pandapower

- Convenient Power System Modelling and Analysis

based on PYPOWER and pandas -

Fraunhofer IWES
Universität Kassel

July 27, 2017
Version 1.4.0
pandapower

Convenient Power System Modelling and Analysis based on PYPOWER and pandas

Lead Developers:
Leon Thurner
Alexander Scheidler

Main Contributors:
Julian Dollichon
Florian Schäfer
Friederike Meier
Jan-Hendrik Menke
Steffen Meinecke
Jakov Krstulović Opara

Further Contributions by:
Tobias Deß
Bastian Junker
Jannis Kupka
Lothar Löwer
Jan Ulfers
Nils Bornhorst
Jonathan Schütt
Elisabeth Drayer

Coordination:
Martin Braun
Johann-Christian Töbermann
Stefan Gehler

Contact:
leon.thurner@uni-kassel.de
alexander.scheidler@iwes.fraunhofer.de
## Contents

1 About pandapower
   1.1 What is pandapower? ................................................. 2
   1.2 Advantages and Contributions .................................. 3
   1.3 A Short Introduction ............................................. 4
   1.4 Unit System and Conventions .................................... 7
   1.5 Tests and Validation .............................................. 8
   1.6 Change Log ...................................................... 16
   1.7 License .......................................................... 19

2 Datastructure and Elements ........................................... 20
   2.1 Empty Network .................................................. 20
   2.2 Bus ..................................................................... 20
   2.3 Line .................................................................... 22
   2.4 Switch .............................................................. 26
   2.5 Load ................................................................... 28
   2.6 Static Generator ................................................. 31
   2.7 External Grid ....................................................... 34
   2.8 Transformer ......................................................... 36
   2.9 Three Winding Transformer ...................................... 41
   2.10 Generator .......................................................... 46
   2.11 Shunt .................................................................. 49
   2.12 Impedance .......................................................... 51
   2.13 Ward .................................................................. 53
   2.14 Extended Ward ..................................................... 55
   2.15 DC Line ................................................................ 57
   2.16 Measurement ....................................................... 59

3 Standard Type Libraries .................................................. 61
   3.1 Basic Standard Types .............................................. 61
   3.2 Manage Standard Types ........................................... 64

4 Power Flow ..................................................................... 67
   4.1 Run a Power Flow .................................................. 67
   4.2 Known Problems and Caveats .................................... 75
   4.3 Diagnostic Function ................................................. 76

5 Short-Circuit .................................................................. 80
   5.1 Running a Short-Circuit Calculation ........................... 80
   5.2 Short-Circuit Currents .............................................. 81
   5.3 Network Elements ................................................... 84

6 State Estimation ............................................................. 89
   6.1 Theoretical Background ............................................ 89
   6.2 Defining Measurements .......................................... 89
   6.3 Running the State Estimation .................................... 90
   6.4 Handling of bad data .............................................. 91
   6.5 Example ............................................................... 92

7 Topological Searches ....................................................... 94
   7.1 Create networkx graph .......................................... 94
   7.2 Topological Searches .............................................. 98
   7.3 Examples .............................................................. 100

8 Networks ....................................................................... 107
   8.1 Example Networks ................................................ 107
   8.2 Simple pandapower test networks ......................... 110
## CONTENTS

8.3 CIGRE Networks ................................................................. 113
8.4 MV Oberrhein ................................................................. 119
8.5 Power System Test Cases ...................................................... 121
8.6 Kerber networks ............................................................. 128
8.7 Synthetic Voltage Control LV Networks .................................. 142
8.8 Dickert LV Networks ......................................................... 147

9 Plotting Networks ............................................................... 150
9.1 Matplotlib Network Plots ..................................................... 150
9.2 Plotly Network Plots .......................................................... 155
9.3 HTML .............................................................................. 163

10 Save and Load Networks ....................................................... 164
10.1 pickle .................................................................. 164
10.2 Excel .................................................................. 164
10.3 Json .................................................................. 165
10.4 SQL .................................................................. 165

11 Converter ............................................................................. 166
11.1 PYPOWER ................................................................. 166
11.2 MATPOWER .............................................................. 168

12 Toolbox .............................................................................. 170
12.1 Result Information ............................................................ 170
12.2 Simulation Setup and Preparation ......................................... 170
12.3 Topology Modification ......................................................... 171
12.4 Item/Element Selection ....................................................... 171
1 About pandapower

pandapower combines the data analysis library pandas and the power flow solver PYPOWER to create an easy to use network calculation program aimed at automation of power system analysis and optimization in distribution and sub-transmission networks.

pandapower is a joint development of the research group Energy Management and Power System Operation, University of Kassel and the Department for Distribution System Operation at the Fraunhofer Institute for Wind Energy and Energy System Technology (IWES), Kassel.

1.1 What is pandapower?

The development of pandapower started as an extension of the widely used power flow solver MATPOWER and its port to python, PYPOWER.

In PYPOWER, the electric attributes of the network are defined in a casefile in the form of a bus/branch model. The bus/branch model formulation is mathematically very close the power flow, which is why it is easy to generate a nodal admittance matrix or other matrices needed for the power flow calculation.

In terms of user friendlyness, there are however some significant drawbacks:

• there is no differentiation between lines and transformers. Furthermore, branch impedances have to be defined in per unit, which is usually not a value directly available from cable or transformer data sheets.
• the casefile only contains pure electrical data. Meta information, such as element names, line lengths or standard types, cannot be saved within the datastructure.
• since there is no API for creating the casefile, networks have to be defined by directly building the matrices.
• the user has to ensure that all bus types (PQ, PV, Slack) are correctly assigned and bus and gen table are coherent.
• power and shunt values can only be assigned as a summed value per bus, the information about individual elements is lost in case of multiple elements at one bus.
• the datastructure is based on matrices, which means deleting one row from the datastructure changes all indices of the following elements.

All these problems make the network definition process prone to errors. pandapower aims to solve these problems by proposing a datastructure based on pandas using PYPOWER to solve the power flow.

pandapower provides

• flexible datastructure for comprehensive modeling of electric power systems
• static electric models for lines, switches, generators, 2/3 winding transformers, ward equivalents etc.
• a convenient interface for static and quasi-static power system analysis

pandapower allows

• automated the creation of complex power system models
• explicit modeling of switches
• solving three phase AC, DC and optimal power flow problems
• topological searches in electric networks
• plotting of structural and/or geographical network plans
• configuring and running state estimation
• static short circuit calculation according to IEC 60909

pandapower does not yet support, but might in the future:

• unbalanced power flow problems
pandapower does not, and most likely never will, support:

- electromagnetic transient simulations
- dynamic short-circuit simulations

If you are interested in contributing to the pandapower project, please contact leon.thurner@uni-kassel.de

### 1.2 Advantages and Contributions

1. **Electric Models**
   - pandapower comes with static equivalent circuit models for lines, 2-Winding transformers, 3-Winding transformers, ward-equivalents etc. (see element documentation for a complete list).
   - Input parameters are intuitive and commonly used model plate parameters (such as line length and resistance per kilometer) instead of parameters like total branch resistance in per unit
   - the pandapower switch model allows modelling of ideal bus-bus switches as well as bus-line / bus-trafo switches
   - the power flow results are processed to include not only the classic power flow results (such as bus voltages and apparent power branch flows), but also line loading or transformer losses

2. **pandapower API**
   - the pandapower API provides create functions for each element to allow automated step-by-step construction of networks
   - the standard type library allows simplified creation of lines, 2-Winding transformers and 3-Winding transformers
   - networks can be saved and loaded to the hard drive with the pickle library

3. **pandapower Datastructure**
   - since variables of any datatype can be stored in the pandas dataframes, electric parameters (integer / float) can be stored together with names (strings), status variables (boolean) etc.
   - variables can be accessed by name instead of by column number of a matrix
   - since all information is stored in pandas tables, all inherent pandas methods can be used to
     - access,
     - query,
     - statistically evaluate,
     - iterate over,
     - visualize,
     - etc.
   - any information that is stored in the pandapower dataframes - be it element parameters, power flow results or a combination of both.

4. **Topological Searches**
   - pandapower networks can be translated into networkx multigraphs for fast topological searches
   - all native networkx algorithms can be used to perform graph searches on pandapower networks
   - pandapower provides some search algorithms specialized on electric power networks

5. **Plotting and Geographical Data**
   - geographical data for buses and lines can be stored in the pandapower datastructure
• networks with geographic information can be plotted using matplotlib
• if no geographical information is available for the buses, generic coordinates can be created through a python-igraph interface

6. State Estimation
• data structure to manage measurements for real-time simulations
• WLS state estimation generates an exact grid state out of unexact measurements
• WLS as the industry standard is a good reference for evaluating new state estimation developments
• bad data detection and filtering methods improve performance of the state estimator

7. Powerflow
• accelerated with a numba implementation that allows very fast construction of nodal point admittance and jacobian matrices
• includes a topology check to allow convergence with unsupplied network areas
• different possibilities for initialization of power flow, including from DC power flow or from previous results

8. Short-Circuit Calculation
• pandapower includes a short-circuit calculation with correction factors according to IEC 60909
• symmetrical three-phase and unsymmetrical two-phase currents can be calculated
• vectorized implementation allows fast calculation of short-circuit currents including branch flow results

1.3 A Short Introduction
pandapower combines the data analysis library pandas and the power flow solver PYPOWER to create an easy to use network calculation tool aimed at automation of analysis and optimization in power systems.

Datastructure
A network in pandapower is represented in a pandapowerNet object, which is a collection of pandas Dataframes. Each dataframe in a pandapowerNet contains the information about one pandapower element, such as line, load transformer etc.

We consider the following simple 3-bus example network as a minimal example:
To create this network in pandapower, we first create an empty pandapower network:

```python
import pandapower as pp
net = pp.create_empty_network()
```

we then create the buses with the given voltage levels:

```python
b1 = pp.create_bus(net, vn_kv=20., name="Bus 1")
b2 = pp.create_bus(net, vn_kv=0.4, name="Bus 2")
b3 = pp.create_bus(net, vn_kv=0.4, name="Bus 3")
```

we then create the bus elements, namely a grid connection at Bus 1 and an load at Bus 3:

```python
pp.create_ext_grid(net, bus=b1, vm_pu=1.02, name="Grid Connection")
pp.create_load(net, bus=b3, p_kw=100, q_kvar=50, name="Load")
```

We now create the branch elements. First, we create the transformer from the type data as it is given in the network description:

```python
tid = pp.create_transformer_from_parameters(net, sn_kva=400.,
                                          hv_bus=b1, lv_bus=b2,
                                          hv_bus=b1, lv_bus=b2,
                                          vn_hv_kv=20., vn_lv_kv=0.4,
                                          vsc_percent=6., vscr_percent=1.425,
                                          10_percent=0.3375, pfe_kw=1.35,
                                          name="Trafo")
```

Note that you do not have to calculate any impedances or tap ratio for the equivalent circuit, this is handled internally by pandapower according to the pandapower transformer model. The transformer model and all other pandapower electric elements are validated against commercial software.

The standard type library allows even easier creation of the transformer. The parameters given above are the
parameters of the transformer “0.4 MVA 20/0.4 kV” from the pandapower basic standard types. The transformer can be created from the standard type library like this:

```python
tid = pp.create_transformer(net, hv_bus=b1, lv_bus=b2, std_type="0.4 MVA 20/0.4 kV", name="Trafo")
```

The same applies to the line, which can either be created by parameters:

```python
pp.create_line_from_parameters(net, from_bus=b2, to_bus=b3,
                             r_ohm_per_km=0.642, x_ohm_per_km=0.083,
                             c_nf_per_km=210, max_i_ka=0.142, name="Line")
```
or from the standard type library:

```python
pp.create_line(net, from_bus=b2, to_bus=b3, length_km=0.1, name="Line",
               std_type="NAYY 4x50 SE")
```

the pandapower representation now looks like this:

This is the version where transformer and line have been created through the standard type libraries, which is why the line has a specified type (cs for cable system) and the transformer has a tap changer, both of which are defined in the type data.

**Running a Power Flow**

A powerflow can be carried out with the `runpp` function:

```python
pp.runpp(net)
```

When a power flow is run, pandapower combines the information of all element tables into one pypower case file and uses pypower to run the power flow. The results are then processed and written back into pandapower:

For the 3-bus example network, the result tables look like this:
You can download the python script that creates this 3-bus system [here](#).

For a more in depth introduction into pandapower modeling and analysis functionality, see the pandapower tutorials about network creation, standard type libraries, power flow, topological searches, plotting and more.

### 1.4 Unit System and Conventions

**Naming Conventions**

Parameters are always named in the form of `<parameter>_uunit`, such as:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>read as</th>
</tr>
</thead>
<tbody>
<tr>
<td>vm_pu</td>
<td>$v_m [pu]$</td>
</tr>
<tr>
<td>loading_percent</td>
<td>$loading [%]$</td>
</tr>
<tr>
<td>pl_kw</td>
<td>$p_l [kw]$</td>
</tr>
<tr>
<td>r_ohm_per_km</td>
<td>$r [Ω/km]$</td>
</tr>
</tbody>
</table>

Constraint parameters are always named with max or min as the prefix to the variable which is constrained, for example:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>read as</th>
</tr>
</thead>
<tbody>
<tr>
<td>min_vm_pu</td>
<td>$v_m^{\text{min}} [pu]$</td>
</tr>
<tr>
<td>max_loading_percent</td>
<td>$loading^{\text{max}} [%]$</td>
</tr>
<tr>
<td>max_p_kw</td>
<td>$p_l^{\text{max}} [kw]$</td>
</tr>
<tr>
<td>min_q_kvar</td>
<td>$q_k^{\text{min}} [kvar]$</td>
</tr>
</tbody>
</table>

It is advised to keep consistent with these naming conventions when extending the framework and introducing new parameters.

**Three Phase System**

For the three phase system, the following conventions apply:

- Voltage values are given as phase-to-phase voltages
- Current values are given as phase currents
- Power values are given as three-phase power flows

The power equation in the three phase system is therefore given as $S = \sqrt{3} \cdot V \cdot I$. 

```python
PandapowerNet

<table>
<thead>
<tr>
<th></th>
<th>index</th>
<th>p_from_kw</th>
<th>q_from_kvar</th>
<th>p_to_kw</th>
<th>q_to_kvar</th>
<th>pl_kw</th>
<th>q_kvar</th>
<th>i_kw</th>
<th>loading_percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>105.39</td>
<td>50.00</td>
<td>100.00</td>
<td>50.00</td>
<td>5.39</td>
<td>0.70</td>
<td>0.167</td>
<td>117.04</td>
</tr>
<tr>
<td>1</td>
<td>1.088</td>
<td>107.265</td>
<td>52.875</td>
<td>105.39</td>
<td>50.09</td>
<td>1.873</td>
<td>1.979</td>
<td>0.003</td>
<td>29.29</td>
</tr>
<tr>
<td>2</td>
<td>0.944</td>
<td>107.265</td>
<td>52.875</td>
<td>105.39</td>
<td>50.09</td>
<td>1.873</td>
<td>1.979</td>
<td>0.003</td>
<td>29.29</td>
</tr>
</tbody>
</table>
```

```python
PandapowerNet

<table>
<thead>
<tr>
<th></th>
<th>index</th>
<th>p_from_kw</th>
<th>q_from_kvar</th>
<th>p_to_kw</th>
<th>q_to_kvar</th>
<th>pl_kw</th>
<th>q_kvar</th>
<th>i_kw</th>
<th>loading_percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>105.39</td>
<td>50.00</td>
<td>100.00</td>
<td>50.00</td>
<td>5.39</td>
<td>0.70</td>
<td>0.167</td>
<td>117.04</td>
</tr>
<tr>
<td>1</td>
<td>1.088</td>
<td>107.265</td>
<td>52.875</td>
<td>105.39</td>
<td>50.09</td>
<td>1.873</td>
<td>1.979</td>
<td>0.003</td>
<td>29.29</td>
</tr>
<tr>
<td>2</td>
<td>0.944</td>
<td>107.265</td>
<td>52.875</td>
<td>105.39</td>
<td>50.09</td>
<td>1.873</td>
<td>1.979</td>
<td>0.003</td>
<td>29.29</td>
</tr>
</tbody>
</table>
```

```python
PandapowerNet

<table>
<thead>
<tr>
<th></th>
<th>index</th>
<th>p_from_kw</th>
<th>q_from_kvar</th>
<th>p_to_kw</th>
<th>q_to_kvar</th>
<th>pl_kw</th>
<th>q_kvar</th>
<th>i_kw</th>
<th>loading_percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>105.39</td>
<td>50.00</td>
<td>100.00</td>
<td>50.00</td>
<td>5.39</td>
<td>0.70</td>
<td>0.167</td>
<td>117.04</td>
</tr>
<tr>
<td>1</td>
<td>1.088</td>
<td>107.265</td>
<td>52.875</td>
<td>105.39</td>
<td>50.09</td>
<td>1.873</td>
<td>1.979</td>
<td>0.003</td>
<td>29.29</td>
</tr>
<tr>
<td>2</td>
<td>0.944</td>
<td>107.265</td>
<td>52.875</td>
<td>105.39</td>
<td>50.09</td>
<td>1.873</td>
<td>1.979</td>
<td>0.003</td>
<td>29.29</td>
</tr>
</tbody>
</table>
```
Since pandapower was developed for distribution systems, all power values are given in kW or kVar.

**Per Unit System**

Bus voltages are given in the per unit system. The per unit values are relative to the phase-to-phase voltages defined in `net.bus.vn_kv` for each bus.

The rated apparent power for the per unit system can be defined with the `net.sn_kva` parameter when creating an empty network. The default value is $S_N = 1000 \text{kVA}$. The value should not be relevant in most applications since all power values are given in physical units.

**Signing System**

For all bus-based power values, the signing is based on the consumer viewpoint:

- Positive active power is power consumption, negative active power is power generation
- Positive reactive power is inductive consumption, negative reactive power is capacitive consumption

The power flow values for branch elements (lines & transformer) are always defined as the power flow into the branch element.

**Frequency**

The frequency can be defined when creating an empty network. The frequency is only used to calculate the shunt admittance of lines, since the line reactance is given directly in ohm per kilometer. The frequency is also relevant when calculating the peak factor $\psi$ in the short circuit calculation.

The standard frequency in pandapower is 50 Hz, and the pandapower standard types are also chosen for 50 Hz systems. If you use a different frequency, please be aware that the line reactance values might not be realistic.

### 1.5 Tests and Validation

#### 1.5.1 Unit Tests

**1.5.2 Test Suite**

pandapower is tested with pytest. There are currently over 220 tests testing all kinds of pandapower functionality. The tests also include automatic validation of pandapower results from power flow or short circuit calculations against commercial software, to ensure that the implementation is correct.

The complete test suite can be run with:

```python
import pandapower.test
pandapower.test.run_all_tests()
```

If all packages are installed correctly, all tests should pass.

#### 1.5.3 Continuous Integration Testing

The tests are continously carried out with Travis CI in Python 2.7, 3.4, 3.5 and 3.6: The test coverage rate is checked with codecov, code quality with codacy:

#### 1.5.4 Model and Loadflow Validation

To ensure that pandapower loadflow results are correct, all pandapower element behaviour is tested against DlgsILENT PowerFactory or PSS Sincal.

There is a result test for each of the pandapower elements that checks loadflow results in pandapower against results from a commercial tools. The results are compared with the following tolerances:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Max. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Max. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voltage Magnitude</td>
<td>0.000001 pu</td>
</tr>
<tr>
<td>Voltage Angle</td>
<td>0.01°</td>
</tr>
<tr>
<td>Current</td>
<td>0.000001 kA</td>
</tr>
<tr>
<td>Power</td>
<td>0.005 kW</td>
</tr>
<tr>
<td>Element Loading</td>
<td>0.001%</td>
</tr>
</tbody>
</table>

#### 1.5.5 Example: Transformer Model Validation

To validate the pandapower transformer model, a transformer is created with the same parameters in pandapower and PowerFactory. To test all aspects of the model we use a transformer with

- both iron and copper losses > 0
- nominal voltages that deviate from the nominal bus voltages at both sides
- an active tap changer
- a voltage angle shift > 0

We use a transformer with the following parameters:

- vsc_percent= 5.0
- vscr_percent = 2.0
- i0_percent = 0.4
- pfe_kw = 2.0
- sn_kva = 400
- vn_hv_kv = 22
- vn_lv_kv = 0.42
- tp_max = 10
- tp_mid = 5
- tp_min = 0
- tp_st_percent = 1.25
- tp_side = “hv”
- tp_pos = 3
- shift_degree = 150

To validate the in_service parameter as well as the transformer switch element, we create three transformers in parallel: one in service, one out of service and one with an open switch in open loop operation. All three transformers are connected to a 20kV / 0.4 kV bus network. The test network then looks like this:
The loadflow result for the exact same network are now compared in pandapower and PowerFactory. It can be seen that both bus voltages:

<table>
<thead>
<tr>
<th>Name</th>
<th>U_Magnitude</th>
<th>U_Angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus1</td>
<td>1.070000</td>
<td>0.000000</td>
</tr>
<tr>
<td>Bus2</td>
<td>1.010750</td>
<td>-0.068061</td>
</tr>
<tr>
<td>Bus3</td>
<td>0.970772</td>
<td>-151.416626</td>
</tr>
</tbody>
</table>

and transformer results:

<table>
<thead>
<tr>
<th>Name</th>
<th>P_LV-side in kW</th>
<th>Q_LV-side in kVAR</th>
<th>P_HV-side in kW</th>
<th>Q_HV-side in kVAR</th>
<th>Current in VA</th>
<th>Current in kA</th>
<th>HV-side in kV</th>
<th>LV-side in kV</th>
<th>loading_percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer</td>
<td>1.774</td>
<td>-0.034</td>
<td>-0.000</td>
<td>0.000</td>
<td>0.0000000000</td>
<td>0.0000000000</td>
<td>1.810150</td>
<td>-0.066862</td>
<td>0.488000</td>
</tr>
<tr>
<td>Transformer</td>
<td>1.774</td>
<td>-0.034</td>
<td>-0.000</td>
<td>0.000</td>
<td>0.0000000000</td>
<td>0.0000000000</td>
<td>1.810150</td>
<td>-0.066862</td>
<td>0.488000</td>
</tr>
</tbody>
</table>

match within the margins defined above.

### 1.5.6 All Test Networks

There is a test network for the validation of each pandapower element in the same way the transformer model is tested.

The PowerFactory file containing all test networks can be downloaded [here](#). The correlating pandapower networks are defined in result_test_network_generator.py in the pandapower/test module. The tests that check pandapower results against PowerFactory results are located in pandapower/test/test_results.py.
1 ABOUT PANDAPOWER

line

load and sgen

trafo
impedance

ward
1 ABOUT PANDAPOWER

xward

switch
1.6 Change Log

1.6.1 [1.4.0] - 2017-07-27

- [ADDED] possibility to save networks to an sql database
- [CHANGED] major change in fileIO: all networks are converted to a uniform dataframe only version before they are saved as excel, json or sql. Old files can still be loaded, but all files saved with v1.4 can only be loaded with v1.4!
- [FIXED] all tests now pass if numba is not installed (although pandapower might be slow without numba)
- [FIXED] state estimation bug with phase shift transformers
- [CHANGED] OPF now raises specific warning if parameters are missing instead of generic exception
- [ADDED]
- [ADDED] Dickert LV Networks

1.6.2 [1.3.1] - 2017-06-16

- [CHANGED] to_pickle saves only python datatypes and no pickle objects
- [ADDED] html representation of pandapower nets
- [ADDED] collections for trafos, loads, ext_grids
- [CHANGED] renamed create_shunt_as_condensator to create_shunt_as_capacitor
- [FIXED] mock problem in create docstrings
- [ADDED] Synthetic Voltage Control LV Networks

1.6.3 [1.3.0] - 2017-05-10

- [ADDED] ZIP loads integrated in power flow
- [ADDED] numba implementation of dissolving switch buses
- [ADDED] Current source representation of full converter elements in short circuit calculations
- [ADDED] Method C for calculation of factor kappa in short circuit calculation
• [CHANGED] Speedup for calculation of branch short circuit currents
• [CHANGED] Branch results for minimum short circuit calculations are calculated as minimal currents
• [ADDED] Interactive plots with plotly
• [CHANGED] included pypower files for power flow and index files
• [FIXED] compatibility with numpy 1.12
• [CHANGED] -1 is a valid value for net.bus_geodata.x
• [CHANGED] allow transformers with negative xk to provide large scale IEEE cases (RTE, PEGASE, Polish)
• [ADDED] large scale IEEE cases (RTE, PEGASE, Polish)
• [ADDED] rated voltage and step variable for shunts
• [ADDED] lagrange multiplier included in bus results after OPF

1.6.4 [1.2.2] - 2017-03-22
• [CHANGED] Minor refactoring in pd2ppc
• [ADDED] Technical Report

1.6.5 [1.2.1] - 2017-03-21
• [FIXED] Readme for PyPi

1.6.6 [1.2.0] - 2017-03-21
• [CHANGED] net.line.imax_ka to net.line.max_i_ka for consistency reasons
• [ADDED] net.line.tp_st_degree for phase shift in trafo tap changers
• [ADDED] sn_kva parameter in create_empty network for per unit system reference power
• [ADDED] parameter parallel for trafo element
• [ADDED] connectivity check for power flow to deal with disconnected network areas
• [ADDED] backward/forward sweep power flow algorithm specially suited for radial and weakly-meshed networks
• [ADDED] linear piece wise and polynomial OPF cost functions
• [ADDED] possibility to make loads controllable in OPF
• [ADDED] to_json and from_json functions to save/load networks with a JSON format
• [ADDED] generator lookup to allow multiple generators at one bus
• [CHANGED] Initialization of calculate_voltage_angles and init for high voltage networks
• [ADDED] bad data detection for state estimation
• [CHANGED] from_ppc: no detect_trafo anymore, several gen at each node possible
• [CHANGED] validate_from_ppc: improved validation behaviour by means of duplicated gen and branch rearangement
• [ADDED] networks: case33bw, case118, case300, case1354pegase, case2869pegase, case9241pegase, GBreducednetwork, GBNetwork, iceland, ciger_network_mv with_der='all’ der
• [ADDED] possibility to add fault impedance for short-circuit current calculation
• [ADDED] branch results for short circuits
• [ADDED] static generator model for short circuits
• [ADDED] three winding transformer model for short circuits
• [FIXED] correctly neglecting shunts and tap changer position for short-circuits
• [ADDED] two phase short-circuit current calculation
• [ADDED] tests for short circuit currents with validation against DIgSILENT PowerFactory

1.6.7 [1.1.1] - 2017-01-12

• [ADDED] installation description and pypi files from github
• [ADDED] automatic inversion of active power limits in convert format to account for convention change in version 1.1.0
• [CHANGED] install_requires in setup.py

1.6.8 [1.1.0] - 2017-01-11

• [ADDED] impedance element can now be used with unsymmetric impedances $z_{ij} \neq z_{ji}$
• [ADDED] dcline element that allows modelling DC lines in PF and OPF
• [ADDED] simple plotting function: call `pp.simple_plot(net)` to directly plot the network
• [ADDED] measurement table for networks. Enables the definition of measurements for real-time simulations.
• [ADDED] estimation module, which provides state estimation functionality with weighted least squares algorithm
• [ADDED] shortcircuit module in beta version for short-circuit calculation according to IEC-60909
• [ADDED] documentation of model validation and tests
• [ADDED] case14, case24_ieee_rts, case39, case57 networks
• [ADDED] mpc and ppc converter
• [CHANGED] convention for active power limits of generators. Generator with max. feed in of 50kW before: $p_{min\_kw}=0$, $p_{max\_kw}=-50$. Now $p_{max\_kw}=0$, $p_{min\_kw}=50$
• [ADDED] DC power flow function `pp.rundcopp`
• [FIXED] bug in create_transformer function for `tp_pos` parameter
• [FIXED] bug in voltage ratio for low voltage side tap changers
• [FIXED] bug in rated voltage calculation for opf line constraints

1.6.9 [1.0.2] - 2016-11-30

• [CHANGED] changed in_service dtype from f8 to bool for shunt, ward, xward
• [CHANGED] included `i_from_ka` and `i_to_ka` in `net.res_line`
• [ADDED] recycle parameter added. `ppc, Ybus, _is_elements` and `bus_lookup` can be reused between multiple powerflows if `recycle["ppc"] == True`, ppc values (P,Q,V) only get updated.
• [FIXED] OPF bugfixes: cost scaling, correct calculation of res_bus.p_kw for sgens
• [ADDED] loadcase added as pypower_extension since unnecessary deepcopies were removed
• [CHANGED] suppress warnings parameter removed from loadflow, casting warnings are automatically suppressed
1.6.10 [1.0.1] - 2016-11-09

- [CHANGED] update short introduction example to include transformer
- [CHANGED] included pypower in setup.py requirements (only pypower, not numpy, scipy etc.)
- [CHANGED] mpc / ppc renamed to ppci / ppc
- [FIXED] MANIFEST.ini includes all relevant doc files and exclude report
- [FIXED] handling of tp_pos parameter in create_trafo and create_trafo3w
- [FIXED] init="result" for open bus-line switches

1.7 License

pandapower is published under the following 3-clause BSD license:

Copyright (c) 2016 by University of Kassel and Fraunhofer Institute for Wind-
→Energy and Power
Systems Technology (IWES) Kassel and individual contributors (see AUTHORS file for-
details).
All rights reserved.

Redistribution and use in source and binary forms, with or without modification,pro-
vided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this
list of conditions
and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this
list of
conditions and the following disclaimer in the documentation and/or other
materials provided
with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors may
be used to
derose or promote products derived from this software without specific prior
written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY,
OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY
WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
2 Datastructure and Elements

A pandapower network consists of an element table for each electric element in the network. Each element table consists of a column for each parameter and a row for each element.

pandapower provides electric models for 13 electric elements, for each of which you can find detailed information about the definition and interpretation of the parameters in the following documentation:

2.1 Empty Network

2.1.1 Create Function

```python
pandapower.create_empty_network(name='', f_hz=50.0, sn_kva=1000.0)
```

This function initializes the pandapower datastructure.

**OPTIONAL:**
- `f_hz` (float, 50.) - power system frequency in hertz
- `name` (string, None) - name for the network

**OUTPUT:**
- `net` (attrdict) - PANDAPOWER attrdict with empty tables:

**EXAMPLE:**
```python
net = create_empty_network()
```

2.2 Bus

See also:

*Unit Systems and Conventions*

2.2.1 Create Function

```python
pandapower.create_bus(net, vn_kv, name=None, index=None, geodata=None, type="b", zone=None, in_service=True, max_vm_pu=nan, min_vm_pu=nan)
```

Adds one bus in table net['bus'].

**INPUT:**
- `net` (pandapowerNet) - The pandapower network in which the element is created
- **OPTIONAL:**
  - `name` (string, default None) - the name for this bus
  - `index` (int, default None) - Force a specified ID if it is available. If None, the index one higher than the highest already existing index is selected.
  - `vn_kv` (float) - The grid voltage level.
  - `geodata` ((x,y)-tuple, default None) - coordinates used for plotting
  - `type` (string, default “b”) - Type of the bus. “n” - auxilary node, “b” - busbar, “m” - muff
  - `zone` (string, None) - grid region
  - `in_service` (boolean) - True for in_service or False for out of service
  - `max_vm_pu` (float, NAN) - Maximum bus voltage in p.u. - necessary for OPF
  - `min_vm_pu` (float, NAN) - Minimum bus voltage in p.u. - necessary for OPF

**OUTPUT:**
- `index` (int) - The unique ID of the created element

**EXAMPLE:**
```python
create_bus(net, name = "bus1")
```
2.2.2 Input Parameters

*net.bus*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Value Range</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td></td>
<td>name of the bus</td>
</tr>
<tr>
<td>vn_kv*</td>
<td>float</td>
<td>&gt; 0</td>
<td>rated voltage of the bus [kV]</td>
</tr>
<tr>
<td>type</td>
<td>string</td>
<td>naming conventions: “n” - node “b” - busbar “m” - muff</td>
<td>type variable to classify buses</td>
</tr>
<tr>
<td>zone</td>
<td>string</td>
<td></td>
<td>can be used to group buses, for example network groups / regions</td>
</tr>
<tr>
<td>max_vm_pu**</td>
<td>float</td>
<td>&gt; 0</td>
<td>Maximum voltage</td>
</tr>
<tr>
<td>min_vm_pu**</td>
<td>float</td>
<td>&gt; 0</td>
<td>Minimum voltage</td>
</tr>
<tr>
<td>in_service*</td>
<td>boolean</td>
<td>True / False</td>
<td>specifies if the bus is in service.</td>
</tr>
</tbody>
</table>

*necessary for executing a power flow calculation **optimal power flow parameter

**Note:** Bus voltage limits can not be set for slack buses and will be ignored by the optimal power flow.

*net.bus_geodata*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>float</td>
<td>x coordinate of bus location</td>
</tr>
<tr>
<td>y</td>
<td>float</td>
<td>y coordinate of bus location</td>
</tr>
</tbody>
</table>

2.2.3 Electric Model

2.2.4 Result Parameters

*net.res_bus*
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>vm_pu</td>
<td>float</td>
<td>voltage magnitude [p.u]</td>
</tr>
<tr>
<td>va_degree</td>
<td>float</td>
<td>voltage angle [degree]</td>
</tr>
<tr>
<td>p_kw</td>
<td>float</td>
<td>resulting active power demand [kW]</td>
</tr>
<tr>
<td>q_kvar</td>
<td>float</td>
<td>resulting reactive power demand [kvar]</td>
</tr>
</tbody>
</table>

The power flow bus results are defined as:

\[
vm_{pu} = |V_{bus}|
\]

\[
va_{degree} = \angle V_{bus}
\]

\[
p_{kw} = Re(\sum_{n=1}^{N} S_{bus,n})
\]

\[
q_{kvar} = Im(\sum_{n=1}^{N} S_{bus,n})
\]

**net.res_bus_est**

The state estimation results are put into **net.res_bus_est** with the same definition as in **net.res_bus**.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>vm_pu</td>
<td>float</td>
<td>voltage magnitude [p.u]</td>
</tr>
<tr>
<td>va_degree</td>
<td>float</td>
<td>voltage angle [degree]</td>
</tr>
<tr>
<td>p_kw</td>
<td>float</td>
<td>resulting active power demand [kW]</td>
</tr>
<tr>
<td>q_kvar</td>
<td>float</td>
<td>resulting reactive power demand [kvar]</td>
</tr>
</tbody>
</table>

**Note:** All power values are given in the consumer system. Therefore a bus with positive p_kw value consumes power while a bus with negative active power supplies power.

### 2.3 Line

**See also:**

*Unit Systems and Conventions Standard Type Libraries*

#### 2.3.1 Create Function

Lines can be either created from the standard type library (create_line) or with custom values (create_line_from_parameters).

```python
pandapower.create_line(net, from_bus, to_bus, length_km, std_type, name=None, index=None, geodata=None, df=1., parallel=1, in_service=True, max_loading_percent=nan)
```

Creates a line element in net["line"] The line parameters are defined through the standard type library.

**INPUT:**

- net - The net within this line should be created
  - from_bus (int) - ID of the bus on one side which the line will be connected with
  - to_bus (int) - ID of the bus on the other side which the line will be connected with
  - length_km (float) - The line length in km
  - std_type (string) - The linetype of a standard line pre-defined in standard_linetypes.

**OPTIONAL:**

- name (string) - A custom name for this line
**index** (int, None) - Force a specified ID if it is available. If None, the index one higher than the highest already existing index is selected.

**geodata** (array, default None, shape= (2L)) - The line geodata of the line. The first row should be the coordinates of bus a and the last should be the coordinates of bus b. The points in the middle represent the bending points of the line

**in_service** (boolean) - True for in_service or False for out of service

**df** (float) - derating factor: maximal current of line in relation to nominal current of line (from 0 to 1)

**parallel** (integer) - number of parallel line systems

**max_loading_percent** (float) - maximum current loading (only needed for OPF)

**OUTPUT:** index (int) - The unique ID of the created line

**EXAMPLE:**
create_line(net, “line1”, from_bus = 0, to_bus = 1, length_km=0.1, std_type=”NAYY 4x50 SE”)

```python
pandapower.create_line_from_parameters(net, from_bus, to_bus, length_km, r_ohm_per_km, x_ohm_per_km, c_nf_per_km, max_i_ka, name=None, index=None, type=None, geodata=None, in_service=True, df=1., parallel=1, max_loading_percent=nan, **kwargs)
```

Creates a line element in net[“line”] from line parameters.

**INPUT:** net - The net within this line should be created

**from_bus** (int) - ID of the bus on one side which the line will be connected with

**to_bus** (int) - ID of the bus on the other side which the line will be connected with

**length_km** (float) - The line length in km

**r_ohm_per_km** (float) - line resistance in ohm per km

**x_ohm_per_km** (float) - line reactance in ohm per km

**c_nf_per_km** (float) - line capacitance in nF per km

**max_i_ka** (float) - maximum thermal current in kA

**OPTIONAL:** name (string) - A custom name for this line

**index** (int, None) - Force a specified ID if it is available. If None, the index one higher than the highest already existing index is selected.

**in_service** (boolean) - True for in_service or False for out of service

**type** (str) - type of line (“ol” for overhead line or “cs” for cable system)

**df** (float) - derating factor: maximal current of line in relation to nominal current of line (from 0 to 1)

**parallel** (integer) - number of parallel line systems

**geodata** (array, default None, shape= (2L)) - The line geodata of the line. The first row should be the coordinates of bus a and the last should be the coordinates of bus b. The points in the middle represent the bending points of the line

**kwarg**s - nothing to see here, go along

**max_loading_percent** (float) - maximum current loading (only needed for OPF)

**OUTPUT:** index (int) - The unique ID of the created line

**EXAMPLE:**
create_line_from_parameters(net, “line1”, from_bus = 0, to_bus = 1, length_km=0.1, r_ohm_per_km =.01, x_ohm_per_km = 0.05, c_nf_per_km = 10, max_i_ka = 0.4)
### 2.3.2 Input Parameters

#### net.line

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Value Range</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td></td>
<td>name of the line</td>
</tr>
<tr>
<td>std_type</td>
<td>string</td>
<td></td>
<td>standard type which can be used to easily define line parameters with the pandapower standard type library</td>
</tr>
<tr>
<td>from_bus*</td>
<td>integer</td>
<td></td>
<td>Index of bus where the line starts</td>
</tr>
<tr>
<td>to_bus*</td>
<td>integer</td>
<td></td>
<td>Index of bus where the line ends</td>
</tr>
<tr>
<td>length_km*</td>
<td>float</td>
<td>&gt; 0</td>
<td>length of the line [km]</td>
</tr>
<tr>
<td>r_ohm_per_km*</td>
<td>float</td>
<td>≥ 0</td>
<td>resistance of the line [Ohm per km]</td>
</tr>
<tr>
<td>x_ohm_per_km*</td>
<td>float</td>
<td>≥ 0</td>
<td>inductance of the line [Ohm per km]</td>
</tr>
<tr>
<td>c_nf_per_km*</td>
<td>float</td>
<td>≥ 0</td>
<td>capacitance of the line [nF per km]</td>
</tr>
<tr>
<td>max_i_ka*</td>
<td>float</td>
<td>&gt; 0</td>
<td>maximal thermal current [kA]</td>
</tr>
<tr>
<td>parallel*</td>
<td>integer</td>
<td>≥ 1</td>
<td>number of parallel line systems</td>
</tr>
<tr>
<td>df*</td>
<td>float</td>
<td>0...1</td>
<td>derating factor (scaling) for max_i_ka</td>
</tr>
<tr>
<td>type</td>
<td>string</td>
<td></td>
<td>type of line</td>
</tr>
<tr>
<td>max_loading_percent*</td>
<td>float</td>
<td>&gt; 0</td>
<td>Maximum loading of the line</td>
</tr>
<tr>
<td>endtemp_degree***</td>
<td>float</td>
<td>&gt; 0</td>
<td>Short-Circuit end temperature of the line</td>
</tr>
<tr>
<td>in_service*</td>
<td>boolean</td>
<td>True / False</td>
<td>specifies if the line is in service</td>
</tr>
</tbody>
</table>

*necessary for executing a power flow calculation **optimal power flow parameter ***short-circuit calculation parameter

**Note:** Defining a line with length zero leads to a division by zero in the power flow and is therefore not allowed. Lines with a very low impedance might lead to convergence problems in the power flow for the same reason. If you want to directly connect two buses, please use the switch element instead of a line with a small impedance!

#### net.line_geodata

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>coords</td>
<td>list</td>
<td>List of (x,y) tuples that mark the inflexion points of the line</td>
</tr>
</tbody>
</table>

### 2.3.3 Electric Model

Lines are modelled with the \( \pi \)-equivalent circuit:
The elements in the equivalent circuit are calculated from the parameters in the net.line dataframe as:

\[
Z = (r_{\text{ohm}\_\text{per}\_\text{km}} + j \cdot x_{\text{ohm}\_\text{per}\_\text{km}}) \cdot \frac{\text{length\_km}}{\text{parallel}}
\]

\[
Y = j \cdot 2\pi f \cdot c_{\text{nf}\_\text{per}\_\text{km}} \cdot 1 \cdot 10^{-9} \cdot \text{length\_km} \cdot \text{parallel}
\]

The power system frequency \( f \) is defined when creating an empty network, the default value is \( f = 50Hz \).

The parameters are then transformed in the per unit system:

\[
Z_N = \frac{V_N^2}{S_N}
\]

\[
\hat{Z} = \frac{Z}{Z_N}
\]

\[
\hat{Y} = Y \cdot Z_N
\]

Where the reference voltage \( V_N \) is the nominal voltage at the from bus and the rated apparent power \( S_N \) is defined system wide in the net object (see Unit Systems and Conventions).

**Note:** pandapower assumes that nominal voltage of from bus and to bus are equal, which means pandapower does not support lines that connect different voltage levels. If you want to connect different voltage levels, either use a transformer or an impedance element.

### 2.3.4 Result Parameters

**net.res_line**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_from_kw</td>
<td>float</td>
<td>active power flow into the line at “from” bus [kW]</td>
</tr>
<tr>
<td>q_from_kvar</td>
<td>float</td>
<td>reactive power flow into the line at “from” bus [kVar]</td>
</tr>
<tr>
<td>p_to_kw</td>
<td>float</td>
<td>active power flow into the line at “to” bus [kW]</td>
</tr>
<tr>
<td>q_to_kvar</td>
<td>float</td>
<td>reactive power flow into the line at “to” bus [kVar]</td>
</tr>
<tr>
<td>pl_kw</td>
<td>float</td>
<td>active power losses of the line [kW]</td>
</tr>
<tr>
<td>ql_kvar</td>
<td>float</td>
<td>reactive power consumption of the line [kVar]</td>
</tr>
<tr>
<td>i_from_ka</td>
<td>float</td>
<td>Current at to bus [kA]</td>
</tr>
<tr>
<td>i_to_ka</td>
<td>float</td>
<td>Current at from bus [kA]</td>
</tr>
<tr>
<td>i_ka</td>
<td>float</td>
<td>Maximum of i_from_ka and i_to_ka [kA]</td>
</tr>
<tr>
<td>loading_percent</td>
<td>float</td>
<td>line loading [%]</td>
</tr>
</tbody>
</table>
The power flow results in the net.res_line table are defined as:

\[
\begin{align*}
    p_{\text{from kw}} &= \Re(\mathbf{E}_{\text{from}} \cdot \mathbf{I}_{\text{from}}^*) \\
    q_{\text{from kvar}} &= \Im(\mathbf{E}_{\text{from}} \cdot \mathbf{I}_{\text{from}}^*) \\
    p_{\text{to kw}} &= \Re(\mathbf{E}_{\text{to}} \cdot \mathbf{I}_{\text{to}}^*) \\
    q_{\text{to kvar}} &= \Im(\mathbf{E}_{\text{to}} \cdot \mathbf{I}_{\text{to}}^*) \\
    p_l_{\text{kw}} &= p_{\text{from kw}} + p_{\text{to kw}} \\
    q_l_{\text{kvar}} &= q_{\text{from kvar}} + q_{\text{to kvar}} \\
    i_{\text{from ka}} &= i_{\text{from}} \\
    i_{\text{to ka}} &= i_{\text{to}} \\
    i_{\text{ka}} &= \max(i_{\text{from}}, i_{\text{to}}) \\
    \text{loading\_percent} &= \frac{i_{\text{ka}}}{i_{\text{max\_ka}} \cdot df \cdot \text{parallel}} \cdot 100
\end{align*}
\]

\text{net.res\_line\_est}

The state estimation results are put into net.res_line_est with the same definition as in net.res_line.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_from_kw</td>
<td>float</td>
<td>active power flow into the line at “from” bus [kW]</td>
</tr>
<tr>
<td>q_from_kvar</td>
<td>float</td>
<td>reactive power flow into the line at “from” bus [kVar]</td>
</tr>
<tr>
<td>p_to_kw</td>
<td>float</td>
<td>active power flow into the line at “to” bus [kW]</td>
</tr>
<tr>
<td>q_to_kvar</td>
<td>float</td>
<td>reactive power flow into the line at “to” bus [kVar]</td>
</tr>
<tr>
<td>p_l_kw</td>
<td>float</td>
<td>active power losses of the line [kW]</td>
</tr>
<tr>
<td>q_l_kvar</td>
<td>float</td>
<td>reactive power consumption of the line [kVar]</td>
</tr>
<tr>
<td>i_from_ka</td>
<td>float</td>
<td>Current at to bus [kA]</td>
</tr>
<tr>
<td>i_to_ka</td>
<td>float</td>
<td>Current at from bus [kA]</td>
</tr>
<tr>
<td>i_ka</td>
<td>float</td>
<td>Maximum of i_from_ka and i_to_ka [kA]</td>
</tr>
<tr>
<td>loading_percent</td>
<td>float</td>
<td>line loading [%]</td>
</tr>
</tbody>
</table>

2.4 Switch

2.4.1 Create Function

\text{pandapower.create\_switch}(\text{net, bus, element, et, closed=True, type=None, name=None, index=None})

Adds a switch in the net[“switch”] table.

Switches can be either between to buses (bus-bus switch) or at the end of a line or transformer element (bus-elememnt switch).

Two buses that are connected through a closed bus-bus switches are fused in the power flow if the switch es closed or separated if the switch is open.

An element that is connected to a bus through a bus-element switch is connected to the bus if the switch is closed or disconnected if the switch is open.

INPUT: net (pandapowerNet) - The net within this transformer should be created

bus - The bus that the switch is connected to

element - index of the element: bus id if et == “b”, line id if et == “l”, trafo id if et == “t”

et - (string) element type: “l” = switch between bus and line, “t” = switch between bus and transformer, “t3” = switch between bus and 3-winding transformer, “b” = switch between two buses

closed (boolean, True) - switch position: False = open, True = closed

type (int, None) - indicates the type of switch: “LS” = Load Switch, “CB” = Circuit Breaker, “LBS” = Load Break Switch or “DS” = Disconnecting Switch
**OPTIONAL:** name (string, default None) - The name for this switch

**OUTPUT:** sid - The unique switch_id of the created switch

**EXAMPLE:** create_switch(net, bus = 0, element = 1, et = 'b', type = "LS")  
create_switch(net, bus = 0, element = 1, et = 'l')

### 2.4.2 Input Parameters

**net.switch**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Value Range</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>bus*</td>
<td>integer</td>
<td>index of connected bus</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>name of the switch</td>
<td></td>
</tr>
</tbody>
</table>
| element*  | integer  | index of the element the switch is connected to:  
- bus index if et = "b"  
- line index if et = "l"  
- trafo index if et = "t" |            |
| et*       | string   | "b" - bus-bus switch  
"l" - bus-line switch  
"t" - bus-trafo switch | element type the switch connects to |
| type      | string   | naming conventions:  
"CB" - circuit breaker  
"LS" - load switch  
"LBS" - load break switch  
"DS" - disconnecting switch | type of switch |
| closed*   | boolean  | "True / False" | signals the switching state of the switch |

*necessary for executing a power flow calculation.

### 2.4.3 Electric Model

**Bus-Bus-Switches:**

Two buses that are connected with a closed bus-bus switches are fused internally for the power flow, open bus-bus switches are ignored:
This has the following advantages compared to modelling the switch as a small impedance:

- there is no voltage drop over the switch (ideal switch)
- no convergence problems due to small impedances / large admittances
- less buses in the admittance matrix

**Bus-Element-Switches:**

When the power flow is calculated internally for every open bus-element switch an auxiliary bus is created in the pypower case file. The pypower branch that corresponds to the element is then connected to this bus. This has the following advantages compared to modelling the switch by setting the element out of service:

- loading current is considered
- information about switch position is preserved
- difference between open switch and out of service line (e.g. faulty line) can be modelled

Closed bus-element switches are ignored:

### 2.5 Load

**See also:**

*Unit Systems and Conventions*

#### 2.5.1 Create Function

```python
pandapower.create_load(net, bus, p_kw, q_kvar=0, const_z_percent=0, const_i_percent=0, sn_kva=nan, name=None, scaling=1., index=None, in_service=True, type=None, max_p_kw=nan, min_p_kw=nan, max_q_kvar=nan, min_q_kvar=nan, controllable=nan)
```

Adds one load in table net["load"].
All loads are modelled in the consumer system, meaning load is positive and generation is negative active power. Please pay attention to the correct signing of the reactive power as well.

**INPUT:** `net` - The net within this load should be created

  `bus` (int) - The bus id to which the load is connected

**OPTIONAL:** `p_kw` (float, default 0) - The real power of the load

  - Positive value -> load
  - Negative value -> generation

`q_kvar` (float, default 0) - The reactive power of the load

`const_z_percent` (float, default 0) - Percentage of `p_kw` and `q_kvar` that will be associated to constant impedance load at rated voltage

`const_i_percent` (float, default 0) - Percentage of `p_kw` and `q_kvar` that will be associated to constant current load at rated voltage

`sn_kva` (float, default None) - Nominal power of the load

`name` (string, default None) - The name for this load

`scaling` (float, default 1.) - An OPTIONAL scaling factor to be set customly

`type` (string, None) - Type variable to classify the load

`index` (int, None) - Force a specified ID if it is available. If None, the index one higher than the highest already existing index is selected.

`in_service` (boolean) - True for in_service or False for out of service

`max_p_kw` (float, default NaN) - Maximum active power load - necessary for controllable loads in for OPF

`min_p_kw` (float, default NaN) - Minimum active power load - necessary for controllable loads in for OPF

`max_q_kvar` (float, default NaN) - Maximum reactive power load - necessary for controllable loads in for OPF

`min_q_kvar` (float, default NaN) - Minimum reactive power load - necessary for controllable loads in for OPF

`controllable` (boolean, default NaN) - States, whether a load is controllable or not. Only respected for OPF

**OUTPUT:** `index` (int) - The unique ID of the created element

**EXAMPLE:** create_load(net, bus=0, p_kw=10., q_kvar=2.)

pandapower.create_load_from_cosphi(net, bus, sn_kva, cos_phi, mode, **kwargs)

Creates a load element from rated power and power factor \(\cos(\phi)\).

**INPUT:** `net` - The net within this static generator should be created

  `bus` (int) - The bus id to which the load is connected

  `sn_kva` (float) - Rated power of the generator

  `cos_phi` (float) - Power factor \(\cos(\phi)\)

  `mode` (str) - "ind" for inductive or "cap" for capacitive behaviour

  **kwargs are passed on to the create_load function

**OUTPUT:** `index` (int) - The unique ID of the created load

All elements are modeled from a consumer point of view. Active power will therefore always be positive, reactive power will be negative for inductive behaviour and positive for capacitive behaviour.
### 2.5.2 Input Parameters

`net.load`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Value Range</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td></td>
<td>name of the load</td>
</tr>
<tr>
<td>bus *</td>
<td>integer</td>
<td></td>
<td>index of connected bus</td>
</tr>
<tr>
<td>p_kw*</td>
<td>float</td>
<td>≥ 0</td>
<td>active power of the load [kW]</td>
</tr>
<tr>
<td>q_kvar*</td>
<td>float</td>
<td></td>
<td>reactive power of the load [kVar]</td>
</tr>
<tr>
<td>const_z_percent*</td>
<td>float</td>
<td>[0, 100]</td>
<td>percentage of p_kw and q_kvar that is associated to constant impedance load at rated voltage [%]</td>
</tr>
<tr>
<td>const_i_percent*</td>
<td>float</td>
<td>[0, 100]</td>
<td>percentage of p_kw and q_kvar that is associated to constant current load at rated voltage [%]</td>
</tr>
<tr>
<td>sn_kva</td>
<td>float</td>
<td>&gt; 0</td>
<td>rated power of the load [kVA]</td>
</tr>
<tr>
<td>scaling *</td>
<td>float</td>
<td>≥ 0</td>
<td>scaling factor for active and reactive power</td>
</tr>
<tr>
<td>in_service*</td>
<td>boolean</td>
<td>True / False</td>
<td>specifies if the load is in service.</td>
</tr>
<tr>
<td>controllable*</td>
<td>bool</td>
<td></td>
<td>States if load is controllable or not, load will not be used as flexibility if it is not controllable</td>
</tr>
<tr>
<td>max_p_kw*</td>
<td>float</td>
<td></td>
<td>Maximum active power</td>
</tr>
<tr>
<td>min_p_kw*</td>
<td>float</td>
<td></td>
<td>Minimum active power</td>
</tr>
<tr>
<td>max_q_kvar*</td>
<td>float</td>
<td></td>
<td>Maximum reactive power</td>
</tr>
<tr>
<td>min_q_kvar*</td>
<td>float</td>
<td></td>
<td>Minimum reactive power</td>
</tr>
</tbody>
</table>

*necessary for executing a power flow calculation.

**Note:** Loads should always have a positive p_kw value, since all power values are given in the consumer system. If you want to model constant generation, use a Static Generator (sgen element) instead of a negative load.

**Note:** The apparent power value sn_kva is provided as additional information for usage in controller or other applications based on panadapower. It is not considered in the power flow!

### 2.5.3 Electric Model

Loads are modelled as PQ-buses in the power flow calculation, with an option to use the so-called ZIP load model, where a load is represented as a composition of constant power (P), constant current (I) and constant impedance (Z):
What part of the load is considered constant with constant power, constant current or constant impedance is defined as follows:

\[
\begin{align*}
    z_{\text{const}} &= \text{const}_{z\_\text{percent}} / 100 \\
    i_{\text{const}} &= \text{const}_{i\_\text{percent}} / 100 \\
    p_{\text{const}} &= (100 - \text{const}_{z\_\text{percent}} - \text{const}_{i\_\text{percent}}) / 100
\end{align*}
\]

The load power values are then defined as:

\[
\begin{align*}
    P_{\text{load}} &= p_{\text{kw}} \cdot \text{scaling} \cdot (p_{\text{const}} + z_{\text{const}} \cdot V^2 + i_{\text{const}} \cdot V) \\
    Q_{\text{load}} &= q_{\text{kvar}} \cdot \text{scaling} \cdot (p_{\text{const}} + z_{\text{const}} \cdot V^2 + i_{\text{const}} \cdot V)
\end{align*}
\]

### 2.5.4 Result Parameters

**net.res_load**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_kw</td>
<td>float</td>
<td>resulting active power demand after scaling and after considering voltage dependence [kW]</td>
</tr>
<tr>
<td>q_kvar</td>
<td>float</td>
<td>resulting reactive power demand after scaling and after considering voltage dependence [kVar]</td>
</tr>
</tbody>
</table>

The power values in the net.res_load table are equivalent to \( P_{\text{load}} \) and \( Q_{\text{load}} \).

### 2.6 Static Generator

See also:

*Unit Systems and Conventions*
2.6.1 Create Function

```python
pandapower.create_sgen(net, bus, p_kw, q_kvar=0, sn_kva=nan, name=None, index=None, scaling=1., type=None, in_service=True, max_p_kw=nan, min_p_kw=nan, max_q_kvar=nan, min_q_kvar=nan, controllable=nan, k=nan, rx=nan)
```

Adds one static generator in table net["sgen"].

Static generators are modelled as negative PQ loads. This element is used to model generators with a constant active and reactive power feed-in. If you want to model a voltage controlled generator, use the generator element instead.

All elements in the grid are modelled in the consumer system, including generators! If you want to model the generation of power, you have to assign a negative active power to the generator. Please pay attention to the correct signing of the reactive power as well.

**INPUT:**
- `net` - The net within this static generator should be created
  - `bus` (int) - The bus id to which the static generator is connected
  - `p_kw` (float) - The real power of the static generator (negative for generation!)

**OPTIONAL:**
- `q_kvar` (float, default 0) - The reactive power of the sgen
- `sn_kva` (float, default None) - Nominal power of the sgen
- `name` (string, default None) - The name for this sgen
- `index` (int, None) - Force a specified ID if it is available. If None, the index one higher than the highest already existing index is selected.
- `scaling` (float, 1.) - An OPTIONAL scaling factor to be set customly
- `type` (string, None) - type variable to classify the static generator
- `in_service` (boolean) - True for in_service or False for out of service
- `controllable` (bool, NaN) - Whether this generator is controllable by the optimal powerflow
- `max_p_kw` (float, default NaN) - Maximum active power injection - necessary for controllable sgens in OPF
- `min_p_kw` (float, default NaN) - Minimum active power injection - necessary for controllable sgens in OPF
- `max_q_kvar` (float, default NaN) - Maximum reactive power injection - necessary for controllable sgens in OPF
- `min_q_kvar` (float, default NaN) - Minimum reactive power injection - necessary for controllable sgens in OPF

**OUTPUT:**
- `index` (int) - The unique ID of the created sgen

**EXAMPLE:**
```
pandapower.create_sgen(net, 1, p_kw = -120)
```

```python
pandapower.create_sgen_from_cosphi(net, bus, sn_kva, cos_phi, mode, **kwargs)
```

Creates an sgen element from rated power and power factor cos(phi).

**INPUT:**
- `net` - The net within this static generator should be created
  - `bus` (int) - The bus id to which the static generator is connected
  - `sn_kva` (float) - rated power of the generator
  - `cos_phi` (float) - power factor cos_phi
  - `mode` (str) - “ind” for inductive or “cap” for capacitive behaviour

**OUTPUT:**
- `index` (int) - The unique ID of the created sgen
All elements including generators are modeled from a consumer point of view. Active power will therefore always be negative, reactive power will be negative for inductive behaviour and positive for capacitive behaviour.

### 2.6.2 Input Parameters

\[ \text{net.sgen} \]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Value Range</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td></td>
<td>name of the static generator</td>
</tr>
<tr>
<td>type</td>
<td>string</td>
<td></td>
<td>type of generator</td>
</tr>
<tr>
<td>bus*</td>
<td>integer</td>
<td></td>
<td>index of connected bus</td>
</tr>
<tr>
<td>p_kw*</td>
<td>float</td>
<td>( \leq 0 )</td>
<td>active power of the static generator [kW]</td>
</tr>
<tr>
<td>q_kvar*</td>
<td>float</td>
<td></td>
<td>reactive power of the static generator [kVar]</td>
</tr>
<tr>
<td>sn_kva</td>
<td>float</td>
<td>( &gt; 0 )</td>
<td>rated power of the static generator [kVA]</td>
</tr>
<tr>
<td>scaling*</td>
<td>float</td>
<td>( \geq 0 )</td>
<td>scaling factor for the active and reactive power</td>
</tr>
<tr>
<td>max_p_kw**</td>
<td>float</td>
<td></td>
<td>Maximum active power</td>
</tr>
<tr>
<td>min_p_kw**</td>
<td>float</td>
<td></td>
<td>Minimum active power</td>
</tr>
<tr>
<td>max_q_kvar**</td>
<td>float</td>
<td></td>
<td>Maximum reactive power</td>
</tr>
<tr>
<td>min_q_kvar**</td>
<td>float</td>
<td></td>
<td>Minimum reactive power</td>
</tr>
<tr>
<td>controllable*</td>
<td>bool</td>
<td></td>
<td>States if sgen is controllable or not, sgen will not be used as a flexibility if it is not controllable</td>
</tr>
<tr>
<td>k***</td>
<td>float</td>
<td>( \geq 0 )</td>
<td>Ratio of nominal current to short circuit current</td>
</tr>
<tr>
<td>rx***</td>
<td>float</td>
<td>( \geq 0 )</td>
<td>R/X ratio for short circuit impedance. Only relevant if type is specified as motor so that sgen is treated as asynchronous motor</td>
</tr>
<tr>
<td>in_service*</td>
<td>boolean</td>
<td>True / False</td>
<td>specifies if the generator is in service.</td>
</tr>
</tbody>
</table>

*necessary for executing a power flow calculation **optimal power flow parameter

### 2.6.3 Electric Model

Static Generators are modelled as PQ-buses in the power flow calculation:
The PQ-Values are calculated from the parameter table values as:

\[ P_{s\text{gen}} = p_{\text{kw}} \cdot \text{scaling} \]
\[ Q_{s\text{gen}} = q_{\text{kvar}} \cdot \text{scaling} \]

**Note:** Static generators should always have a negative \( p_{\text{kw}} \) value, since all power values are given in the consumer system. If you want to model constant power consumption, please use the load element instead of a static generator with positive active power value. If you want to model a voltage controlled generator, use the generator element.

**Note:** The apparent power value \( s_{\text{kva}} \) is provided as additional information for usage in controller or other applications based on panadapower. It is not considered in the power flow!

### 2.6.4 Result Parameters

\textit{net.res_sgen}

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_{\text{kw}} )</td>
<td>float</td>
<td>resulting active power demand after scaling [kW]</td>
</tr>
<tr>
<td>( q_{\text{kvar}} )</td>
<td>float</td>
<td>resulting reactive power demand after scaling [kVar]</td>
</tr>
</tbody>
</table>

The power values in the net.res_sgen table are equivalent to \( P_{s\text{gen}} \) and \( Q_{s\text{gen}} \).

### 2.7 External Grid

See also:

\textit{Unit Systems and Conventions}

#### 2.7.1 Create Function

\texttt{pandapower.create_ext_grid}(net, bus, \texttt{vm\_pu=1.0}, \texttt{va\_degree=0.}, \texttt{name=None}, \texttt{in\_service=True}, \texttt{s\_sc\_max\_mva=nan}, \texttt{s\_sc\_min\_mva=nan}, \texttt{rx\_max=nan}, \texttt{rx\_min=nan}, \texttt{max\_p\_kw=nan}, \texttt{min\_p\_kw=nan}, \texttt{max\_q\_kvar=nan}, \texttt{min\_q\_kvar=nan}, \texttt{index=None})

Creates an external grid connection.

External grids represent the higher level power grid connection and are modelled as the slack bus in the power flow calculation.

**INPUT:**\texttt{net} - pandapower network

\texttt{bus} (int) - bus where the slack is connected

**OPTIONAL:** \texttt{vm\_pu} (float, default 1.0) - voltage at the slack node in per unit

\texttt{va\_degree} (float, default 0.) - voltage angle at the slack node in degrees*

\texttt{name} (string, default None) - name of of the external grid

\texttt{in\_service} (boolean) - True for \texttt{in\_service} or False for out of service

\texttt{Sk\_max} - maximal short circuit apparent power **

\texttt{SK\_min} - maximal short circuit apparent power **
RX_max - maximal R/X-ratio **  
RK_min - minimal R/X-ratio **

max_p_kw (float, default NaN) - Maximum active power injection. Only respected for OPF  
min_p_kw (float, default NaN) - Minimum active power injection. Only respected for OPF  
max_q_kvar (float, default NaN) - Maximum reactive power injection. Only respected for OPF  
min_q_kvar (float, default NaN) - Minimum reactive power injection. Only respected for OPF  
* only considered in loadflow if calculate_voltage_angles = True  
** only needed for short circuit calculations  
EXAMPLE: create_ext_grid(net, 1, voltage = 1.03)

2.7.2 Input Parameters

net.ext_grid

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Value Range</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td></td>
<td>name of the external grid</td>
</tr>
<tr>
<td>bus*</td>
<td>integer</td>
<td></td>
<td>index of connected bus</td>
</tr>
<tr>
<td>vm_pu*</td>
<td>float</td>
<td>&gt; 0</td>
<td>voltage set point [p.u]</td>
</tr>
<tr>
<td>va_degree*</td>
<td>float</td>
<td></td>
<td>angle set point [degree]</td>
</tr>
<tr>
<td>max_p_kw**</td>
<td>float</td>
<td></td>
<td>Maximum active power</td>
</tr>
<tr>
<td>min_p_kw**</td>
<td>float</td>
<td></td>
<td>Minimum active power</td>
</tr>
<tr>
<td>max_q_kvar**</td>
<td>float</td>
<td></td>
<td>Maximum reactive power</td>
</tr>
<tr>
<td>min_q_kvar**</td>
<td>float</td>
<td></td>
<td>Minimum reactive power</td>
</tr>
<tr>
<td>s_sc_max_mva***</td>
<td>float</td>
<td>&gt; 0</td>
<td>maximum short circuit power provision [MVA]</td>
</tr>
<tr>
<td>s_sc_min_mva***</td>
<td>float</td>
<td>&gt; 0</td>
<td>minimum short circuit power provision [MVA]</td>
</tr>
<tr>
<td>rx_max***</td>
<td>float</td>
<td>0...1</td>
<td>maximum R/X ratio of short-circuit impedance</td>
</tr>
<tr>
<td>rx_min***</td>
<td>float</td>
<td>0...1</td>
<td>minimum R/X ratio of short-circuit impedance</td>
</tr>
<tr>
<td>in_service*</td>
<td>boolean</td>
<td>True / False</td>
<td>specifies if the external grid is in service.</td>
</tr>
</tbody>
</table>

*necessary for executing a power flow calculation **optimal power flow parameter ***short-circuit calculation parameter

2.7.3 Electric Model

The external grid is modelled as a voltage source in the power flow calculation, which means the node the grid is connected to is treated as a slack node:
with:

\[ v_{bus} = v_{mpu} \cdot e^{j\theta} \]
\[ \theta = \text{shift degree} \cdot \frac{\pi}{180} \]

### 2.7.4 Result Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_kw</td>
<td>float</td>
<td>active power supply at the external grid [kW]</td>
</tr>
<tr>
<td>q_kvar</td>
<td>float</td>
<td>reactive power supply at the external grid [kVar]</td>
</tr>
</tbody>
</table>

Active and reactive power feed-in / consumption at the slack node is a result of the power flow:

\[ p_{kw} = P_{eg} \]
\[ q_{kvar} = Q_{eg} \]

**Note:** All power values are given in the consumer system, therefore p_kw is positive if the external grid is absorbing power and negative if it is supplying power.

### 2.8 Transformer

See also:

*Unit Systems and Conventions Standard Type Libraries*

#### 2.8.1 Create Function

Transformers can be either created from the standard type library (create_transformer) or with custom values (create_transformer_from_parameters).

\[
\text{pandapower.create_transformer} (\text{net}, \text{hv_bus}, \text{lv_bus}, \text{std_type}, \text{name=None}, \text{tp_pos=nan}, \\
\text{in_service=True}, \text{index=None}, \text{max_loading_percent=nan}, \\
\text{parallel=1})
\]

Creates a two-winding transformer in table net["trafo"]. The trafo parameters are defined through the standard type library.
INPUT: net - The net within this transformer should be created
  hv_bus (int) - The bus on the high-voltage side on which the transformer will be connected to
  lv_bus (int) - The bus on the low-voltage side on which the transformer will be connected to
  std_type - The used standard type from the standard type library

OPTIONAL: name (string, None) - A custom name for this transformer
  tp_pos (int, nan) - current tap position of the transformer. Defaults to the medium position (tp_mid)
  in_service (boolean, True) - True for in_service or False for out of service
  index (int, None) - Force a specified ID if it is available. If None, the index one higher than the highest already existing index is selected.
  max_loading_percent (float) - maximum current loading (only needed for OPF)

OUTPUT: index (int) - The unique ID of the created transformer

EXAMPLE: create_transformer(net, hv_bus = 0, lv_bus = 1, name = “trafo1”, std_type = “0.4 MVA 10/0.4 kV”)

pandapower.create_transformer_from_parameters(net, hv_bus, lv_bus, sn_kva, vn_hv_kv, vn_lp_kv, vscr_percent, vsc_percent, pfe_kw, i0_percent, shift_degree=0, tp_side=None, tp_mid=nan, tp_max=nan, tp_min=nan, tp_st_percent=nan, tp_st_degree=nan, tp_pos=nan, in_service=True, name=None, index=None, max_loading_percent=nan, parallel=1, **kwargs)

Creates a two-winding transformer in table net[“trafo”]. The trafo parameters are defined through the standard type library.

INPUT: net - The net within this transformer should be created
  hv_bus (int) - The bus on the high-voltage side on which the transformer will be connected to
  lv_bus (int) - The bus on the low-voltage side on which the transformer will be connected to
  sn_kva (float) - rated apparent power
  vn_hv_kv (float) - rated voltage on high voltage side
  vn_lp_kv (float) - rated voltage on low voltage side
  vscr_percent (float) - real part of relative short-circuit voltage
  vsc_percent (float) - relative short-circuit voltage
  pfe_kw (float) - iron losses in kW
  i0_percent (float) - open loop losses in percent of rated current

OPTIONAL: in_service (boolean) - True for in_service or False for out of service
  parallel (integer) - number of parallel transformers
  name (string) - A custom name for this transformer
  shift_degree (float) - Angle shift over the transformer*
  tp_side (string) - position of tap changer (“hv”, “lv”)
  tp_pos (int, nan) - current tap position of the transformer. Defaults to the medium position (tp_mid)
  tp_mid (int, nan) - tap position where the transformer ratio is equal to the ration of the rated voltages
  tp_max (int, nan) - maximal allowed tap position
**tp_min** (int, nan): minimal allowed tap position

**tp_st_percent** (int) - tap step in percent

**index** (int, None) - Force a specified ID if it is available. If None, the index one higher than the highest already existing index is selected.

**kwargs** - nothing to see here, go along

* only considered in loadflow if calculate_voltage_angles = True

**max_loading_percent** (float) - maximum current loading (only needed for OPF)

**OUTPUT:** **index** (int) - The unique ID of the created transformer

**EXAMPLE:** `create_transformer_from_parameters(net, hv_bus=0, lv_bus=1, name="trafo1", sn_kva=40, vn_hv_kv=110, vn_lv_kv=10, vsc_percent=10, vscr_percent=0.3, pfe_kw=30, i0_percent=0.1, shift_degree=30)`

### 2.8.2 Input Parameters

`net.trafo`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Value Range</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td></td>
<td>name of the transformer</td>
</tr>
<tr>
<td>std_type</td>
<td>string</td>
<td></td>
<td>transformer standard type name</td>
</tr>
<tr>
<td>hv_bus</td>
<td>integer</td>
<td></td>
<td>high voltage bus index of the transformer</td>
</tr>
<tr>
<td>lv_bus</td>
<td>integer</td>
<td></td>
<td>low voltage bus index of the transformer</td>
</tr>
<tr>
<td>sn_kva</td>
<td>float</td>
<td>&gt; 0</td>
<td>rated apparent power of the transformer [kVA]</td>
</tr>
<tr>
<td>vn_hv_kv</td>
<td>float</td>
<td>&gt; 0</td>
<td>rated voltage at high voltage bus [kV]</td>
</tr>
<tr>
<td>vn_lv_kv</td>
<td>float</td>
<td>&gt; 0</td>
<td>rated voltage at low voltage bus [kV]</td>
</tr>
<tr>
<td>vsc_percent</td>
<td>float</td>
<td>&gt; 0</td>
<td>short circuit voltage [%]</td>
</tr>
<tr>
<td>vscr_percent</td>
<td>float</td>
<td>≥ 0</td>
<td>real component of short circuit voltage [%]</td>
</tr>
<tr>
<td>pfe_kw</td>
<td>float</td>
<td>≥ 0</td>
<td>iron losses [kW]</td>
</tr>
<tr>
<td>i0_percent</td>
<td>float</td>
<td>≥ 0</td>
<td>open loop losses in [%]</td>
</tr>
<tr>
<td>shift_degree</td>
<td>float</td>
<td></td>
<td>transformer phase shift angle</td>
</tr>
<tr>
<td>tp_side</td>
<td>string</td>
<td>&quot;hv&quot;, &quot;lv&quot;</td>
<td>defines if tap changer is at the high- or low voltage side</td>
</tr>
<tr>
<td>tp_mid</td>
<td>integer</td>
<td></td>
<td>rated tap position</td>
</tr>
<tr>
<td>tp_min</td>
<td>integer</td>
<td></td>
<td>minimum tap position</td>
</tr>
<tr>
<td>tp_max</td>
<td>integer</td>
<td></td>
<td>maximum tap position</td>
</tr>
<tr>
<td>tp_st_percent</td>
<td>float</td>
<td>&gt; 0</td>
<td>tap step size [%]</td>
</tr>
<tr>
<td>tp_pos</td>
<td>integer</td>
<td></td>
<td>current position of tap changer</td>
</tr>
<tr>
<td>max_loading_percent</td>
<td>float</td>
<td>&gt; 0</td>
<td>Maximum loading of the transformer with respect to sn_kva and its corresponding current at 1.0 p.u.</td>
</tr>
</tbody>
</table>

*necessary for executing a power flow calculation **optimal power flow parameter

**Note:** The transformer loading constraint for the optimal power flow corresponds to the option `trafo_loading="current"`.

### 2.8.3 Electric Model

The equivalent circuit used for the transformer can be set in the power flow with the parameter “trafo_model”.
Transformer Ratio:
The magnitude of the transformer ratio is given as:

\[ n = \frac{V_{\text{ref,HV,transformer}}}{V_{\text{ref,LV,transformer}}} \cdot \frac{V_{\text{ref,LV,bus}}}{V_{\text{ref,HV,bus}}} \]

The reference voltages of the high- and low voltage buses are taken from the net.bus table. If no tap changer is defined, the reference voltage of the transformer is taken directly from the transformer table:

\[ V_{\text{ref,HV,transformer}} = v_{n,hv,kv} \]
\[ V_{\text{ref,LV,transformer}} = v_{n,lv,kv} \]

If a tap changer is defined, the reference voltage is multiplied with the tap factor:

\[ n_{\text{tap}} = 1 + (tp_{\text{pos}} - tp_{\text{mid}}) \cdot \frac{tp_{\text{st,percent}}}{100} \]

On which side the reference voltage is adapted depends on the \textit{tp\_side} variable:

<table>
<thead>
<tr>
<th>\text{tp_side}</th>
<th>\text{hv}</th>
<th>\text{lv}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{V}_{n,HV,\text{transformer}}</td>
<td>\text{vn}<em>{hv,kv} \cdot n</em>{\text{tap}}</td>
<td>\text{vn}_{hv,kv}</td>
</tr>
<tr>
<td>\text{V}_{n,\text{LV,transformer}}</td>
<td>\text{vn}_{lv,kv}</td>
<td>\text{vn}<em>{lv,kv} \cdot n</em>{\text{tap}}</td>
</tr>
</tbody>
</table>

\textbf{Note:} The variables \textit{tp\_min} and \textit{tp\_max} are not considered in the power flow. The user is responsible to ensure that \textit{tp\_min} < \textit{tp\_pos} < \textit{tp\_max}!
Phase Shift:

If the power flow is run with voltage_angles=True, the complex ratio is given as:

\[ n = n \cdot e^{j\theta} \]
\[ \theta = \text{shift\_degree} \cdot \frac{\pi}{180} \]

Otherwise, the ratio does not include a phase shift:

\[ n = n \]

Impedances:

The short-circuit impedance is calculated as:

\[ z_k = \frac{\text{vsc\_percent}}{100} \cdot \frac{1000}{\text{sn\_kva}} \]
\[ r_k = \frac{\text{vsc\_percent}}{100} \cdot \frac{1000}{\text{sn\_kva}} \]
\[ x_k = \sqrt{z_k^2 - r_k^2} \]
\[ \bar{z}_k = r_k + j \cdot x_k \]

The magnetising admittance is calculated as:

\[ y_m = \frac{i0\_percent}{100} \cdot \frac{1000}{\text{sn\_kva}} \cdot \frac{1000}{\text{sn\_kva}} \]
\[ g_m = \frac{\text{pf\_kw}}{\text{sn\_kva}} \cdot \frac{100}{\text{sn\_kva}} \cdot \frac{1000}{\text{sn\_kva}} \]
\[ b_m = \sqrt{y_m^2 - g_m^2} \]
\[ y_m = g_m - j \cdot b_m \]

The values calculated in that way are relative to the rated values of the transformer. To transform them into the per unit system, they have to be converted to the rated values of the network:

\[ Z_N = \frac{V_N^2}{S_N} \]
\[ Z_{\text{ref,trafo}} = \frac{vN\_lv\_kv^2 \cdot 1000}{\text{sn\_kva}} \]
\[ \bar{z} = \bar{z}_k \cdot \frac{Z_{\text{ref,trafo}}}{Z_N} \]
\[ y = y_m \cdot \frac{Z_{\text{ref,trafo}}}{Z_N} \]

Where the reference voltage \( V_N \) is the nominal voltage at the low voltage side of the transformer and the rated apparent power \( S_N \) is defined system wide in the net object (see Unit Systems and Conventions).

### 2.8.4 Result Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_hv_kw</td>
<td>float</td>
<td>active power flow at the high voltage transformer bus [kW]</td>
</tr>
<tr>
<td>q_hv_kvar</td>
<td>float</td>
<td>reactive power flow at the high voltage transformer bus [kVar]</td>
</tr>
<tr>
<td>p_lv_kw</td>
<td>float</td>
<td>active power flow at the low voltage transformer bus [kW]</td>
</tr>
<tr>
<td>q_lv_kvar</td>
<td>float</td>
<td>reactive power flow at the low voltage transformer bus [kVar]</td>
</tr>
<tr>
<td>pl_kw</td>
<td>float</td>
<td>active power losses of the transformer [kW]</td>
</tr>
<tr>
<td>ql_kvar</td>
<td>float</td>
<td>reactive power consumption of the transformer [kvar]</td>
</tr>
<tr>
<td>i_hv_ka</td>
<td>float</td>
<td>current at the high voltage side of the transformer [kA]</td>
</tr>
<tr>
<td>i_lv_ka</td>
<td>float</td>
<td>current at the low voltage side of the transformer [kA]</td>
</tr>
<tr>
<td>loading_percent</td>
<td>float</td>
<td>load utilization relative to rated power [%]</td>
</tr>
</tbody>
</table>
\[ p_{hv\_kw} = Re(v_{hv} \cdot i_{hv}) \]
\[ q_{hv\_kvar} = Im(v_{hv} \cdot i_{hv}) \]
\[ p_{lv\_kw} = Re(v_{lv} \cdot i_{lv}) \]
\[ q_{lv\_kvar} = Im(v_{lv} \cdot i_{lv}) \]
\[ p_{lkw} = p_{hv\_kw} + p_{lv\_kw} \]
\[ q_{lkw} = q_{hv\_kvar} + q_{lv\_kvar} \]
\[ i_{hv\_ka} = i_{hv} \]
\[ i_{lv\_ka} = i_{lv} \]

The definition of the transformer loading depends on the trafo-loading parameter of the power flow. For trafo-loading="current", the loading is calculated as:

\[ \text{loading\_percent} = \max \left( \frac{i_{hv} \cdot v_{hv} \cdot \text{sn\_kva}}{\text{sn\_kva}}, \frac{i_{lv} \cdot v_{lv} \cdot \text{sn\_kva}}{\text{sn\_kva}} \right) \cdot 100 \]

For trafo-loading="power", the loading is defined as:

\[ \text{loading\_percent} = \max \left( \frac{i_{hv} \cdot v_{hv} \cdot \text{sn\_kva}}{\text{sn\_kva}}, \frac{i_{lv} \cdot v_{lv} \cdot \text{sn\_kva}}{\text{sn\_kva}} \right) \cdot 100 \]

### 2.9 Three Winding Transformer

See also:

*Unit Systems and Conventions Standard Type Libraries*

### 2.9.1 Create Function

```python
pandapower.create_transformer3w(net, hv_bus, mv_bus, lv_bus, std_type, name=None, tp_pos=nan, in_service=True, index=None, max_loading_percent=nan)
```

Creates a three-winding transformer in table net[“trafo3w”]. The trafo parameters are defined through the standard type library.

**INPUT:**
- `net` - The net within this transformer should be created
  - `hv_bus` (int) - The bus on the high-voltage side on which the transformer will be connected to
  - `mv_bus` (int) - The medium voltage bus on which the transformer will be connected to
  - `lv_bus` (int) - The bus on the low-voltage side on which the transformer will be connected to
  - `std_type` - The used standard type from the standard type library

**OPTIONAL:**
- `name` (string) - A custom name for this transformer
- `tp_pos` (int, nan) - current tap position of the transformer. Defaults to the medium position (tp_mid)
- `in_service` (boolean) - True for in_service or False for out of service
- `index` (int, None) - Force a specified ID if it is available. If None, the index one higher than the highest already existing index is selected.
- `max_loading_percent` (float) - maximum current loading (only needed for OPF)

**OUTPUT:**
- `index` (int) - The unique ID of the created transformer

**EXAMPLE:**
```
create_transformer3w(net, hv_bus = 0, mv_bus = 1, lv_bus = 2, name = “trafo1”, std_type = “63/25/38 MV A 110/20/10 kV”)
```
pandapower.create_transformer3w_from_parameters (net, hv_bus, mv_bus, lv_bus, vn_hv_kv, vn_mv_kv, vn_lv_kv, sn_hv_kva, sn_mv_kva, sn_lv_kva, vsc_hv_percent, vsc_mv_percent, vsc_lv_percent, vscr_hv_percent, vscr_mv_percent, vscr_lv_percent, pfe_kw, i0_percent, shift_mv_degree=0., shift_lv_degree=0., tp_side=None, tp_st_percent=nan, tp_pos=nan, tp_mid=nan, tp_max=nan, tp_min=nan, name=None, in_service=True, index=None, max_loading_percent=nan)

Adds a three-winding transformer in table net["trafo3w"].

**Input:** net (pandapowerNet) - The net within this transformer should be created

- **hv_bus** (int) - The bus on the high-voltage side on which the transformer will be connected to
- **mv_bus** (int) - The bus on the middle-voltage side on which the transformer will be connected to
- **lv_bus** (int) - The bus on the low-voltage side on which the transformer will be connected to
- **vn_hv_kv** (float) - rated voltage on high voltage side
- **vn_mv_kv** (float) - rated voltage on medium voltage side
- **vn_lv_kv** (float) - rated voltage on low voltage side
- **sn_hv_kva** (float) - rated apparent power on high voltage side
- **sn_mv_kva** (float) - rated apparent power on medium voltage side
- **sn_lv_kva** (float) - rated apparent power on low voltage side
- **vsc_hv_percent** (float) - short circuit voltage from high to medium voltage
- **vsc_mv_percent** (float) - short circuit voltage from medium to low voltage
- **vsc_lv_percent** (float) - short circuit voltage from high to low voltage
- **vscr_hv_percent** (float) - real part of short circuit voltage from high to medium voltage
- **vscr_mv_percent** (float) - real part of short circuit voltage from medium to low voltage
- **vscr_lv_percent** (float) - real part of short circuit voltage from high to low voltage
- **pfe_kw** (float) - iron losses
- **i0_percent** (float) - open loop losses

**OPTIONAL:**

- **shift_mv_degree** (float, 0) - angle shift to medium voltage side*
- **shift_lv_degree** (float, 0) - angle shift to low voltage side*
- **tp_st_percent** (float) - Tap step in percent
- **tp_side** (string, None) - “hv”, “mv”, “lv”
- **tp_mid** (int, nan) - default tap position
- **tp_min** (int, nan) - Minimum tap position
- **tp_max** (int, nan) - Maximum tap position
- **tp_pos** (int, nan) - current tap position of the transformer. Defaults to the medium position (tp_mid)
- **name** (string, None) - Name of the 3-winding transformer
- **in_service** (boolean, True) - True for in_service or False for out of service
* only considered in loadflow if calculate_voltage_angles = True **The model currently only supports one tap-changer per 3W Transformer.

max_loading_percent (float) - maximum current loading (only needed for OPF)

OUTPUT: trafo_id - The unique trafo_id of the created 3W transformer

Example: create_transformer3w_from_parameters(net, hv_bus=0, mv_bus=1, lv_bus=2, name="trafo1", sn_hv_kva=40, sn_mv_kva=20, sn_lv_kva=20, vn_hv_kv=110, vn_mv_kv=20, vn_lv_kv=10, vsc_hv_percent=10, vsc_mv_percent=11, vsc_lv_percent=12, vscr_hv_percent=0.3, vscr_mv_percent=0.31, vscr_lv_percent=0.32, pfe_kw=30, i0_percent=0.1, shift_mv_degree=30, shift_lv_degree=30)

Note: All short circuit voltages are given relative to the maximum apparent power flow. For example vsc_hv_percent is the short circuit voltage from the high to the medium level, it is given relative to the minimum of the rated apparent power in high and medium level: min(sn_hv_kva, sn_mv_kva). This is consistent with most commercial network calculation software (e.g. PowerFactory). Some tools (like PSS Sincal) however define all short circuit voltages relative to the overall rated apparent power of the transformer: max(sn_hv_kva, sn_mv_kva, sn_lv_kva). You might have to convert the values depending on how the short-circuit voltages are defined.

2.9.2 Input Parameters

### net.trafo3w

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Value Range</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td></td>
<td>name of the transformer</td>
</tr>
<tr>
<td>hv_bus*</td>
<td>integer</td>
<td></td>
<td>high voltage bus index of the transformer</td>
</tr>
<tr>
<td>mv_bus</td>
<td>integer</td>
<td></td>
<td>medium voltage bus index of the transformer</td>
</tr>
<tr>
<td>lv_bus*</td>
<td>integer</td>
<td></td>
<td>low voltage bus index of the transformer</td>
</tr>
<tr>
<td>vn_hv_kv*</td>
<td>float</td>
<td></td>
<td>rated voltage at high voltage bus [kV]</td>
</tr>
<tr>
<td>vn_mv_kv*</td>
<td>float &gt; 0</td>
<td></td>
<td>rated voltage at medium voltage bus [kV]</td>
</tr>
<tr>
<td>vn_lv_kv*</td>
<td>float &gt; 0</td>
<td></td>
<td>rated voltage at low voltage bus [kV]</td>
</tr>
<tr>
<td>sn_hv_kva*</td>
<td>float &gt; 0</td>
<td></td>
<td>rated apparent power on high voltage side [kVA]</td>
</tr>
<tr>
<td>sn_mv_kva*</td>
<td>float &gt; 0</td>
<td></td>
<td>rated apparent power on medium voltage side [kVA]</td>
</tr>
<tr>
<td>sn_lv_kva*</td>
<td>float &gt; 0</td>
<td></td>
<td>rated apparent power on low voltage side [kVA]</td>
</tr>
<tr>
<td>vsc_hv_percent*</td>
<td>float &gt; 0</td>
<td></td>
<td>short circuit voltage from high to medium voltage [%]</td>
</tr>
<tr>
<td>vsc_mv_percent*</td>
<td>float &gt; 0</td>
<td></td>
<td>short circuit voltage from medium to low voltage [%]</td>
</tr>
<tr>
<td>vsc_lv_percent*</td>
<td>float &gt; 0</td>
<td></td>
<td>short circuit voltage from high to low voltage [%]</td>
</tr>
<tr>
<td>vscr_hv_percent*</td>
<td>float ≥ 0</td>
<td></td>
<td>real part of short circuit voltage from high to medium voltage [%]</td>
</tr>
<tr>
<td>vscr_mv_percent*</td>
<td>float ≥ 0</td>
<td></td>
<td>real part of short circuit voltage from medium to low voltage [%]</td>
</tr>
<tr>
<td>vscr_lv_percent*</td>
<td>float ≥ 0</td>
<td></td>
<td>real part of short circuit voltage from high to low voltage [%]</td>
</tr>
<tr>
<td>pfe_kw*</td>
<td>float ≥ 0</td>
<td></td>
<td>iron losses [kW]</td>
</tr>
<tr>
<td>i0_percent*</td>
<td>float ≥ 0</td>
<td></td>
<td>open loop losses [%]</td>
</tr>
<tr>
<td>tp_side</td>
<td>string</td>
<td>&quot;hv&quot;, &quot;mv&quot;, &quot;lv&quot;</td>
<td>defines if tap changer is positioned on high-medium- or low voltage side</td>
</tr>
<tr>
<td>tp_mid</td>
<td>integer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2 DATASTRUCTURE AND ELEMENTS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Value Range</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>tp_min</td>
<td>integer</td>
<td></td>
<td>minimum tap position</td>
</tr>
<tr>
<td>tp_max</td>
<td>integer</td>
<td></td>
<td>maximum tap position</td>
</tr>
<tr>
<td>tp_st_percent</td>
<td>float</td>
<td>&gt; 0</td>
<td>tap step size [%]</td>
</tr>
<tr>
<td>tp_pos</td>
<td>integer</td>
<td></td>
<td>current position of tap changer</td>
</tr>
<tr>
<td>in_service*</td>
<td>boolean</td>
<td>True/False</td>
<td>specifies if the transformer is in service.</td>
</tr>
</tbody>
</table>

*necessary for executing a power flow calculation.

**Note:** Three Winding Transformer loading can not yet be constrained with the optimal power flow.

### 2.9.3 Electric Model

Three Winding Transformers are modelled by three two-winding transformers:

![Electric Model Diagram]

The parameters of the three transformers are defined as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>hv_bus</td>
<td>hv_bus</td>
<td>auxiliary bus</td>
<td>auxiliary bus</td>
</tr>
<tr>
<td>lv_bus</td>
<td>auxiliary bus</td>
<td>mv_bus</td>
<td>lv_bus</td>
</tr>
<tr>
<td>sn_kva</td>
<td>sn_hv_kva</td>
<td>sn_mv_kva</td>
<td>sn_lv_kva</td>
</tr>
<tr>
<td>vn_hv_kv</td>
<td>vn_hv_kv</td>
<td>vn_hv_kv</td>
<td>vn_hv_kv</td>
</tr>
<tr>
<td>vn_lv_kv</td>
<td>vn_hv_kv</td>
<td>vn_mv_kv</td>
<td>vn_lv_kv</td>
</tr>
<tr>
<td>vsc_percent</td>
<td>(v_{k,t1})</td>
<td>(v_{k,t2})</td>
<td>(v_{k,t3})</td>
</tr>
<tr>
<td>vsr_percent</td>
<td>(v_{r,t1})</td>
<td>(v_{r,t2})</td>
<td>(v_{r,t3})</td>
</tr>
<tr>
<td>pfe_kw</td>
<td>pfe_kw</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>i0_percent</td>
<td>i0_percent</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>shift_degree</td>
<td>shift_degree</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The definition of the two winding transformer parameter can be found [here](#).

To calculate the short-circuit voltages \(v_{k,t1..t3}\) and \(v_{r,t1..t3}\), first all short-circuit voltages are converted to the high
voltage level:

\[ v'_{k,h} = \text{vsc}_{\text{hv}} \cdot \text{sn}_{\text{hv}} \]

\[ v'_{k,m} = \text{vsc}_{\text{mv}} \cdot \frac{\text{sn}_{\text{mv}}}{\text{sn}_{\text{mv}}} \]

\[ v'_{k,l} = \text{vsc}_{\text{lv}} \cdot \frac{\text{sn}_{\text{hv}}}{\text{sn}_{\text{hv}}} \]

The short-circuit voltages of the three transformers are then calculated as follows:

\[ v'_{k,t1} = \frac{1}{2} (v'_{k,h} + v'_{k,l} - v'_{k,m}) \]

\[ v'_{k,t2} = \frac{1}{2} (v'_{k,m} + v'_{k,h} - v'_{k,l}) \]

\[ v'_{k,t3} = \frac{1}{2} (v'_{k,m} + v'_{k,l} - v'_{k,h}) \]

Since these voltages are given relative to the high voltage side, they have to be transformed back to the voltage level of each transformer:

\[ v_{k,t1} = v'_{k,t1} \cdot \frac{\text{sn}_{\text{mv}}}{\text{sn}_{\text{hv}}} \cdot \frac{\text{sn}_{\text{lv}}}{\text{sn}_{\text{hv}}} \]

\[ v_{k,t2} = v'_{k,t2} \cdot \frac{\text{sn}_{\text{mv}}}{\text{sn}_{\text{hv}}} \cdot \frac{\text{sn}_{\text{hv}}}{\text{sn}_{\text{hv}}} \]

\[ v_{k,t3} = v'_{k,t3} \cdot \frac{\text{sn}_{\text{mv}}}{\text{sn}_{\text{hv}}} \cdot \frac{\text{sn}_{\text{lv}}}{\text{sn}_{\text{hv}}} \]

The real part of the short-circuit voltage is calculated in the same way.

**Note:** All short circuit voltages are given relative to the maximum apparent power flow. For example, \( \text{vsc}_{\text{hv}} \) is the short circuit voltage from the high to the medium level, it is given relative to the minimum of the rated apparent power in high and medium level: \( \min(\text{sn}_{\text{hv}}, \text{sn}_{\text{mv}}) \). This is consistent with most commercial network calculation software (e.g. PowerFactory). Some tools (like PSS Sincal) however define all short circuit voltages relative to the overall rated apparent power of the transformer: \( \max(\text{sn}_{\text{hv}}, \text{sn}_{\text{mv}}, \text{sn}_{\text{lv}}) \). You might have to convert the values depending on how the short-circuit voltages are defined.

The tap changer adapts the nominal voltages of the transformer in the equivalent to the 2W-Model:

<table>
<thead>
<tr>
<th>( V_{n,\text{HV, transformer}} )</th>
<th>( \text{tp}_\text{side}=&quot;\text{hv}&quot; )</th>
<th>( \text{tp}_\text{side}=&quot;\text{mv}&quot; )</th>
<th>( \text{tp}_\text{side}=&quot;\text{lv}&quot; )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 1 \cdot n_{\text{tap}} )</td>
<td>( \text{in}<em>{\text{hv}} \cdot n</em>{\text{tap}} )</td>
<td>( \text{in}<em>{\text{hv}} \cdot n</em>{\text{tap}} )</td>
<td>( \text{in}<em>{\text{hv}} \cdot n</em>{\text{tap}} )</td>
</tr>
<tr>
<td>( V_{n,\text{MV, transformer}} )</td>
<td>( \text{in}<em>{\text{mv}} \cdot n</em>{\text{tap}} )</td>
<td>( \text{in}<em>{\text{mv}} \cdot n</em>{\text{tap}} )</td>
<td>( \text{in}<em>{\text{mv}} \cdot n</em>{\text{tap}} )</td>
</tr>
<tr>
<td>( V_{n,\text{LV, transformer}} )</td>
<td>( \text{in}<em>{\text{lv}} \cdot n</em>{\text{tap}} )</td>
<td>( \text{in}<em>{\text{lv}} \cdot n</em>{\text{tap}} )</td>
<td>( \text{in}<em>{\text{lv}} \cdot n</em>{\text{tap}} )</td>
</tr>
</tbody>
</table>

\[ n_{\text{tap}} = 1 + (\text{tp}_{\text{pos}} - \text{tp}_{\text{mid}}) \cdot \frac{\text{tp}_{\text{st,percent}}}{100} \]

**See also:**

MVA METHOD FOR 3-WINDING TRANSFORMER

**2.9.4 Result Parameters**

net.res_trafo3w
### Parameter, Datatype, Explanation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_hv_kw</td>
<td>float</td>
<td>active power flow at the high voltage transformer bus [kW]</td>
</tr>
<tr>
<td>q_hv_kvar</td>
<td>float</td>
<td>reactive power flow at the high voltage transformer bus [kVar]</td>
</tr>
<tr>
<td>p_mv_kw</td>
<td>float</td>
<td>active power flow at the medium voltage transformer bus [kW]</td>
</tr>
<tr>
<td>q_mv_kvar</td>
<td>float</td>
<td>reactive power flow at the medium voltage transformer bus [kVar]</td>
</tr>
<tr>
<td>p_lv_kw</td>
<td>float</td>
<td>active power flow at the low voltage transformer bus [kW]</td>
</tr>
<tr>
<td>q_lv_kvar</td>
<td>float</td>
<td>reactive power flow at the low voltage transformer bus [kVar]</td>
</tr>
<tr>
<td>pl_kw</td>
<td>float</td>
<td>active power losses of the transformer [kW]</td>
</tr>
<tr>
<td>ql_kvar</td>
<td>float</td>
<td>reactive power consumption of the transformer [kvar]</td>
</tr>
<tr>
<td>i_hv_ka</td>
<td>float</td>
<td>current at the high voltage side of the transformer [kA]</td>
</tr>
<tr>
<td>i_mv_ka</td>
<td>float</td>
<td>current at the medium voltage side of the transformer [kA]</td>
</tr>
<tr>
<td>i_lv_ka</td>
<td>float</td>
<td>current at the low voltage side of the transformer [kA]</td>
</tr>
<tr>
<td>loading_percent</td>
<td>float</td>
<td>transformer utilization [%]</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
p_{hv\_kw} &= Re(\bar{v}_{hv} \cdot \bar{i}_{hv}) \\
q_{hv\_kvar} &= Im(\bar{v}_{hv} \cdot \bar{i}_{hv}) \\
p_{mv\_kw} &= Re(\bar{v}_{mv} \cdot \bar{i}_{mv}) \\
q_{mv\_kvar} &= Im(\bar{v}_{mv} \cdot \bar{i}_{mv}) \\
p_{lv\_kw} &= Re(\bar{v}_{lv} \cdot \bar{i}_{lv}) \\
q_{lv\_kvar} &= Im(\bar{v}_{lv} \cdot \bar{i}_{lv}) \\
pl\_kw &= p_{hv\_kw} + p_{lv\_kw} \\
ql\_kvar &= q_{hv\_kvar} + q_{lv\_kvar} \\
i_{hv\_ka} &= i_{hv} \\
i_{mv\_ka} &= i_{mv} \\
i_{lv\_ka} &= i_{lv} \\
\end{align*}
\]

The definition of the transformer loading depends on the trafo_loading parameter of the power flow.

For trafo_loading=\textasciitilde current\textasciitilde, the loading is calculated as:

\[
\text{loading\_percent} = \max(\frac{i_{hv} \cdot v_{hv\_kv}}{sn_{hv\_kva}}, \frac{i_{mv} \cdot v_{mv\_kv}}{sn_{mv\_kva}}, \frac{i_{lv} \cdot v_{lv\_kv}}{sn_{lv\_kva}}) \cdot 100
\]

For trafo_loading=\textasciitilde power\textasciitilde, the loading is defined as:

\[
\text{loading\_percent} = \max(\frac{i_{hv} \cdot v_{hv}}{sn_{hv\_kva}}, \frac{i_{mv} \cdot v_{mv}}{sn_{mv\_kva}}, \frac{i_{lv} \cdot v_{lv}}{sn_{lv\_kva}}) \cdot 100
\]

### 2.10 Generator

See also:

*Unit Systems and Conventions*

#### 2.10.1 Create Function

\[
pandapower.create_gen(net, bus, p_kw, vm\_pu=1., sn\_kva=nan, name=None, index=None, max_q_kvar=nan, min_q_kvar=nan, min_p_kw=nan, max_p_kw=nan, scaling=1., type=None, controllable=nan, vn\_kv=nan, xdss=nan, rdss=nan, cos\_phi=nan, in\_service=True)
\]

Adds a generator to the network.

Generators are always modelled as voltage controlled PV nodes, which is why the input parameter is active power and a voltage set point. If you want to model a generator as PQ load with fixed reactive power and variable voltage, please use a static generator instead.
**INPUT:** net - The net within this generator should be created
  
  **bus** (int) - The bus id to which the generator is connected
  
  **OPTIONAL:** p_kw (float, default 0) - The real power of the generator (negative for generation!)
  
  **vm_pu** (float, default 0) - The voltage set point of the generator.
  
  **sn_kva** (float, None) - Nominal power of the generator
  
  **name** (string, None) - The name for this generator
  
  **index** (int, None) - Force a specified ID if it is available. If None, the index one higher than the highest already existing index is selected.
  
  **scaling** (float, 1.0) - scaling factor which for the active power of the generator
  
  **type** (string, None) - type variable to classify generators
  
  **controllable** (bool, NaN) - Whether this generator is controllable by the optimal powerflow
  
  **vn_kv** (float, NaN) - Rated voltage of the generator for short-circuit calculation
  
  **xdss** (float, NaN) - Subtransient generator reactance for short-circuit calculation
  
  **rdss** (float, NaN) - Subtransient generator resistance for short-circuit calculation
  
  **cos_phi** (float, NaN) - Rated cosine phi of the generator for short-circuit calculation
  
  **in_service** (bool, True) - True for in_service or False for out of service
  
  **max_p_kw** (float, default NaN) - Maximum active power injection - necessary for OPF
  
  **min_p_kw** (float, default NaN) - Minimum active power injection - necessary for OPF
  
  **max_q_kvar** (float, default NaN) - Maximum reactive power injection - necessary for OPF
  
  **min_q_kvar** (float, default NaN) - Minimum reactive power injection - necessary for OPF
  
  **OUTPUT:** index (int) - The unique ID of the created generator
  
  **EXAMPLE:** create_gen(net, 1, p_kw = -120, vm_pu = 1.02)

### 2.10.2 Input Parameters

`net.gen`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Value Range</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td></td>
<td>name of the generator</td>
</tr>
<tr>
<td>type</td>
<td>string</td>
<td></td>
<td>type variable to classify generators</td>
</tr>
<tr>
<td>bus*</td>
<td>integer</td>
<td></td>
<td>index of connected bus</td>
</tr>
<tr>
<td>p_kw*</td>
<td>float</td>
<td>≤ 0</td>
<td>the real power of the generator [kW]</td>
</tr>
<tr>
<td>vm_pu*</td>
<td>float</td>
<td></td>
<td>voltage set point of the generator [p.u]</td>
</tr>
<tr>
<td>sn_kva</td>
<td>float</td>
<td>&gt; 0</td>
<td>nominal power of the generator [kVA]</td>
</tr>
<tr>
<td>min_q_kvar</td>
<td>float</td>
<td></td>
<td>minimal reactive power of the generator [kVar]</td>
</tr>
<tr>
<td>max_q_kvar</td>
<td>float</td>
<td></td>
<td>maximal reactive power of the generator [kVar]</td>
</tr>
<tr>
<td>scaling*</td>
<td>float</td>
<td>≤ 0</td>
<td>scaling factor for the active power</td>
</tr>
<tr>
<td>max_p_kw**</td>
<td>float</td>
<td></td>
<td>Maximum active power</td>
</tr>
<tr>
<td>Parameter</td>
<td>Datatype</td>
<td>Value Range</td>
<td>Explanation</td>
</tr>
<tr>
<td>--------------</td>
<td>----------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>min_p_kw</td>
<td>float</td>
<td></td>
<td>Minimum active power</td>
</tr>
<tr>
<td>max_q_kvar</td>
<td>float</td>
<td></td>
<td>Maximum reactive power</td>
</tr>
<tr>
<td>min_q_kvar</td>
<td>float</td>
<td></td>
<td>Minimum reactive power</td>
</tr>
<tr>
<td>controllable</td>
<td>bool</td>
<td>True/False</td>
<td>States if a gen is controllable or not. Currently gens must be controllable, because there is no method to respect uncontrollable gens yet.</td>
</tr>
<tr>
<td>vn_kv</td>
<td>float</td>
<td></td>
<td>Rated voltage of the generator</td>
</tr>
<tr>
<td>xdss</td>
<td>float</td>
<td>&gt; 0</td>
<td>Subtransient generator reactance</td>
</tr>
<tr>
<td>rdss</td>
<td>float</td>
<td>&gt; 0</td>
<td>Subtransient generator reactance</td>
</tr>
<tr>
<td>cos_phi</td>
<td>float</td>
<td>0 ≤ 1</td>
<td>Subtransient generator reactance</td>
</tr>
<tr>
<td>in_service</td>
<td>boolean</td>
<td>True / False</td>
<td>Rated generator cosine phi specifies if the generator is in service.</td>
</tr>
</tbody>
</table>

*necessary for executing a power flow calculation **optimal power flow parameter ***short-circuit calculation parameter

**Note:** Active power should normally be negative to model a voltage controlled generator, since all power values are given in the load reference system. A generator with positive active power represents a voltage controlled machine. If you want to model constant generation without voltage control, use the Static Generator element.

### 2.10.3 Electric Model

Generators are modelled as PV-nodes in the power flow:

![Electric Model Diagram](image)

Voltage magnitude and active power are defined by the input parameters in the generator table:

\[
P_{gen} = p\_kw \times scaling
\]

\[
v_{bus} = v_m\_pu
\]

### 2.10.4 Result Parameters

net.res_gen

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_kw</td>
<td>float</td>
<td>resulting active power demand after scaling [kW]</td>
</tr>
<tr>
<td>q_kvar</td>
<td>float</td>
<td>resulting reactive power demand after scaling [kVar]</td>
</tr>
</tbody>
</table>
## Datastructure and Elements

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>va_degree</td>
<td>float</td>
<td>generator voltage angle [degree]</td>
</tr>
<tr>
<td>vm_pu</td>
<td>float</td>
<td>voltage at the generator [p.u]</td>
</tr>
</tbody>
</table>

The power flow returns reactive generator power and generator voltage angle:

\[
\begin{align*}
    p_{\text{kw}} &= P_{\text{gen}} \\
    q_{\text{kvar}} &= Q_{\text{gen}} \\
    \text{va\_degree} &= \angle V_{\text{bus}} \\
    \text{vm\_degree} &= |V_{\text{bus}}|
\end{align*}
\]

**Note:** If the power flow is run with the enforce_qlims option and the generator reactive power exceeds / underruns the maximum / minimum reactive power limit, the generator is converted to a static generator with the maximum / minimum reactive power as constant reactive power generation. The voltage at the generator bus is then no longer equal to the voltage set point defined in the parameter table.

### 2.11 Shunt

**See also:**

*Unit Systems and Conventions*

#### 2.11.1 Create Function

```python
def create_shunt(net, bus, q_kvar, p_kw=0.0, vn_kv=None, step=1, name=None, in_service=True, index=None)
```

Creates a shunt element

**INPUT:**
- `net` (pandapowerNet) - The pandapower network in which the element is created
- `bus` - bus number of bus to whom the shunt is connected to
- `p_kw` - shunt active power in kW at v= 1.0 p.u.
- `q_kvar` - shunt susceptance in kVAR at v= 1.0 p.u.

**OPTIONAL:**
- `vn_kv` (float, None) - rated voltage of the shunt. Defaults to rated voltage of connected bus
- `step` (int, 1) - step of shunt with which power values are multiplied
- `name` (str, None) - element name
- `in_service` (boolean, True) - True for in_service or False for out of service
- `index` (int, None) - Force a specified ID if it is available. If None, the index one higher than the highest already existing index is selected.

**OUTPUT:**
- `index` (int) - The unique ID of the created shunt

**EXAMPLE:**
```python
create_shunt(net, 0, 20)
```

```python
def create_shunt_as_capacitor(net, bus, q_kvar, loss_factor, **kwargs)
```

Creates a shunt element representing a capacitor bank.

**INPUT:**
- `net` (pandapowerNet) - The pandapower network in which the element is created
- `bus` - bus number of bus to whom the shunt is connected to
- `q_kvar` (float) - reactive power of the capacitor bank at rated voltage
- `loss_factor` (float) - loss factor tan(delta) of the capacitor bank
**kwargs are passed to the create_shunt function

**OUTPUT:** index (int) - The unique ID of the created shunt

### 2.11.2 Input Parameters

**net.shunt**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Value Range</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td></td>
<td>name of the shunt</td>
</tr>
<tr>
<td>bus*</td>
<td>integer</td>
<td>≥ 0</td>
<td>index of bus where the impedance starts</td>
</tr>
<tr>
<td>p_kw*</td>
<td>float</td>
<td>≥ 0</td>
<td>shunt active power in kW at v= 1.0 p.u.</td>
</tr>
<tr>
<td>q_kvar*</td>
<td>float</td>
<td></td>
<td>shunt reactive power in kvar at v= 1.0 p.u.</td>
</tr>
<tr>
<td>vn_kv*</td>
<td>float</td>
<td>&gt; 0</td>
<td>rated voltage of the shunt element</td>
</tr>
<tr>
<td>step*</td>
<td>integer</td>
<td>≥ 1</td>
<td>step position of the shunt</td>
</tr>
<tr>
<td>in_service*</td>
<td>boolean</td>
<td>True / False</td>
<td>specifies if the shunt is in service.</td>
</tr>
</tbody>
</table>

*necessary for executing a power flow calculation.

### 2.11.3 Electric Model

The power values are given at \( v = 1 \) pu and are scaled linearly with the number of steps:

\[
S'_{\text{shunt,ref}} = (p_{\text{kw}} + j \cdot q_{\text{kvar}}) \cdot \text{step}
\]

Since \( S'_{\text{shunt,ref}} \) is the apparent power at the nominal voltage, we know that:

\[
Y_{\text{shunt}} = \frac{S'_{\text{shunt,ref}}}{V_{\text{nom}}^2}
\]

Converting to the per unit system results in:

\[
y_{\text{shunt}} = \frac{S'_{\text{shunt,ref}}}{V_N^2} \cdot Z_N
\]

\[
y_{\text{shunt}} = \frac{S'_{\text{shunt,ref}}}{V_N^2} \cdot \frac{V_N^2}{S_N}
\]

\[
y_{\text{shunt}} = \frac{S'_{\text{shunt,ref}}}{S_N}
\]

with the reference values for the per unit system as defined in *Unit Systems and Conventions.*


### 2.11.4 Result Parameters

*net.res_shunt*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_kw</td>
<td>float</td>
<td>shunt active power consumption [kW]</td>
</tr>
<tr>
<td>q_kvar</td>
<td>float</td>
<td>shunt reactive power consumption [kVAR]</td>
</tr>
<tr>
<td>vm_pu</td>
<td>float</td>
<td>voltage magnitude at shunt bus [pu]</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
    p_{kw} &= \text{Re}(v_{bus} \cdot i_{shunt}) \\
    q_{kvar} &= \text{Im}(v_{bus} \cdot i_{shunt}) \\
    v_{m pu} &= v_{bus}
\end{align*}
\]

### 2.12 Impedance

**See also:**

*Unit Systems and Conventions*

#### 2.12.1 Create Function

*net.impedance*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Value Range</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td></td>
<td>name of the impedance</td>
</tr>
<tr>
<td>from_bus*</td>
<td>integer</td>
<td></td>
<td>index of bus where the impedance starts</td>
</tr>
<tr>
<td>to_bus*</td>
<td>integer</td>
<td></td>
<td>index of bus where the impedance ends</td>
</tr>
<tr>
<td>rft_pu*</td>
<td>float</td>
<td>&gt; 0</td>
<td>resistance of the impedance from 'from’ to ‘to’ bus [p.u]</td>
</tr>
<tr>
<td>xft_pu*</td>
<td>float</td>
<td>&gt; 0</td>
<td>reactance of the impedance from ‘from’ to ‘to’ bus [p.u]</td>
</tr>
<tr>
<td>rtf_pu*</td>
<td>float</td>
<td>&gt; 0</td>
<td>resistance of the impedance from ‘to’ to ‘from’ bus [p.u]</td>
</tr>
<tr>
<td>xtf_pu*</td>
<td>float</td>
<td>&gt; 0</td>
<td>reactance of the impedance from ‘to’ to ‘from’ bus [p.u]</td>
</tr>
</tbody>
</table>

```
pandapower.create_impedance(net, from_bus, to_bus, rft_pu, xft_pu, sn_kva, rtf_pu=None, xtf_pu=None, name=None, in_service=True, index=None)
```

Creates an per unit impedance element

**INPUT:** *net* (pandapowerNet) - The pandapower network in which the element is created

- **from_bus** (int) - starting bus of the impedance
- **to_bus** (int) - ending bus of the impedance
- **r_pu** (float) - real part of the impedance in per unit
- **x_pu** (float) - imaginary part of the impedance in per unit
- **sn_kva** (float) - rated power of the impedance in kVA

**OUTPUT:**

impedance id
2.12.3 Electric Model

The impedance is modelled as a longitudinal per unit impedance with \( z_{ft} \neq z_{tf} \):

\[
\begin{align*}
Z_{ft} &= (r_{ft\_pu} + j \cdot x_{ft\_pu}) \cdot \frac{S_N}{sn\_kva} \\
Z_{tf} &= (r_{ft\_pu} + j \cdot x_{tf\_pu}) \cdot \frac{S_N}{sn\_kva}
\end{align*}
\]

where \( S_N \) is the reference power of the per unit system (see Unit Systems and Conventions).

The asymmetric impedance results in an asymmetric nodal point admittance matrix:

\[
\begin{bmatrix}
Y_{00} & \cdots & \cdots & Y_{nn} \\
\vdots & \ddots & y_{ft} & \vdots \\
\vdots & y_{tf} & \ddots & \vdots \\
Y_{n0} & \cdots & \cdots & Y_{nn}
\end{bmatrix}
\]

2.12.4 Result Parameters

\( net\_res\_impedance \)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_from_kw</td>
<td>float</td>
<td>active power flow into the impedance at “from” bus [kW]</td>
</tr>
<tr>
<td>q_from_kvar</td>
<td>float</td>
<td>reactive power flow into the impedance at “from” bus [kVAR]</td>
</tr>
<tr>
<td>p_to_kw</td>
<td>float</td>
<td>active power flow into the impedance at “to” bus [kW]</td>
</tr>
<tr>
<td>q_to_kvar</td>
<td>float</td>
<td>reactive power flow into the impedance at “to” bus [kVAR]</td>
</tr>
<tr>
<td>pl_kw</td>
<td>float</td>
<td>active power losses of the impedance [kW]</td>
</tr>
<tr>
<td>ql_kvar</td>
<td>float</td>
<td>reactive power consumption of the impedance [kVAR]</td>
</tr>
<tr>
<td>i_from_ka</td>
<td>float</td>
<td>current at from bus [kA]</td>
</tr>
<tr>
<td>i_to_ka</td>
<td>float</td>
<td>current at to bus [kA]</td>
</tr>
</tbody>
</table>
\[
i_{\text{from}_k} = i_{\text{from}} \\
i_{\text{to}_k} = i_{\text{to}} \\
p_{\text{from}_k} = \text{Re}(v_{\text{from}} \cdot -\overline{I}_{\text{from}_k}) \\
q_{\text{from}_k} = \text{Im}(v_{\text{from}} \cdot -\overline{I}_{\text{from}_k}) \\
p_{\text{to}_k} = \text{Re}(v_{\text{to}} \cdot I_{\text{to}_k}) \\
q_{\text{to}_k} = \text{Im}(v_{\text{to}} \cdot I_{\text{to}_k}) \\
pl_k = p_{\text{from}_k} + p_{\text{to}_k} \\
ql_k = q_{\text{from}_k} + q_{\text{to}_k}
\]

### 2.13 Ward

See also:

*Unit Systems and Conventions*

#### 2.13.1 Create Function

```python
pandapower.create_ward(net, bus, ps_kw, qs_kvar, pz_kw, qz_kvar, name=None, in_service=True, index=None)
```

Creates a ward equivalent.

A ward equivalent is a combination of an impedance load and a PQ load.

**INPUT:** 
- `net` (pandapowernet) - The pandapower net within the element should be created
- `bus` (int) - bus of the ward equivalent
- `ps_kw` (float) - active power of the PQ load
- `qs_kvar` (float) - reactive power of the PQ load
- `pz_kw` (float) - active power of the impedance load in kW at 1.pu voltage
- `qz_kvar` (float) - reactive power of the impedance load in kVar at 1.pu voltage

**OUTPUT:** ward id

#### 2.13.2 Input Parameters

```
net.ward
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Value Range</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td></td>
<td>name of the ward equivalent</td>
</tr>
<tr>
<td>bus*</td>
<td>integer</td>
<td></td>
<td>index of connected bus</td>
</tr>
<tr>
<td>ps_kw*</td>
<td>float</td>
<td></td>
<td>constant active power demand [kW]</td>
</tr>
<tr>
<td>qs_kvar*</td>
<td>float</td>
<td></td>
<td>constant reactive power demand [kVar]</td>
</tr>
<tr>
<td>pz_kw*</td>
<td>float</td>
<td></td>
<td>constant impedance active power demand at 1.0 pu [kW]</td>
</tr>
<tr>
<td>qz_kvar*</td>
<td>float</td>
<td></td>
<td>constant impedance reactive power demand at 1.0 pu [kVar]</td>
</tr>
<tr>
<td>in_service*</td>
<td>boolean</td>
<td>True / False</td>
<td>specifies if the ward equivalent is in service.</td>
</tr>
</tbody>
</table>

*necessary for executing a power flow calculation.
2.13.3 Electric Model

The ward equivalent is a combination of a constant apparent power consumption and a constant impedance load. The constant apparent power is given by:

\[
\begin{align*}
P_{\text{const}} &= p_{s\_kw} \\
Q_{\text{const}} &= q_{s\_kvar}
\end{align*}
\]

The shunt admittance part of the ward equivalent is calculated as described here:

\[
y_{\text{shunt}} = \frac{p_{z\_kw} + j \cdot q_{z\_kvar}}{S_N}
\]

2.13.4 Result Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_kw</td>
<td>float</td>
<td>active power demand of the ward equivalent [kW]</td>
</tr>
<tr>
<td>q_kvar</td>
<td>float</td>
<td>reactive power demand of the ward equivalent [kVar]</td>
</tr>
<tr>
<td>vm_pu</td>
<td>float</td>
<td>voltage at the ward bus [p.u]</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
vm_{\_pu} &= v_{bus} \\
p_{\_kw} &= P_{\text{const}} + \text{Re}(\frac{V_{bus}^2}{Y_{\text{shunt}}}) \\
q_{\_kvar} &= Q_{\text{const}} + \text{Im}(\frac{V_{bus}^2}{Y_{\text{shunt}}})
\end{align*}
\]
2.14 Extended Ward

See also:

Unit Systems and Conventions

2.14.1 Create Function

```python
def create_xward(net, bus, ps_kw, qs_kvar, pz_kw, qz_kvar, r_ohm, x_ohm, vm pu, in_service=True, name=None, index=None):
    Creates an extended ward equivalent.
    A ward equivalent is a combination of an impedance load, a PQ load and as voltage source with an internal impedance.

    INPUT: net - The pandapower net within the impedance should be created
           bus (int) - bus of the ward equivalent
           ps_kw (float) - active power of the PQ load
           qs_kvar (float) - reactive power of the PQ load
           pz_kw (float) - active power of the impedance load in kW at 1.pu voltage
           qz_kvar (float) - reactive power of the impedance load in kVar at 1.pu voltage
           vm pu (float)

    OUTPUT: xward id
```

2.14.2 Result Parameters

```plaintext
net.xward

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Value Range</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>name*</td>
<td>string</td>
<td></td>
<td>name of the extended ward equivalent</td>
</tr>
<tr>
<td>bus*</td>
<td>integer</td>
<td></td>
<td>index of connected bus</td>
</tr>
<tr>
<td>ps_kw*</td>
<td>float</td>
<td></td>
<td>constant active power demand [kW]</td>
</tr>
<tr>
<td>qs_kvar*</td>
<td>float</td>
<td></td>
<td>constant reactive power demand [kVar]</td>
</tr>
<tr>
<td>pz_kw*</td>
<td>float</td>
<td></td>
<td>constant impedance active power demand at 1.0 pu [kW]</td>
</tr>
<tr>
<td>qz_kvar*</td>
<td>float</td>
<td></td>
<td>constant impedance reactive power demand at 1.0 pu [kVar]</td>
</tr>
<tr>
<td>r_pu*</td>
<td>float</td>
<td>&gt; 0</td>
<td>internal resistance of the voltage source [p.u]</td>
</tr>
<tr>
<td>x_pu*</td>
<td>float</td>
<td>&gt; 0</td>
<td>internal reactance of the voltage source [p.u]</td>
</tr>
<tr>
<td>vm pu*</td>
<td>float</td>
<td>&gt; 0</td>
<td>voltage source set point [p.u]</td>
</tr>
<tr>
<td>in_service*</td>
<td>boolean</td>
<td>True / False</td>
<td>specifies if the extended ward equivalent is in service.</td>
</tr>
</tbody>
</table>
```

*necessary for executing a power flow calculation.

2.14.3 Electric Model

The extended ward equivalent is a ward equivalent: with additional PV-node with an internal resistance.
The constant apparent power is given by:

\[ P_{\text{const}} = ps\_kw \]
\[ Q_{\text{const}} = qs\_kvar \]

The shunt admittance part of the extended ward equivalent is calculated as described here:

\[ y_{\text{shunt}} = \frac{pz\_kw + j \cdot qz\_kvar}{SN} \]

The internal resistance is defined as:

\[ z_{\text{int}} = r\_pu + j \cdot x\_pu \]

The internal voltage source is modelled as a PV-node (generator) with:

\[ p\_kw = 0 \]
\[ vm\_pu = vm\_pu \]

### 2.14.4 Result Parameters

**net.res_xward**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_kw</td>
<td>float</td>
<td>active power demand of the ward equivalent [kW]</td>
</tr>
<tr>
<td>q_kvar</td>
<td>float</td>
<td>reactive power demand of the ward equivalent [kVar]</td>
</tr>
<tr>
<td>vm_pu</td>
<td>float</td>
<td>voltage at the ward bus [p.u]</td>
</tr>
</tbody>
</table>

\[ vm\_pu = v_{bus} \]
\[ p\_kw = P_{\text{const}} + Re\left(\frac{V_{bus}^2}{Y_{\text{shunt}}}\right) + Re\left(I_{\text{int}} \cdot V_{bus}\right) \]
\[ q_{\text{kvar}} = Q_{\text{const}} + Im\left(\frac{V_{bus}^2}{Y_{\text{shunt}}} + I_{\text{int}} \cdot V_{bus}\right) \]
2.15 DC Line

See also:

Unit Systems and Conventions

2.15.1 Create Function

```python
pandapower.create_dcline(net, from_bus, to_bus, p_kw, loss_percent, loss_kw,
vm_from_pu, vm_to_pu, index=None, name=None,
max_p_kw=nan, min_q_from_kvar=nan, min_q_to_kvar=nan,
max_q_from_kvar=nan, max_q_to_kvar=nan, in_service=True)
```

Creates a dc line.

**INPUT:**
- `from_bus` (int) - ID of the bus on one side which the line will be connected with
- `to_bus` (int) - ID of the bus on the other side which the line will be connected with
- `loss_percent` - (float) Standard deviation in the same unit as the measurement.
- `loss_kw` - (int) Index of bus. Determines the position of the measurement for line/transformer measurements (bus == from_bus: measurement at from_bus; same for to_bus)
- `vm_from_pu` - (int, None) Index of measured element, if element_type is “line” or “transformer”.
- `vm_to_pu` - (int, None) Index of measured element, if element_type is “line” or “transformer”.

**OPTIONAL:**
- `index` (int, None) - Force a specified ID if it is available. If None, the index one higher than the highest already existing index is selected.
- `name` (str, None) - A custom name for this dc line
- `in_service` (boolean) - True for in_service or False for out of service
  - `max_p_kw` - Maximum active power flow. Only respected for OPF
  - `min_q_from_kvar` - Minimum reactive power at from bus. Necessary for OPF
  - `min_q_to_kvar` - Minimum reactive power at to bus. Necessary for OPF
  - `max_q_from_kvar` - Maximum reactive power at from bus. Necessary for OPF
  - `max_q_to_kvar` - Maximum reactive power at to bus. Necessary for OPF

**OUTPUT:**
- `index` (int) - The unique ID of the created element

**EXAMPLE:**
```
create_dcline(net, from_bus=0, to_bus=1, p_kw=1e4, loss_percent=1.2, loss_kw=25,
vm_from_pu=1.01, vm_to_pu=1.02)
```

2.15.2 Input Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Value Range</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td></td>
<td>name of the generator</td>
</tr>
<tr>
<td>from_bus*</td>
<td>integer</td>
<td></td>
<td>Index of bus where the dc line starts</td>
</tr>
<tr>
<td>to_bus*</td>
<td>integer</td>
<td></td>
<td>Index of bus where the dc line ends</td>
</tr>
<tr>
<td>p_kw*</td>
<td>float</td>
<td>&gt; 0</td>
<td>Active power transmitted from ‘from_bus’ to ‘to_bus’</td>
</tr>
<tr>
<td>loss_percent*</td>
<td>float</td>
<td>&gt; 0</td>
<td>Relative transmission loss in percent of active power transmission</td>
</tr>
<tr>
<td>loss_kw*</td>
<td>float</td>
<td>&gt; 0</td>
<td>Total transmission loss in kW</td>
</tr>
</tbody>
</table>
### 2 Data Structure and Elements

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Value Range</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>vm_from_pu*</td>
<td>float</td>
<td>&gt; 0</td>
<td>Voltage setpoint at from bus</td>
</tr>
<tr>
<td>vm_to_pu*</td>
<td>float</td>
<td>&gt; 0</td>
<td>Voltage setpoint at to bus</td>
</tr>
<tr>
<td>max_p_kw*</td>
<td>float</td>
<td>&gt; 0</td>
<td>Maximum active power transmission</td>
</tr>
<tr>
<td>min_q_from_kvar*</td>
<td>float</td>
<td>&gt; 0</td>
<td>Minimum reactive power at from bus</td>
</tr>
<tr>
<td>max_q_from_kvar*</td>
<td>float</td>
<td>&gt; 0</td>
<td>Maximum reactive power at from bus</td>
</tr>
<tr>
<td>min_q_to_kvar*</td>
<td>float</td>
<td>&gt; 0</td>
<td>Minimum reactive power at to bus</td>
</tr>
<tr>
<td>max_q_to_kvar*</td>
<td>float</td>
<td>&gt; 0</td>
<td>Maximum reactive power at to bus</td>
</tr>
<tr>
<td>in_service*</td>
<td>bool</td>
<td>True/False</td>
<td>Specifies if DC line is in service</td>
</tr>
</tbody>
</table>

* necessary for executing a power flow calculation ** optimal power flow parameter

**Note:** DC line is only able to model one-directional loadflow for now, which is why p_kw / max_p_kw have to be > 0.

#### 2.15.3 Electric Model

A DC line is modelled as two generators in the loadflow:

![Diagram of DC line model](image)

The active power at the from side is defined by the parameters in the dcline table. The active power at the to side is equal to the active power on the from side minus the losses of the DC line.

The voltage control with reactive power works just as described for the generator model. Maximum and Minimum reactive power limits are considered in the OPF, and in the PF if it is run with enforce_q_lims=True.

#### 2.15.4 Result Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_from_kw</td>
<td>float</td>
<td>Active power flow into the line at 'from_bus' [kW]</td>
</tr>
<tr>
<td>q_from_kvar</td>
<td>float</td>
<td>Reactive power flow into the line at 'from_bus' [kVar]</td>
</tr>
<tr>
<td>p_to_kw</td>
<td>float</td>
<td>Active power flow into the line at 'to_bus' [kW]</td>
</tr>
<tr>
<td>q_to_kvar</td>
<td>float</td>
<td>Reactive power flow into the line at 'to_bus' [kVar]</td>
</tr>
<tr>
<td>pl_kw</td>
<td>float</td>
<td>Active power losses of the line [kW]</td>
</tr>
<tr>
<td>vm_from_pu</td>
<td>float</td>
<td>Voltage magnitude at 'from_bus' [p.u]</td>
</tr>
<tr>
<td>va_from_deg</td>
<td>float</td>
<td>Voltage angle at 'from_bus' [degree]</td>
</tr>
</tbody>
</table>
### 2.16 Measurement

#### 2.16.1 Create Function

pandapower.create_measurement(net, type, element_type, value, std_dev, bus, element=None, check_existing=True, index=None, name=None)

Creates a measurement, which is used by the estimation module. Possible types of measurements are: v, p, q, i

**INPUT:**
- **type** (string) - Type of measurement. “v”, “p”, “q”, “i” are possible.
- **element_type** (string) - Clarifies which element is measured. “bus”, “line”, “transformer” are possible.
- **value** (float) - Measurement value. Units are “kW” for P, “kVar” for Q, “p.u.” for V, “A” for I. Generation is a positive bus power injection, consumption negative.
- **std_dev** (float) - Standard deviation in the same unit as the measurement.
- **bus** (int) - Index of bus. Determines the position of the measurement for line/transformer measurements (bus == from_bus: measurement at from_bus; same for to_bus)
- **element** (int, None) - Index of measured element, if element_type is “line” or “transformer”.

**OPTIONAL:**
- **check_existing** (bool) - Check for and replace existing measurements for this bus and type. Set it to false for performance improvements which can cause unsafe behaviour.
- **name** (str, None) - name of measurement.

**OUTPUT:**
- (int) Index of measurement

**EXAMPLE:** 500 kW load measurement with 10 kW standard deviation on bus 0: create_measurement(net, “p”, “bus”, -500., 10., 0)

#### 2.16.2 Input Parameters

**net.measurement**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Value Range</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>vm_to_pu</td>
<td>float</td>
<td></td>
<td>voltage magnitude at 'to_bus' [p.u]</td>
</tr>
<tr>
<td>va_to_degree</td>
<td>float</td>
<td></td>
<td>voltage angle at 'to_bus' [degree]</td>
</tr>
<tr>
<td>Parameter</td>
<td>Datatype</td>
<td>Value Range</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>type</td>
<td>string</td>
<td>“p” “q” “i” “v”</td>
<td>Defines what physical quantity is measured</td>
</tr>
<tr>
<td>element_type</td>
<td>string</td>
<td>“bus” “line” “transformer”</td>
<td>Defines which element type is equipped with the measurement</td>
</tr>
<tr>
<td>value</td>
<td>float</td>
<td></td>
<td>Measurement value</td>
</tr>
<tr>
<td>std_dev</td>
<td>float</td>
<td></td>
<td>Standard deviation (same unit as measurement)</td>
</tr>
<tr>
<td>bus</td>
<td>int</td>
<td>must be in net.bus.index</td>
<td>Defines the bus at which the measurement is placed. For line or transformer measurement it defines the side at which the measurement is placed (from_bus or to_bus).</td>
</tr>
<tr>
<td>element</td>
<td>int</td>
<td>must be in net.line.index or net.trafo.index</td>
<td>If the element_type is “line” or “transformer”, element is the index of the relevant element. For “bus” measurements it is None (default)</td>
</tr>
<tr>
<td>check_existing</td>
<td>bool</td>
<td></td>
<td>Checks if a measurement of the type already exists and overwrites it. If set to False, the measurement may be added twice (unsafe behaviour), but the performance increases</td>
</tr>
<tr>
<td>index</td>
<td>int</td>
<td></td>
<td>Defines a specific index for the new measurement (if possible)</td>
</tr>
</tbody>
</table>
3 Standard Type Libraries

Lines and transformers have two different categories of parameters: parameter that depend on the specific element (like the length of a line or the bus to which a transformer is connected to etc.) and parameter that only depend on the type of line or transformer which is used (like the rated power of a transformer or the resistance per kilometer line).

The standard type library provides a database of different types for transformer and lines, so that you only have to chose a certain type and not define all parameters individually for each line or transformer. The standard types are saved in the network as a dictionary in the form of:

```
net.std_types = {"line": {"standard_type": {"parameter": value, ...}},
                   "trafo": {"standard_type": {"parameter": value, ...}},
                   "trafo3w": {"standard_type": {"parameter": value, ...}}}
```

The create_line and create_transformer functions use this database when you create a line or transformer with a certain standard type. You can also use the standard type functions directly to create new types in the database, directly load type data, change types or check if a certain type exists. You can also add additional type parameters which are not added to the pandas table by default (e.g. diameter of the conductor).

For a introduction on how to use the standard type library, see the interactive tutorial on standard types.

3.1 Basic Standard Types

Every pandapower network comes with a default set of standard types.

**Note:** The pandapower standard types are compatible with 50 Hz systems, please be aware that the standard type values might not be realistic for 60 Hz (or other) power systems.
### 3.1.1 Lines

<table>
<thead>
<tr>
<th>r_ohm_per_km</th>
<th>x_ohm_per_km</th>
<th>c_nf_per_km</th>
<th>max_i_ka</th>
<th>type</th>
<th>q_mm²</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.194</td>
<td>0.315</td>
<td>11.25</td>
<td>0.47</td>
<td>ol</td>
<td>149</td>
</tr>
<tr>
<td>0.194</td>
<td>0.41</td>
<td>8.75</td>
<td>0.47</td>
<td>ol</td>
<td>149</td>
</tr>
<tr>
<td>0.194</td>
<td>0.337</td>
<td>10.5</td>
<td>0.47</td>
<td>ol</td>
<td>149</td>
</tr>
<tr>
<td>1.8769</td>
<td>0.35</td>
<td>1</td>
<td>0.105</td>
<td>ol</td>
<td>149</td>
</tr>
<tr>
<td>0.1571</td>
<td>0.4</td>
<td>8.8</td>
<td>0.535</td>
<td>ol</td>
<td>184</td>
</tr>
<tr>
<td>0.1571</td>
<td>0.33</td>
<td>10.75</td>
<td>0.535</td>
<td>ol</td>
<td>184</td>
</tr>
<tr>
<td>1.2012</td>
<td>0.355</td>
<td>11.25</td>
<td>0.14</td>
<td>ol</td>
<td>24</td>
</tr>
<tr>
<td>0.1158</td>
<td>0.39</td>
<td>9</td>
<td>0.645</td>
<td>ol</td>
<td>243</td>
</tr>
<tr>
<td>0.1158</td>
<td>0.32</td>
<td>11</td>
<td>0.645</td>
<td>ol</td>
<td>243</td>
</tr>
<tr>
<td>0.0949</td>
<td>0.38</td>
<td>9.2</td>
<td>0.74</td>
<td>ol</td>
<td>305</td>
</tr>
<tr>
<td>0.5939</td>
<td>0.3</td>
<td>12.2</td>
<td>0.21</td>
<td>ol</td>
<td>48</td>
</tr>
<tr>
<td>0.5939</td>
<td>0.372</td>
<td>9.5</td>
<td>0.21</td>
<td>ol</td>
<td>48</td>
</tr>
<tr>
<td>0.059</td>
<td>0.285</td>
<td>10</td>
<td>0.96</td>
<td>ol</td>
<td>490</td>
</tr>
<tr>
<td>0.059</td>
<td>0.253</td>
<td>11</td>
<td>0.96</td>
<td>ol</td>
<td>490</td>
</tr>
<tr>
<td>0.306</td>
<td>0.29</td>
<td>13.2</td>
<td>0.35</td>
<td>ol</td>
<td>94</td>
</tr>
<tr>
<td>0.306</td>
<td>0.33</td>
<td>10.75</td>
<td>0.35</td>
<td>ol</td>
<td>94</td>
</tr>
<tr>
<td>0.306</td>
<td>0.35</td>
<td>10</td>
<td>0.35</td>
<td>ol</td>
<td>94</td>
</tr>
<tr>
<td>0.153</td>
<td>0.166</td>
<td>112</td>
<td>0.366</td>
<td>cs</td>
<td>120</td>
</tr>
<tr>
<td>0.099</td>
<td>0.136</td>
<td>125</td>
<td>0.457</td>
<td>cs</td>
<td>185</td>
</tr>
<tr>
<td>0.075</td>
<td>0.149</td>
<td>135</td>
<td>0.526</td>
<td>cs</td>
<td>240</td>
</tr>
<tr>
<td>0.06</td>
<td>0.144</td>
<td>144</td>
<td>0.588</td>
<td>cs</td>
<td>300</td>
</tr>
<tr>
<td>0.161</td>
<td>0.117</td>
<td>273</td>
<td>0.362</td>
<td>cs</td>
<td>185</td>
</tr>
<tr>
<td>0.122</td>
<td>0.112</td>
<td>304</td>
<td>0.421</td>
<td>cs</td>
<td>240</td>
</tr>
<tr>
<td>0.313</td>
<td>0.132</td>
<td>216</td>
<td>0.252</td>
<td>cs</td>
<td>95</td>
</tr>
<tr>
<td>0.225</td>
<td>0.08</td>
<td>264</td>
<td>0.242</td>
<td>cs</td>
<td>120</td>
</tr>
<tr>
<td>0.208</td>
<td>0.08</td>
<td>261</td>
<td>0.27</td>
<td>cs</td>
<td>150</td>
</tr>
<tr>
<td>0.642</td>
<td>0.083</td>
<td>210</td>
<td>0.142</td>
<td>cs</td>
<td>50</td>
</tr>
</tbody>
</table>

### 3.1.2 Transformers

### 3.1.3 Three Winding Transformers

<table>
<thead>
<tr>
<th>sn_hv_kva</th>
<th>sn_mv_kva</th>
<th>sn_lv_kva</th>
<th>vn_hv_kv</th>
<th>vn_mv_kv</th>
<th>vn_lv_kv</th>
<th>vsc_hv_percent</th>
<th>vsc_mv_percent</th>
<th>vsc_lv_percent</th>
<th>vscr_hv_percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>63/25/38 MVA 110/10/10 kV</td>
<td>63000</td>
<td>25000</td>
<td>38000</td>
<td>110</td>
<td>20</td>
<td>10.4</td>
<td>10.4</td>
<td>10.4</td>
<td>0.28</td>
</tr>
<tr>
<td>63/25/38 MVA 110/20/10 kV</td>
<td>63000</td>
<td>25000</td>
<td>38000</td>
<td>110</td>
<td>20</td>
<td>10.4</td>
<td>10.4</td>
<td>10.4</td>
<td>0.28</td>
</tr>
<tr>
<td></td>
<td>vscr_mv_percent</td>
<td>vscr_lv_percent</td>
<td>ple_kw</td>
<td>t0_percent</td>
<td>shift_mv_degree</td>
<td>shift_lv_degree</td>
<td>tp_side</td>
<td>tp_mid</td>
<td>tp_min</td>
</tr>
<tr>
<td>-----</td>
<td>----------------</td>
<td>----------------</td>
<td>--------</td>
<td>------------</td>
<td>----------------</td>
<td>----------------</td>
<td>---------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>63/25/38 MVA 110/10/10 kV</td>
<td>0.32</td>
<td>0.35</td>
<td>35</td>
<td>0.89</td>
<td>0</td>
<td>0</td>
<td>hv</td>
<td>0</td>
<td>-10</td>
</tr>
<tr>
<td>63/25/38 MVA 110/20/10 kV</td>
<td>0.32</td>
<td>0.35</td>
<td>35</td>
<td>0.89</td>
<td>0</td>
<td>0</td>
<td>hv</td>
<td>0</td>
<td>-10</td>
</tr>
</tbody>
</table>
3.2 Manage Standard Types

3.2.1 Show all Available Standard Types

```python
pandapower.available_std_types(net, element='line')
```

Returns all standard types available for this network as a table.

**INPUT:**
- `net` - pandapower Network
- `element` - type of element ("line" or "trafo")

**OUTPUT:**
- `typedata` - table of standard type parameters

3.2.2 Create Standard Type

```python
pandapower.create_std_type(net, data, name, element='line', overwrite=True)
```

Creates type data in the type database. The parameters that are used for the loadflow have to be at least contained in data. These parameters are:

- `c_nf_per_km`, `r_ohm_per_km`, `x_ohm_per_km` and `max_i_ka` (for lines)
- `sn_kva`, `vn_hv_kv`, `vn_lv_kv`, `vsc_percent`, `vscr_percent`, `pfe_kw`, `i0_percent`, `shift_degree*` (for transformers)
- `sn_hv_kva`, `sn_mv_kva`, `sn_lv_kva`, `vn_hv_kv`, `vn_mv_kv`, `vn_lv_kv`, `vsc_hv_percent`, `vsc_mv_percent`, `vsc_lv_percent`, `vsc_hv_percent`, `vsc_mv_percent`, `vsc_lv_percent`, `pfe_kw`, `i0_percent`, `shift_mv_degree*`, `shift_lv_degree*` (for 3-winding-transformers)

Additional parameters can be added and later loaded into pandapower with the function `parameter_from_std_type`.

* only considered in loadflow if calculate_voltage_angles=True

The standard type is saved into the pandapower library of the given network by default.

**INPUT:**
- `net` - The pandapower network
- `data` - dictionary of standard type parameters
- `name` - name of the standard type as string
- `element` - "line", "trafo" or "trafo3w"

**EXAMPLE:**

```python
>>> line_data = {"c_nf_per_km": 0, "r_ohm_per_km": 0.642, "x_ohm_per_km": 0.083, "max_i_ka": 0.142, "type": "cs", "q_mm2": 50}
>>> pp.create_std_type(net, line_data, "NAYY 4×50 SE", element='line')
```

```python
pandapower.create_std_types(net, data, element='line', overwrite=True)
```

Creates multiple standard types in the type database.

**INPUT:**
- `net` - The pandapower network
- `data` - dictionary of standard type parameter sets
- `element` - "line", "trafo" or "trafo3w"

**EXAMPLE:**

```python
>>> linetypes = {"typ1": {"r_ohm_per_km": 0.01, "x_ohm_per_km": 0.02, "c_nf_per_km": 10, "max_i_ka": 0.4, "type": "cs"},
               "typ2": {"r_ohm_per_km": 0.015, "x_ohm_per_km": 0.01, "c_nf_per_km": 30, "max_i_ka": 0.3, "type": "cs"}}
>>> pp.create_std_types(net, data=linetypes, element='line')
```
3.2.3 Copy Standard Types

`pandapower.copy_std_types(to_net, from_net, element='line', overwrite=True)`

Transfers all standard types of one network to another.

**INPUT:**
- `to_net` - The pandapower network to which the standard types are copied
- `from_net` - The pandapower network from which the standard types are taken
- `element` - “line” or “trafo”
- `overwrite` - if True, overwrites standard types which already exist in `to_net`

3.2.4 Load Standard Types

`pandapower.load_std_type(net, name, element='line')`

Loads linetype data from the linetypes database. Issues a warning if linetype is unknown.

**INPUT:**
- `net` - The pandapower network
- `name` - name of the standard type as string
- `element` - “line” or “trafo”

**OUTPUT:** `typedata` - dictionary containing type data

3.2.5 Check if Standard Type Exists

`pandapower.std_type_exists(net, name, element='line')`

Checks if a standard type exists.

**INPUT:**
- `net` - pandapower Network
- `name` - name of the standard type as string
- `element` - type of element (“line” or “trafo”)  

**OUTPUT:** `exists` - True if standard type exists, False otherwise

3.2.6 Change Standard Type

`pandapower.change_std_type(net, eid, name, element='line')`

Changes the type of a given element in pandapower. Changes only parameter that are given for the type.

**INPUT:**
- `net` - pandapower network
- `eid` - element index (either line or transformer index)
- `element` - type of element (“line” or “trafo”)
- `name` - name of the new standard type

3.2.7 Load Additional Parameter from Library

`pandapower.parameter_from_std_type(net, parameter, element='line', fill=None)`

Adds additional parameters, which are not included in the original pandapower datastructure but are available in the standard type database to the pandapower net.

**INPUT:**
- `net` - pandapower network
- `parameter` - name of parameter as string
- `element` - type of element (“line” or “trafo”)
fill - fill-value that is assigned to all lines/trafos without a value for the parameter, either because the line/trafo has no type or because the type does not have a value for the parameter

3.2.8 Find Standard Type

pandapower.find_std_type_by_parameter(net, data, element='line', epsilon=0.0)

Searches for a std_type that fits all values given in the data dictionary with the margin of epsilon.

INPUT: net - pandapower network
data - dictionary of standard type parameters
element - type of element ("line" or "trafo")
epsilon - tolerance margin for parameter comparison

OUTPUT: fitting_types - list of fitting types or empty list

3.2.9 Delete Standard Type

pandapower.delete_std_type(net, name, element='line')

Deletes standard type parameters from database.

INPUT: net - pandapower Network
name - name of the standard type as string
element - type of element ("line" or "trafo")
4 Power Flow

The power flow is the most important static network calculation operation. This section shows you how to run different power flows (AC, DC, OPF), what known problems and caveats there are and how you can identify problems using the pandapower diagnostic function.

4.1 Run a Power Flow

pandapower provides an AC powerflow, DC powerflow and an OPF.

4.1.1 Power Flow

pandapower uses PYPOWER to solve the power flow problem:

```
pandapower.runpp(net, algorithm='nr', calculate_voltage_angles='auto', init='auto',
max_iteration='auto', tolerance_kva=1e-05, trafo_model='t',
trafo_loading='current', enforce_q_lims=False, numba=True, recycle=None,
check_connectivity=True, r_switch=0.0, voltage_depend_loads=True,
delta_q=0, **kwargs)
```

Runs PANDAPOWER AC Flow

**INPUT:** net - The pandapower format network

**OPTIONAL:** algorithm (str, “nr”) - algorithm that is used to solve the power flow problem.

The following algorithms are available:

- “nr” newton-raphson (pypower implementation with numba accelerations)
- “bfsw” backward/forward sweep (specially suited for radial and weakly-meshed networks)
- “gs” gauss-seidel (pypower implementation)
- “fdbx” (pypower implementation)
- “fdxb” (pypower implementation)

**calculate_voltage_angles** (bool, “auto”) - consider voltage angles in loadflow calculation

If True, voltage angles of ext_grids and transformer shifts are considered in the loadflow calculation. Considering the voltage angles is only necessary in meshed networks that are usually found in higher networks. That's why calculate_voltage_angles in “auto” mode defaults to:

- True, if the network voltage level is above 70 kV
- False otherwise

The network voltage level is defined as the maximum rated voltage in the network that is connected to a line.

**init** (str, “auto”) - initialization method of the loadflow pandapower supports four methods for initializing the loadflow:
• “auto” - init defaults to “dc” if calculate_voltage_angles is True or “flat” otherwise
• “flat” - flat start with voltage of 1.0pu and angle of 0° at all PQ-buses and 0° for PV buses as initial solution
• “dc” - initial DC loadflow before the AC loadflow. The results of the DC loadflow are used as initial solution for the AC loadflow.
• “results” - voltage vector of last loadflow from net.res_bus is used as initial solution. This can be useful to accelerate convergence in iterative loadflows like time series calculations.

Considering the voltage angles might lead to non-convergence of the power flow in flat start. That is why in “auto” mode, init defaults to “dc” if calculate_voltage_angles is True or “flat” otherwise

**max_iteration** (int, “auto”) - maximum number of iterations carried out in the power flow algorithm.

In “auto” mode, the default value depends on the power flow solver:
- 10 for “nr”
- 100 for “bfsw”
- 1000 for “gs”
- 30 for “fdbx”
- 30 for “fdxb”

**tolerance_kva** (float, 1e-5) - loadflow termination condition referring to P / Q mismatch of node power in kva

**trafo_model** (str, “t”) - transformer equivalent circuit model pandapower provides two equivalent circuit models for the transformer:
- “t” - transformer is modeled as equivalent with the T-model.
- “pi” - transformer is modeled as equivalent PI-model. This is not recommended, since it is less exact than the T-model. It is only recommended for validation with other software that uses the pi-model.

**trafo_loading** (str, “current”) - mode of calculation for transformer loading

Transformer loading can be calculated relative to the rated current or the rated power. In both cases the overall transformer loading is defined as the maximum loading on the two sides of the transformer.
- “current” - transformer loading is given as ratio of current flow and rated current of the transformer. This is the recommended setting, since thermal as well as magnetic effects in the transformer depend on the current.
- “power” - transformer loading is given as ratio of apparent power flow to the rated apparent power of the transformer.

**enforce_q_lims** (bool, False) - respect generator reactive power limits

If True, the reactive power limits in net.gen.max_q_kvar/min_q_kvar are respected in the loadflow. This is done by running a second loadflow if reactive power limits are violated at any generator, so that the runtime for the loadflow will increase if reactive power has to be curtailed.

Note: enforce_q_lims only works if algorithm=“nr”!

**numba** (bool, True) - Activation of numba JIT compiler in the newton solver

If set to True, the numba JIT compiler is used to generate matrices for the powerflow, which leads to significant speed improvements.

**recycle** (dict, none) - Reuse of internal powerflow variables for time series calculation
Contains a dict with the following parameters:

- **_is_elements**: If True in service elements are not filtered again and are taken from the last result in net["_is_elements"]
- **ppc**: If True the ppc is taken from net["_ppc"] and gets updated instead of reconstructed entirely
- **Ybus**: If True the admittance matrix (Ybus, Yf, Yt) is taken from ppc["internal"] and not reconstructed

**check_connectivity** (bool, True) - Perform an extra connectivity test after the conversion from pandapower to PYPOWER

If True, an extra connectivity test based on SciPy Compressed Sparse Graph Routines is performed. If check finds unsupplied buses, they are set out of service in the ppc

**r_switch** (float, 0.0) - resistance of bus-bus-switches. If impedance is zero, buses connected by a closed bus-bus switch are fused to model an ideal bus. Otherwise, they are modelled as branches with resistance r_switch.

**voltage_depend_loads** (bool, True) - consideration of voltage-dependent loads. If False, net.load.const_z_percent and net.load.const_i_percent are not considered, i.e. net.load.p_kw and net.load.q_kvar are considered as constant-power loads.

**delta_q** - Reactive power tolerance for option “enforce_q_lims” in kvar - helps convergence in some cases.

**kwargs** - options to use for PYPOWER.runpf

**Warning**: Neglecting voltage angles is only valid in radial networks! pandapower was developed for distribution networks, which is why omitting the voltage angles is the default. However be aware that voltage angle differences in networks with multiple galvatically coupled external grids lead to balancing power flows between slack nodes. That is why voltage angles always have to be considered in meshed network, such as in the sub-transmission level!

**Note**: If you are interested in the pypower casefile that pandapower is using for power flow, you can find it in net["_ppc"]). However all necessary informations are written into the pandapower format net, so the pandapower user should not usually have to deal with pypower.

### 4.1.2 DC Power Flow

**Warning**: To run an AC power flow with DC power flow initialization, use the AC power flow with init="dc".

pandapower uses PYPOWER to solve the DC power flow problem:

```python
import pandapower as pp
import pyppower as ppy

net = pp.runpp(net, init='dc')
```

**pandapower.runpp** runs PANDAPOWER DC Flow

**INPUT**: net - The pandapower format network

**OPTIONAL**: **trafo_model** (str, “t”) - transformer equivalent circuit model pandapower provides two equivalent circuit models for the transformer:
• “t” - transformer is modeled as equivalent with the T-model. This is consistent with PowerFactory and is also more accurate than the PI-model. We recommend using this transformer model.

• “pi” - transformer is modeled as equivalent PI-model. This is consistent with Sincal, but the method is questionable since the transformer is physically T-shaped. We therefore recommend the use of the T-model.

**trafo-loading** (str, “current”) - mode of calculation for transformer loading

Transformer loading can be calculated relative to the rated current or the rated power. In both cases the overall transformer loading is defined as the maximum loading on the two sides of the transformer.

• “current”- transformer loading is given as ratio of current flow and rated current of the transformer. This is the recommended setting, since thermal as well as magnetic effects in the transformer depend on the current.

• “power” - transformer loading is given as ratio of apparent power flow to the rated apparent power of the transformer.

**recycle** (dict, none) - Reuse of internal powerflow variables for time series calculation

Contains a dict with the following parameters: _is_elements: If True in service elements are not filtered again and are taken from the last result in net”_is_elements”] ppc: If True the ppc (PYPOWER case file) is taken from net”_ppc”] and gets updated instead of reconstructed entirely Ybus: If True the admittance matrix (Ybus, Yf, Yt) is taken from ppc”internal”] and not reconstructed

**check_connectivity** (bool, False) - Perform an extra connectivity test after the conversion from pandapower to PYPOWER

If true, an extra connectivity test based on SciPy Compressed Sparse Graph Routines is performed. If check finds unsupplied buses, they are put out of service in the PYPOWER matrix

**r_switch** (float, 0.0) - resistance of bus-bus-switches. If impedance is zero, buses connected by a closed bus-bus switch are fused to model an ideal bus. Otherwise, they are modelled as branches with resistance r_switch

**kwargs** - options to use for PYPOWER.runpf

---

**Note:** If you are interested in the pypower casefile that pandapower is using for power flow, you can find it in net”_ppc”]. However all necessary informations are written into the pandapower format net, so the pandapower user should not usually have to deal with pypower.

---

### 4.1.3 Optimal Power Flow

Pandapower provides an interface for AC and DC optimal power flow calculations. In the following, it is presented how the optimisation problem can be formulated with the pandapower data format.

**Note:** We highly recommend the tutorials for the usage of the optimal power flow.

---

### 4.1.4 Optimisation problem

The equation describes the basic formulation of the optimal power flow problem. The pandapower optimal power flow can be constrained by either, AC and DC loadflow equations. The branch constraints represent the maximum apparent power loading of transformers and the maximum line current loadings. The bus constraints can contain maximum and minimum voltage magnitude and angle. For the external grid, generators, loads, DC lines and static
generators, the maximum and minimum active resp. reactive power can be considered as operational constraints for the optimal power flow. The constraints are defined element wise in the respective element tables.

\[
\min \sum_{i \in \text{gen,sgen,load,extgrid}} P_i \ast f_i(P_i)
\]

subject to

Loadflow equations

branch constraints

bus constraints

**operational power constraints**

**Generator Flexibilities / Operational power constraints**

The active and reactive power generation of generators, loads, dc lines and static generators can be defined as a flexibility for the OPF.

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Defined in</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P_{\text{min},g} \leq P_g \leq P_{\text{max},g}, g \in \text{gen} )</td>
<td>net.gen.min_p_kw / net.gen.max_p_kw</td>
</tr>
<tr>
<td>(Q_{\text{min},g} \leq Q_g \leq Q_{\text{max},g}, g \in \text{gen} )</td>
<td>net.gen.min_q_kvar / net.gen.max_q_kvar</td>
</tr>
<tr>
<td>(P_{\text{min},sg} \leq P_{sg} \leq P_{\text{max},sg}, sg \in \text{sgen} )</td>
<td>net.sgen.min_p_kw / net.sgen.max_p_kw</td>
</tr>
<tr>
<td>(Q_{\text{min},sg} \leq Q_{sg} \leq Q_{\text{max},sg}, sg \in \text{sgen} )</td>
<td>net.sgen.min_q_kvar / net.sgen.max_q_kvar</td>
</tr>
<tr>
<td>(P_{\text{max},sg}, g \in \text{dcline} )</td>
<td>net.dcline.max_p_kw</td>
</tr>
<tr>
<td>(Q_{\text{min},sg} \leq Q_{sg} \leq Q_{\text{max},sg}, g \in \text{dcline} )</td>
<td>net.dcline.min_q_from_kvar / net.dcline.max_q_from_kvar</td>
</tr>
<tr>
<td>(P_{\text{min},eg} \leq P_{eg} \leq P_{\text{max},eg}, eg \in \text{extgrid} )</td>
<td>net.ext_grid.min_p_kw / net.ext_grid.max_p_kw</td>
</tr>
<tr>
<td>(Q_{\text{min},eg} \leq Q_{eg} \leq Q_{\text{max},eg}, eg \in \text{extgrid} )</td>
<td>net.ext_grid.min_q_kvar / net.ext_grid.max_q_kvar</td>
</tr>
<tr>
<td>(P_{\text{min},ld} \leq P_{ld} \leq P_{\text{max},ld}, ld \in \text{load} )</td>
<td>net.sgen.min_p_kw / net.sgen.max_p_kw</td>
</tr>
<tr>
<td>(Q_{\text{min},ld} \leq Q_{ld} \leq Q_{\text{max},ld}, ld \in \text{load} )</td>
<td>net.sgen.min_q_kvar / net.sgen.max_q_kvar</td>
</tr>
</tbody>
</table>

**Note:** Defining operational constraints is indispensable for the OPF, it will not start if constraints are not defined.

**Network Constraints**

The network constraints contain constraints for bus voltages and branch flows:

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Defined in</th>
</tr>
</thead>
<tbody>
<tr>
<td>(V_{\text{min},j} \leq V_{j} \leq V_{\text{min},j}, j \in \text{bus} )</td>
<td>net.bus.min_vm_pu / net.bus.max_vm_pu</td>
</tr>
<tr>
<td>(L_k \leq L_{\text{max},k}, k \in \text{trafo} )</td>
<td>net.trafo.max_loading_percent</td>
</tr>
<tr>
<td>(L_l \leq L_{\text{max},l}, l \in \text{line} )</td>
<td>net.line.max_loading_percent</td>
</tr>
<tr>
<td>(L_l \leq L_{\text{max},l}, l \in \text{trafo3w} )</td>
<td>net.trafo3w.max_loading_percent</td>
</tr>
</tbody>
</table>

The defaults are 100% loading for branch elements and +-0.1 p.u. for bus voltages.
4.1.5 Cost functions

The cost function is specified element wise and is organized in tables as well, which makes the parametrization user friendly. There are two options formulating a cost function for each element: A piecewise linear function with SnS data points.

\[
f_{\text{pwll}}(p) = f_0 + \frac{f_{n+1} - f_0}{p_{n+1} - p_0} (p - p_0), \quad (p_0, f_0), \quad p_0 < p < p_1 \\
... \\
(p_{n-1}, f_{n-1}), \quad p_{n-1} < p < p_n \\
\]

\[
f_{\text{pwll}}(q) = f_1 + \frac{f_2 - f_1}{q_2 - q_1} (q - q_1)
\]

Piecewise linear cost functions can be specified using `create_piecewise_linear_costs()`:

```python
pandapower.create_piecewise_linear_cost(net, element, element_type, data_points, type='p', index=None)
```

**Creates an entry for piecewise linear costs for an element. The currently supported elements are**
- Generator
- External Grid
- Static Generator
- Load
- Dcline

**INPUT:**
- `element` (int) - ID of the element in the respective element table
- `element_type` (string) - Type of element ["gen", "sgen", "ext_grid", "load", "dcline"] are possible
- `data_points` - (numpy array) Numpy array containing n data points (see example)

**OPTIONAL:**
- `type` (string) - Type of cost ["p", "q"] are allowed
- `index` (int, index) - Force a specified ID if it is available. If None, the index one higher than the highest already existing index is selected.

**OUTPUT:**
- `index` (int) - The unique ID of created cost entry

**EXAMPLE:**
`create_piecewise_linear_cost(net, 0, "load", np.array([[0, 0], [75, 50], [150, 100]]))`

**NOTE:** costs for reactive power can only be quadratic, linear or constant. No higher grades supported.

The other option is to formulate a n-polynomial cost function:

\[
f_{\text{pol}}(p) = c_np^n + ... + c_1p + c_0 \\
f_{\text{pol}}(q) = c_2q^2 + c_1q + c_0
\]

Polynomial cost functions can be specified using `create_polynomial_cost()`:

```python
pandapower.create_polynomial_cost(net, element, element_type, coefficients, type='p', index=None)
```

**Creates an entry for polynomial costs for an element. The currently supported elements are**
- Generator
- External Grid
- Static Generator
- Load
- Dcline
INPUT: `element` (int) - ID of the element in the respective element table

`element_type` (string) - Type of element ["gen", "s gen", "ext_grid", "load", "dcline"] are possible

`data_points` - (numpy array) Numpy array containing n cost coefficients (see example)

**type** - "p" or “q”

OPTIONAL: `type` - (string) - Type of cost ["p", “q"] are allowed

`index` (int, None) - Force a specified ID if it is available. If None, the index one higher than the highest already existing index is selected.

OUTPUT: `index` (int) - The unique ID of created cost entry

EXAMPLE: create_polynomial_cost(net, 0, “gen”, np.array([0, 1, 0]))

---

**Note:** Please note, that polynomial costs for reactive power can only be quadratic, linear or constant. Piecewise linear cost functions for reactive power are not working at the moment with 2 segments or more. Loads can only have 2 data points in their piecewise linear cost function for active power.

Active and reactive power costs are calculated separately. The costs of all types are summed up to determine the overall costs for a grid state.

### 4.1.6 Parametrisation of the calculation

The internal solver uses the interior point method. By default, the initial state is the center of the operational constraints. Another option would be to initialize the optimization with a valid loadflow solution. For optimization of a timeseries, this warm start possibility could imply a significant speedup. This is not yet provided in the actual version, but could be an useful extension in the future. Another parametrisation for the AC OPF is, if voltage angles should be considered, which is the same option than for the loadflow calculation with pandapower.runpp:

```python
pandapower.runpp(net, verbose=False, calculate_voltage_angles=False, check_connectivity=False, suppress_warnings=True, r_switch=0.0, delta=1e-10, **kwargs)
```

Runs the pandapower Optimal Power Flow. Flexibilities, constraints and cost parameters are defined in the `pandapower` element tables.

Flexibilities can be defined in net.sgen / net.gen /net.load net.sgen.controllable if a static generator is controllable. If False, the active and reactive power are assigned as in a normal power flow. If True, the following flexibilities apply:

- net.sgen.min_p_kw / net.sgen.max_p_kw
- net.sgen.min_q_kvar / net.sgen.max_q_kvar
- net.load.min_p_kw / net.load.max_p_kw
- net.load.min_q_kvar / net.load.max_q_kvar
- net.gen.min_p_kw / net.gen.max_p_kw
- net.gen.min_q_kvar / net.gen.max_q_kvar
- net.ext_grid.min_p_kw / net.ext_grid.max_p_kw
- net.ext_grid.min_q_kvar / net.ext_grid.max_q_kvar
- net.dcline.min_q_to_kvar / net.dcline.max_q_to_kvar / net.dcline.min_q_from_kvar / net.dcline.max_q_from_kvar

Controllable loads behave just like controllable static generators. It must be stated if they are controllable. Otherwise, they are not respected as flexibilities. DC lines are controllable per default.

Network constraints can be defined for buses, lines and transformers the elements in the following columns:
• net.bus.min_vm_pu / net.bus.max_vm_pu
• net.line.max_loading_percent
• net.trafo.max_loading_percent
• net.trafo3w.max_loading_percent

How these costs are combined into a cost function depends on the cost_function parameter.

**INPUT:** net - The pandapower format network

**OPTIONAL:**
- **verbose** (bool, False) - If True, some basic information is printed
- **suppress_warnings** (bool, True) - suppress warnings in pypower

If set to True, warnings are disabled during the loadflow. Because of the way data is processed in pypower, ComplexWarnings are raised during the loadflow. These warnings are suppressed by this option, however keep in mind all other pypower warnings are suppressed, too.

**References:**

### 4.1.7 DC Optimal Power Flow

The dc optimal power flow is a linearized optimization of the grid state. It offers two cost function options, that are fitting special use cases. To understand the usage, the OPF tutorial is recommended (see tutorial).

**pandapower.runcopp**
```python
(net, verbose=False, check_connectivity=True, suppress_warnings=True,
 r_switch=0.0, delta=1e-10, **kwargs)
```

Runs the pandapower Optimal Power Flow. Flexibilities, constraints and cost parameters are defined in the pandapower element tables.

Flexibilities for generators can be defined in net.sgen / net.gen. net.sgen.controllable / net.gen.controllable signals if a generator is controllable. If False, the active and reactive power are assigned as in a normal power flow. If yes, the following flexibilities apply:

• net.sgen.min_p_kw / net.sgen.max_p_kw
• net.gen.min_p_kw / net.gen.max_p_kw
• net.load.min_p_kw / net.load.max_p_kw

Network constraints can be defined for buses, lines and transformers the elements in the following columns: - net.line.max_loading_percent - net.trafo.max_loading_percent - net.trafo3w.max_loading_percent

**INPUT:** net - The pandapower format network

**OPTIONAL:**
- **verbose** (bool, False) - If True, some basic information is printed
- **suppress_warnings** (bool, True) - suppress warnings in pypower

If set to True, warnings are disabled during the loadflow. Because of the way data is processed in pypower, ComplexWarnings are raised during the loadflow. These warnings are suppressed by this option, however keep in mind all other pypower warnings are suppressed, too.
Flexibilities, costs and constraints (except voltage constraints) are handled as in the Optimal Power Flow. Voltage constraints are not considered in the DC OPF, since voltage magnitudes are not part of the linearized power flow equations.

**Note:** If you are interested in the pypower casefile that pandapower is using for power flow, you can find it in net["_ppc_opf"]. However all necessary informations are written into the pandpower format net, so the pandapower user should not usually have to deal with pypower.

### 4.2 Known Problems and Caveats

#### 4.2.1 Voltage Controlling Elements

It is generally possible to have several generators and external grids in one network. Buses also might have several bus-elements (ext_grid, load, sgen etc.) connected to them:

![Diagram of possible connections](image)

It is however not possible to connect multiple ext_grids and gens at one bus, since this would convergence problems in PYPOWER:

![Diagram of impossible connections](image)

The pandapower API will prevent you from adding a second voltage controlling element to a bus, so you should not be able to build the networks pictured above through the pandapower API.

It is also not allowed to add two voltage controlled elements to buses which are connected through a closed bus-bus switch, since those buses are fused internally and therefore the same bus in PYPOWER (see switch model):

![Diagram of impossible connections](image)

#### 4.2.2 Zero Impedance Branches

Branches with zero impedance will lead to a non-converging power flow:
This is due to the fact that the power flow is based on admittances, which would be infinite for an impedance of zero. The same problem might occur with impedances very close to zero.

Zero impedance branches occur for:

- lines with length_km = 0
- lines with r_ohm_per_km = 0 and x_ohm_per_km = 0
- transformers with vsc_percent=0

If you want to directly connect to buses without voltage drop, use a bus-bus switch.

### 4.3 Diagnostic Function

A power flow calculation on a pandapower network can fail to converge for a vast variety of reasons, which often makes debugging difficult, annoying and time consuming. To help with that, the diagnostic function automatically checks pandapower networks for the most common issues leading to errors. It provides logging output and diagnoses with a controllable level of detail.

```
 Tool for diagnosis of pandapower networks. Identifies possible reasons for non converging loadflows.

INPUT: net (pandapowerNet) : pandapower network

OPTIONAL:

- report_style (string, 'detailed') : style of the report, that gets output in the console
  - 'detailed': full report with high level of additional descriptions
  - 'compact': more compact report, containing essential information only
  - 'None': no report

- warnings_only (boolean, False): Filters logging output for warnings
  - True: logging output for errors only
  - False: logging output for all checks, regardless if errors were found or not

- return_result_dict (boolean, True): returns a dictionary containing all check results
  - True: returns dict with all check results
  - False: no result dict
```
• **overload_scaling_factor** (float, 0.001): downscaling factor for loads and generation for overload check

• **lines_min_length_km** (float, 0): minimum length_km allowed for lines

• **lines_min_z_ohm** (float, 0): minimum z_ohm allowed for lines

• **nom_voltage_tolerance** (float, 0.3): highest allowed relative deviation between nominal voltages and bus voltages

**OUTPUT:**

• **diag_results** (dict): dict that contains the indeces of all elements where errors were found

  Format: {'check_name': check_results}

**EXAMPLE:**

```python
<<< pandapower.diagnostic(net, report_style='compact', warnings_only=True)
```

Usage ist very simple: Just call the function and pass the net you want to diagnose as an argument. Optionally you can specify if you want detailed logging output or summaries only and if the diagnostic should log all checks performed vs. errors only.

### 4.3.1 Check functions

The diagnostic function includes the following checks:

• invalid values (e.g. negative element indeces)

• check, if at least one external grid exists

• check, if there are buses with more than one gen and/or ext_grid

• overload: tries to run a power flow calculation with loads scaled down to 10%

• switch_configuration: tries to run a power flow calculation with all switches closed

• inconsistent voltages: checks, if there are lines or switches that connect different voltage levels

• lines with impedance zero

• closed switches between in_service and out_of_service buses

• components whose nominal voltages differ from the nominal voltages of the buses they’re connected to

• elements, that are disconnected from the network

• usage of wrong reference system for power values of loads and gens

### 4.3.2 Logging Output

Here are a few examples of what logging output looks like:

**detailed_report = True/False**

Both reports show the same result, but on the left hand picture with detailed information, on the right hand picture summary only.
4.3.3 Result Dictionary

Aditionally all check results are returned in a dict to allow simple access to the indeces of all element where errors were found.
<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Size</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>busses_mult_gen_set_grid</td>
<td>NoneType</td>
<td>1</td>
<td>None</td>
</tr>
<tr>
<td>deviating_nominal_voltages</td>
<td>dist</td>
<td>1</td>
<td>{'Traffic': {'My_DBS': [], 'My_1x_switched': [2], '1x_DBS': []}}</td>
</tr>
<tr>
<td>ext_grid</td>
<td>NoneType</td>
<td>1</td>
<td>None</td>
</tr>
<tr>
<td>inconsistent_voltages</td>
<td>dist</td>
<td>2</td>
<td>['line': [5, 4, 7], 'outlines': [280, 286, 308]]</td>
</tr>
<tr>
<td>invalid_values</td>
<td>dist</td>
<td>8</td>
<td>['bus': [16, 'un_kw', '20.00'], 'ext_grid': [], 'gen': [], 'line': [20, 'length_km', '20.00'], [20, 'length_km', '10.00'], [20, 'length_km', '10.00'], [20, 'length_km', '10.00'], 'load': [], 'gen': [], 'switch': [20, 'closed', 1.5], 'trans': []]</td>
</tr>
<tr>
<td>isolated_sections</td>
<td>dist</td>
<td>2</td>
<td>['isolated_sections': [40], [58], [52], [52], [52], [52], [56], [57], [51]], 'lines_both_switches_open': [29]}</td>
</tr>
<tr>
<td>lines_with_impedance_tens</td>
<td>list</td>
<td>1</td>
<td>[9]</td>
</tr>
<tr>
<td>overload</td>
<td>NoneType</td>
<td>1</td>
<td>None</td>
</tr>
<tr>
<td>problematic_switches</td>
<td>NoneType</td>
<td>1</td>
<td>None</td>
</tr>
<tr>
<td>wrong_reference_system</td>
<td>NoneType</td>
<td>1</td>
<td>None</td>
</tr>
<tr>
<td>wrong_switch_configuration</td>
<td>NoneType</td>
<td>1</td>
<td>None</td>
</tr>
</tbody>
</table>
5 Short-Circuit

The short-circuit module is used to calculate short-circuits according to DIN/IEC EN 60909.

5.1 Running a Short-Circuit Calculation

The short circuit calculation is carried out with the calc_sc function:

```python
calc_sc(net, fault='3ph', case='max', lv_tol_percent=10, topology='auto', ip=False, ith=False, tk_s=1.0, kappa_method='C', r_fault_ohm=0.0, x_fault_ohm=0.0, branch_results=True)
```

Calculates minimal or maximal symmetrical short-circuit currents. The calculation is based on the method of the equivalent voltage source according to DIN/IEC EN 60909. The initial short-circuit alternating current \(ikss\) is the basis of the short-circuit calculation and is therefore always calculated. Other short-circuit currents can be calculated from \(ikss\) with the conversion factors defined in DIN/IEC EN 60909.

The output is stored in the net.res_bus_sc table as a short_circuit current for each bus.

**INPUT:**
- net (pandapowerNet) pandapower Network
- *fault* (str, 3ph) type of fault
  - “3ph” for three-phase
  - “2ph” for two-phase short-circuits
- case (str, “max”)  
  - “max” for maximal current calculation
  - “min” for minimal current calculation
- lv_tol_percent (int, 10) voltage tolerance in low voltage grids
  - 6 for 6% voltage tolerance
  - 10 for 10% voltage tolerance
- ip (bool, False) if True, calculate aperiodic short-circuit current
- Ith (bool, False) if True, calculate equivalent thermical short-circuit current Ith
- topology (str, “auto”) define option for meshing (only relevant for ip and ith)
  - “meshed” - it is assumed all buses are supplied over multiple paths
  - “radial” - it is assumed all buses are supplied over exactly one path
  - “auto” - topology check for each bus is performed to see if it is supplied over multiple paths
- tk_s (float, 1) failure clearing time in seconds (only relevant for ith)
- r_fault_ohm (float, 0) fault resistance in Ohm
- x_fault_ohm (float, 0) fault reactance in Ohm
- consider_sgens (bool, True) defines if short-circuit contribution of static generators should be considered or not

**OUTPUT:**

**EXAMPLE:**
calc_sc(net)
print(net.res_bus_sc)
import pandapower.shortcircuit as sc
import pandapower.networks as nw

net = nw.mv_oberrhein()
net.ext_grid["s_sc_min_mva"] = 100
net.ext_grid["rx_min"] = 0.1

net.line["endtemp_degree"] = 20
sc.calc_sc(net, case="min")
print(net.res_bus_sc)

5.2 Short-Circuit Currents

The short-circuit currents are calculated with the equivalent voltage source at the fault location. For an explanation of the theory behind short-circuit calculations according to IEC 60909 please refer to the norm or secondary literature:

See also:

IEC 60909-0:2016 Short-circuit currents in three-phase a.c. systems

According to the IEC 60909 on openelectrical

pandapower currently implements symmetrical and two-phase faults. One phase faults and two-phase faults with earthing are not yet available.

5.2.1 Initial Short-Circuit Current

The general ohmic network equation is given as:

The SC is calculated in two steps:

• calculate the SC contribution \( I_{kI}^{II} \) of all voltage source elements

• calculate the SC contribution \( I_{kII}^{II} \) of all current source elements

These two currents are then combined into the total initial SC current \( I_{k}^{II} = I_{kI}^{II} + I_{kII}^{II} \).

5.2.2 Equivalent Voltage Source

For the short-circuit calculation with the equivalent voltage source, all voltage sources are replaced by one equivalent voltage source \( V_Q \) at the fault location. The voltage magnitude at the fault bus is assumed to be:

\[
V_Q = \begin{cases} 
\frac{c \cdot V_N}{\sqrt{3}} & \text{for three phase short circuit currents} \\
\frac{c \cdot V_N}{2} & \text{for two phase short circuit currents}
\end{cases}
\]

where \( V_N \) is the nominal voltage at the fault bus and \( c \) is the voltage correction factor, which accounts for operational deviations from the nominal voltage in the network.

The voltage correction factors \( c_{min} \) for minimum and \( c_{max} \) for maximum short-circuit currents are defined for each bus depending on the voltage level. In the low voltage level, there is an additional distinction between networks with a tolerance of 6% vs. a tolerance of 10% for \( c_{max} \):

<table>
<thead>
<tr>
<th>Voltage Level</th>
<th>( c_{min} )</th>
<th>( c_{max} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 1 kV</td>
<td>0.95</td>
<td>1.05</td>
</tr>
<tr>
<td>Tolerance 6%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tolerance 10%</td>
<td>1.00</td>
<td>1.10</td>
</tr>
<tr>
<td>&gt; 1 kV</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5 SHORT-CIRCUIT

5.2.3 Voltage Source Contribution

To calculate the contribution of all voltage source elements, the following assumptions are made:

1. Operational currents at all buses are neglected
2. All current source elements are neglected
3. The voltage at the fault bus is equal to \( V_Q \)

For the calculation of a short-circuit at bus \( j \), this yields the following network equations:

\[
\begin{bmatrix}
Y_{11} & \cdots & Y_{n1} \\
\vdots & \ddots & \vdots \\
Y_{1n} & \cdots & Y_{nn}
\end{bmatrix}
\begin{bmatrix}
V_1 \\
\vdots \\
V_Q \end{bmatrix}
= 
\begin{bmatrix}
0 \\
\vdots \\
I''_{kj} \end{bmatrix}
\]

where \( I''_{kj} \) is the voltage source contribution of the short-circuit current at bus \( j \). The voltages at all non-fault buses and the current at the fault bus are unknown. To solve for \( I''_{kj} \), we multiply with the inverted nodal point admittance matrix (impedance matrix):

\[
\begin{bmatrix}
V_1 \\
\vdots \\
V_Q \end{bmatrix}
= 
\begin{bmatrix}
Z_{11} & \cdots & Z_{n1} \\
\vdots & \ddots & \vdots \\
Z_{1n} & \cdots & Z_{nn}
\end{bmatrix}
\begin{bmatrix}
0 \\
\vdots \\
I''_{kj} \end{bmatrix}
\]

The short-circuit current for bus \( m \) is now given as:

\[
I''_{km} = \frac{V_Q}{Z_{jj}}
\]

To calculate the vector of the short-circuit currents at all buses, the equation can be expanded as follows:

\[
\begin{bmatrix}
V_Q1 \\
\vdots \\
V_Qn
\end{bmatrix}
= 
\begin{bmatrix}
Z_{11} & \cdots & Z_{n1} \\
\vdots & \ddots & \vdots \\
Z_{1n} & \cdots & Z_{nn}
\end{bmatrix}
\begin{bmatrix}
I''_{k11} \\
\vdots \\
I''_{kn}
\end{bmatrix}
\]

which yields:

\[
\begin{bmatrix}
I''_{k11} \\
\vdots \\
I''_{kn}
\end{bmatrix}
= 
\begin{bmatrix}
\frac{V_Q1}{Z_{11} + Z_{fault}} \\
\vdots \\
\frac{V_Qn}{Z_{nn} + Z_{fault}}
\end{bmatrix}
\]

In that way, all short-circuit currents can be calculated at once with one inversion of the nodal point admittance matrix.

In case a fault impedance is specified, it is added to the diagonal of the impedance matrix. The short-circuit currents at all buses are then calculated as:

\[
\begin{bmatrix}
I''_{k11} \\
\vdots \\
I''_{kn}
\end{bmatrix}
= 
\begin{bmatrix}
\frac{V_Q1}{Z_{11} + Z_{fault}} \\
\vdots \\
\frac{V_Qn}{Z_{nn} + Z_{fault}}
\end{bmatrix}
\]
5.2.4 Current Source Contribution

To calculate the current source component of the SC current, all voltage sources are short circuited and only current sources are considered. The bus currents are then given as:

\[
\begin{bmatrix}
I_1 \\
\vdots \\
I_m \\
\vdots \\
I_n
\end{bmatrix} =
\begin{bmatrix}
0 \\
\vdots \\
I_{kIj} \\
\vdots \\
0
\end{bmatrix} - 
\begin{bmatrix}
I'_{kC1} \\
\vdots \\
I'_{kCj} \\
\vdots \\
I'_{kCn}
\end{bmatrix} = 
\begin{bmatrix}
-I''_{kC1} \\
\vdots \\
I''_{kIj} - I''_{kCj} \\
\vdots \\
-I''_{kCn}
\end{bmatrix}
\]

where \( I''_{kC} \) are the SC currents that are fed in by converter element at each bus and \( I''_{kIj} \) is the contribution of converter elements at the fault bus \( j \). With the voltage at the fault bus known to be zero, the network equations are given as:

\[
\begin{bmatrix}
V_1 \\
\vdots \\
0 \\
\vdots \\
V_n
\end{bmatrix} = 
\begin{bmatrix}
Z_{11} & \cdots & \cdots & \cdots & Z_{n1} \\
\vdots & \ddots & \vdots & \vdots & \vdots \\
0 & \ddots & Z_{jj} & \vdots & \vdots \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
Z_{1n} & \cdots & \cdots & Z_{nn}
\end{bmatrix} 
\begin{bmatrix}
-I''_{kC1} \\
\vdots \\
I''_{kIj} - I''_{kCj} \\
\vdots \\
-I''_{kCn}
\end{bmatrix}
\]

From which row \( j \) of the equation yields:

\[
0 = Z_{jj} \cdot I''_{kIj} - \sum_{m=1}^{n} Z_{jm} \cdot I''_{kCj}
\]

which can be converted into:

\[
I''_{kIj} = \frac{1}{Z_{jj}} \sum_{m=1}^{n} Z_{jm} \cdot I''_{kCm}
\]

To calculate all SC currents for faults at each bus simultaneously, this can be generalized into the following matrix equation:

\[
\begin{bmatrix}
I''_{kI1} \\
\vdots \\
I''_{kI1n}
\end{bmatrix} = 
\begin{bmatrix}
Z_{11} & \cdots & \cdots & \cdots & Z_{n1} \\
\vdots & \ddots & \vdots & \vdots & \vdots \\
0 & \ddots & Z_{jj} & \vdots & \vdots \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
Z_{1n} & \cdots & \cdots & Z_{nn}
\end{bmatrix} 
\begin{bmatrix}
\frac{I''_{kI1}}{Z_{11}} \\
\vdots \\
\frac{I''_{kI1n}}{Z_{nn}}
\end{bmatrix}
\]

5.2.5 Peak Short-Circuit Current

5.2.6 Current Calculation

The peak short-circuit current is calculated as:

\[
\begin{bmatrix}
i_{p,1} \\
\vdots \\
i_{p,n}
\end{bmatrix} = \sqrt{2} \left( \begin{bmatrix}
\kappa_1 \\
\vdots \\
\kappa_n
\end{bmatrix} \begin{bmatrix}
I''_{kI1} \\
\vdots \\
I''_{kI1n}
\end{bmatrix} + \begin{bmatrix}
I''_{kC1} \\
\vdots \\
I''_{kCn}
\end{bmatrix} \right)
\]

where \( \kappa \) is the peak factor.
5.2.7 Peak Factor $\kappa$

In radial networks, $\kappa$ is given as:

$$\kappa = 1.02 + 0.98e^{-3R/X}$$

where $R/X$ is the R/X ratio of the equivalent short-circuit impedance $Z_k$ at the fault location.

In meshed networks, the standard defines three possibilities for the calculation of $\kappa$:

- Method A: Uniform Ratio R/X
- Method B: R/X ratio at short-circuit location
- Method C: Equivalent frequency

The user can chose between Methods B and C when running a short circuit calculation. Method C yields the most accurate results according to the standard and is therefore the default option. Method A is only suited for estimated manual calculations with low accuracy and therefore not implemented in pandapower.

**Method C: Equivalent frequency**

For method C, the same formula for $\kappa$ is used as for radial grids. The R/X value that is inserter is however not the

**Method B: R/X Ratio at short-circuit location**

For method B, $\kappa$ is given as:

$$\kappa = [1.02 + 0.98e^{-3R/X}] \cdot 1.15$$

while being limited with $\kappa_{\text{min}} < \kappa < \kappa_{\text{max}}$ depending on the voltage level:

<table>
<thead>
<tr>
<th>Voltage Level</th>
<th>$\kappa_{\text{min}}$</th>
<th>$\kappa_{\text{max}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 1 kV</td>
<td>1.0</td>
<td>1.8</td>
</tr>
<tr>
<td>&gt; 1 kV</td>
<td></td>
<td>2.0</td>
</tr>
</tbody>
</table>

5.2.8 Thermal Short-Circuit Current

5.2.9 Current Calculation

The equivalent thermal current is calculated as:

$$\begin{bmatrix} I_{\text{th},1} \\ \vdots \\ I_{\text{th},n} \end{bmatrix} = \begin{bmatrix} \sqrt{m_1 + n_1} \\ \vdots \\ \sqrt{m_n + n_n} \end{bmatrix} \begin{bmatrix} I''_{k,1} \\ \vdots \\ I''_{k,n} \end{bmatrix}$$

where $m$ and $n$ represent the dc and ac part of the thermal load.

5.2.10 Correction Factors $m$ and $n$

For short-circuit currents far from synchronous generators, the factors are given as:

$$n = 1 \cdot m = \frac{1}{2 \cdot f \cdot T_k \cdot \ln(\kappa - 1)} \left[e^{3fT_k \cdot \ln(\kappa - 1)} - 1 \right]$$

where $\kappa$ is the peak factor defined here and $T_k$ is the duration of the short-circuit current that can be defined as a parameter when running the short-circuit calculation.

5.3 Network Elements

Correction factors for generator and branch elements are implemented as defined in the IEC 60909 standard. The results for all elements are tested against commercial software to ensure that correction factors are correctly
applied.

5.3.1 Voltage Source Elements

Voltage source elements are represented by their internal voltage source with an internal resistance $Z_k$:

![Diagram of voltage source element]

since the voltage source is moved to the fault location for with methodology of the equivalent voltage source, the bus elements can be reduced to a single shunt impedance:

$$Z_i = R_i + jX_i$$

The contribution of loads and shunts are negligible according to the standard and therefore neglected in the short-circuit calculation.

5.3.2 External Grid

When calculating maximum short-circuit currents, the impedance of an external grid connection is given as:

$$Z_{k, eg} = \frac{c_{max}}{s_{sc_{max} mva}}$$

$$x_{k, eg} = \frac{z_{sg}}{\sqrt{1 + r_{x_{max}}^2}}$$

$$r_{k, eg} = r_{x_{max}} \cdot x_{sg}$$

where $r_{x_{max}}$ and $s_{sc_{max} mva}$ are parameters in the ext_grid table and $c_{max}$ is the \textit{voltage correction factor} of the external grid bus.

In case of minimal short-circuit currents, the impedance is calculated accordingly:

$$Z_{k, eg} = \frac{c_{min}}{s_{sc_{min} mva}}$$

$$x_{k, eg} = \frac{z_{sg}}{\sqrt{1 + r_{x_{min}}^2}}$$

$$r_{k, eg} = r_{x_{min}} \cdot x_{sg}$$
5.3.3 Asynchronous Motor

Asynchronous motors can be considered by setting the type variable of an sgen element to “motor”. The internal impedance is then calculated as:

\[
Z_{k,m} = \frac{1}{k} \cdot \frac{v_{n_kv}^2 \cdot 1000}{s_{n_kva}}
\]

\[
X_{k,m} = \frac{Z_{sg}}{\sqrt{1 + r_x^2}}
\]

\[
R_{k,m} = r_x \cdot X_{sg}
\]

where \( s_{n_kva} \) is the rated power of the motor, \( k \) is the ratio of nominal to short circuit current and \( r_x \) is the R/X ratio of the motor. \( v_{n_kv} \) is the rated voltage of the bus the motor is connected to.

5.3.4 Synchronous Generator

Synchronous generators are considered with the short-circuit impedance of:

\[
Z_{k,gen} = K_G \cdot (R_d'' + jX_d'')
\]

The short-circuit impedance is calculated as:

\[
z_k = x_{dss}
\]

The generator correction factor \( K_G \) is given as:

\[
K_G = \frac{V_{N,gen}}{V_{N,bus}} \cdot \frac{c_{max}}{1 + x_{dss} \cdot \sin(\varphi)}
\]

where \( V_{N,bus} \) is the rated voltage of the bus the generator is connected to and \( V_{N,gen} \) is the rated voltage of the generator which is defined by the parameter \( s_{n_kva} \) in the gen table. The rated phasor angle \( \varphi \) is given as:

\[
\varphi = \arccos(c_{os_phi})
\]

where \( c_{os_phi} \) is defined in the gen table.

5.3.5 Current Source Elements

Full converter elements, such as PV plants or wind parks, are modeled as current sources:

\[
I_k = -j \cdot \frac{k \cdot s_{n_kva}}{\sqrt{3} \cdot v_{n_kv}}
\]

where \( s_{n_kva} \) is the rated power of the generator and \( k \) is the ratio of nominal to short circuit current. \( v_{n_kv} \) is the rated voltage of the bus the generator is connected to.
5.3.6 Branch Elements

Branches are represented by a single short circuit impedance:

\[ Z_{\text{b}} \]

Shunt admittances are neglected for all branch elements.

5.3.7 Line

\[ R_k = r_{\text{ohm}_\text{per}_\text{km}} \cdot \frac{\text{length}_{\text{km}}}{\text{parallel}} \cdot K_L \]

\[ X_k = x_{\text{ohm}_\text{per}_\text{km}} \cdot \frac{\text{length}_{\text{km}}}{\text{parallel}} \]

where the correction factor for the short-circuit resistance \( K_L \) is defined as:

\[ K_L = \begin{cases} 1 & \text{for maximum short-circuit calculations} \\ 1 + 0.04K^{-1}(\text{endtemp}_\text{degree} - 20^\circ C) & \text{for minimum short-circuit calculations} \end{cases} \]

The end temperature in degree after a fault has to be defined with the parameter endtemp_degree in the line table.

5.3.8 Two-Winding Transformer

The short-circuit impedance is calculated as:

\[ z_k = \frac{\text{vsc}_{\text{percent}}}{100} \cdot \frac{1000}{\text{sn}_{\text{kva}}} \cdot K_T \]

\[ r_k = \frac{\text{vscr}_{\text{percent}}}{100} \cdot \frac{1000}{\text{sn}_{\text{kva}}} \cdot K_T \]

\[ x_k = \sqrt{z_k^2 - r_k^2} \]

where the correction factor \( K_T \) is defined in the standard as:

\[ K_T = 0.95 \frac{c_{\text{max}}}{1 + 0.6x_T} \]

where \( c_{\text{max}} \) is the voltage correction factor on the low voltage side of the transformer and \( x_T \) is the transformer impedance relative to the rated values of the transformer.

The ratio of the transformer is considered to be the nominal ratio, the tap changer positions are not considered according to the standard.

5.3.9 Three-Winding Transformer

Three Winding Transformers are modelled by three two-winding transformers:
The conversion from one two to three two winding transformer parameter is described here.

For the short-circuit calculation, the loss parameters are neglected and the transformer correction factor is applied for the equivalent two-winding transformers as follows:

\[ v'_{k,t1} = \frac{1}{2} (v'_{k,h} \cdot K_{T,h} + v'_{k,l} \cdot K_{T,l} - v'_{k,m} \cdot K_{T,m}) \]
\[ v'_{k,t2} = \frac{1}{2} (v'_{k,m} \cdot K_{T,m} + v'_{k,l} \cdot K_{T,l} - v'_{k,h} \cdot K_{T,h}) \]
\[ v'_{k,t3} = \frac{1}{2} (v'_{k,m} \cdot K_{T,m} + v'_{k,l} \cdot K_{T,l} - v'_{k,h} \cdot K_{T,h}) \]

Note that the correction factor has to be applied to the transformers before the wye-delta and not on the resulting two-winding transformers.

5.3.10 Impedance

The impedance element is a generic element that is not described in the standard. It is considered in the short-circuit calculation just as in the power flow as described here.
6 State Estimation

The module provides a state estimation for pandapower networks.

6.1 Theoretical Background

State Estimation is a process to estimate the electrical state of a network by eliminating inaccuracies and errors from measurement data. Various measurements are placed around the network and transferred to the operational control center via SCADA. Unfortunately measurements are not perfect: There are tolerances for each measurement device, which lead to an inherent inaccuracy in the measurement value. Analog transmission of data can change the measurement values through noise. Faulty devices can return completely wrong measurement values. To account for the measurement errors, the state estimation processes all available measurements and uses a regression method to identify the likely real state of the electrical network. The output of the state estimator is therefore a set of voltage absolutes and voltage angles for all buses in the grid. The input is the network in pandapower format and a number of measurements.

6.1.1 Amount of Measurements

There is a minimum amount of required measurements necessary for the regression to be mathematically possible. Assuming the network contains \( n \) buses, the network is then described by \( 2n \) variables, namely \( n \) voltage absolute values and \( n \) voltage angles. A slack bus serves as the reference, its voltage angle is set to zero or the value provided in the corresponding \texttt{net.ext_grid.va_degree} entry (see \texttt{init} parameter) and is not altered in the estimation process. The voltage angles of the other network buses are relative to the voltage angles of the connected slack bus. The state estimation therefore has to find \( 2n - k \) variables, where \( k \) is the number of defined slack buses. The minimum amount of measurements \( m_{min} \) needed for the method to work is therefore:

\[
m_{min} = 2n - k
\]

To perform well however, the number of redundant measurements should be higher. A value of \( m \approx 4n \) is often considered reasonable for practical purposes.

6.1.2 Standard Deviation

Since each measurement device may have a different tolerance and a different path length it has to travel to the control center, the accuracy of each measurement can be different. Therefore each measurement is assigned an accuracy value in the form of a standard deviation. Typical measurement errors are 1 % for voltage measurements and 1-3 % for power measurements.

For a more in-depth explanation of the internals of the state estimation method, please see the following sources:

See also:


6.2 Defining Measurements

Measurements are defined via the pandapower “create_measurement” function. There are different physical properties, which can be measured at different elements. The following lists and table clarify the possible combinations. Bus power injection measurements are given in the producer system. Generated power is positive, consumed power is negative.

Types of Measurements

- “v” for voltage measurements (in per-unit)
- “p” for active power measurements (in kW)
• “q” for reactive power measurements (in kVar)
• “i” for electrical current measurements at a line (in A)

Element Types
• “bus” for bus measurements
• “line” for line measurements
• “transformer” for transformer measurements

Available Measurements per Element

<table>
<thead>
<tr>
<th>Element Type</th>
<th>Available Measurement Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>bus</td>
<td>v, p, q</td>
</tr>
<tr>
<td>line</td>
<td>i, p, q</td>
</tr>
<tr>
<td>transformer</td>
<td>i, p, q</td>
</tr>
</tbody>
</table>

The “create_measurement” function is defined as follows:

```
pandapower.create.create_measurement(net, type, element_type, value, std_dev, bus, element=None, check_existing=True, index=None, name=None)
```

Creates a measurement, which is used by the estimation module. Possible types of measurements are: v, p, q, i

INPUT: type (string) - Type of measurement. “v”, “p”, “q”, “i” are possible.

  element_type (string) - Clarifies which element is measured. “bus”, “line”, “transformer” are possible.

  value (float) - Measurement value. Units are “kW” for P, “kVar” for Q, “p.u.” for V, “A” for I.
  Generation is a positive bus power injection, consumption negative.

  std_dev (float) - Standard deviation in the same unit as the measurement.

  bus (int) - Index of bus. Determines the position of the measurement for line/transformer measurements (bus == from_bus: measurement at from_bus; same for to_bus)

  element (int, None) - Index of measured element, if element_type is “line” or “transformer”.

OPTIONAL: check_existing (bool) - Check for and replace existing measurements for this bus and type.
  Set it to false for performance improvements which can cause unsafe behaviour.

  name (str, None) - name of measurement.

OUTPUT: (int) Index of measurement

EXAMPLE: 500 kW load measurement with 10 kW standard deviation on bus 0: create_measurement(net, "p", "bus", -500., 10., 0)

6.3 Running the State Estimation

The state estimation can be used with the wrapper function “estimate”, which prevents the need to deal with the state_estimation class object and functions. It can be imported from “estimation.state_estimation”.

```
pandapower.estimation.estimate(net, init='flat', tolerance=1e-06, maximum_iterations=10, calculate_voltage_angles=True, ref_power=1000000.0)
```

Wrapper function for WLS state estimation.

INPUT: net - The net within this line should be created.

  init - (string) Initial voltage for the estimation. ‘flat’ sets 1.0 p.u. / 0° for all buses, ‘results’ uses the values from res_bus_est if available and ‘slack’ considers the slack bus voltage (and optionally, angle) as the initial values. Default is ‘flat’.
OPTIONAL: **tolerance** - (float) - When the maximum state change between iterations is less than tolerance, the process stops. Default is 1e-6.

**maximum_iterations** - (integer) - Maximum number of iterations. Default is 10.

**calculate_voltage_angles** - (boolean) - Take into account absolute voltage angles and phase shifts in transformers, if init is ‘slack’. Default is True.

**OUTPUT: successful** (boolean) - Was the state estimation successful?

### 6.4 Handling of bad data

**Note:** The bad data removal is not very robust at this time. Please treat the results with caution!

The state estimation class allows additionally the removal of bad data, especially single or non-interacting false measurements. For detecting bad data the Chi-squared distribution is used to identify the presence of them. Afterwards follows the largest normalized residual test that identifies the actual measurements which will be removed at the end. Both methods are combined in the `perform_rn_max_test` function that is part of the state estimation class. To access it, the following wrapper function `remove_bad_data` has been created.

```python
pandapower.estimation.remove_bad_data(net, init='flat', tolerance=1e-06, maximum_iterations=10, calculate_voltage_angles=True, rn_max_threshold=3.0, ref_power=1000000.0)
```

Wrapper function for bad data removal.

**INPUT: net** - The net within this line should be created.

**init** - (string) Initial voltage for the estimation. ‘flat’ sets 1.0 p.u. / 0° for all buses, ‘results’ uses the values from res_bus_est if available and ‘slack’ considers the slack bus voltage (and optionally, angle) as the initial values. Default is ‘flat’.

**OPTIONAL: tolerance** - (float) - When the maximum state change between iterations is less than tolerance, the process stops. Default is 1e-6.

**maximum_iterations** - (integer) - Maximum number of iterations. Default is 10.

**calculate_voltage_angles** - (boolean) - Take into account absolute voltage angles and phase shifts in transformers, if init is ‘slack’. Default is True.

**rn_max_threshold** (float) - Identification threshold to determine if the largest normalized residual reflects a bad measurement (default value of 3.0)

**chi2_prob_false** (float) - probability of error / false alarms (default value: 0.05)

**OUTPUT: successful** (boolean) - Was the state estimation successful?

Nevertheless the Chi-squared test is available as well to allow a identification of topology errors or, as explained, false measurements. It is named as `chi2_analysis`. The detection’s result of present bad data of the Chi-squared test is stored internally as `bad_data_present` (boolean, class member variable) and returned by the function call.

```python
pandapower.estimation.chi2_analysis(net, init='flat', tolerance=1e-06, maximum_iterations=10, calculate_voltage_angles=True, chi2_prob_false=0.05, ref_power=1000000.0)
```

Wrapper function for the chi-squared test.

**INPUT: net** - The net within this line should be created.

**init** - (string) Initial voltage for the estimation. ‘flat’ sets 1.0 p.u. / 0° for all buses, ‘results’ uses the values from res_bus_est if available and ‘slack’ considers the slack bus voltage (and optionally, angle) as the initial values. Default is ‘flat’.
**OPTIONAL:**  
- **tolerance**  
  - (float) - When the maximum state change between iterations is less than tolerance, the process stops. Default is $1e^{-6}$.

- **maximum_iterations**  
  - (integer) - Maximum number of iterations. Default is 10.

- **calculate_voltage_angles**  
  - (boolean) - Take into account absolute voltage angles and phase shifts in transformers, if init is ‘slack’. Default is True.

**OUTPUT:**  
- **bad_data_detected**  
  - (boolean) - Returns true if bad data has been detected

Background information about this topic can be sourced from the following literature:


### 6.5 Example

As an example, we will define measurements for a simple pandapower network `net` with 4 buses. Bus 4 is out-of-service. The external grid is connected at bus 1.

There are multiple measurements available, which have to be defined for the state estimator. There are two voltage measurements at buses 1 and 2. There are two power measurements (active and reactive power) at bus 2. There are also line power measurements at bus 1. The measurements are both for active and reactive power and are located on the line from bus 1 to bus 2 and from bus 1 to bus 3. This yields the following code:

```python
pp.create_measurement(net, "v", "bus", 1.006, .004, bus1)  # V at bus 1
pp.create_measurement(net, "v", "bus", 0.968, .004, bus2)  # V at bus 2
pp.create_measurement(net, "p", "bus", -501, 10, bus2)  # P at bus 2
pp.create_measurement(net, "q", "bus", -286, 10, bus2)  # Q at bus 2
pp.create_measurement(net, "p", "line", 888, 8, bus=bus1, element=line1)  # Pline (bus 1 -> bus 2) at bus 1
pp.create_measurement(net, "p", "line", 1173, 8, bus=bus1, element=line2)  # Pline (bus 1 -> bus 3) at bus 1
pp.create_measurement(net, "q", "line", 568, 8, bus=bus1, element=line1)  # Qline (bus 1 -> bus 2) at bus 1
pp.create_measurement(net, "q", "line", 663, 8, bus=bus1, element=line2)  # Qline (bus 1 -> bus 3) at bus 1
```

Now that the data is ready, the state_estimation can be initialized and run. We want to use the flat start condition, in which all voltages are set to 1.0 p.u..

```python
success = estimate(net, init="flat")
V, delta = net.res_bus_est.vm_pu, net.res_bus_est.va_degree
```

The resulting variables now contain the voltage absolute values in `V`, the voltage angles in `delta`, an indication of success in `success`. The bus power injections can be accessed similarly with `net.res_bus_est.p_kw` and `net.res_bus_est.q_kvar`. Line data is also available in the same format as defined in `res_line`.

If we like to check our data for fault measurements, and exclude them in our state estimation, we use the following code:

```python
success_rn_max = remove_bad_data(net, init="flat")
V_rn_max, delta_rn_max = net.res_bus_est.vm_pu, net.res_bus_est.va_degree
```
In the case that we only like to know if there is a likelihood of fault measurements (probability of fault can be adjusted), the Chi-squared test should be performed separately. If the test detects the possibility of fault data, the value of the added class member variable `bad_data_present` would be `true` as well as the boolean variable `success_chi2` that is used here:

```python
success_chi2 = chi2_analysis(net, init="flat")
```
7 Topological Searches

pandapower provides the possibility of graph searches using the networkx package, which is “a Python language software package for the creation, manipulation, and study of the structure, dynamics, and function of complex networks.” (see NetworkX documentation http://networkx.github.io/documentation/networkx-1.10/index.html)

pandapower provides a function to translate pandapower networks into networkx graphs. Once the electric network is translated into an abstract networkx graph, all network operations that are available in networkx can be used to analyse the network. For example you can find the shortest path between two nodes, find out if two areas in a network are connected to each other or if there are cycles in a network. For a complete list of all NetworkX algorithms see http://networkx.github.io/documentation/networkx-1.10/reference/algorithms.html

pandapower also provides some search algorithms specialized for electric networks, such as finding all buses that are connected to a slack node.

7.1 Create networkx graph

The basis of all topology functions is the conversion of a pandapower network into a NetworkX MultiGraph. A MultiGraph is a simplified representation of a network’s topology, reduced to nodes and edges. Busses are being represented by nodes (Note: only buses with in_service = 1 appear in the graph), edges represent physical connections between buses (typically lines or trafos). Multiple parallel edges between nodes are possible.

This is a very simple example of a pandapower network being converted to a MultiGraph. (Note: The MultiGraph’s shape is completely arbitrary since MultiGraphs have no inherent shape unless geodata is provided.)

Nodes have the same indices as the buses they originate from. Edges are defined by the nodes they connect. Additionally nodes and edges can hold key/value attribute pairs.

The following attributes get transferred into the MultiGraph:

| lines | trafos |
Apart from these there are no element attributes contained in the MultiGraph!

Creating a multigraph from a pandapower network

The function create_nxgraph function from the pandapower.topology package allows you to convert a pandapower network into a MultiGraph:

\[
\text{pandapower.topology.create_nxgraph}\left(\text{net}, \text{respect_switches=True, include_lines=True, include_trafos=True, n gobuses=None, notravbuses=None, multi=True}\right)
\]

Converts a pandapower network into a NetworkX graph, which is a simplified representation of a network’s topology, reduced to nodes and edges. Busses are being represented by nodes (Note: only buses with in_service = 1 appear in the graph), edges represent physical connections between buses (typically lines or trafos).

**INPUT:** net (pandapowerNet) - variable that contains a pandapower network

**OPTIONAL:**
- **respect_switches** (boolean, True) - True: open line switches are being considered (no edge between nodes)
- **include_lines** (boolean, True) - determines, whether lines get converted to edges
- **include_trafos** (boolean, True) - determines, whether trafos get converted to edges
- **nogobuses** (integer/list, None) - nogobuses are not being considered in the graph
- **notravbuses** (integer/list, None) - lines connected to these buses are not being considered in the graph
- **multi** (boolean, True) - True: The function generates a NetworkX MultiGraph, which allows multiple parallel edges between nodes False: NetworkX Graph (no multiple parallel edges)

**OUTPUT:** mg - Returns the required NetworkX graph

**EXAMPLE:** import pandapower.topology as top

\[
\text{mg = top.create_nx_graph(\text{net}, respect_switches = False)} \# \text{converts the pandapower network \text{“net” to a MultiGraph. Open switches will be ignored.}}
\]

**Examples**

\[
\text{create_nxgraph(\text{net}, respect_switches = False)}
\]
create_nxgraph(net, include_lines = False)

create_nxgraph(net, include_trafos = False)
create_nxgraph(net, nogobuses = [4])

trafos are excluded \rightarrow lines are the only edges

create_nxgraph(net, notravbuses = [4])

bus 4 is a nogobus \rightarrow bus 4 and lines connected to it are not being considered in the graph
7.2 Topological Searches

Once you converted your network into a MultiGraph there are several functions to perform topological searches and analyses at your disposal. You can either use the general-purpose functions that come with NetworkX (see http://networkx.github.io/documentation/networkx-1.10/reference/algorithms.html) or topology's own ones which are specialized on electrical networks.

7.2.1 calc_distance_to_bus

pandapower.topology.calc_distance_to_bus(net, bus, respect_switches=True, nogobuses=None, notravbuses=None)

Calculates the shortest distance between a source bus and all buses connected to it.

INPUT: net (pandapowerNet) - Variable that contains a pandapower network.
        bus (integer) - Index of the source bus.

OPTIONAL:

        respect_switches (boolean, True) - True: open line switches are being considered
        (no edge between nodes)
        False: open line switches are being ignored

        nogobuses (integer/list, None) - nogobuses are not being considered
        notravbuses (integer/list, None) - lines connected to these buses are not being considered

OUTPUT:

        dist - Returns a pandas series with containing all distances to the source bus in km.

EXAMPLE: import pandapower.topology as top
dist = top.calc_distance_to_bus(net, 5)
7.2.2 connected_component

\texttt{pandapower.topology.connected\_component}(mg, bus, notravbuses=[])

Finds all buses in a NetworkX graph that are connected to a certain bus.

**INPUT:**
- \texttt{mg} (NetworkX graph) - NetworkX Graph or MultiGraph that represents a pandapower network.
- \texttt{bus} (integer) - Index of the bus at which the search for connected components originates

**OPTIONAL:**
- \texttt{notravbuses} (list/set) - Indices of notravbuses: lines connected to these buses are not being considered in the graph

**OUTPUT:**
- \texttt{cc} (generator) - Returns a generator that yields all buses connected to the input bus

**EXAMPLE:**
```python
import pandapower.topology as top
g = top.create_nx_graph(net)
cc = top.connected_component(g, 5)
```

7.2.3 connected_components

\texttt{pandapower.topology.connected\_components}(mg, notravbuses=set())

Clusters all buses in a NetworkX graph that are connected to each other.

**INPUT:**
- \texttt{mg} (NetworkX graph) - NetworkX Graph or MultiGraph that represents a pandapower network.

**OPTIONAL:**
- \texttt{notravbuses} (set) - Indices of notravbuses: lines connected to these buses are not being considered in the graph

**OUTPUT:**
- \texttt{cc} (generator) - Returns a generator that yields all clusters of buses connected to each other.

**EXAMPLE:**
```python
import pandapower.topology as top
g = top.create_nx_graph(net)
cc = top.connected_components(net, 5)
```

7.2.4 unsupplied_buses

\texttt{pandapower.topology.unsupplied\_buses}(net, mg=None, in_service_only=False, slacks=None, respect_switches=True)

Finds buses, that are not connected to an external grid.

**INPUT:**
- \texttt{net} (pandapowerNet) - variable that contains a pandapower network

**OPTIONAL:**
- \texttt{mg} (NetworkX graph) - NetworkX Graph or MultiGraph that represents a pandapower network.
- \texttt{in\_service\_only} (boolean) - Indicates whether only buses with in service status should be considered.
- \texttt{slacks} (list/set) - Indices of slacks.
- \texttt{respect\_switches} (boolean) - Indicates whether to respect open switches.

**OUTPUT:**
- \texttt{ub} (set) - unsupplied buses

**EXAMPLE:**
```python
import pandapower.topology as top
top.unsupplied_buses(net)
```

7.2.5 determine_stubs

\texttt{pandapower.topology.determine\_stubs}(net, roots=None, mg=None, respect_switches=False)

Finds stubs in a network. Open switches are being ignored. Results are being written in a new column in the bus table (“on\_stub”) and line table (“is\_stub”) as True/False value.
7 TOPOLOGICAL SEARCHES

INPUT: net (pandapowerNet) - Variable that contains a pandapower network.

OPTIONAL:

roots (integer/list, None) - Indices of buses that should be excluded (by default, the ext_grid buses will be set as roots)

EXAMPLE: import pandapower.topology as top
top.determine_stubs(net, roots = [0, 1])

7.3 Examples

The combination of a suitable MultiGraph and the available topology functions enables you to perform a wide range of topological searches and analyses.

Here are a few examples of what you can do:

basic example network

```python
import pandapower as pp

net = pp.create_empty_network()

net.create_bus(net, name = "110 kV bar", vn_kv = 110, type = 'b')
net.create_bus(net, name = "20 kV bar", vn_kv = 20, type = 'b')
net.create_bus(net, name = "bus 2", vn_kv = 20, type = 'b')
net.create_bus(net, name = "bus 3", vn_kv = 20, type = 'b')
net.create_bus(net, name = "bus 4", vn_kv = 20, type = 'b')
net.create_bus(net, name = "bus 5", vn_kv = 20, type = 'b')
net.create_bus(net, name = "bus 6", vn_kv = 20, type = 'b')

net.create_ext_grid(net, 0, vm_pu = 1)

net.create_line(net, name = "line 0", from_bus = 1, to_bus = 2, length_km = 1, std_type = "NAYY 150")
net.create_line(net, name = "line 1", from_bus = 2, to_bus = 3, length_km = 1, std_type = "NAYY 150")
net.create_line(net, name = "line 2", from_bus = 3, to_bus = 4, length_km = 1, std_type = "NAYY 150")
net.create_line(net, name = "line 3", from_bus = 4, to_bus = 5, length_km = 1, std_type = "NAYY 150")
net.create_line(net, name = "line 4", from_bus = 5, to_bus = 6, length_km = 1, std_type = "NAYY 150")
net.create_line(net, name = "line 5", from_bus = 6, to_bus = 1, length_km = 1, std_type = "NAYY 150")

net.create_transformer_from_parameters(net, hv_bus = 0, lv_bus = 1, i0_percent= 0.038, pfe_kw = 11.6,
vscr_percent = 0.322, sn_kva = 40000.0, vn_lv_kv = 22.0,
vn_hv_kv = 110.0, vsc_percent = 17.8)

net.create_load(net, 2, p_kw = 1000, q_kvar = 200, name = "load 0")
net.create_load(net, 3, p_kw = 1000, q_kvar = 200, name = "load 1")
net.create_load(net, 4, p_kw = 1000, q_kvar = 200, name = "load 2")
net.create_load(net, 5, p_kw = 1000, q_kvar = 200, name = "load 3")
net.create_load(net, 6, p_kw = 1000, q_kvar = 200, name = "load 4")

net.create_switch(net, bus = 1, element = 0, et = 'l')
net.create_switch(net, bus = 2, element = 0, et = 'l')
net.create_switch(net, bus = 2, element = 1, et = 'l')
net.create_switch(net, bus = 3, element = 1, et = 'l')
net.create_switch(net, bus = 3, element = 2, et = 'l')
net.create_switch(net, bus = 4, element = 2, et = 'l')
```
### 7.3.1 Using NetworkX algorithms: shortest path

For many basic network analyses the algorithms that come with the NetworkX package will work just fine and you won’t need one of the specialised topology functions. Finding the shortest path between two buses is a good example for that.

```python
import pandapower.topology as top
import networkx as nx

mg = top.create_nxgraph(net)
x.shortest_path(mg, 0, 5)
```

Out: `[0, 1, 6, 5]`

#### 7.3.2 Find disconnected buses

With `unsupplied_buses` you can easily find buses that are not connected to an external grid.

```python
import pandapower.topology as top
net.switch.closed.at[11] = 0
top.unsupplied_buses(net)
```

Out: `{5, 6}`
7.3.3 Calculate distances between buses

calc_distance_to_bus allows you to calculate the distance (= shortest network route) from one bus all other ones. This is possible since line lengths are being transferred into the MultiGraph as an edge attribute. (Note: bus-bus-switches and trafos are interpreted as edges with length = 0)

```python
import pandapower.topology as top
net.switch.closed.at[6] = 1
net.switch.closed.at[8] = 0
top.calc_distance_to_bus(net, 1)
```

```
Out:
0 0
1 0
2 1
3 2
4 3
5 4
6 1
```

**Interpretation:** The distance between bus 1 and itself is 0 km. Bus 1 is also 0 km away from bus 0, since they are connected with a transformer. The shortest path between bus 1 and bus 5 is 4 km long.
7.3.4 Find connected buses with the same voltage level

```python
import pandapower.topology as top
mg_no_trafos = top.create_nxgraph(net, include_trafos = False)
cc = top.connected_components(mg_no_trafos)

In : next(cc)
Out : {0}
In : next(cc)
Out : {1, 2, 3, 4, 5, 6}
```
7.3.5 Find rings and ring sections

Another example of what you can do with the right combination of input arguments when creating the MultiGraph is finding rings and ring sections in your network. To achieve that for our example network, the trafo buses needs to be set as a nogobuses. With `respect_switches = True` you get the ring sections, with `respect_switches = False` the whole ring.

```python
import pandapower.topology as top

mg_ring_sections = top.create_nxgraph(net, nogobuses = [0, 1])
cc_ring_sections = top.connected_components(mg_ring_sections)
```

In : `next(cc_ring_sections)`
Out : `{2, 3, 4}`

In : `next(cc_ring_sections)`
Out : `{5, 6}`

**pandapower network**

**MultiGraph**

![Diagram of a network with buses and lines]

Busses 0 and 1 are nogobuses —→ only bussesthat belong to the same ring section are connected to each other.

```python
import pandapower.topology as top

mg_ring = top.create_nxgraph(net, respect_switches = False, nogobuses = [0,1])
cc_ring = top.connected_components(mg_ring)
```

In : `next(cc_ring)`
Out : `{2, 3, 4, 5, 6}`
7.3.6 Find stubs

determine_stubs lets you identify buses and lines that are stubs. Open switches are being ignored. Busses that you want to exclude should be defined as roots. Ext_grid buses are roots by default.

This is a small extension for the example network:

```python
pp.create_bus(net, name = "bus 7", vn_kv = 20, type = 'b')
pp.create_bus(net, name = "bus 8", vn_kv = 20, type = 'b')
pp.create_line(net, name = "line 6", from_bus = 6, to_bus = 7, length_km = 1, std_type = "NAYY 150")
pp.create_line(net, name = "line 7", from_bus = 7, to_bus = 8, length_km = 1, std_type = "NAYY 150")
pp.create_load(net, 7, p_kw = 1000, q_kvar = 200, name = "load 5")
pp.create_load(net, 8, p_kw = 1000, q_kvar = 200, name = "load 6")

import pandapower.topology as top
top.determine_stubs(net, roots = [0,1])
```

In: net.bus

Out:

<table>
<thead>
<tr>
<th>name</th>
<th>vn_kv</th>
<th>min_vm_pu</th>
<th>max_vm_pu</th>
<th>type</th>
<th>zone</th>
<th>in_service</th>
<th>auf_stich</th>
</tr>
</thead>
<tbody>
<tr>
<td>bus 7</td>
<td>20</td>
<td>NaN</td>
<td>NaN</td>
<td>b</td>
<td>None</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>bus 8</td>
<td>20</td>
<td>NaN</td>
<td>NaN</td>
<td>b</td>
<td>None</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>bus 2</td>
<td>20</td>
<td>NaN</td>
<td>NaN</td>
<td>b</td>
<td>None</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>bus 3</td>
<td>20</td>
<td>NaN</td>
<td>NaN</td>
<td>b</td>
<td>None</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>bus 4</td>
<td>20</td>
<td>NaN</td>
<td>NaN</td>
<td>b</td>
<td>None</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>bus 5</td>
<td>20</td>
<td>NaN</td>
<td>NaN</td>
<td>b</td>
<td>None</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>bus 6</td>
<td>20</td>
<td>NaN</td>
<td>NaN</td>
<td>b</td>
<td>None</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>bus 7</td>
<td>20</td>
<td>NaN</td>
<td>NaN</td>
<td>b</td>
<td>None</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>bus 8</td>
<td>20</td>
<td>NaN</td>
<td>NaN</td>
<td>b</td>
<td>None</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

In: net.line
Out:

<table>
<thead>
<tr>
<th>name</th>
<th>std_type</th>
<th>from_bus</th>
<th>to_bus</th>
<th>length_km</th>
<th>r_ohm_per_km</th>
<th>x_ohm_per_km</th>
<th>c_nf_per_km</th>
<th>max_i_ka</th>
<th>type</th>
<th>in_service</th>
<th>is_stich</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NAYY 150</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0.206</td>
<td>0.091</td>
<td></td>
<td></td>
<td>cs</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>1</td>
<td>NAYY 150</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>0.206</td>
<td>0.091</td>
<td></td>
<td></td>
<td>cs</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>2</td>
<td>NAYY 150</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>0.206</td>
<td>0.091</td>
<td></td>
<td></td>
<td>cs</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>3</td>
<td>NAYY 150</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>0.206</td>
<td>0.091</td>
<td></td>
<td></td>
<td>cs</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>4</td>
<td>NAYY 150</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td>0.206</td>
<td>0.091</td>
<td></td>
<td></td>
<td>cs</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>5</td>
<td>NAYY 150</td>
<td>6</td>
<td>7</td>
<td>1</td>
<td>0.206</td>
<td>0.091</td>
<td></td>
<td></td>
<td>cs</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>6</td>
<td>NAYY 150</td>
<td>7</td>
<td>8</td>
<td>1</td>
<td>0.206</td>
<td>0.091</td>
<td></td>
<td></td>
<td>cs</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

The open switch is being ignored. Buses 0 and 1 are roots.
8 Networks

Besides creating your own grids through the pandapower API pandapower provides synthetic and Benchmark networks through the networks module. The pandapower networks modul contains simple test networks, randomly generated networks, CIGRE test networks, IEEE case files and synthetic networks from the dissertation of Georg Kerber and Lindner et. al..

You can find documentation for the individual modules here:

8.1 Example Networks

There are two example networks available. The simple example network shows the basic principles of how to create a pandapower network. If you like to study a more advanced and thus more complex network, please take a look at the more multi-voltage level example network.

8.1.1 Simple Example Network

The following example contains all basic elements that are supported by the pandapower format. It is a simple example to show the basic principles of creating a pandapower network.

pandapower.networks.example_simple()  
Returns the simple example network from the pandapower tutorials.

OUTPUT: net - simple example network

EXAMPLE:

```python
>>> import pandapower.networks
>>> net = pandapower.networks.example_simple()
```

The stepwise creation of this network is shown in the pandapower tutorials.

8.1.2 Multi-Voltage Level Example Network

The following example contains all elements that are supported by the pandapower format. It is a more realistic network than the simple example and of course more complex. Using typically voltage levels for european distribution networks (high, medium and low voltage) the example relates characteristic topologies, utility types, line lengths and generator type distribution to the various voltage levels. To set network size limits the quantity of nodes in every voltage level is restricted and one medium voltage open ring and only two low voltage feeder are considered. Other feeders are represented by equivalent loads. As an example one double busbar and one single busbar are considered.

pandapower.networks.example_multivoltage()  
Returns the multivoltage example network from the pandapower tutorials.

OUTPUT: net - multivoltage example network

EXAMPLE:

```python
>>> import pandapower.networks
>>> net = pandapower.networks.example_multivoltage()
```
The stepwise creation of this network is shown in the pandapower tutorials.

### 8.2 Simple pandapower test networks

#### 8.2.1 Four load branch

**pandapower.networks.panda_four_load_branch()**

This function creates a simple six bus system with four radial low voltage nodes connected to a medium voltage slack bus. At every low voltage node the same load is connected.

**OUTPUT:** net - Returns the required four load system

**EXAMPLE:**
```python
import pandapower.networks as pn
net_four_load = pn.panda_four_load_branch()
```

![Diagram of four load branch network](image)

#### 8.2.2 Four loads with branches out

**pandapower.networks.four_loads_with_branches_out()**

This function creates a simple ten bus system with four radial low voltage nodes connected to a medium voltage slack bus. At every of the four radial low voltage nodes another low voltage node with a load is connected via cable.

**OUTPUT:** net - Returns the required four load system with branches

**EXAMPLE:**
```python
import pandapower.networks as pn
net_four_load_with_branches = pn.four_loads_with_branches_out()
```
8.2.3 Four bus system

`pandapower.networks.simple_four_bus_system()`

This function creates a simple four bus system with two radial low voltage nodes connected to a medium voltage slack bus. At both low voltage nodes the a load and a static generator is connected.

**OUTPUT**: `net` - Returns the required four bus system

**EXAMPLE**: import pandapower.networks as pn

    net_simple_four_bus = pn.simple_four_bus_system()
8.2.4 Medium voltage open ring

`pandapower.networks.simple_mv_open_ring_net()`

This function creates a simple medium voltage open ring network with loads at every medium voltage node. As an example this function is used in the topology and diagnostic docu.

**OUTPUT:** net - Returns the required simple medium voltage open ring network

**EXAMPLE:** import pandapower.networks as pn
net_simple_open_ring = pn.simple_mv_open_ring_net()
8.3 CIGRE Networks

CIGRE-Networks were developed by the CIGRE Task Force C6.04.02 to “facilitate the analysis and validation of new methods and techniques” that aim to “enable the economic, robust and environmentally responsible integration of DER” (Distributed Energy Resources). CIGRE-Networks are a set of comprehensive reference systems to allow the “analysis of DER integration at high voltage, medium voltage and low voltage and at the desired degree of detail”.

Note: Source for this network is the final Report of Task Force C6.04.02: “Benchmark Systems for Network Integration of Renewable and Distributed Energy Resources”, 2014


8.3.1 High voltage transmission network

```python
import pandapower.networks as pn
# You have to specify a length for the connection line between buses 6a and 6b
net = pn.create_cigre_network_hv(length_km_6a_6b)
```

This pandapower network includes the following parameter tables:
- shunt (3 elements)
- trafo (6 elements)
8.3.2 Medium voltage distribution network

```python
import pandapower.networks as pn
net = pn.create_cigre_network_mv(with_der=False)
```

This pandapower network includes the following parameter tables:
- switch (8 elements)
- load (18 elements)
- ext_grid (1 elements)
- line (15 elements)
- trafo (2 elements)
- bus (15 elements)
- bus_geodata (15 elements)
8.3.3 Medium voltage distribution network with PV and Wind DER

**Note:** This network contains additional 9 distributed energy resources compared to medium voltage distribution network:

- 8 photovoltaic generators
- 1 wind turbine

Compared to the case study of CIGRE Task Force C6.04.02 paper all pv and wind energy resources are considered but 2 Batteries, 2 residential fuel cells, 1 CHP diesel and 1 CHP fuel cell are neglected. Although the case study mentions the High Voltage as 220 kV, we assume 110 kV again because of no given 220 kV-Trafo data.

```python
import pandapower.networks as pn

net = pn.create_cigre_network_mv(with_der="pv_wind")
```
This pandapower network includes the following parameter tables:
- switch (8 elements)
- load (18 elements)
- ext_grid (1 element)
- sgen (9 elements)
- line (15 elements)
- trafo (2 elements)
- bus (15 elements)
- bus_geodata (15 elements)

8.3.4 Medium voltage distribution network with all DER

Note: This network contains additional 15 distributed energy resources compared to medium voltage distribution network:
- 8 photovoltaic generators
• 1 wind turbine
• 2 Batteries
• 2 residential fuel cells
• 1 CHP diesel
• 1 CHP fuel cell

Compared to the case study of CIGRE Task Force C6.04.02 paper all distributed energy resources are considered. Although the case study mentions the High Voltage as 220 kV, we assume 110 kV again because of no given 220 kV-Trafo data.

```python
import pandapower.networks as pn

net = pn.create_cigre_network_mv(with_der="all")

'''
This pandapower network includes the following parameter tables:
- switch (8 elements)
- load (18 elements)
- ext_grid (1 elements)
- sgen (15 elements)
- line (15 elements)
- trafo (2 elements)
- bus (15 elements)
- bus_geodata (15 elements)
'''
```
8.3.5 Low voltage distribution network

```python
import pandapower.networks as pn
net = pn.create_cigre_network_lv()
```

This pandapower network includes the following parameter tables:

- switch (3 elements)
- load (15 elements)
- ext_grid (1 elements)
- line (37 elements)
- trafo (3 elements)
- bus (44 elements)
- bus_geodata (44 elements)
8.4 MV Oberrhein

Note: The MV Oberrhein network is a generic network assembled from openly available data supplemented with parameters based on experience.

`pandapower.networks.mv_oberrhein()`

_loads the Oberrhein network, a generic 20 kV network serviced by two 25 MVA HV/MV transformer stations. The network supplies 141 HV/MV substations and 6 MV loads through four MV feeders. The network layout is meshed, but the network is operated as a radial network with 6 open sectioning points.

The network can be loaded with two different worst case scenarios for load and generation, which are defined by scaling factors for loads / generators as well as tap positions of the HV/MV transformers. These worst case scenarios are a good starting point for working with this network, but you are of course free to parametrize the network for your use case.

The network also includes geographical information of lines and buses for plotting.

**OPTIONAL:** scenario - (str, “load”): defines the scaling for load and generation

- “load”: high load scenario, load = 0.6 / sgen = 0, trafo taps [-2, -3]
- “generation”: high feed-in scenario: load = 0.1, generation = 0.8, trafo taps [0, 0]

`cosphi_load` - (str, 0.98): cosine(phi) of the loads

`cosphi_sgen` - (str, 1.0): cosine(phi) of the static generators
**include_substations** - (bool, False): if True, the transformers of the MV/LV level are modelled, otherwise the loads representing the LV networks are connected directly to the MV node

**OUTPUT: net** - pandapower network

**EXAMPLE:**

```python
import pandapower.networks
net = pandapower.networks.mv_oberrhein("generation")
```

The geographical representation of the network looks like this:

![Geographical representation of the network](image)

The different colors of the MV/LV stations indicate the feeders which are galvanically separated by open switches. If you are interested in how to make plots such as these, check out the pandapower tutorial on plotting.

The power flow results of the network in the different worst case scenarios look like this:

![Power flow results](image)
As you can see, the network is designed to comply with a voltage band of $0.975 < u < 1.03$ and line loading of $<60\%$ in the high load case (for n-1 security) and $<100\%$ in the low load case.

## 8.5 Power System Test Cases

**Note:** All Power System Test Cases were converted from PYPOWER or MATPOWER case files.

### 8.5.1 Case 4gs

```python
pandapower.networks.case4gs()
```

This is the 4 bus example from J. J. Grainger and W. D. Stevenson, Power system analysis. McGraw-Hill, 1994. pp. 337-338. Its data origin is PYPOWER.

**OUTPUT:** net - Returns the required ieee network case4gs

**EXAMPLE:** import pandapower.networks as pn

```python
net = pn.case4gs()
```

### 8.5.2 Case 6ww

```python
pandapower.networks.case6ww()
```


**OUTPUT:** net - Returns the required ieee network case6ww

**EXAMPLE:** import pandapower.networks as pn

```python
net = pn.case6ww()
```

### 8.5.3 Case 9

```python
pandapower.networks.case9()
```

Calls the pickle file case9.p which data origin is PYPOWER. This network was published in Anderson and Fouad’s book ‘Power System Control and Stability’ for the first time in 1980.

**OUTPUT:** net - Returns the required ieee network case9

**EXAMPLE:** import pandapower.networks as pn

```python
net = pn.case9()
```

### 8.5.4 Case 14

```python
pandapower.networks.case14()
```

Calls the pickle file case14.p which data origin is PYPOWER. This network was converted from IEEE Common Data Format (ieee14cdf.txt) on 20-Sep-2004 by cdf2matp, rev. 1.11, to matpower format and finally converted to pandapower format by pandapower.converter.from_ppc. The vn_kv was adapted considering the proposed voltage levels in Washington case 14.
OUTPUT: net - Returns the required ieee network case14

EXAMPLE: import pandapower.networks as pn
net = pn.case14()

8.5.5 Case 30

pandapower.networks.case30()

This function calls the pickle file case30.p which data origin is PYPOWER. Some more information about this network are given by Washington case 30 and Illinois University case 30.

OUTPUT: net - Returns the required ieee network case30

EXAMPLE: import pandapower.networks as pn
net = pn.case30()

8.5.6 Case 33bw

pandapower.networks.case33bw()

Calls the pickle file case33bw.p which data is provided by MATPOWER. The data origin is the paper M. Baran, F. Wu, Network reconfiguration in distribution systems for loss reduction and load balancing IEEE Transactions on Power Delivery, 1989.

OUTPUT: net - Returns the required ieee network case33bw

EXAMPLE: import pandapower.networks as pn
net = pn.case33bw()

8.5.7 Case 39

pandapower.networks.case39()

Calls the pickle file case39.p which data origin is PYPOWER. This network was published the first time in G. Bills et al., On-line stability analysis study, RP 90-1, E. P. R. I. North American Rockwell Corporation, Edison Electric Institute, Ed. IEEE Press, Oct. 1970,. Some more information about this network are given by Illinois University case 39. Because the Pypower data origin proposes vn_kv=345 for all nodes the transformers connect node of the same voltage level.

OUTPUT: net - Returns the required ieee network case39

EXAMPLE: import pandapower.networks as pn
net = pn.case39()
8.5.8 Case 57

pandapower.networks.case57 (vn_kv_area1=115, vn_kv_area2=500, vn_kv_area3=138, vn_kv_area4=345, vn_kv_area5=230, vn_kv_area6=161)

This function provides the ieee case57 network with the data origin PYPOWER case 57. Some more information about this network are given by Illinois University case 57. Because the Py_power data origin proposes no vn_kv some assumption must be made. There are six areas with coinciding voltage level. These are:

- area 1 with coinciding voltage level comprises node 1-17
- area 2 with coinciding voltage level comprises node 18-20
- area 3 with coinciding voltage level comprises node 21-24 + 34-40 + 44-51
- area 4 with coinciding voltage level comprises node 25 + 30-33
- area 5 with coinciding voltage level comprises node 41-43 + 56-57
- area 6 with coinciding voltage level comprises node 52-55 + 26-29

OUTPUT: net - Returns the required ieee network case57

EXAMPLE:
import pandapower.networks as pn
net = pn.case57()

8.5.9 Case 89pegase

pandapower.networks.case89pegase ()

Calls the pickle file case89pegase.p which data is provided by MATPOWER. The data origin are the paper C. Josz, S. Fliscounakis, J. Maenght, P. Panciatici, AC power flow data in MATPOWER and QCQP for-
mat: iTesla, RTE snapshots, and PEGASE, 2016 and S. Fliscounakis, P. Panciatici, F. Capitanescu, and L.
Wehenkel, Contingency ranking with respect to overloads in very large power systems taking into account
4909-4917, Nov 2013..

OUTPUT: net - Returns the required ieee network case89pegase

EXAMPLE:
import pandapower.networks as pn
net = pn.case89pegase()

8.5.10 Case 118

pandapower.networks.case118 ()

Calls the pickle file case118.p which data origin is PYPOWER. Some more information about this network
are given by Washington case 118 and Illinois University case 118.

OUTPUT: net - Returns the required ieee network case118

EXAMPLE:
import pandapower.networks as pn
net = pn.case118()
8.5.11 Case 145

pandapower.networks.case145()

Calls the pickle file case145.p which data origin is MATPOWER. This data is converted by MATPOWER 5.1 using CDF2MPC on 18-May-2016 from 'dd50cdf.txt'.

OUTPUT: net - Returns the required ieee network case145

EXAMPLE:
import pandapower.networks as pn
net = pn.case145()

8.5.12 Case 300

pandapower.networks.case300()

Calls the pickle file case300.p which data origin is PYPOWER. Some more information about this network are given by Washington case 300 and Illinois University case 300.

OUTPUT: net - Returns the required ieee network case300

EXAMPLE:
import pandapower.networks as pn
net = pn.case300()

8.5.13 Case 1354pegase

pandapower.networks.case1354pegase()

This grid represents a part of the European high voltage transmission network. The data is provided by MATPOWER. The data origin are the paper C. Josz, S. Fliscounakis, J. Maenght, P. Panciatici, AC power flow data in MATPOWER and QCQP format: iTesla, RTE snapshots, and PEGASE, 2016 and S. Fliscounakis, P. Panciatici, F. Capitanescu, and L. Wehenkel, Contingency ranking with respect to overloads in very large power systems taking into account uncertainty, preventive, and corrective actions, IEEE Transactions on Power Systems, vol. 28, no. 4, pp. 4909-4917, Nov 2013..

OUTPUT: net - Returns the required ieee network case1354pegase

EXAMPLE:
import pandapower.networks as pn
net = pn.case1354pegase()

8.5.14 Case 1888rte

pandapower.networks.case1888rte(ref_bus_idx=1246)

This case accurately represents the size and complexity of French very high voltage and high voltage transmission network. The data is provided by MATPOWER. The data origin is the paper C. Josz, S. Fliscounakis, J. Maenght, P. Panciatici, AC power flow data in MATPOWER and QCQP format: iTesla, RTE snapshots, and PEGASE, 2016.

OPTIONAL:

ref_bus_idx - Since the MATPOWER case provides a reference bus without connected generator, because a distributed slack is assumed, to convert the data to pandapower, another bus has been assumed as reference bus. Via ‘ref_bus_idx’ the User can choose a reference bus, which should have a generator connected to. Please be aware that by changing the reference bus to another bus than the proposed default value, maybe a powerflow does not converge anymore!
8.5.15 Case 2848rte

pandapower.networks.case2848rte(ref_bus_idx=271)

This case accurately represents the size and complexity of French very high voltage and high voltage transmission network. The data is provided by MATPOWER. The data origin is the paper C. Josz, S. Flis- counakis, J. Maenght, P. Panciatici, AC power flow data in MATPOWER and QCQP format: iTesla, RTE snapshots, and PEGASE, 2016.

OPTIONAL:

ref_bus_idx - Since the MATPOWER case provides a reference bus without connected generator, because a distributed slack is assumed, to convert the data to pandapower, another bus has been assumed as reference bus. Via ‘ref_bus_idx’ the User can choose a reference bus, which should have a generator connected to. Please be aware that by changing the reference bus to another bus than the proposed default value, maybe a powerflow does not converge anymore!

OUTPUT: net - Returns the required ieee network case2848rte

EXAMPLE: import pandapower.networks as pn

net = pn.case2848rte()

8.5.16 Case 2869pegase

pandapower.networks.case2869pegase()

This grid represents a part of the European high voltage transmission network. The data is provided by MATPOWER. The data origin is the paper C. Josz, S. Fliscounakis, J. Maenght, P. Panciatici, AC power flow data in MATPOWER and QCQP format: iTesla, RTE snapshots, and PEGASE, 2016 and S. Fliscounakis, P. Panciatici, F. Capitanescu, and L. Wehenkel, Contingency ranking with respect to overloads in very large power systems taking into account uncertainty, preventive, and corrective actions, IEEE Transactions on Power Systems, vol. 28, no. 4, pp. 4909-4917, Nov 2013..

OUTPUT: net - Returns the required ieee network case2869pegase

EXAMPLE: import pandapower.networks as pn

net = pn.case2869pegase()

8.5.17 Case 3120sp

pandapower.networks.case3120sp()

This case represents the Polish 400, 220 and 110 kV networks during summer 2008 morning peak conditions. The data was provided by Roman Korab <roman.korab@polsl.pl> and to pandapower converted from MATPOWER.

OUTPUT: net - Returns the required ieee network case3120sp

EXAMPLE: import pandapower.networks as pn

net = pn.case3120sp()
8.5.18 Case 6470rte

pandapower.networks.case6470rte(ref_bus_idx=5988)
This case accurately represents the size and complexity of French very high voltage and high voltage transmission network. The data is provided by MATPOWER. The data origin is the paper C. Josz, S. Fliscounakis, J. Maenght, P. Panciatici, AC power flow data in MATPOWER and QCQP format: iTesla, RTE snapshots, and PEGASE, 2016.

OPTIONAL:
ref_bus_idx - Since the MATPOWER case provides a reference bus without connected generator, because a distributed slack is assumed, to convert the data to pandapower, another bus has been assumed as reference bus. Via ‘ref_bus_idx’ the User can choose a reference bus, which should have a generator connected to. Please be aware that by changing the reference bus to another bus than the proposed default value, maybe a powerflow does not converge anymore!

OUTPUT: net - Returns the required ieee network case6470rte

EXAMPLE: import pandapower.networks as pn
net = pn.case6470rte()

8.5.19 Case 6495rte

pandapower.networks.case6495rte(ref_bus_idx=None)
This case accurately represents the size and complexity of French very high voltage and high voltage transmission network. The data is provided by MATPOWER. The data origin is the paper C. Josz, S. Fliscounakis, J. Maenght, P. Panciatici, AC power flow data in MATPOWER and QCQP format: iTesla, RTE snapshots, and PEGASE, 2016.

OPTIONAL:
ref_bus_idx - Since the MATPOWER case provides a reference bus without connected generator, because a distributed slack is assumed, to convert the data to pandapower, another buses (6077, 6161, 6305, 6306, 6307, 6308) has been assumed as reference bus. Via ‘ref_bus_idx’ the User can choose a reference bus, which should have a generator connected to. Please be aware that by changing the reference bus to another bus than the proposed default value, maybe a powerflow does not converge anymore!

OUTPUT: net - Returns the required ieee network case6495rte

EXAMPLE: import pandapower.networks as pn
net = pn.case6495rte()

8.5.20 Case 6515rte

pandapower.networks.case6515rte(ref_bus_idx=6171)
This case accurately represents the size and complexity of French very high voltage and high voltage transmission network. The data is provided by MATPOWER. The data origin is the paper C. Josz, S. Fliscounakis, J. Maenght, P. Panciatici, AC power flow data in MATPOWER and QCQP format: iTesla, RTE snapshots, and PEGASE, 2016.

OPTIONAL:
**8.5.21 Case 9241pegase**

```python
import pandapower.networks as pn
net = pn.case9241pegase()
```

This grid represents a part of the European high voltage transmission network. The data is provided by MATPOWER. The data origin are the paper C. Josz, S. Fliscounakis, J. Maeght, P. Panciatici, AC power flow data in MATPOWER and QCQP format: iTesla, RTE snapshots, and PEGASE, 2016 and S. Fliscounakis, P. Panciatici, F. Capitanescu, and L. Wehenkel, Contingency ranking with respect to overloads in very large power systems taking into account uncertainty, preventive, and corrective actions, IEEE Transactions on Power Systems, vol. 28, no. 4, pp. 4909-4917, Nov 2013.

**OUTPUT:** `net` - Returns the required ieee network case9241pegase

**EXAMPLE:**
```python
import pandapower.networks as pn
net = pn.case9241pegase()
```

---

**8.5.22 Case GB network**

```python
import pandapower.networks as pn
net = pn.GBnetwork()
```

Calls the pickle file GBnetwork.p which data is provided by W. A. Bukhsh, Ken McKinnon, Network data of real transmission networks, April 2013. This data represents detailed model of electricity transmission network of Great Britian (GB). It consists of 2224 nodes, 3207 branches and 394 generators. This data is obtained from publically available data on National grid website. The data was originally pointing out by Manolis Belivanis, University of Strathclyde.

**OUTPUT:** `net` - Returns the required ieee network GBreducednetwork

**EXAMPLE:**
```python
import pandapower.networks as pn
net = pn.GBnetwork()
```

---

**8.5.23 Case GB reduced network**

```python
import pandapower.networks as pn
net = pn.GBreducednetwork()
```

Calls the pickle file GBreducednetwork.p which data is provided by W. A. Bukhsh, Ken McKinnon, Network data of real transmission networks, April 2013. This data is a representative model of electricity transmission network in Great Britain (GB). It was originally developed at the University of Strathclyde in 2010.

**OUTPUT:** `net` - Returns the required ieee network GBreducednetwork

**EXAMPLE:**
```python
import pandapower.networks as pn
net = pn.GBreducednetwork()
```
### 8.5.24 Case iceland

```python
calls the pickle file iceland.p which data is provided by W. A. Bukhsh, Ken McKinnon, Network data ofeal transmission networks, April 2013. This data represents electricity transmission network of Iceland.
It consists of 118 nodes, 206 branches and 35 generators. It was originally developed in PSAT format by
Patrick McNabb, Durham University in January 2011.
```

**OUTPUT:** net - Returns the required ieee network iceland

**EXAMPLE:**
```python
import pandapower.networks as pn
net = pn.iceland()
```

### 8.6 Kerber networks

The kerber networks are based on the grids used in the dissertation “Aufnahmefähigkeit von Niederspan-
nungsverteilnetzen für die Einspeisung aus Photovoltaikanlagen” (Capacity of low voltage distribution networks
with increased feed-in of photovoltaic power) by Georg Kerber. The following introduction shows the basic idea
behind his network concepts and demonstrate how you can use them in pandapower.

“The increasing amount of new distributed power plants demands a reconsideration of conventional planning
strategies in all classes and voltage levels of the electrical power networks. To get reliable results on loadability
of low voltage networks statistically firm network models are required. A strategy for the classification of low
voltage networks, exemplary results and a method for the generation of reference networks are shown.” (source:
http:/mediatum.ub.tum.de/doc/681082/681082.pdf)

#### 8.6.1 Average Kerber networks

**Kerber Landnetze:**
- Low number of loads per transformer station
- High proportion of agriculture and industry
- Typical network topologies: line

**Kerber Dorfnetz:**
- Higher number of loads per transformer station (compared to Kerber Landnetze)
- Lower proportion of agriculture and industry
- Typical network topologies: line, open ring

**Kerber Vorstadtnetze:**
- Highest number of loads per transformer station (compared to Kerber Landnetze/Dorfnetz)
- no agriculture and industry
- high building density
- Typical network topologies: open ring, meshed networks

See also:
- Georg Kerber, Aufnahmefähigkeit von Niederspannungsverteilnetzen für die Einspeisung aus Photo-
voltaikkleinanlagen, Dissertation
- Georg Kerber, Statistische Analyse von NS-Verteilungsnetzen und Modellierung von Referenznetzen

<table>
<thead>
<tr>
<th>Kerber Landnetze</th>
<th>Lines</th>
<th>Total Length</th>
<th>Loads</th>
<th>Installed Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freileitung 1</td>
<td>13</td>
<td>0.273 km</td>
<td>13</td>
<td>104 kW</td>
</tr>
</tbody>
</table>
You can include the kerber networks by simply using:

```python
import pandapower.networks as pn
net1 = pn.create_kerber_net()
```

**8.6.2 Kerber Landnetze**

```python
import pandapower.networks as pn
net1 = pn.create_kerber_landnetz_freileitung_1()

This pandapower network includes the following parameter tables:
- load (13 elements) p_load_in_kw=8, q_load_in_kw=0
- bus (15 elements)
- line (13 elements) std_type="Al 120", l_lines_in_km=0.021
- trafo (1 elements) std_type="0.125 MVA 10/0.4 kV Dyn5 ASEA"
- ext_grid (1 elements)

net2 = pn.create_kerber_landnetz_freileitung_2()

This pandapower network includes the following parameter tables:
- load (8 elements) p_load_in_kw=8, q_load_in_kw=0
- bus (10 elements)
- line (8 elements) std_type="AL 50", l_lines_1_in_km=0.038, l_lines_2_in_km=0.081
- trafo (1 elements) std_type="0.125 MVA 10/0.4 kV Dyn5 ASEA"
- ext_grid (1 elements)
```
import pandapower.networks as pn

net1 = pn.create_kerber_landnetz_kabel_1()

This pandapower network includes the following parameter tables:
- load (8 elements) p_load_in_kw=8, q_load_in_kw=0
- bus (18 elements)
- line (16 elements) std_type="NAYY 150", std_type_branchout_line="NAYY 50"
- trafo (1 elements) std_type = "0.125 MVA 10/0.4 kV Dyn5 ASEA"
- ext_grid (1 elements)

net2 = pn.create_kerber_landnetz_kabel_2()

This pandapower network includes the following parameter tables:
- load (14 elements) p_load_in_kw=8, q_load_in_kw=0
- bus (30 elements)
- line (28 elements) std_type="NAYY 150", std_type_branchout_line="NAYY 50"
- trafo (1 elements) std_type="0.125 MVA 10/0.4 kV Dyn5 ASEA"
- ext_grid (1 elements)
8.6.3 Kerber Dorfnetz

```python
import pandapower.networks as pn
net = pn.create_kerber_dorfnetz()
'''
This pandapower network includes the following parameter tables:
  - load (57 elements) p_load_in_kw=6, q_load_in_kw=0
  - bus (116 elements)
  - line (114 elements) std_type="NAYY 150"; std_type_branchout_line="NAYY 50"
  - trafo (1 elements) std_type="0.4 MVA 10/0.4 kV Yyn6 4 ASEA"
  - ext_grid (1 elements)
'''
```
8.6.4 Kerber Vorstadtnetze

```python
import pandapower.networks as pn

net1 = pn.create_kerber_vorstadtnetz_kabel_1()

This pandapower network includes the following parameter tables:
- load (146 elements) p_load_in_kw=2, q_load_in_kw=0
- bus (294 elements)
- line (292 elements) std_type="NAYY 150", std_type_branchout_line_1="NAYY 50", std_type_branchout_line_2="NYY 35"
- trafo (1 elements) std_type="0.63 MVA 20/0.4 kV Yyn6 wnr ASEA"
- ext_grid (1 elements)
```
import pandapower.networks as pn

net2 = pn.create_kerber_vorstadtnetz_kabel_2()

This pandapower network includes the following parameter tables:
- load (144 elements) \( p_{\text{load in kw}}=2, \quad q_{\text{load in kw}}=0 \)
- bus (290 elements)
- line (288 elements) \( \text{std type}=\text{"NAYY 150"}, \quad \text{std type branchout line 1}=\text{"NAYY 50"}, \quad \text{std type branchout line 2}=\text{"NYY 35"} \)
- trafo (1 elements) "\text{std type}=0.63 \text{ MVA 20/0.4 kV } \text{Yyn6 } \text{wnr } \text{ASEA}" 
- ext_grid (1 elements)
The typical kerber networks represent the most common low-voltage distribution grids. To produce statements of universal validity or check limit value, a significant part of all existing grids have to be involved. The following grids obtain special builds of parameters (very high line length, great number of branches or high loaded transformers). These parameters results in high loaded lines and low voltage magnitudes within the extreme network. By including the extreme networks, kerber reached the 95% confidence interval.

Therefore 95% of all parameter results in an considered distribution grid are equal or better compared to the outcomes from kerber extreme networks. Besides testing for extreme parameters you are able to check for functional capability of reactive power control. Since more rare network combination exist, the total number of extreme grids is higher than the amount of typical kerber networks.

See also:

- Georg Kerber, Aufnahmefähigkeit von Niederspannungsverteilnetzen für die Einspeisung aus Photovoltaikkleinanlagen, Dissertation
- Georg Kerber, Statistische Analyse von NS-Verteilungsnetzen und Modellierung von Referenznetzen

<table>
<thead>
<tr>
<th>Kerber Landnetze</th>
<th>Lines</th>
<th>Total Length</th>
<th>Loads</th>
<th>Installed Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freileitung 1</td>
<td>26</td>
<td>0.312 km</td>
<td>26</td>
<td>208 kW</td>
</tr>
<tr>
<td>Freileitung 2</td>
<td>27</td>
<td>0.348 km</td>
<td>27</td>
<td>216 kW</td>
</tr>
<tr>
<td>Kabel 1</td>
<td>52</td>
<td>1.339 km</td>
<td>26</td>
<td>208 kW</td>
</tr>
<tr>
<td>Kabel 2</td>
<td>54</td>
<td>1.435 km</td>
<td>27</td>
<td>216 kW</td>
</tr>
<tr>
<td>Lines</td>
<td>Total Length</td>
<td>Loads</td>
<td>Installed Power</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>-------</td>
<td>-----------------</td>
<td></td>
</tr>
<tr>
<td><strong>Kerber Dorfnetze</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kabel 1</td>
<td>116</td>
<td>3.088 km</td>
<td>58</td>
<td>348 kW</td>
</tr>
<tr>
<td>Kabel 2</td>
<td>234</td>
<td>6.094 km</td>
<td>117</td>
<td>702 kW</td>
</tr>
<tr>
<td><strong>Vorstadtnetze</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kabel_a Type 1</td>
<td>290</td>
<td>3.296 km</td>
<td>145</td>
<td>290 kW</td>
</tr>
<tr>
<td>Kabel_b Type 1</td>
<td>290</td>
<td>4.019 km</td>
<td>145</td>
<td>290 kW</td>
</tr>
<tr>
<td>Kabel_c Type 2</td>
<td>382</td>
<td>5.256 km</td>
<td>191</td>
<td>382 kW</td>
</tr>
<tr>
<td>Kabel_d Type 2</td>
<td>384</td>
<td>5.329 km</td>
<td>192</td>
<td>384 kW</td>
</tr>
</tbody>
</table>

The Kerber extreme networks are categorized into two groups:

**Type I**: Kerber networks with extreme lines

**Type II**: Kerber networks with extreme lines and high loaded transformer

**Note**: Note that all Kerber extreme networks (no matter what type / territory) consist of various branches, linetypes or line length.

---

**8.6.6 Extreme Kerber Landnetze**

```python
import pandapower.networks as pn

'''Extrem Landnetz Freileitung Typ I'''
net = pn.kb_extrem_landnetz_freileitung()

'''Extrem Landnetz Kabel Typ I'''
net = pn.kb_extrem_landnetz_kabel()
```
import pandapower.networks as pn

'''Extrem Landnetz Freileitung Typ II'''
net = pn.kb_extrem_landnetz_freileitung_trafo()

'''Extrem Landnetz Kabel Typ II'''
net = pn.kb_extrem_landnetz_kabel_trafo()
8.6.7 Extreme Kerber Dorfnetze

```python
import pandapower.networks as pn

"""Extrem Dorfnetz Kabel Typ I"
net = pn.kb_extrem_dorfnetz()
```
import pandapower.networks as pn

"""Extrem Dorfnetz Kabel Typ II"
net = pn.kb_extrem_dorfnetz_trafo()
8.6.8 Extreme Kerber Vorstadnetze

```python
import pandapower.networks as pn

'''Extrem Vorstadtnetz Kabel_a Typ I'''
et = pn.kb_extrem_vorstadtnetz_1()
```
Extrem Vorstadtnetz Kabel_a Typ I

```python
import pandapower.networks as pn

'''Extrem Vorstadtnetz Kabel_b Typ I'''
net = pn.kb_extrem_vorstadtnetz_2()

Extrem Vorstadtnetz Kabel_b Typ I
```
import pandapower.networks as pn

'''Extrem Vorstadtnetz Kabel_c Typ II'''
net = pn.kb_extrem_vorstadtnetz_trafo_1()

---

import pandapower.networks as pn

'''Extrem Vorstadtnetz Kabel_d Typ II'''
net = pn.kb_extrem_vorstadtnetz_trafo_2()
8.7 Synthetic Voltage Control LV Networks

This function creates a LV network from M. Lindner, C. Aigner, R. Witzmann, F. Wirtz, I. Berber, M. Gödde and R. Frings. “Aktuelle Musternetze zur Untersuchung von Spannungsproblemen in der Niederspannung”. 14. Symposium Energieinnovation TU Graz. 2014 which are representative, synthetic grids for voltage control analysis. According to Lindner the household loads are 5.1 kW and the special loads are 7.9 kW. The user is suggested to assume load distribution and load profile generation.

**OPTIONAL:**

`network_class` (str, ‘rural_1’) - specify which type of network will be created. Must be in ['rural_1', 'rural_2', 'village_1', 'village_2', 'suburb_1'].

**OUTPUT:**

`net` - returns the required synthetic voltage control lv network

**EXAMPLE:**

```python
import pandapower.networks as nw
net = nw.create_synthetic_voltage_control_lv_network()
```
**network_class: "rural_1"**

**line types: cables**

<table>
<thead>
<tr>
<th>Position</th>
<th>active Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strang 2 / NVP 1</td>
<td>6.9 kW</td>
</tr>
<tr>
<td>Strang 2 / NVP 2</td>
<td>15.3 kW</td>
</tr>
<tr>
<td>Strang 2 / NVP 4</td>
<td>29.6 kW</td>
</tr>
<tr>
<td>Strang 3 / NVP 4</td>
<td>15.8 kW</td>
</tr>
<tr>
<td>Strang 3 / NVP 5</td>
<td>25.3 kW</td>
</tr>
</tbody>
</table>

**Special Loads**

Strang 2 / NVP 4
Strang 3 / NVP 2

**Decentral Energy Resources**

<table>
<thead>
<tr>
<th>Position</th>
<th>active Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strang 2 / NVP 1</td>
<td>6.9 kW</td>
</tr>
<tr>
<td>Strang 2 / NVP 2</td>
<td>15.3 kW</td>
</tr>
<tr>
<td>Strang 2 / NVP 4</td>
<td>29.6 kW</td>
</tr>
<tr>
<td>Strang 3 / NVP 4</td>
<td>15.8 kW</td>
</tr>
<tr>
<td>Strang 3 / NVP 5</td>
<td>25.3 kW</td>
</tr>
</tbody>
</table>
network_class: "rural_2"  
line types: OHL dominated

<table>
<thead>
<tr>
<th>Special Loads</th>
<th>Decentral Energy Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>Position</td>
</tr>
<tr>
<td>Strang 1 / NVP 1</td>
<td>Strang 1 / NVP 1</td>
</tr>
<tr>
<td>Strang 2 / NVP 3</td>
<td>Strang 2 / NVP 3</td>
</tr>
<tr>
<td>Strang 3 / NVP 2</td>
<td>Strang 3 / NVP 2</td>
</tr>
<tr>
<td></td>
<td>Strang 3 / NVP 3</td>
</tr>
</tbody>
</table>
### Special Loads

<table>
<thead>
<tr>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

### Decentral Energy Resources

<table>
<thead>
<tr>
<th>Position</th>
<th>active Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strang 1 / NVP 6</td>
<td>29.8 kW</td>
</tr>
<tr>
<td>Strang 1 / NVP 2</td>
<td>16 kW</td>
</tr>
<tr>
<td>Strang 1 / NVP 3</td>
<td>4.6 kW</td>
</tr>
<tr>
<td>Strang 1 / NVP 6</td>
<td>19 kW</td>
</tr>
<tr>
<td>Strang 1 / NVP 8</td>
<td>29 kW</td>
</tr>
<tr>
<td>Strang 2 / NVP 1</td>
<td>16 kW</td>
</tr>
<tr>
<td>Strang 2 / NVP 2</td>
<td>5.2 kW</td>
</tr>
<tr>
<td>Strang 2 / NVP 3</td>
<td>19 kW</td>
</tr>
<tr>
<td>Strang 2 / NVP 5</td>
<td>12 kW</td>
</tr>
<tr>
<td>Strang 2 / NVP 10</td>
<td>10 kW</td>
</tr>
<tr>
<td>Strang 2 / NVP 12</td>
<td>8 kW</td>
</tr>
<tr>
<td>Strang 3 / NVP 1</td>
<td>12.63 kW</td>
</tr>
<tr>
<td>Strang 3 / NVP 2</td>
<td>30 kW</td>
</tr>
<tr>
<td>Strang 4 / NVP 3</td>
<td>10 kW</td>
</tr>
<tr>
<td>Strang 4 / NVP 4</td>
<td>33 kW</td>
</tr>
<tr>
<td>Strang 4 / NVP 10</td>
<td>8 kW</td>
</tr>
</tbody>
</table>
8.8 Dickert LV Networks

```python
pandapower.networks.create_dickert_lv_network(feeders_range='short',
linetype='cable',
customer='single',
case='good',
trafo_type_name='0.4 MVA 20/0.4 kV',
trafo_type_data=None)
```

This function creates a LV network from J. Dickert, M. Domagk and P. Schegner. “Benchmark low voltage distribution networks based on cluster analysis of actual grid properties”. PowerTech, 2013 IEEE Grenoble. This LV network will have one to three feeders connected to MV-LV-Trafo. To connect more feeders with respect to the optional given parameters ‘feeders_range’, ‘linetype’, ‘customer’ and ‘case’, the ‘create_dickert_lv_feeders’ function can be executed. The given ‘preferred lines for feeders’ are used, knowing that there are some other standard types mentioned as well.

Since the paper focusses on LV grids structure, load powers and MV connection are neglected, so that the user should identify appropriate assumptions for trafo and load parameters. ‘trafo_type_name’ and ‘trafo_type_data’ can be set directly in this function. By default, the load powers are calculated with coincidence factor, derived with normal distributed peak system demand, described in Dickert, Schegner - ‘Residential Load Models for Network Planning Purposes’, Modern Electric Power Systems 2010, Wroclaw, Poland, with the given example assumptions:

- $c_{\inf} = 0.1$
- $P_{\text{max}1} = 10 \text{ kW}$
- powerfactor = 0.95 ind. (in range of 0.9 to 1)

OPTIONAL:
feeders_range (str, ‘short’) - feeder length, which can be (‘short’, ‘middle’, ‘long’)
linetype (str, ‘cable’) - the are different feeders provided for ‘cable’ or ‘C&OHL’
customer (str, ‘single’) - type of customers (‘single’ or ‘multiple’) supplied by the feeders
case (str, ‘good’) - case of supply mission, which can be (‘good’, ‘average’, ‘bad’)
trafo_type_name (str, ‘0.4 MVA 20/0.4 kV’) - name of the HV-MV-Trafo standard type
trafo_type_data (dict, None) - if ‘trafo_type_name’ is not in pandapower standard types, the data of this new trafo types must be given here in pandapower trafo type way

OUTPUT:
net (pandapowerNet) - Returns the required dickert lv network

EXAMPLE:
import pandapower.networks as pn
net = pn.create_dickert_lv_network()

For all given Dickert LV Networks (in numbers: 12) the number of delivery points of tapped line and the distance between delivery points is given in this table:

<table>
<thead>
<tr>
<th>Feeders range</th>
<th>Line type</th>
<th>Customer</th>
<th>Case</th>
<th>d_DP</th>
<th>n_DP</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
<td>cable</td>
<td>single</td>
<td>good</td>
<td>60</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>average</td>
<td>120</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>bad</td>
<td>80</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>multiple</td>
<td>good</td>
<td>80</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>average</td>
<td>50</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>bad</td>
<td>40</td>
<td>10</td>
</tr>
<tr>
<td>middle</td>
<td>cable</td>
<td>multiple</td>
<td>good</td>
<td>40</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>average</td>
<td>35</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>bad</td>
<td>30</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C&amp;OHL</td>
<td>good</td>
<td>50</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>average</td>
<td>45</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>bad</td>
<td>40</td>
<td>16</td>
</tr>
<tr>
<td>long</td>
<td>cable</td>
<td>multiple</td>
<td>good</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>average</td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>bad</td>
<td>30</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C&amp;OHL</td>
<td>good</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>average</td>
<td>40</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>bad</td>
<td>40</td>
<td>40</td>
</tr>
</tbody>
</table>

The next figure shows the topology of the paper’s example with lv network with mid-range, cable type and in good case:
Figure 10. Example of benchmark LV-feeders for mid-range cable
9 Plotting Networks

pandapower includes enables plotting networks with two plotting packages: Matplotlib and Plotly.

9.1 Matplotlib Network Plots

pandapower provides the functionality to translate pandapower network elements into matplotlib collections. The different collections for lines, buses or transformers can than be drawn with pyplot.

If no coordinates are available for the buses, pandapower provides possibility to create generic coordinates through the igraph package. If no geocoordinates are available for the lines, they can be plotted as direct connections between the buses.

9.1.1 Simple Plotting

The function simple_plot() can be used for simple plotting. For advanced possibilities see the tutorials

```python
pandapower.plotting.simple_plot(net, respect_switches=False, line_width=1.0, bus_size=1.0, ext_grid_size=1.0, scale_size=True, bus_color='b', line_color='grey', trafo_color='g', ext_grid_color='y', library='igraph')
```

Plots a pandapower network as simple as possible. If no geodata is available, artificial geodata is generated. For advanced plotting see the tutorial

**INPUT:** net - The pandapower format network.

**OPTIONAL:** respect_switches (bool, False) - Respect switches if artificial geodata is created

- line_width (float, 1.0) - width of lines
- bus_size (float, 1.0) - Relative size of buses to plot.
  - The value bus_size is multiplied with mean_distance_between_buses, which equals the distance between the max geocoord and the min divided by 200. mean_distance_between_buses = sum((net['bus_geodata'].max() • net['bus_geodata'].min()) / 200)
  - net['bus_geodata'].min()) / 200
- ext_grid_size (float, 1.0) - Relative size of ext_grids to plot.
  - See bus sizes for details. Note: ext_grids are plottet as rectangles
- scale_size (bool, True) - Flag if bus_size and ext_grid_size will be scaled with respect to grid mean distances
- bus_color (String, colors[0]) - Bus Color. Init as first value of color palette. Usually colors[0] = “b”.
- line_color (String, ‘grey’) - Line Color. Init is grey
- trafo_color (String, ‘g’) - Trafo Color. Init is green
- ext_grid_color (String, ‘y’) - External Grid Color. Init is yellow

9.1.2 Create Collections

Matplotlib collections can be created from pandapower networks with the following functions:
9.1.3 Bus Collections

```python
pandapower.plotting.create_bus_collection(net, buses=None, size=5, marker='o',
patch_type='circle', colors=None, z=None,
cmap=None, norm=None, infofunc=None,
picker=False, bus_geodata=None, **kwargs)
```

Creates a matplotlib patch collection of pandapower buses.

**Input:** `net` (pandapowerNet) - The pandapower network

**OPTIONAL:** `buses` (list, None) - The buses for which the collections are created. If None, all buses in the network are considered.

- `size` (int, 5) - patch size
- `marker` (str, “o”) - patch marker
- `patch_type` (str, “circle”) - patch type, can be
  - “circle” for a circle
  - “rect” for a rectangle
  - “poly<n>” for a polygon with n edges
- `infofunc` (function, None) - infofunction for the patch element
- `colors` (list, None) - list of colors for every element
- `cmap` - colormap for the patch colors
- `bus_geodata` (DataFrame, None) - coordinates to use for plotting If None, net["bus_geodata"] is used
- `picker` - picker argument passed to the patch collection
- **kwargs - key word arguments are passed to the patch function

9.1.4 Branch Collections

```python
pandapower.plotting.create_line_collection(net, lines=None, line_geodata=None,
use_bus_geodata=False, infofunc=None,
cmap=None, norm=None, picker=False, z=None, cbar_title='Line Loading [%]',
**kwargs)
```

Creates a matplotlib line collection of pandapower lines.

**Input:** `net` (pandapowerNet) - The pandapower network

**OPTIONAL:** `lines` (list, None) - The lines for which the collections are created. If None, all lines in the network are considered.

- `line_geodata` (DataFrame, None) - coordinates to use for plotting If None, net["line_geodata"] is used
- `infofunc` (function, None) - infofunction for the patch element
- **kwargs - key word arguments are passed to the patch function

```python
pandapower.plotting.create_trafo_collection(net, trafos=None, bus_geodata=None,
**kwargs)
```

Creates a matplotlib line collection of pandapower transformers.

**Input:** `net` (pandapowerNet) - The pandapower network

**OPTIONAL:** `trafos` (list, None) - The transformers for which the collections are created. If None, all transformers in the network are considered.

**kwargs - key word arguments are passed to the patch function
9.1.5 Create Colormaps

9.1.6 Discrete

pandapower.plotting.cmap_discrete(cmap_list)
Can be used to create a discrete colormap.

**INPUT:**

- cmap_list (list) - list of tuples, where each tuple represents one range. Each tuple has the form of ((from, to), color).

**OUTPUT:**

- cmap - matplotlib colormap
- norm - matplotlib norm object

**EXAMPLE:**

```python
>>> from pandapower.plotting import cmap_discrete, create_line_trace, draw_traces
>>> cmap_list = [((20, 50), "green"), ((50, 70), "yellow"), ((70, 100), "red")]
>>> cmap, norm = cmap_discrete(cmap_list)
>>> lc = create_line_trace(net, cmap=cmap, norm=norm)
>>> draw_traces([lc])
```

9.1.7 Continuous

pandapower.plotting.cmap_continuous(cmap_list)
Can be used to create a continuous colormap.

**INPUT:**

- cmap_list (list) - list of tuples, where each tuple represents one color. Each tuple has the form of (center, color). The colorbar is a linear segmentation of the colors between the centers.

**OUTPUT:**

- cmap - matplotlib colormap
- norm - matplotlib norm object

**EXAMPLE:**

```python
>>> from pandapower.plotting import cmap_continuous, create_bus_trace, draw_traces
>>> cmap_list = [(0.97, "blue"), (1.0, "green"), (1.03, "red")]
>>> cmap, norm = cmap_continuous(cmap_list)
>>> bc = create_bus_trace(net, cmap=cmap, norm=norm)
>>> draw_traces([bc])
```

9.1.8 Draw Collections

pandapower.plotting.draw_collections(collections, figsize=(10, 8), ax=None, plot_colorbars=True, set_aspect=True)
Draws matplotlib collections which can be created with the create collection functions.

**Input:**
- collections (list) - iterable of collection objects

**OPTIONAL:**
- figsize (tuple, (10,8)) - figsize of the matplotlib figure
- ax (axis, None) - matplotlib axis object to plot into, new axis is created if None
9.1.9 Generic Coordinates

If there are no geocoordinates in a network, generic coordinates can be created. There are two possibilities:

- with python-igraph: http://igraph.org/python/ (recommended)
- with networkx and graphviz (http://www.graphviz.org)

Generically created geocoordinates can then be plotted in the same way as real geocoordinates.

```
pandapower.plotting.create_generic_coordinates(net, mg=None, library='igraph', respect_switches=False)
```

This function will add arbitrary geo-coordinates for all buses based on an analysis of branches and rings. It will remove out of service buses/lines from the net. The coordinates will be created either by igraph or by using networkx library.

**INPUT:** net - pandapower network

**OPTIONAL:** mg - Existing networkx multigraph, if available. Convenience to save computation time.

library - “igraph” to use igraph package or “networkx” to use networkx package

**OUTPUT:** net - pandapower network with added geo coordinates for the buses

**EXAMPLE:** net = create_generic_coordinates(net)

Example plot with mv_oberrhein network from the pandapower.networks package as geographical plan (respect_switches=False):
and as structural plan (respect_switches=True):
9.2 Plotly Network Plots

pandapower provides interactive network plots using Plotly. These plots are built with arguments and functionalities to be as much as possible analogous with pandapower’s matplotlib plotting library. There is a functionality to translate pandapower network elements into plotly collections (traces). The different collections for lines, buses or transformers can than be drawn.

In order to get idea about interactive plot features and possibilities see the tutorial.

If a network has geocoordinates, there is a possibility to represent interactive plots on Mapbox maps.

**Note:** Plots on Mapbox maps are available only considering you have a Mapbox account and a Mapbox Access Token. After getting a mabox token it can be set to pandapower as the following
9 PLOTTING NETWORKS

9.2.1 Built-in plot functions

In order to get idea about interactive plot features and possibilities see the tutorial.

9.2.2 Simple Plotting

The function simple_plotly() can be used for a simple interactive plotting.

```python
from pandapower.plotting.plotly.mapbox_plot import set_mapbox_token
set_mapbox_token('<token>')
```

```python
pandapower.plotting.plotly.simple_plotly(net, respect_switches=True,
use_line_geodata=None, on_map=False,
projection=None, map_style='basic',
figsize=1, aspectratio='auto',
line_width=1, bus_size=10,
ext_grid_size=20.0, bus_color='blue',
line_color='grey', trafo_color='green',
ext_grid_color='yellow')
```

Plots a pandapower network as simple as possible in plotly. If no geodata is available, artificial geodata is generated. For advanced plotting see the tutorial

**INPUT:** net - The pandapower format network. If none is provided, mv_oberrhein() will be plotted as an example

**OPTIONAL:** respect_switches (bool, True) - Respect switches when artificial geodata is created

use_line_geodata* (bool, True) - defines if lines patches are based on net.line_geodata of the lines (True) or on net.bus_geodata of the connected buses (False)

on_map (bool, False) - enables using mapbox plot in plotly. If provided geodata are not real geo-coordinates in lon/lat form, on_map will be set to False.

projection (String, None) - defines a projection from which network geo-data will be transformed to lat-long. For each projection a string can be found at [http://spatialreference.org/ref/epsg/](http://spatialreference.org/ref/epsg/)

map_style (str, ‘basic’) - enables using mapbox plot in plotly

- ‘streets’
- ‘bright’
- ‘light’
- ‘dark’
- ‘satellite’

figsize (float, 1) - aspectratio is multiplied by it in order to get final image size

aspectratio (tuple, ‘auto’) - when ‘auto’ it preserves original aspect ratio of the network geodata any custom aspectration can be given as a tuple, e.g. (1.2, 1)

line_width (float, 1.0) - width of lines

bus_size (float, 10.0) - size of buses to plot.

ext_grid_size (float, 20.0) - size of ext_grids to plot.

See bus sizes for details. Note: ext_grids are plotted as rectangles

bus_color (String, "blue") - Bus Color. Init as first value of color palette.

line_color (String, ‘grey’) - Line Color. Init is grey
**trafo_color** (String, ‘green’) - Trafo Color. Init is green

**ext_grid_color** (String, ‘yellow’) - External Grid Color. Init is yellow

Example plot with `mv_oberrhein` network from the pandapower.networks package:

```python
from pandapower.plotting.plotly import simple_plotly
from pandapower.networks import mv_oberrhein
net = mv_oberrhein()
simple_plotly(net)
```

Example simple plot on a map:

```python
net = mv_oberrhein()
simple_plotly(net, on_map=True, projection='epsg:31467')
```
9.2.3 Network coloring according to voltage levels

The function vlevel_plotly() is used to plot a network colored and labeled according to voltage levels.

```python
pandapower.plotting.plotly.vlevel_plotly(net, respect_switches=True,
                                        use_line_geodata=None, colors_dict=None,
                                        on_map=False, projection=None,
                                        map_style='basic', figsize=1, aspectratio='auto',
                                        line_width=2, bus_size=10)
```

Plots a pandapower network in plotly using lines/buses colors according to the voltage level they belong to.

If no geodata is available, artificial geodata is generated. For advanced plotting see the tutorial.

**INPUT:** net - The pandapower format network. If none is provided, mv_oberrhein() will be plotted as an example

**OPTIONAL:** respect_switches (bool, True) - Respect switches when artificial geodata is created

- use_line_geodata (bool, True) - defines if lines patches are based on net.line_geodata of the lines (True) or on net.bus_geodata of the connected buses (False)

- colors_dict* (dict, None) - dictionary for customization of colors for each voltage level in the form:
  ```python
dictionary = {
    'voltage_kv1': 'color1',
    'voltage_kv2': 'color2',
    ...}
```

- on_map (bool, False) - enables using mapbox plot in plotly If provided geodata are not real geo-coordinates in lon/lat form, on_map will be set to False.

- projection (String, None) - defines a projection from which network geo-data will be transformed to lat-long. For each projection a string can be found at http://spatialreference.org/ref/epsg/

- map_style (str, 'basic') - enables using mapbox plot in plotly
  ```
  • 'streets'
  • 'bright'
  • 'light'
  • 'dark'
  • 'satellite'
  ```

- figsize (float, 1) - aspectratio is multiplied by it in order to get final image size

- aspectratio (tuple, ‘auto’) - when ‘auto’ it preserves original aspect ratio of the network geodata any custom aspectration can be given as a tuple, e.g. (1.2, 1)

- line_width (float, 1.0) - width of lines

- bus_size (float, 10.0) - size of buses to plot.
Example plot with mv_oberrhein network from the pandapower.networks package:

```python
from pandapower.plotting.plotly import vlevel_plotly
from pandapower.networks import mv_oberrhein
net = mv_oberrhein()
vlevel_plotly(net)
```

9.2.4 Power Flow results

The function `pf_res_plotly()` is used to plot a network according to power flow results where a colormap is used to represent line loading and voltage magnitudes. For advanced possibilities see the tutorials.

```python
pandapower.plotting.plotly.pf_res_plotly(net, cmap='Jet', use_line_geodata=None,
on_map=False, projection=None,
map_style='basic', figsize=1, aspect_ratio='auto', line_width=2,
bus_size=10)
```

Plots a pandapower network in plotly using colormap for coloring lines according to line loading and buses according to voltage in p.u. If no geodata is available, artificial geodata is generated. For advanced plotting see the tutorial

**INPUT:** net - The pandapower format network. If none is provided, mv_oberrhein() will be plotted as an example

**OPTIONAL:** respect_switches (bool, False) - Respect switches when artificial geodata is created

- `cmap` (str, True) - name of the colormap
- `colors_dict` (dict, None) - by default 6 basic colors from default color palette is used. Otherwise, user can define a dictionary in the form: voltage_kV : color
- `on_map` (bool, False) - enables using mapbox plot in plotly If provided geodata are not real geo-coordinates in lon/lat form, on_map will be set to False.
- `projection` (String, None) - defines a projection from which network geo-data will be transformed to lat-long. For each projection a string can be found at http://spatialreference.org/ref/epsg/

**map_style** (str, ‘basic’) - enables using mapbox plot in plotly
- 'streets'
- 'bright'
- 'light'
- 'dark'
• ‘satellite’

**figsize** (float, 1) - aspect ratio is multiplied by it in order to get final image size

**aspectratio** (tuple, ‘auto’) - when ‘auto’ it preserves original aspect ratio of the network geodata any custom aspect ratio can be given as a tuple, e.g. (1.2, 1)

**line_width** (float, 1.0) - width of lines

**bus_size** (float, 10.0) - size of buses to plot.

Example power flow results plot:

```python
from pandapower.plotting.plotly import pf_res_plotly
from pandapower.networks import mv_oberrhein
net = mv_oberrhein()
pf_res_plotly(net)
```

Power flow results on a map:

```python
net = mv_oberrhein()
pf_res_plotly(net, on_map=True, projection='epsg:31467', map_style='dark')
```
9.2.5 Create & Draw Traces

Plotly traces can be created from pandapower networks with the following functions.

9.2.6 Bus Traces

```python
pandapower.plotting.plotly.create_bus_trace(net, buses=None, size=5, patch_type='circle', color='blue', infofunc=None, trace_name='buses', legendgroup=None, cmap=None, cmap_vals=None, cbar_title=None, cmin=None, cmax=None)
```

Creates a plotly trace of pandapower buses.

**INPUT:** net (pandapowerNet) - The pandapower network

**OPTIONAL:** buses (list, None) - The buses for which the collections are created. If None, all buses in the network are considered.

  - size (int, 5) - patch size
  - patch_type (str, “circle”) - patch type, can be
    - “circle” for a circle
    - “square” for a rectangle
    - “diamond” for a diamond
    - much more pathc types at https://plot.ly/python/reference/#scatter-marker
  - infofunc (list, None) - hoverinfo for each trace element
  - trace_name (String, “buses”) - name of the trace which will appear in the legend
  - color (String, “blue”) - color of buses in the trace
  - cmap (String, None) - name of a colormap which exists within plotly (Greys, YIGnBu, Greens, YlOrRd, Bluered, RdBu, Reds, Blues, Picnic, Rainbow, Portland, Jet, Hot, Blackbody, Earth, Electric, Viridis) alternatively a custom discrete colormap can be used
  - cmap_vals (list, None) - values used for coloring using colormap
  - cbar_title (String, None) - title for the colorbar
  - cmin (float, None) - colorbar range minimum
  - cmax (float, None) - colorbar range maximum

9.2.7 Branch Traces

```python
pandapower.plotting.plotly.create_line_trace(net, lines=None, use_line_geodata=True, respect_switches=False, width=1.0, color='grey', infofunc=None, trace_name='lines', legendgroup=None, cmap=None, cbar_title=None, show_colorbar=True, cmap_vals=None, cmin=None, cmax=None)
```

Creates a plotly trace of pandapower lines.

**INPUT:** net (pandapowerNet) - The pandapower network
9 PLOTTING NETWORKS

**OPTIONAL:**  
*lines* (list, None) - The lines for which the collections are created. If None, all lines in the network are considered.

*width* (int, 1) - line width

*respect_switches* (bool, False) - flag for consideration of disconnected lines

*infofunc* (list, None) - hoverinfo for each line

*trace_name* (String, "lines") - name of the trace which will appear in the legend

*color* (String, "grey") - color of lines in the trace

*legendgroup* (String, None) - defines groups of layers that will be displayed in a legend e.g. groups according to voltage level (as used in `vlevel_plotly`)

*colormap* (String, None) - name of a colormap which exists within plotly if set to True default *Jet* colormap is used, alternative colormaps : Greys, YlGnBu, Greens, YlOrRd, Bluered, RdBu, Reds, Blues, Picnic, Rainbow, Portland, Jet, Hot, Blackbody, Earth, Electric, Viridis

*show_colorbar* (bool, False) - flag for showing or not corresponding colorbar

*show_colorbar* (bool, False) - flag for showing or not corresponding colorbar

*pandapower.plotting.plotly.create_trafo_trace(net, trafos=None, color='green', width=5, infofunc=None, cmap=None, trace_name='trafos', cmin=None, cmax=None, cmap_vals=None)*

creates a plotly trace of pandapower trafos.

**INPUT:**  
*net* (pandapowerNet) - The pandapower network

**OPTIONAL:**  
*trafos* (list, None) - The trafos for which the collections are created. If None, all trafos in the network are considered.

*width* (int, 5) - line width

*infofunc* (list, None) - hoverinfo for each line

*trace_name* (String, "lines") - name of the trace which will appear in the legend

*color* (String, "green") - color of lines in the trace

*colormap* (bool, False) - name of a colormap which exists within plotly (Greys, YlGnBu, Greens, YlOrRd, Bluered, RdBu, Reds, Blues, Picnic, Rainbow, Portland, Jet, Hot, Blackbody, Earth, Electric, Viridis)

*show_colorbar* (bool, False) - flag for showing or not corresponding colorbar

*pandapower.plotting.plotly.draw_traces(traces, on_map=False, map_style='basic', showlegend=True, figsize=1, aspectratio='auto')*

plots all the traces (which can be created using `create_bus_trace()`, `create_line_trace()`, `create_trafo_trace()`) to PLOTLY (see https://plot.ly/python/)

9.2.8 Draw Traces
INPUT: traces - list of dicts which correspond to plotly traces generated using: create_bus_trace, create_line_trace, create_trafo_trace

OPTIONAL: on_map (bool, False) - enables using mapbox plot in plotly
map_style (str, 'basic') - enables using mapbox plot in plotly
  • 'streets'
  • 'bright'
  • 'light'
  • 'dark'
  • 'satellite'
showlegend (bool, 'True') - enables legend display
figsize (float, 1) - aspectratio is multiplied by it in order to get final image size
aspectratio (tuple, 'auto') - when 'auto' it preserves original aspect ratio of the network geodata any custom aspectration can be given as a tuple, e.g. (1.2, 1)

9.2.9 Transforming network geodata from any projection to lat/long

In case network geodata are not in The World Geodetic System (WGS84), that is latitude/longitude format, but in some of the map-projections, it may be converted to lat/long by providing name of the projection (in the form 'epsg:<projection_number>' according to spatialreference). A sample of converting geodata from mv_oberrhein network can be found in the tutorial.

pandapower.plotting.plotly.geo_data_to_latlong (net, projection)
Transforms network’s geodata (in net.bus_geodata and net.line_geodata) from specified projection to lat/long (WGS84).

INPUT: net (pandapowerNet) - The pandapower network
projection (String) - projection from which geodata are transformed to lat/long. some examples
  • "epsg:31467" - 3-degree Gauss-Kruger zone 3
  • "epsg:2032" - NAD27(CGQ77) / UTM zone 18N
  • "epsg:2190" - Azores Oriental 1940 / UTM zone 26N

9.3 HTML

pandapower.plotting.to_html (net, filename, respect_switches=True, include_lines=True, include_trafos=True, show_tables=True)
Saves a pandapower Network to an html file.

INPUT: net (dict) - The pandapower format network
filename (string) - The absolute or relative path to the input file.

OPTIONAL:

respect_switches (boolean, True) - True: open line switches are being considered
  (no edge between nodes)
False: open line switches are being ignored
include_lines (boolean, True) - determines, whether lines get converted to edges
include_trafos (boolean, True) - determines, whether trafos get converted to edges
show_tables (boolean, True) - shows pandapower element tables
## 10 Save and Load Networks

### 10.1 pickle

**pandapower.to_pickle** *(net, filename)*

Saves a pandapower Network with the pickle library.

**INPUT:**
- `net` (dict) - The pandapower format network
- `filename` (string) - The absolute or relative path to the output file or an writable file-like object

**EXAMPLE:**

```python
>>> pp.to_pickle(net, os.path.join("C:\", "example_folder", "example1.p"))  # absolute path
>>> pp.to_pickle(net, "example2.p")  # relative path
```

**pandapower.from_pickle** *(filename, convert=True)*

Load a pandapower format Network from pickle file

**INPUT:**
- `filename` (string or file) - The absolute or relative path to the input file or file-like object

**OUTPUT:**
- `net` (dict) - The pandapower format network

**EXAMPLE:**

```python
>>> net1 = pp.from_pickle(os.path.join("C:\", "example_folder", "example1.p"))  # absolute path
>>> net2 = pp.from_pickle("example2.p")  # relative path
```

### 10.2 Excel

**pandapower.to_excel** *(net, filename, include_empty_tables=False, include_results=True)*

Saves a pandapower Network to an excel file.

**INPUT:**
- `net` (dict) - The pandapower format network
- `filename` (string) - The absolute or relative path to the output file

**OPTIONAL:**
- `include_empty_tables` (bool, False) - empty element tables are saved as excel sheet
- `include_results` (bool, True) - results are included in the excel sheet

**EXAMPLE:**

```python
>>> pp.to_excel(net, os.path.join("C:\", "example_folder", "example1.xlsx"))  # absolute path
>>> pp.to_excel(net, "example2.xlsx")  # relative path
```

**pandapower.from_excel** *(filename, convert=True)*

Load a pandapower network from an excel file

**INPUT:**
- `filename` (string) - The absolute or relative path to the input file.

**OUTPUT:**
- `convert` (bool) - use the convert format function to

**INPUT:**
- `convert` (bool) - use the convert format function to

**EXAMPLE:**

```python
>>> net1 = pp.from_excel(os.path.join("C:\", "example_folder", "example1.xlsx"))  # absolute path
>>> net2 = pp.from_excel("example2.xlsx")  # relative path
```
10.3 Json

```
pandapower.to_json(net, filename=None)
```
Saves a pandapower Network in JSON format. The index columns of all pandas DataFrames will be saved in ascending order. net elements which name begins with “_” (internal elements) will not be saved. Std types will also not be saved.

**INPUT:** net (dict) - The pandapower format network

```
filename (string or file) - The absolute or relative path to the output file or file-like object
```

**EXAMPLE:**
```
>>> pp.to_json(net, "example.json")
```

```
pandapower.from_json(filename, convert=True)
```
Load a pandapower network from a JSON file. The index of the returned network is not necessarily in the same order as the original network. Index columns of all pandas DataFrames are sorted in ascending order.

**INPUT:** filename (string or file) - The absolute or relative path to the input file or file-like object

**OUTPUT:** convert (bool) - use the convert format function to

```
net (dict) - The pandapower format network
```

**EXAMPLE:**
```
>>> net = pp.from_json("example.json")
```

10.4 SQL

```
pandapower.to_sqlite(net, filename)
pandapower.from_sqlite(filename, netname="")
```
11 Converter

Pandapower provides some very useful converters which enable an exchange of network data with other Power System analysis tools. These tools are:

11.1 PYPOWER

The following functions are provided to enable a network data exchange with PYPOWER.

```python
pandapower.converter.from_ppc(ppc, f_hz=50, validate_conversion=False)
```

This function converts pypower case files to pandapower net structure.

**INPUT:**
- `ppc`: The pypower case file.

**OPTIONAL:**
- `f_hz` (float, 50) - The frequency of the network.
- `validate_conversion` (bool, False) - If True, validate_from_ppc is run after conversion.

**OUTPUT:**
- `net`: pandapower net.

**EXAMPLE:**

```python
import pandapower.converter as cv
from pypower import case4gs
ppc_net = case4gs.case4gs()
pp_net = cv.from_ppc(ppc_net, f_hz=60)
```

```python
pandapower.converter.validate_from_ppc(ppc_net, pp_net)
```

This function validates the pypower case files to pandapower net structure conversion via a comparison of loadflow calculations.

**INPUT:**
- `ppc_net` - The pypower case file which already contains the pypower powerflow results.
- `pp_net` - The pandapower network.

**OPTIONAL:**
- `max_diff_values` - Dict of maximal allowed difference values. The keys must be ‘vm_pu’, ‘va_degree’, ‘p_branch_kw’, ‘q_branch_kvar’, ‘p_gen_kw’ and ‘q_gen_kvar’ and the values floats.

**OUTPUT:**
- `conversion_success` - conversion_success is returned as False if pypower or pandapower cannot calculate a powerflow or if the maximum difference values (max_diff_values ) cannot be hold.

**EXAMPLE:**

```python
import pandapower.converter as cv
pp_net = cv.from_ppc(ppc_net, f_hz=50)
conversion_success = cv.validate_from_ppc(ppc_net, pp_net)
```
NOTE:

The user has to take care that the loadflow results already are included in the provided ppc_net.

```
pandapower.converter.to_ppc(net, calculate_voltage_angles=False, trafo_model='t',
                          r_switch=0.0, check_connectivity=True, voltage_dependLoads=True, init='results')
```

This function converts a pandapower net to a pypower case file.

INPUT:

net - The pandapower net.

OPTIONAL:

calculate_voltage_angles (bool, False) - consider voltage angles in loadflow calculation

If True, voltage angles of ext_grids and transformer shifts are considered in the loadflow calculation. Considering the voltage angles is only necessary in meshed networks that are usually found in higher networks.

trafo_model (str, “t”) - transformer equivalent circuit model pandapower provides two equivalent circuit models for the transformer:

- “t” - transformer is modeled as equivalent with the T-model.
- “pi” - transformer is modeled as equivalent PI-model. This is not recommended, since it is less exact than the T-model. It is only recommended for validation with other software that uses the pi-model.

r_switch (float, 0.0) - resistance of bus-bus-switches. If impedance is zero, buses connected by a closed bus-bus switch are fused to model an ideal bus. Otherwise, they are modelled as branches with resistance r_switch.

check_connectivity (bool, True) - Perform an extra connectivity test after the conversion from pandapower to PYPOWER

If True, an extra connectivity test based on SciPy Compressed Sparse Graph Routines is performed. If check finds unsupplied buses, they are set out of service in the ppc

voltage_depend_loads (bool, True) - consideration of voltage-dependent loads. If False, net.load.const_z_percent and net.load.const_i_percent are not considered, i.e. net.load.p_kw and net.load.q_kvar are considered as constant-power loads.

init (str, “results”) - initialization method of the converter pandapower ppc converter supports two methods for initializing the converter:

- “flat” - flat start with voltage of 1.0pu and angle of 0° at all PQ-buses and 0° for PV buses as initial solution
- “results” - voltage vector from net.res_bus is used as initial solution.

OUTPUT:

ppc - The Pypower casefile for usage with pypower

EXAMPLE:

```
import pandapower.converter as pc
import pandapower.networks as pn
net = pn.case9()
ppc = pc.to_ppc(net)
```
11.2 MATPOWER

To communicate to MATPOWER to exchange network data these functions are available.

```python
classConverterFactory

from mpc_file, f_hz=50, casename_mpc_file='mpc', validate_conversion=False)
This function converts a matpower case file (.mat) version 2 to a pandapower net.

import pandapower as pp
pp_net = cv.from_mpc('case9.mat', f_hz=60)
```

INPUT:

<table>
<thead>
<tr>
<th><strong>mpc_file</strong></th>
<th>path to a matpower case file (.mat).</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>f_hz</strong></td>
<td>(int, 50) - The frequency of the network.</td>
</tr>
<tr>
<td><strong>casename_mpc_file</strong></td>
<td>(str, ‘mpc’) - If mpc_file does not contain the arrays “gen”, “branch” and “bus” it will use the sub-struct casename_mpc_file</td>
</tr>
</tbody>
</table>

OUTPUT:

| **net** | The pandapower network |

EXAMPLE:

```python
import pandapower as pp
pp_net = cv.from_mpc('case9.mat', f_hz=60)
```

```python
class Converter

from mpc_file, net, filename=None, init='results', calculate_voltage_angles=False, trafo_model='t', mode='pf')
This function converts a pandapower net to a matpower case files (.mat) version 2. Note: python is 0-based while Matlab is 1-based.

import pandapower as pp
pp_net = cv.to_mpc(net, filename=None, init='results', calculate_voltage_angles=False, trafo_model='t', mode='pf')
```

INPUT:

| **net** | The pandapower net. |

OPTIONAL:

| **filename** | (None) - File path + name of the mat file which will be created. If None the mpc will only be returned |
| **init** | (str, “results”) - initialization method of the loadflow For the conversion to a mpc, the following options can be chosen: |
| **calculate_voltage_angles** | (bool, False) - copy the voltage angles from pandapower to the mpc |
| **trafo_model** | (str, “t”) - transformer equivalent circuit model pandapower provides two equivalent circuit models for the transformer: |

Note: python is 0-based while Matlab is 1-based.
• “t” - transformer is modelled as equivalent with the T-model. This is consistent with Power-Factory and is also more accurate than the PI-model. We recommend using this transformer model.

• “pi” - transformer is modelled as equivalent PI-model. This is consistent with Sincal, but the method is questionable since the transformer is physically T-shaped. We therefore recommend the use of the T-model.

EXAMPLE:

```python
import pandapower.converter as pc
import pandapower.networks as pn
net = pn.case9()
pc.to_mpc(net)
```
12 Toolbox

The pandapower toolbox is a collection of helper functions that are implemented for the pandapower framework. It is designed for functions of common application that fit nowhere else. Have a look at the available functions to save yourself the effort of maybe implementing something twice. If you develop some functionality which could be interesting to other users as well and do not fit into one of the specialized packages, feel welcome to add your contribution. To improve overview functions are loosely grouped by functionality, please adhere to this notion when adding your own functions and feel free to open new groups as needed.

**Note:** If you implement a function that might be useful for others, it is mandatory to add a short docstring to make browsing the toolbox practical. Ideally further comments if appropriate and a reference of authorship should be added as well.

### 12.1 Result Information

**pandapower.lf_info**(net, numv=1, numi=2)
Prints some basic information of the results in a net (max/min voltage, max trafo load, max line load).

**OPTIONAL:**
- numv (integer, 1) - maximal number of printed maximal respectively minimal voltages
- numi (integer, 2) - maximal number of printed maximal loading at trafos or lines

**pandapower.opf_task**(net)
Prints some basic information of the optimal powerflow task.

**pandapower.switch_info**(net, sidx)
Prints what buses and elements are connected by a certain switch.

**pandapower.overloaded_lines**(net, max_load=100)
Returns the results for all lines with loading_percent > max_load or None, if there are none.

**pandapower.violated_buses**(net, min_vm_pu, max_vm_pu)
Returns all bus indices where vm_pu is not within min_vm_pu and max_vm_pu or returns None, if there are none of those buses.

**pandapower.nets_equal**(x, y, check_only_results=False, tol=1e-14)
Compares the DataFrames of two networks. The networks are considered equal if they share the same keys and values, except of the ‘et’ (elapsed time) entry which differs depending on runtime conditions and entries stating with ‘_’.

### 12.2 Simulation Setup and Preparation

**pandapower.convert_format**(net)
Converts old nets to new format to ensure consistency. The converted net is returned.

**pandapower.add_zones_to_elements**(net, elements=[‘line’, ‘trafo’, ‘ext_grid’, ‘switch’])
Adds zones to elements, inferring them from the zones of buses they are connected to.

**pandapower.create_continuous_bus_index**(net, start=0)
Creates a continuous bus index starting at zero and replaces all references of old indices by the new ones.

**pandapower.set_scaling_by_type**(net, scalings, scale_load=True, scale_sgen=True)
Sets scaling of loads and/or sgens according to a dictionary mapping type to a scaling factor. Note that the type-string is case sensitive. E.g. scaling = {“pv”: 0.8, “bhkw”: 0.6}
• scalings – A dictionary containing a mapping from element type to
• scale_load –
• scale_sgen –

12.3 Topology Modification

pandapower.set_isolated_areas_out_of_service(net)
Set all isolated buses and all elements connected to isolated buses out of service.

pandapower.drop_inactive_elements(net)
Drops any elements not in service AND any elements connected to inactive buses.

pandapower.drop_buses(net, buses)
Drops buses and by default safely drops all elements connected to them as well.

pandapower.drop Trafford(net, Trafford)
Deletes all Trafford and in the given list of indices and removes any switches connected to it.

pandapower.drop_lines(net, Lines)
Deletes all lines and their geodata in the given list of indices and removes any switches connected to it.

pandapower.fuse_buses(net, b1, b2, drop=True)
Reroutes any connections to buses in b2 to the given bus b1. Additionally drops the buses b2, if drop=True (default).

pandapower.set_element_status(net, buses, in_service)
Sets buses and all elements connected to them in or out of service.

pandapower.select_subnet(net, buses, include_switch_buses=False, include_results=False, keep_everything_else=False)
Selects a subnet by a list of bus indices and returns a net with all elements connected to them.

pandapower.close_switch_at_line_with_two_open_switches(net)
Finds lines that have opened switches at both ends and closes one of them. Function is usually used when optimizing section points to prevent the algorithm from ignoring isolated lines.

12.4 Item/Element Selection

pandapower.get_element_index(net, element, name, exact_match=True)
Returns the element(s) identified by a name or regex and its element-table.

INPUT: net - pandapower network
element - Table to get indices from ("line", "bus", "trafo" etc.)
name - Name of the element to match.

OPTIONAL:
exact_match (boolean, True) - True: Expects exactly one match, raises
UserWarning otherwise.
False: returns all indices matching the name/pattern

OUTPUT: index - The indices of matching element(s).

pandapower.next_bus(net, bus, element_id, et='line', **kwargs)
Returns the index of the second bus an element is connected to, given a first one. E.g. the from_bus given the to_bus of a line.

pandapower.get_connected_elements(net, element, buses, respect_switches=True, respect_in_service=False)
Returns elements connected to a given bus.
**INPUT:** net (pandapowerNet)
  
  `element` (string, name of the element table)
  
  `buses` (single integer or iterable of ints)
  
**OPTIONAL:**
  
  `respect_switches` (boolean, True) - True: open switches will be respected  
  False: open switches will be ignored

  `respect_in_service` (boolean, False) - True: in_service status of connected lines will be respected  
  False: in_service status will be ignored
  
**OUTPUT:** `connected_elements` (set) - Returns connected elements.
  
`pandapower.get_connected_buses(net, buses, consider=('l', 's', 't'), respect_switches=True, respect_in_service=False)`
  
Returns buses connected to given buses. The source buses will NOT be returned.

**INPUT:** net (pandapowerNet)
  
  `buses` (single integer or iterable of ints)
  
**OPTIONAL:**
  
  `respect_switches` (boolean, True) - True: open switches will be respected  
  False: open switches will be ignored

  `respect_in_service` (boolean, False) - True: in_service status of connected buses will be respected  
  False: in_service status will be ignored

  `consider` (iterable, (“l”, “s”, “t”)) - Determines, which types of connections will be considered. l: lines s: switches t: trafos
  
**OUTPUT:** `cl` (set) - Returns connected buses.
  
`pandapower.get_connected_switches(net, buses, consider=('b', 'l', 't'), status='all')`
  
Returns switches connected to given buses.

**INPUT:** net (pandapowerNet)
  
  `buses` (single integer or iterable of ints)
  
**OPTIONAL:**
  
  `respect_switches` (boolean, True) - True: open switches will be respected  
  False: open switches will be ignored

  `consider` (iterable, (“l”, “s”, “t”)) - Determines, which types of connections will be considered. l: lines s: switches t: trafos

  `status` (string, (“all”, “closed”, “open”)) - Determines, which switches will be considered
  
**OUTPUT:** `cl` (set) - Returns connected buses.