TECHNICAL REPORT

pandapower

- Convenient Power System Modelling and Analysis

based on PYPOWER and pandas -

Fraunhofer IWES
Universität Kassel

March 22, 2017

Version 1.2.2

Energiemanagement und Betrieb elektrischer Netze

Department of Energy Management and Power System Operation

University of Kassel, Germany

www.uni-kassel.de/eecs/e2n

Fraunhofer IWES

Fraunhofer Institute for Wind Energy and Energy System Technology (IWES)

Department for Distribution System Operation

Kassel, Germany

www.iwes.fraunhofer.de

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

Further Contributions by:
Tobias Deß
Bastian Junker
Jannis Kupka
Lothar Löwer
Jan Ulffers
Nils Bornhorst
Jonathan Schütt
Elisabeth Drayer
Jakov Krstulović Opara

Coordination:
Martin Braun
Johann-Christian Töbermann
Stefan Gehler

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

Contents

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 lenghts or standard types, canot 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

- automized 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

- RMS simulation

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 automized 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 specialiced 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 possiblities 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

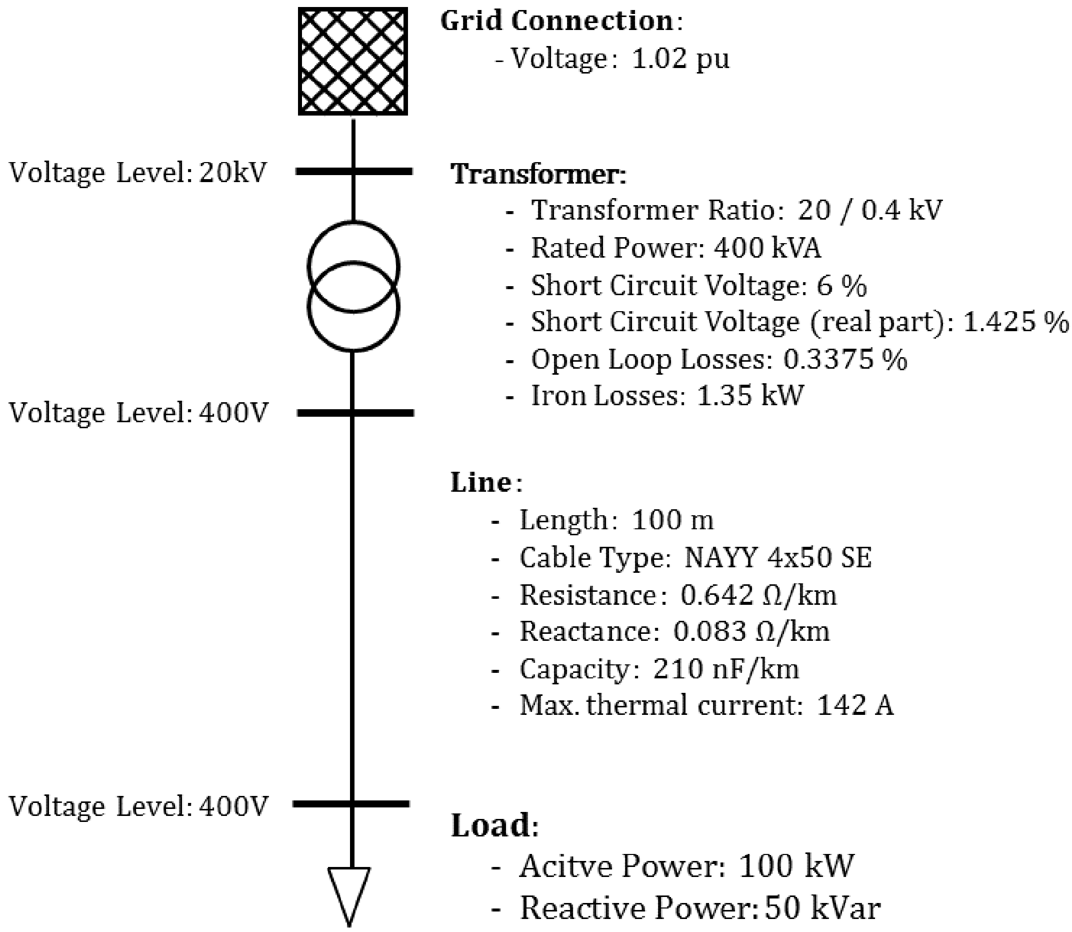
1.3 A Short Introduction

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:

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

we then create the buses with the given voltage levels:

```
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:

```
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:

```
tid = pp.create_transformer_from_parameters(net, sn_kva=400.,
                                            hv_bus=b1, lv_bus=b2,
                                            vn_hv_kv=20., vn_lv_kv=0.4,
                                            vsc_percent=6., vscr_percent=1.425,
                                            i0_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:

```
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:

```
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:

```
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:

PandapowerNet

bus

index	name	vn_kv	type	in_service
0	Bus 1	20.0	b	True
1	Bus 2	0.4	b	True
2	Bus 3	0.4	b	True

trafo

index	name	std_type	hv_bus	lv_bus	sn_kva	vn_hv_kv	vn_lv_kv	vsc_percent	vscr_percent	pfe_kw	i0_percent	shift_degree	tp_side	tp_min	tp_mid	tp_max	tp_pos	in_service
0	Trafo	0.4 MVA 20/0.4	0	1	400	20.0	0.4	6.0	1.425	1.35	0.3375	150	hv	0	-2	2	0	True

line

index	name	std_type	from_bus	to_bus	length_km	r_ohm_per_km	x_ohm_per_km	c_nf_per_km	imax_ka	df	parallel	type	in_service
0	Line 1	NAYY 4x50 SE	1	2	0.1	0.642	0.083	210	0.142	1.0	1	cs	True

load

index	name	bus	p_kw	q_kvar	sn_kva	scaling	in_service
0	Load	2	100.0	50.0	NaN	1.0	True

ext_grid

index	name	bus	vm_pu	va_degree	in_service
0	Grid Connection	0	1.02	0.0	True

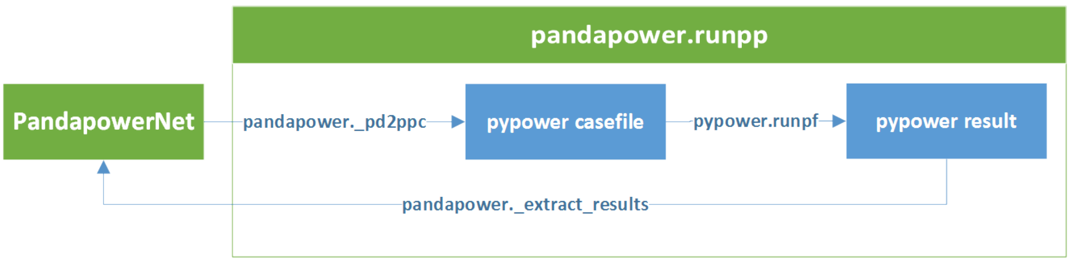
This is the version where transformer and line have been created through the standard type libraries, which is why the line has a specified type (cs for cable system) and the transformer has a tap changer, both of which are defined in the type data.

Running a Power Flow

A powerflow can be carried out with the runpp function:

```
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:

PandapowerNet

res_bus

index	vm_pu	va_degree	p_kw	q_kvar
0	1.0200	0.000	-107.27	-52.68
1	1.0088	-0.76	0.00	0.00
2	0.9644	0.12	100.00	50.00

res_trafo

index	p_from_kw	q_from_kvar	p_to_kw	q_to_kvar	pl_kw	ql_kvar	i_lv_ka	i_hv_ka	loading_percent
0	107.265	52.675	-105.392	-50.696	1.873	1.979	0.003	0.167	29.29

res_line

index	p_from_kw	q_from_kvar	p_to_kw	q_to_kvar	pl_kw	ql_kvar	i_ka	loading_percent
0	105.39	50.70	-100.00	-50.00	5.39	0.70	0.167	117.84

res_load

index	p_kw	q_kvar
0	100.00	50.00

res_ext_grid

index	p_kw	q_kvar
0	-107.27	-52.68

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>_<unit>, such as:

Parameter	read as
vm_pu	$v_m[pu]$
loading_percent	$loading[\%]$
pl_kw	$p_l[kw]$
r_ohm_per_km	$r[\Omega/km]$

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

Parameter	read as
min_vm_pu	$v_m^{min}[pu]$
max_loading_percent	$loading^{max}[\%]$
max_p_kw	$p^{max}[kw]$
min_q_kvar	$q^{min}[kvar]$

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 $S = \sqrt{3} \cdot V \cdot I$.

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.

Internally, pandapower calculates with a nominal apparent power of $S_N = 1MVA$ for the per unit system, which however should not be relevant for the user 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 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

pandapower is tested with pytest. There are currently over 100 tests testing all kinds of pandapower functionality.

The complete test suite can be run with:

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

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

pandapower is tested with python 2.7:

```
============================= test session starts =============================
platform win32 -- Python 2.7.12, pytest-3.0.5, py-1.4.32, pluggy-0.4.0
collected 139 items

..\..\Documents\pandapower\pandapower\test\api\test_auxiliary.py .
..\..\Documents\pandapower\pandapower\test\api\test_diagnostic.py ................
↪...
..\..\Documents\pandapower\pandapower\test\api\test_file_io.py ..
..\..\Documents\pandapower\pandapower\test\api\test_std_types.py .........
..\..\Documents\pandapower\pandapower\test\api\test_toolbox.py .....
..\..\Documents\pandapower\pandapower\test\converter\test_from_ppc.py ....
..\..\Documents\pandapower\pandapower\test\converter\test_to_ppc.py .
..\..\Documents\pandapower\pandapower\test\estimation\test_wls_estimation.py ......
↪.....
..\..\Documents\pandapower\pandapower\test\loadflow\test_results.py ..............
↪...
..\..\Documents\pandapower\pandapower\test\loadflow\test_runpp.py ......
..\..\Documents\pandapower\pandapower\test\loadflow\test_scenarios.py .....
..\..\Documents\pandapower\pandapower\test\networks\test_cigre_networks.py ...
..\..\Documents\pandapower\pandapower\test\networks\test_create_example.py ..
..\..\Documents\pandapower\pandapower\test\networks\test_ieee_cases.py ........
..\..\Documents\pandapower\pandapower\test\networks\test_kerber_networks.py .......
↪..........
```

```
..\..\Documents\pandapower\pandapower\test\networks\test_simple_pandapower_test_
↪networks.py ....
..\..\Documents\pandapower\pandapower\test\opf\test_basic.py ......
..\..\Documents\pandapower\pandapower\test\opf\test_curtailment.py .
..\..\Documents\pandapower\pandapower\test\opf\test_dcline.py ..
..\..\Documents\pandapower\pandapower\test\opf\test_oberrhein.py .
..\..\Documents\pandapower\pandapower\test\shortcircuit\test_line_gen.py ..
..\..\Documents\pandapower\pandapower\test\shortcircuit\test_line_transformer.py ..
↪..
..\..\Documents\pandapower\pandapower\test\shortcircuit\test_meshing_detection.py .
↪...

========================= 139 passed in 88.13 seconds =========================
```

python 3.4:

```
============================= test session starts =============================
platform win32 -- Python 3.4.4, pytest-3.0.5, py-1.4.32, pluggy-0.4.0
hp.pandapower.shortcircuit - WARNING: WARNING: pandapower shortcircuit module is␣
↪in beta stadium, proceed with caution!
collected 139 items

..\..\Documents\pandapower\pandapower\test\api\test_auxiliary.py .
..\..\Documents\pandapower\pandapower\test\api\test_diagnostic.py ................
↪..
..\..\Documents\pandapower\pandapower\test\api\test_file_io.py ..
..\..\Documents\pandapower\pandapower\test\api\test_std_types.py .........
..\..\Documents\pandapower\pandapower\test\api\test_toolbox.py .....
..\..\Documents\pandapower\pandapower\test\converter\test_from_ppc.py ....
..\..\Documents\pandapower\pandapower\test\converter\test_to_ppc.py .
..\..\Documents\pandapower\pandapower\test\estimation\test_wls_estimation.py ......
↪.....
..\..\Documents\pandapower\pandapower\test\loadflow\test_results.py ..............
↪...
..\..\Documents\pandapower\pandapower\test\loadflow\test_runpp.py ......
..\..\Documents\pandapower\pandapower\test\loadflow\test_scenarios.py .....
..\..\Documents\pandapower\pandapower\test\networks\test_cigre_networks.py ...
..\..\Documents\pandapower\pandapower\test\networks\test_create_example.py ..
..\..\Documents\pandapower\pandapower\test\networks\test_ieee_cases.py .........
..\..\Documents\pandapower\pandapower\test\networks\test_kerber_networks.py ......
↪.........
..\..\Documents\pandapower\pandapower\test\networks\test_simple_pandapower_test_
↪networks.py ....
..\..\Documents\pandapower\pandapower\test\opf\test_basic.py ......
..\..\Documents\pandapower\pandapower\test\opf\test_curtailment.py .
..\..\Documents\pandapower\pandapower\test\opf\test_dcline.py ..
..\..\Documents\pandapower\pandapower\test\opf\test_oberrhein.py .
..\..\Documents\pandapower\pandapower\test\shortcircuit\test_line_gen.py ..
..\..\Documents\pandapower\pandapower\test\shortcircuit\test_line_transformer.py ..
↪..
..\..\Documents\pandapower\pandapower\test\shortcircuit\test_meshing_detection.py .
↪...

========================= 139 passed in 92.21 seconds =========================
```

and python 3.5:

```
============================= test session starts =============================
platform win32 -- Python 3.5.1, pytest-3.0.5, py-1.4.31, pluggy-0.4.0
collecting 129 itemshp.pandapower.shortcircuit - WARNING: WARNING: pandapower␣
↪shortcircuit module is in beta stadium, proceed with caution!
collected 139 items
```

```
..\pandapower\test\api\test_auxiliary.py .
..\pandapower\test\api\test_diagnostic.py ....................
..\pandapower\test\api\test_file_io.py ..
..\pandapower\test\api\test_std_types.py .........
..\pandapower\test\api\test_toolbox.py .....
..\pandapower\test\converter\test_from_ppc.py ....
..\pandapower\test\converter\test_to_ppc.py .
..\pandapower\test\estimation\test_wls_estimation.py ..........
..\pandapower\test\loadflow\test_results.py .................
..\pandapower\test\loadflow\test_runpp.py ......
..\pandapower\test\loadflow\test_scenarios.py .....
..\pandapower\test\networks\test_cigre_networks.py ...
..\pandapower\test\networks\test_create_example.py ..
..\pandapower\test\networks\test_ieee_cases.py .........
..\pandapower\test\networks\test_kerber_networks.py ................
..\pandapower\test\networks\test_simple_pandapower_test_networks.py ....
..\pandapower\test\opf\test_basic.py .......
..\pandapower\test\opf\test_curtailment.py .
..\pandapower\test\opf\test_dcline.py ..
..\pandapower\test\opf\test_oberrhein.py .
..\pandapower\test\shortcircuit\test_line_gen.py ..
..\pandapower\test\shortcircuit\test_line_transformer.py ....
..\pandapower\test\shortcircuit\test_meshing_detection.py ....

======================== 139 passed in 72.78 seconds ========================
```

1.5.2 Model and Loadflow Validation

To ensure that pandapower loadflow results are correct, all pandapower element behaviour is tested against DIgSILENT 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:

Parameter	**Max. Deviation**
Voltage Magnitude	0.000001 pu
Voltage Angle	0.01 °
Current	0.000001 kA
Power	0.005 kW
Element Loading	0.001%

1.5.3 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

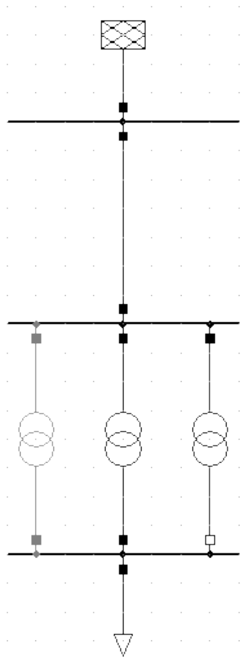
- both iron and copper losses > 0
- nominal voltages that deviate from the nominal bus voltages at both sides
- an active tap changer
- a voltage angle shift > 0

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, on 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:

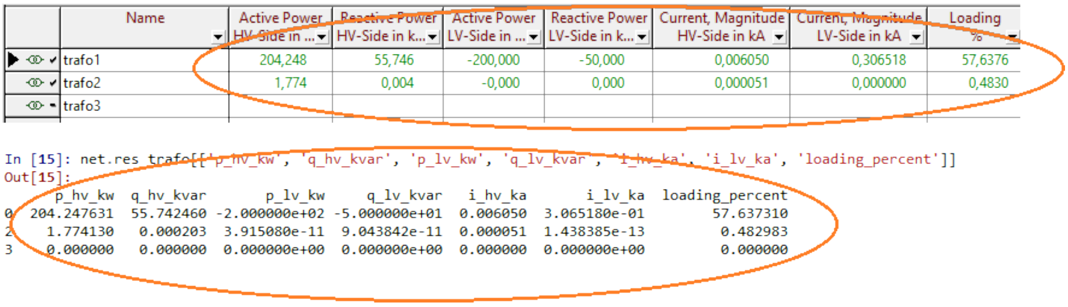


The loadflow result for the exact same network are now compared in pandapower and PowerFactory. It can be seen that both bus voltages:

Name	u, Magnitude p.u.	U, Angle deg
Bus1	1,010000	0,000000
Bus2	1,010150	-0,066861
Bus3	0,970773	-151,416621

```
In [12]: net.res_bus[["vm_pu", "va_degree"]]
Out[12]:
     vm_pu   va_degree
0  1.010000    0.000000
3  1.010150   -0.066862
4  0.970773 -151.416627
```

and transformer results:



	Name	Active Power HV-Side in ...	Reactive Power HV-Side in k...	Active Power LV-Side in ...	Reactive Power LV-Side in k...	Current, Magnitude HV-Side in kA	Current, Magnitude LV-Side in kA	Loading %
	trafo1	204,248	55,746	-200,000	-50,000	0,006050	0,306518	57,6376
	trafo2	1,774	0,004	-0,000	0,000	0,000051	0,000000	0,4830
	trafo3							

```
In [15]: net.res_trafo[['p_hv_kw', 'q_hv_kvar', 'p_lv_kw', 'q_lv_kvar', 'i_hv_ka', 'i_lv_ka', 'loading_percent']]
Out[15]:
     p_hv_kw  q_hv_kvar       p_lv_kw     q_lv_kvar   i_hv_ka       i_lv_ka  loading_percent
0  204.247631  55.742460 -2.000000e+02 -5.000000e+01  0.006050  3.065180e-01        57.637310
2    1.774130   0.000203  3.915080e-11  9.043842e-11  0.000051  1.438385e-13         0.482983
3    0.000000   0.000000  0.000000e+00  0.000000e+00  0.000000  0.000000e+00         0.000000
```

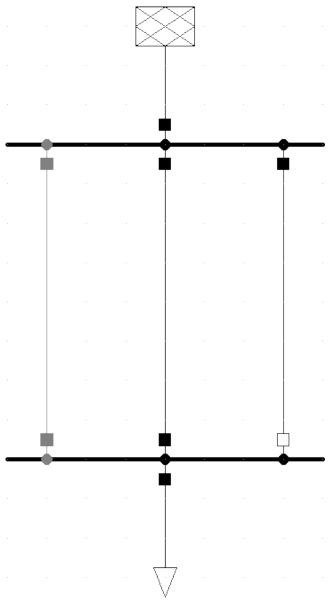
match within the margins defined above.

1.5.4 All Test Networks

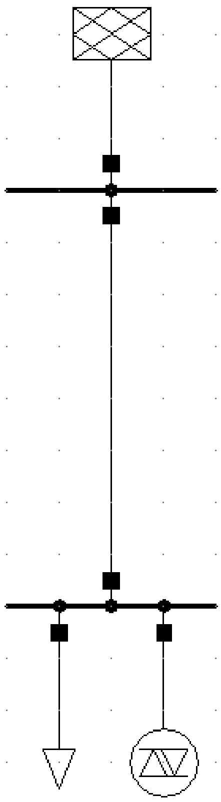
There is a test network for the validation of each pandapower element in the same way the transformer model is tested.

The PowerFactory file containing all test networks can be downloaded `here`. The correlating pandapower networks are defined in result_test_network_generatory.py in the pandapower/test module. The tests that check pandapower results against PowerFactory results are located in pandapower/test/test_results.py.

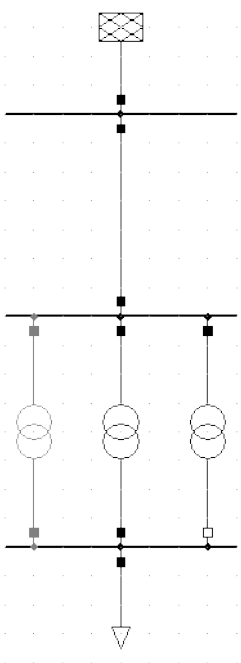
line



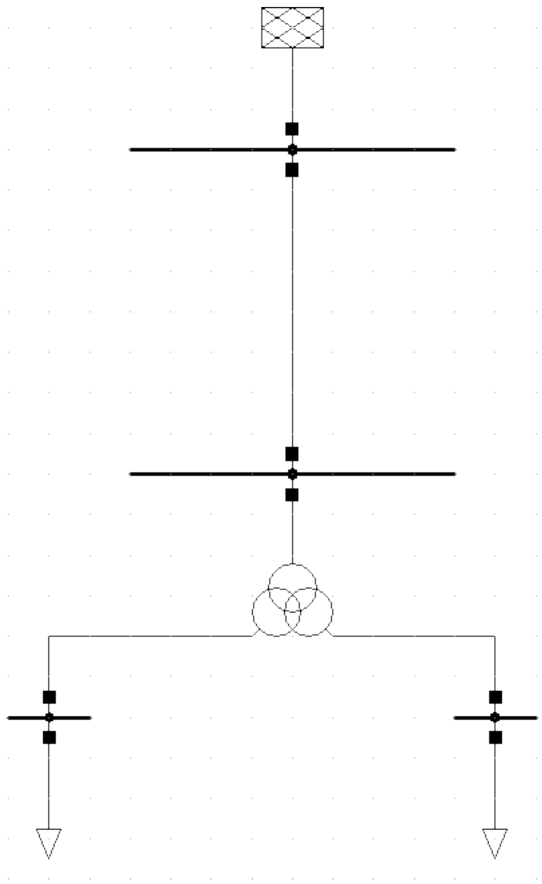
load and sgen



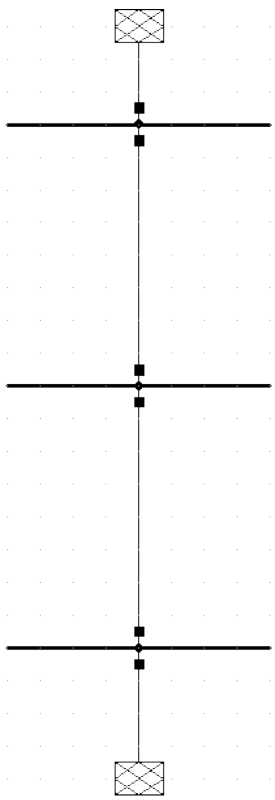
trafo



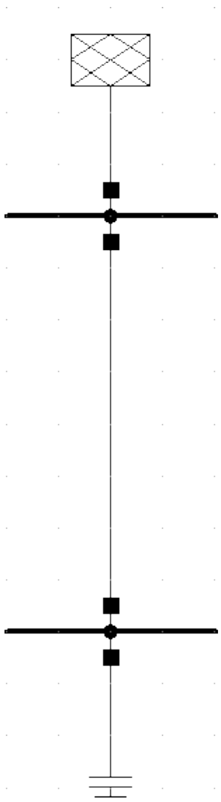
trafo3w



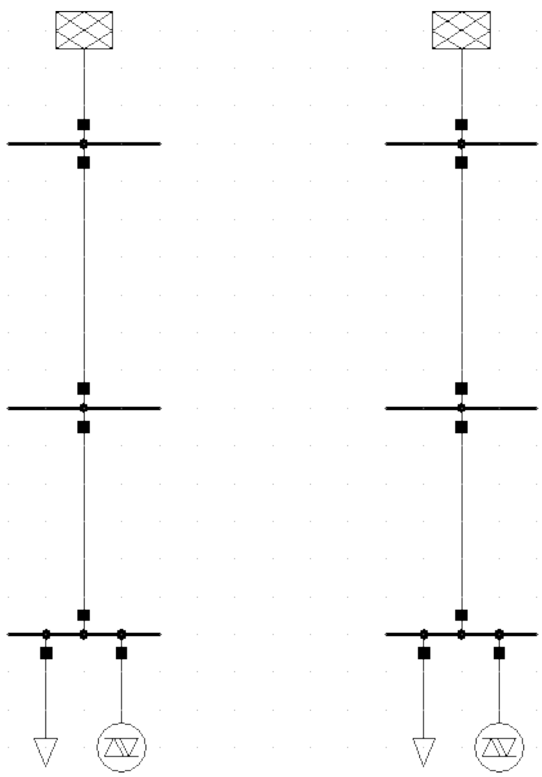
ext_grid



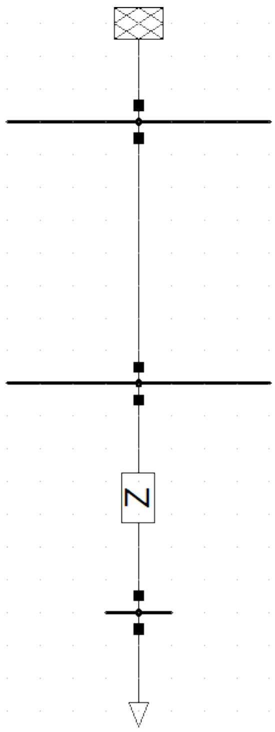
shunt



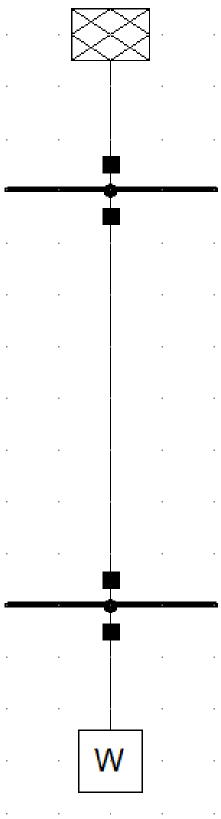
gen



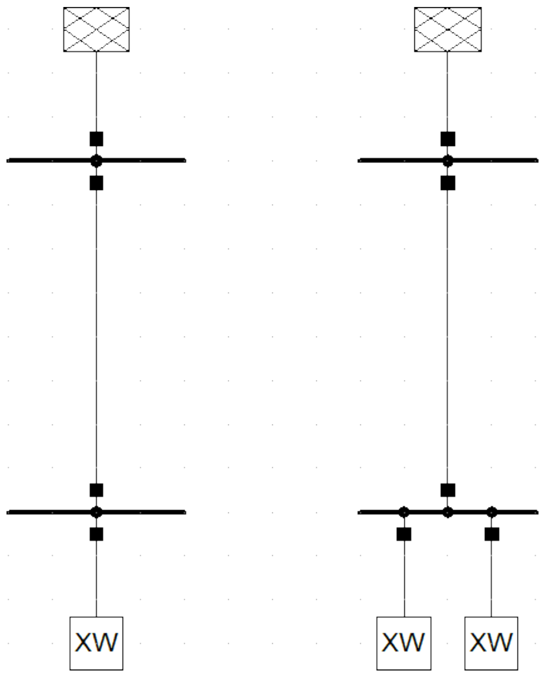
impedance



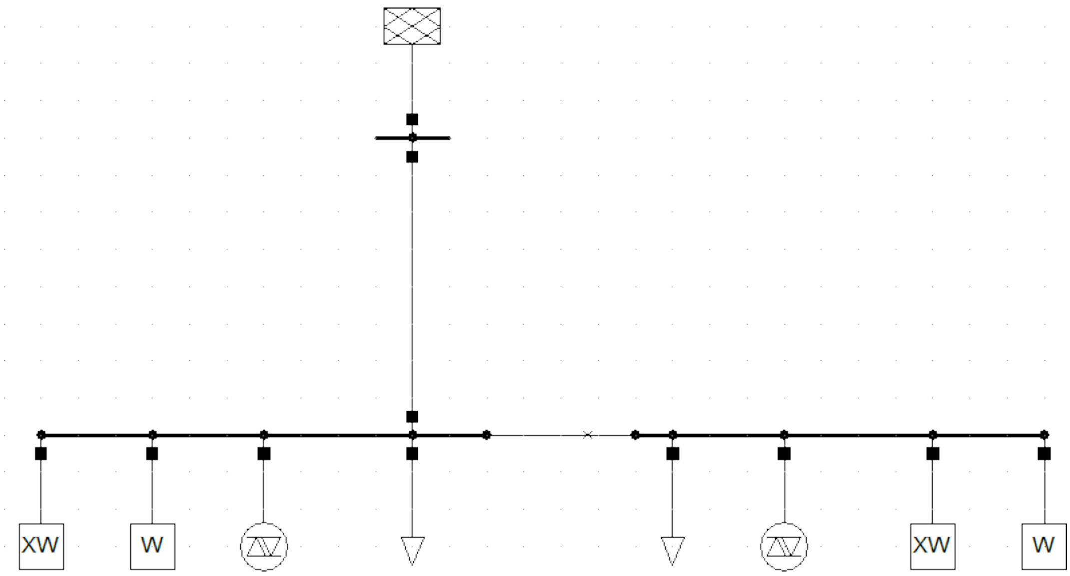
ward



xward



switch



1.6 Change Log

1.6.1 [1.2.2] - 2017-03-22

- [CHANGED] Minor refactoring in pd2ppc
- [ADDED] Technical Report

1.6.2 [1.2.1] - 2017-03-21

- [FIXED] Readme for PyPi

1.6.3 [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, cigre_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

1.6.4 [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

1.6.5 [1.1.0] - 2017-01-11

- [ADDED] impedance element can now be used with unsymetric impedances zij != zji
- [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

1.6.6 [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] supress warnings parameter removed from loadflow, casting warnings are automatically supressed

1.6.7 [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:

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

`pandapower.create_empty_network(name=None, 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: net = create_empty_network()

2.2 Bus

See also:

Unit Systems and Conventions

2.2.1 Create Function

`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, **kwargs)`

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

Busses are the nodes of the network that all other elements connect to.

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

vn_kv (float) - The grid voltage level.

busgeodata ((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

OUTPUT: **eid** (int) - The index of the created element

EXAMPLE: create_bus(net, name = "bus1")

2.2.2 Input Parameters

net.bus

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

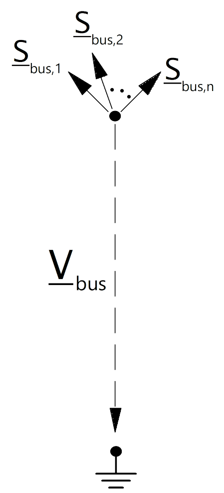
*necessary for executing a power flow calculation **optimal power flow parameter

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

net.bus_geodata

Parameter	Datatype	Explanation
x	float	x coordinate of bus location
y	float	y coordinate of bus location

2.2.3 Electric Model



2.2.4 Result Parameters

net.res_bus

Parameter	Datatype	Explanation
vm_pu	float	voltage magnitude [p.u]
va_degree	float	voltage angle [degree]
p_kw	float	resulting active power demand [kW]
q_kvar	float	resulting reactive power demand [kvar]

The power flow bus results are defined as:

$$
\begin{aligned}
vm\_pu &= |\underline{V}_{bus}| \\
va\_degree &= \angle \underline{V}_{bus} \\
p\_kw &= Re(\sum_{n=1}^{N} \underline{S}_{bus,n}) \\
q\_kvar &= Im(\sum_{n=1}^{N} \underline{S}_{bus,n})
\end{aligned}
$$

net.res_bus_est

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

Parameter	Datatype	Explanation
vm_pu	float	voltage magnitude [p.u]
va_degree	float	voltage angle [degree]
p_kw	float	resulting active power demand [kW]
q_kvar	float	resulting reactive power demand [kvar]

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).

pandapower.**create_line**(*net*, *from_bus*, *to_bus*, *length_km*, *std_type*, *name=None*, *index=None*, *geodata=None*, *df=1.0*, *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) - Force a specified ID if it is available

geodata (array, default None, shape= (,2L)) - The linegeodata 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: **line_id** - The unique line_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”)

`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=1.0, 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) - Force a specified ID if it is available

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

type (str) - type of line (“oh” 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 linegeodata 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

kwargs - nothing to see here, go along

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

OUTPUT: **line_id** - The unique line_id of the created line

EXAMPLE: create_line_from_parameters(net, “line1”, from_bus = 0, to_bus = 1, lenght_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

Parameter	Datatype	Value Range	Explanation
name	string		name of the line
std_type	string		standard type which can be used to easily define line parameters with the pandapower standard type library
from_bus*	integer		Index of bus where the line starts

Parameter	Datatype	Value Range	Explanation
to_bus*	integer		Index of bus where the line ends
length_km*	float	> 0	length of the line [km]
r_ohm_per_km*	float	≥ 0	resistance of the line [Ohm per km]
x_ohm_per_km*	float	≥ 0	inductance of the line [Ohm per km]
c_nf_per_km*	float	≥ 0	capacitance of the line [nF per km]
max_i_ka*	float	> 0	maximal thermal current [kA]
parallel*	integer	≥ 1	number of parallel line systems
df*	float	0...1	derating factor (scaling) for max_i_ka
type	string	Naming conventions: *"ol"* - overhead line *"cs"* - underground cable system	type of line
max_loading_percent**	float	> 0	Maximum loading of the line
endtemp_degree***	float	> 0	Short-Circuit end temperature of the line
in_service*	boolean	True / False	specifies if the line is in service.

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

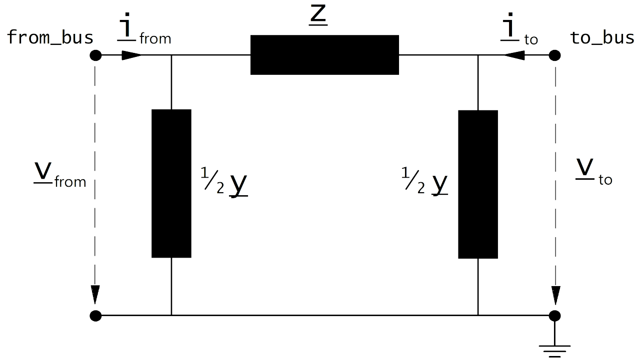
Note: Defining a line with length zero leads to a division by zero in the power flow and is therefore not allowed. Lines with a very low impedance might lead to convergence problems in the power flow for the same reason. If you want to directly connect two buses, please use the switch element instead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

net.line_geodata

Parameter	Datatype	Explanation
coords	list	List of (x,y) tuples that mark the inflexion points of the line

2.3.3 Electric Model

Lines are modelled with the π-equivalent circuit:



The elements in the equivalent circuit are calculated from the parameters in the net.line dataframe as:

$$\underline{Z} = (r\_ohm\_per\_km + j \cdot x\_ohm\_per\_km) \cdot \frac{length\_km}{parallel}$$

$$\underline{Y} = j \cdot 2\pi f \cdot c\_nf\_per\_km \cdot 1 \cdot 10^-9 \cdot length\_km \cdot parallel$$

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:

$$Z_N = \frac{V_N^2}{S_N}$$
$$\underline{z} = \frac{\underline{Z}}{Z_N}$$
$$\underline{y} = \underline{Y} \cdot Z_N$$

Where $S_N = 1\ MVA$ (see *Unit Systems and Conventions*) and U_N is the nominal voltage at the from bus.

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

Parameter	Datatype	Explanation
p_from_kw	float	active power flow into the line at "from" bus [kW]
q_from_kvar	float	reactive power flow into the line at "from" bus [kVar]
p_to_kw	float	active power flow into the line at "to" bus [kW]
q_to_kvar	float	reactive power flow into the line at "to" bus [kVar]
pl_kw	float	active power losses of the line [kW]
ql_kvar	float	reactive power consumption of the line [kVar]
i_from_ka	float	Current at to bus [kA]
i_to_ka	float	Current at from bus [kA]
i_ka	float	Maximum of i_from_ka and i_to_ka [kA]
loading_percent	float	line loading [%]

The power flow results in the net.res_line table are defined as:

$$\begin{aligned}
p_from_kw &= Re(\underline{v}_{from} \cdot \underline{i}^*_{from}) \\
q_from_kvar &= Im(\underline{v}_{from} \cdot \underline{i}^*_{from}) \\
p_to_kw &= Re(\underline{v}_{to} \cdot \underline{i}^*_{to}) \\
q_to_kvar &= Im(\underline{v}_{to} \cdot \underline{i}^*_{to}) \\
pl_kw &= p_from_kw + p_to_kw \\
ql_kvar &= q_from_kvar + q_to_kvar \\
i_from_ka &= i_{from} \\
i_to_ka &= i_{to} \\
i_ka &= max(i_{from}, i_{to}) \\
loading_percent &= \frac{i_ka}{imax_ka \cdot df \cdot parallel} \cdot 100
\end{aligned}$$

net.res_line_est

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

Parameter	Datatype	Explanation
p_from_kw	float	active power flow into the line at "from" bus [kW]
q_from_kvar	float	reactive power flow into the line at "from" bus [kVar]

Parameter	Datatype	Explanation
p_to_kw	float	active power flow into the line at "to" bus [kW]
q_to_kvar	float	reactive power flow into the line at "to" bus [kVar]
pl_kw	float	active power losses of the line [kW]
ql_kvar	float	reactive power consumption of the line [kVar]
i_from_ka	float	Current at to bus [kA]
i_to_ka	float	Current at from bus [kA]
i_ka	float	Maximum of i_from_ka and i_to_ka [kA]
loading_percent	float	line loading [%]

2.4 Switch

2.4.1 Create Function

pandapower.**create_switch**(*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

Parameter	Datatype	Value Range	Explanation
bus*	integer		index of connected bus
name	string		name of the switch

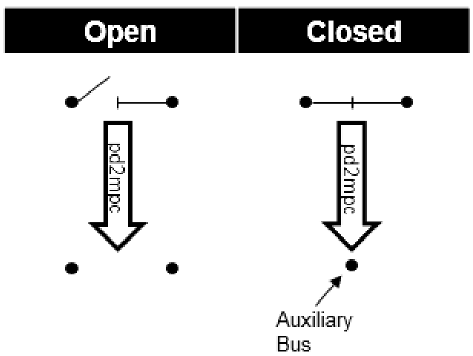
Parameter	Datatype	Value Range	Explanation
element*	integer		index of the element the switch is connected to: - bus index if et = *"b"* - line index if et = *"l"* - trafo index if et = *"t"*
et*	string	*"b"* - bus-bus switch *"l"* - bus-line switch *"t"* - bus-trafo switch	element type the switch connects 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:

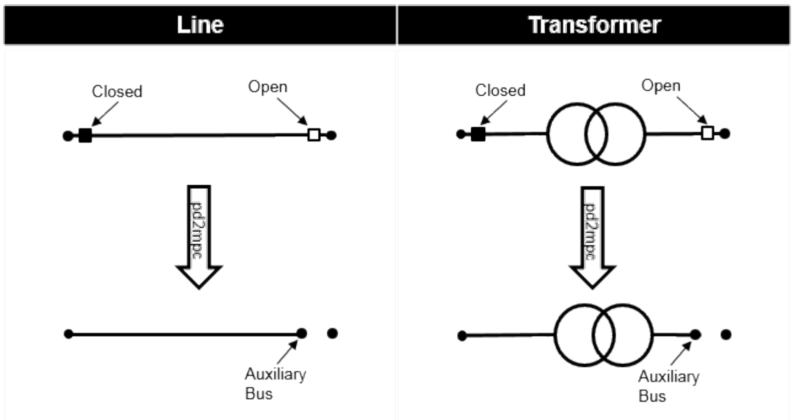
- there is no voltage drop over the switch (ideal switch)
- no convergence problems due to small impedances / large admittances
- less buses in the admittance matrix

Bus-Element-Switches:

When the power flow is calculated internally for every open bus-element switch an auxilary 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

`pandapower.`**`create_load`**(*net, bus, p_kw, q_kvar=0, sn_kva=nan, name=None, scaling=1.0, 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

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

- postive value -> load
- negative value -> generation

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 the specified index. If None, the next highest available index is used

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

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

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

2.5.2 Input Parameters

net.load

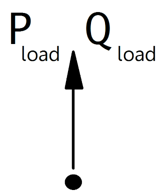
Parameter	Datatype	Value Range	Explanation
name	string		name of the load
bus *	integer		index of connected bus
p_kw*	float	≥ 0	active power of the load [kW]
q_kvar*	float		reactive power of the load [kVar]
sn_kva	float	> 0	rated power of the load [kVA]
scaling *	float	≥ 0	scaling factor for active and reactive power
in_service*	boolean	True / False	specifies if the load is in service.
controllable**	bool		States if load is controllable or not, load will not be used as a flexibilty if it is not controllable

*necessary for executing a power flow calculation.

Note: Loads are not yet respected by the optimal power flow as a flexibility.

2.5.3 Electric Model

Loads are modelled as PQ-buses in the power flow calculation:



The PQ-Values are calculated from the parameter table values as:

$$P_{load} = p\_kw \cdot scaling$$
$$Q_{load} = q\_kvar \cdot scaling$$

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!

2.5.4 Result Parameters

net.res_load

Parameter	Datatype	Explanation
p_kw	float	resulting active power demand after scaling [kW]
q_kvar	float	resulting reactive power demand after scaling [kVar]

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

2.6 Static Generator

See also:

Unit Systems and Conventions

2.6.1 Create Function

pandapower.**create_sgen**(*net, bus, p_kw, q_kvar=0, sn_kva=nan, name=None, index=None, scaling=1.0, type=None, in_service=True, max_p_kw=nan, min_p_kw=nan, max_q_kvar=nan, min_q_kvar=nan, controllable=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

OPTIONAL: p_kw (float, default 0) - The real power of the static generator (negative for generation!)

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 the specified index. If None, the next highest available index is used

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

OUTPUT: index - The unique id of the created sgen

EXAMPLE: create_sgen(net, 1, p_kw = -120)

2.6.2 Input Parameters

net.sgen

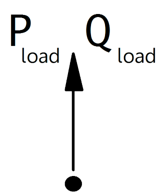
Parameter	Datatype	Value Range	Explanation
name	string		name of the static generator

Parameter	Datatype	Value Range	Explanation
type	string	naming conventions: *"PV"* - photovoltaic system *"WP"* - wind power system *"CHP"* - combined heating and power system	type of generator
bus*	integer		index of connected bus
p_kw*	float	≤ 0	active power of the static generator [kW]
q_kvar*	float		reactive power of the static generator [kVar]
sn_kva	float	> 0	rated power ot the static generator [kVA]
scaling*	float	≥ 0	scaling factor for the active and reactive power
max_p_kw**	float		Maximum active power
min_p_kw**	float		Minimum active power
max_q_kvar**	float		Maximum reactive power
min_q_kvar**	float		Minimum reactive power
controllable**	bool		States if sgen is controllable or not, sgen will not be used as a flexibilty if it is not controllable
in_service*	boolean	True / False	specifies if the generator is in service.

*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:

$$P_{sgen} = p\_kw \cdot scaling$$
$$Q_{sgen} = q\_kvar \cdot scaling$$

Note: Static generators should always have a negative p_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 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!

2.6.4 Result Parameters

net.res_sgen

Parameter	Datatype	Explanation
p_kw	float	resulting active power demand after scaling [kW]
q_kvar	float	resulting reactive power demand after scaling [kVar]

The power values in the net.res_sgen table are equivalent to P_{sgen} and Q_{sgen}.

2.7 External Grid

See also:

Unit Systems and Conventions

2.7.1 Create Function

pandapower.**create_ext_grid**(*net, bus, vm_pu=1.0, va_degree=0.0, name=None, in_service=True, s_sc_max_mva=nan, s_sc_min_mva=nan, rx_max=nan, rx_min=nan, max_p_kw=nan, min_p_kw=nan, max_q_kvar=nan, min_q_kvar=nan, 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: **net** - pandapower network

bus (int) - bus where the slack is connected

OPTIONAL: **vm_pu** (float, default 1.0) - voltage at the slack node in per unit

va_degree (float, default 0.) - name of of the external grid*

name (string, default None) - name of of the external grid

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

Sk_max - maximal short circuit apparent power **

SK_min - maximal short circuit apparent power **

RX_max - maximal R/X-ratio **

RK_min - minimal R/X-ratio **

* 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

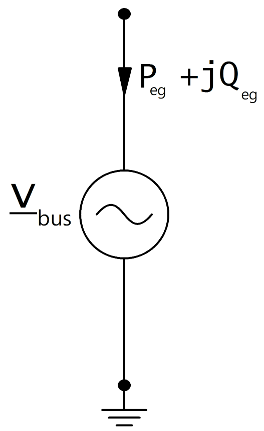
Parameter	Datatype	Value Range	Explanation
name	string		name of the external grid
bus*	integer		index of connected bus
vm_pu*	float	> 0	voltage set point [p.u]
va_degree*	float		angle set point [degree]

Parameter	Datatype	Value Range	Explanation
max_p_kw**	float		Maximum active power
min_p_kw**	float		Minimum active power
max_q_kvar**	float		Maximum reactive power
min_q_kvar**	float		Minimum reactive power
s_sc_max_mva***	float	> 0	maximum short circuit power provision [MVA]
s_sc_min_mva***	float	> 0	minimum short circuit power provision [MVA]
rx_max***	float	0...1	maxium R/X ratio of short-circuit impedance
rx_min***	float	0...1	minimum R/X ratio of short-circuit impedance
in_service*	boolean	True / False	specifies if the external grid is in service.

*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:

$$\underline{v}_{bus} = vm\_pu \cdot e^{j \cdot \theta}$$
$$\theta = shift\_degree \cdot \frac{\pi}{180}$$

2.7.4 Result Parameters

net.res_ext_grid

Parameter	Datatype	Explanation
p_kw	float	active power supply at the external grid [kW]
q_kvar	float	reactive power supply at the external grid [kVar]

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).

`pandapower.`**`create_transformer`**(*net*, *hv_bus*, *lv_bus*, *std_type*, *name=None*, *tp_pos=nan*, *in_service=True*, *index=None*, *max_loading_percent=nan*, *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) - Force a specified ID if it is available

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

OUTPUT: trafo_id - The unique trafo_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_lv_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_lv_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

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) - Force a specified ID if it is available

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: trafo_id - The unique trafo_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

Parameter	**Datatype**	**Value Range**	**Explanation**
name	string		name of the transformer
std_type	string		transformer standard type name
hv_bus*	integer		high voltage bus index of the transformer
lv_bus*	integer		low voltage bus index of the transformer
sn_kva*	float	> 0	rated apparent power of the transformer [kVA]
vn_hv_kv*	float	> 0	rated voltage at high voltage bus [kV]
vn_lv_kv*	float	> 0	rated voltage at low voltage bus [kV]
vsc_percent*	float	> 0	short circuit voltage [%]
vscr_percent*	float	≥ 0	real component of short circuit voltage [%]
pfe_kw*	float	≥ 0	iron losses [kW]
i0_percent*	float	≥ 0	open loop losses in [%]
shift_degree*	float		transformer phase shift angle
tp_side	string	"hv", "lv"	defines if tap changer is at the high- or low voltage side
tp_mid	integer		rated tap position
tp_min	integer		minimum tap position

Parameter	Datatype	Value Range	Explanation
tp_max	integer		maximum tap position
tp_st_percent	float	>0	tap step size [%]
tp_pos	integer		current position of tap changer
max_loading_percent**	float	>0	Maximum loading of the transformer with respect to sn_kva and its corresponding current at 1.0 p.u.
in_service*	boolean	True / False	specifies if the transformer is in service.

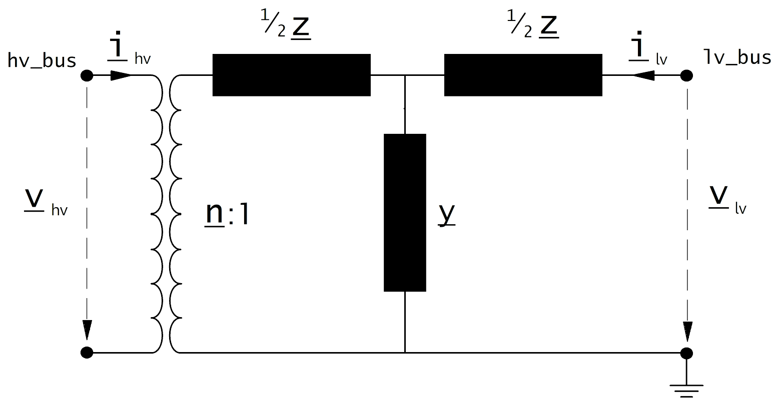
*necessary for executing a power flow calculation **optimal power flow parameter

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

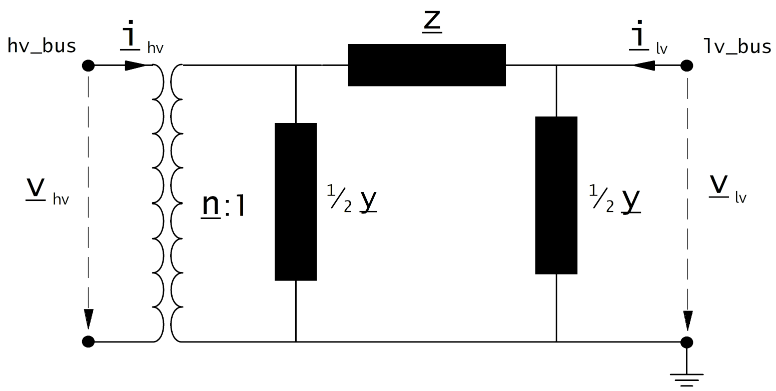
2.8.3 Electric Model

The equivalent circuit used for the transformer can be set in the power flow with the parameter "trafo_model".

trafo_model='t':



trafo_model='pi':



Transformer Ratio:

The magnitude of the transformer ratio is given as:

$$n = \frac{V_{ref,HV,transformer}}{V_{ref,LV,transformer}} \cdot \frac{V_{ref,LV bus}}{V_{ref,HV bus}}$$

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:

$$V_{ref,HV,transformer} = vn\_hv\_kv$$
$$V_{ref,LV,transformer} = vn\_lv\_kv$$

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

$$n_{tap} = 1 + (tp\_pos - tp\_mid) \cdot \frac{tp\_st\_percent}{100}$$

On which side the reference voltage is adapted depends on the tp_side variable:

	tp_side="hv"	**tp_side="lv"**
$V_{n,HV,transformer}$	$vnh\_kv \cdot n_{tap}$	vnh_kv
$V_{n,LV,transformer}$	vnl_kv	$vnl\_kv \cdot n_{tap}$

Note: The variables tp_min and tp_max are not considered in the power flow. The user is responsible to ensure that tp_min < tp_pos < tp_max!

Phase Shift:

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

$$\underline{n} = n \cdot e^{j \cdot \theta}$$
$$\theta = shift\_degree \cdot \frac{\pi}{180}$$

Otherwise, the ratio does not include a phase shift:

$$\underline{n} = n$$

Impedances:

The short-circuit impedance is calculated as:

$$z_k = \frac{vsc\_percent}{100} \cdot \frac{1000}{sn\_kva}$$
$$r_k = \frac{vscr\_percent}{100} \cdot \frac{1000}{sn\_kva}$$
$$x_k = \sqrt{z^2 - r^2}$$
$$\underline{z}_k = r_k + j \cdot x_k$$

The magnetising admittance is calculated as:

$$y_m = \frac{i0\_percent}{100}$$
$$g_m = \frac{pfe\_kw}{sn\_kva \cdot 1000} \cdot \frac{1000}{sn\_kva}$$
$$b_m = \sqrt{y_m^2 - g_m^2}$$
$$\underline{y_m} = g_m - j \cdot b_m$$

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:

$$Z_N = \frac{V_N^2}{S_N}$$
$$Z_{ref,trafo} = \frac{vn\_lv\_kv^2 \cdot 1000}{sn\_kva}$$
$$\underline{z} = \underline{z}_k \cdot \frac{Z_{ref,trafo}}{Z_N}$$
$$\underline{y} = \underline{y}_m \cdot \frac{Z_N}{Z_{ref,trafo}}$$

Where $S_N = 1\ MVA$ (see *Unit Systems and Conventions*) and V_N is the nominal bus voltage at the low voltage side of the transformer.

2.8.4 Result Parameters

net.res_trafo

Parameter	Datatype	Explanation
p_hv_kw	float	active power flow at the high voltage transformer bus [kW]
q_hv_kvar	float	reactive power flow at the high voltage transformer bus [kVar]
p_lv_kw	float	active power flow at the low voltage transformer bus [kW]
q_lv_kvar	float	reactive power flow at the low voltage transformer bus [kVar]
pl_kw	float	active power losses of the transformer [kW]
ql_kvar	float	reactive power consumption of the transformer [kvar]
i_hv_ka	float	current at the high voltage side of the transformer [kA]
i_lv_ka	float	current at the low voltage side of the transformer [kA]
loading_percent	float	load utilization relative to rated power [%]

$$p\_hv\_kw = Re(\underline{v}_{hv} \cdot \underline{i}_{hv}^*)$$
$$q\_hv\_kvar = Im(\underline{v}_{hv} \cdot \underline{i}_{hv}^*)$$
$$p\_lv\_kw = Re(\underline{v}_{lv} \cdot \underline{i}_{lv}^*)$$
$$q\_lv\_kvar = Im(\underline{v}_{lv} \cdot \underline{i}_{lv}^*)$$
$$pl\_kw = p\_hv\_kw + p\_lv\_kw$$
$$ql\_kvar = q\_hv\_kvar + q\_lv\_kvar$$
$$i\_hv\_ka = i_{hv}$$
$$i\_lv\_ka = i_{lv}$$

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:

$$loading\_percent = max(\frac{i_{hv} \cdot vn\_hv\_kv}{sn\_kva}, \frac{i_{lv} \cdot vn\_lv\_kv}{sn\_kva}) \cdot 100$$

For trafo_loading=”power”, the loading is defined as:

$$loading\_percent = max(\frac{i_{hv} \cdot v_{hv}}{sn\_kva}, \frac{i_{lv} \cdot v_{lv}}{sn\_kva}) \cdot 100$$

2.9 Three Winding Transformer

See also:

Unit Systems and Conventions Standard Type Libraries

2.9.1 Create Function

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) - Force a specified ID if it is available

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

OUTPUT: trafo_id - The unique trafo_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 MVA 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=0.0*, *shift_lv_degree=0.0*, *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 ciruit 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

Parameter	Datatype	Value Range	Explanation
name	string		name of the transformer
hv_bus*	integer		high voltage bus index of the transformer
mv_bus	integer		medium voltage bus index of the transformer
lv_bus*	integer		low voltage bus index of the transformer

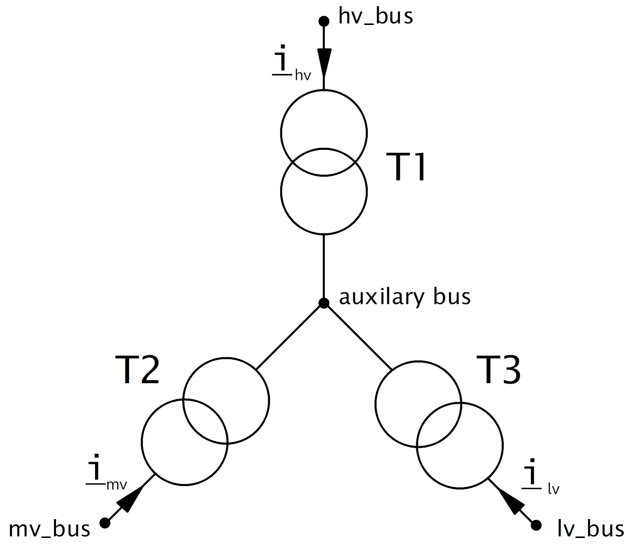
Parameter	Datatype	Value Range	Explanation
vn_hv_kv*	float		rated voltage at high voltage bus [kV]
vn_mv_kv*	float	> 0	rated voltage at medium voltage bus [kV]
vn_lv_kv*	float	> 0	rated voltage at low voltage bus [kV]
sn_hv_kva*	float	> 0	rated apparent power on high voltage side [kVA]
sn_mv_kva*	float	> 0	rated apparent power on medium voltage side [kVA]
sn_lv_kva*	float	> 0	rated apparent power on low voltage side [kVA]
vsc_hv_percent*	float	> 0	short circuit voltage from high to medium voltage [%]
vsc_mv_percent*	float	> 0	short circuit voltage from medium to low voltage [%]
vsc_lv_percent*	float	> 0	short circuit voltage from high to low voltage [%]
vscr_hv_percent*	float	≥ 0	real part of short circuit voltage from high to medium voltage [%]
vscr_mv_percent*	float	≥ 0	real part of short circuit voltage from medium to low voltage [%]
vscr_lv_percent*	float	≥ 0	real part of short circuit voltage from high to low voltage [%]
pfe_kw*	float	≥ 0	iron losses [kW]
i0_percent*	float	≥ 0	open loop losses [%]
tp_side	string	“hv”, “mv”, “lv”	defines if tap changer is positioned on high- medium- or low voltage side
tp_mid	integer		
tp_min	integer		minimum tap position
tp_max	integer		maximum tap position
tp_st_percent	float	> 0	tap step size [%]
tp_pos	integer		current position of tap changer
in_service*	boolean	True/False	specifies if the transformer is in service.

*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:



The parameters of the three transformers are defined as follows:

	T1	**T2**	**T3**
hv_bus	hv_bus	auxiliary bus	auxiliary bus
lv_bus	auxiliary bus	mv_bus	lv_bus
sn_kva	sn_hv_kva	sn_mv_kva	sn_lv_kva
vn_hv_kv	vn_hv_kv	vn_hv_kv	vn_hv_kv
vn_lv_kv	vn_hv_kv	vn_mv_kv	vn_lv_kv
vsc_percent	$v_{k,t1}$	$v_{k,t2}$	$v_{k,t3}$
vscr_percent	$v_{r,t1}$	$v_{r,t2}$	$v_{r,t3}$
pfe_kw	pfe_kw	0	0
i0_percent	i0_percent	0	0
shift_degree	shift_degree	0	0

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:

$$
\begin{aligned}
v'_{k,h} &= vsc\_hv\_percent \\
v'_{k,m} &= vsc\_mv\_percent \cdot \frac{sn\_hv\_kva}{sn\_mv\_kva} \\
v'_{k,l} &= vsc\_lv\_percent \cdot \frac{sn\_hv\_kva}{sn\_lv\_kva}
\end{aligned}
$$

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

$$
\begin{aligned}
v'_{k,t1} &= \frac{1}{2}(v'_{k,h} + v'_{k,l} - v'_{k,m}) \\
v'_{k,t2} &= \frac{1}{2}(v'_{k,m} + v'_{k,h} - v'_{k,l}) \\
v'_{k,t3} &= \frac{1}{2}(v'_{k,m} + v'_{k,l} - v'_{k,h})
\end{aligned}
$$

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

$$v_{k,t1} = v'_{k,t1}$$
$$v_{k,t2} = v'_{k,t2} \cdot \frac{sn\_mv\_kva}{sn\_hv\_kva}$$
$$v_{k,t3} = v'_{k,t3} \cdot \frac{sn\_lv\_kva}{sn\_hv\_kva}$$

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 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.

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

	tp_side="hv"	**tp_side="mv"**	**tp_side="lv"**
$V_{n,HV,transformer}$	$vnh\_kv \cdot n_{tap}$	vnh_kv	vnh_kv
$V_{n,MV,transformer}$	vnm_kv	$vnm\_kv \cdot n_{tap}$	vnm_kv
$V_{n,LV,transformer}$	vnl_kv	vnl_kv	$vnl\_kv \cdot n_{tap}$

with

$$n_{tap} = 1 + (tp\_pos - tp\_mid) \cdot \frac{tp\_st\_percent}{100}$$

See also:

MVA METHOD FOR 3-WINDING TRANSFORMER

2.9.4 Result Parameters

net.res_trafo3w

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

$$
\begin{aligned}
p\_hv\_kw &= Re(\underline{v}_{hv} \cdot \underline{i}_{hv}) \\
q\_hv\_kvar &= Im(\underline{v}_{hv} \cdot \underline{i}_{hv}) \\
p\_mv\_kw &= Re(\underline{v}_{mv} \cdot \underline{i}_{mv}) \\
q\_mv\_kvar &= Im(\underline{v}_{mv} \cdot \underline{i}_{mv}) \\
p\_lv\_kw &= Re(\underline{v}_{lv} \cdot \underline{i}_{lv}) \\
q\_lv\_kvar &= Im(\underline{v}_{lv} \cdot \underline{i}_{lv}) \\
pl\_kw &= p\_hv\_kw + p\_lv\_kw \\
ql\_kvar &= q\_hv\_kvar + q\_lv\_kvar \\
i\_hv\_ka &= i_{hv} \\
i\_mv\_ka &= i_{mv} \\
i\_lv\_ka &= i_{lv}
\end{aligned}
$$

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:

$$loading\_percent = max(\frac{i_{hv} \cdot vn\_hv\_kv}{sn\_hv\_kva}, \frac{i_{mv} \cdot vn\_mv\_kv}{sn\_mv\_kva}, \frac{i_{lv} \cdot vn\_lv\_kv}{sn\_lv\_kva}) \cdot 100$$

For trafo_loading="power", the loading is defined as:

$$loading\_percent = max(\frac{i_{hv} \cdot v_{hv}}{sn\_hv\_kva}, \frac{i_{mv} \cdot v_{mv}}{sn\_mv\_kva}, \frac{i_{lv} \cdot v_{lv}}{sn\_lv\_kva}) \cdot 100$$

2.10 Generator

See also:

Unit Systems and Conventions

2.10.1 Create Function

pandapower.**create_gen**(*net, bus, p_kw, vm_pu=1.0, sn_kva=nan, name=None, index=None, max_q_kvar=nan, min_q_kvar=nan, min_p_kw=nan, max_p_kw=nan, scaling=1.0, 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 the specified index. If None, the next highest available index is used

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

OUTPUT: **index** - 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

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

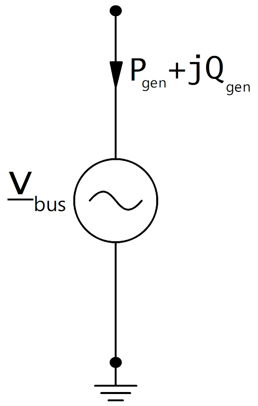
*necessary for executing a power flow calculation **optimal power flow parameter ***short-circuit calculation parameter

---

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:



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

$$P_{gen} = p\_kw * scaling$$
$$v_{bus} = vm\_pu$$

2.10.4 Result Parameters

net.res_gen

Parameter	Datatype	Explanation
p_kw	float	resulting active power demand after scaling [kW]
q_kvar	float	resulting reactive power demand after scaling [kVar]
va_degree	float	generator voltage angle [degree]
vm_pu	float	voltage at the generator [p.u]

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

$$p\_kw = P_{gen}$$
$$q\_kvar = Q_{gen}$$
$$va\_degree = \angle \underline{v}_{bus}$$
$$vm\_degree = |\underline{v}_{bus}|$$

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

`pandapower.`**`create_shunt`**(*net*, *bus*, *q_kvar*, *p_kw=0.0*, *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: **name** (str, None) - element name

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

OUTPUT: shunt id

EXAMPLE: create_shunt(net, 0, 20)

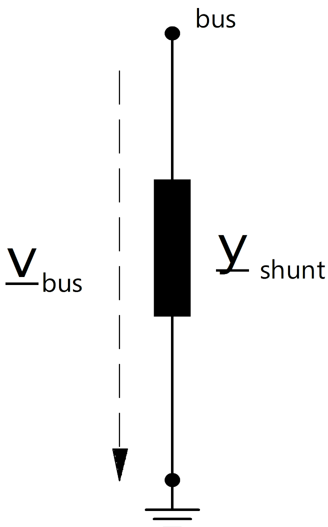
2.11.2 Input Parameters

net.shunt

Parameter	Datatype	Value Range	Explanation
name	string		name of the shunt
bus*	integer		index of bus where the impedance starts
p_kw*	float	≥ 0	shunt active power in kW at v= 1.0 p.u.
q_kvar*	float		shunt reactive power in kvar at v= 1.0 p.u.
in_service*	boolean	True / False	specifies if the shunt is in service.

*necessary for executing a power flow calculation.

2.11.3 Electric Model



The power values are given at $v = 1pu$ or $V = V_N$:

$$\underline{S}_{shunt,ref} = p\_kw + j \cdot q\_kvar$$

Since $\underline{S}_{shunt,ref}$ is the apparent power at the nominal voltage, we know that:

$$\underline{S}_{shunt,ref} = \frac{\underline{Y}_{shunt}}{V_N^2}$$
$$\underline{Y}_{shunt} = \frac{\underline{S}_{shunt,ref}}{V_N^2}$$

Converting to the per unit system results in:

$$\begin{aligned}\underline{y}_{shunt} &= \frac{\underline{S}_{shunt,ref}}{V_N^2} \cdot Z_N \\ &= \frac{\underline{S}_{shunt,ref}}{V_N^2} \cdot \frac{V_N^2}{S_N} \\ &= \frac{\underline{S}_{shunt,ref}}{S_N}\end{aligned}$$

with $S_N = 1\ MVA$ (see *Unit Systems and Conventions*).

2.11.4 Result Parameters

net.res_shunt

Parameter	Datatype	Explanation
p_kw	float	shunt active power consumption [kW]
q_kvar	float	shunt reactive power consumption [kVAr]
vm_pu	float	voltage magnitude at shunt bus [pu]

$$\begin{aligned}p\_kw &= Re(\underline{v}_{bus} \cdot \underline{i}_{shunt}) \\ q\_kvar &= Im(\underline{v}_{bus} \cdot \underline{i}_{shunt}) \\ vm\_pu &= v_{bus}\end{aligned}$$

2.12 Impedance

See also:

Unit Systems and Conventions

2.12.1 Create Function

`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

2.12.2 Input Parameters

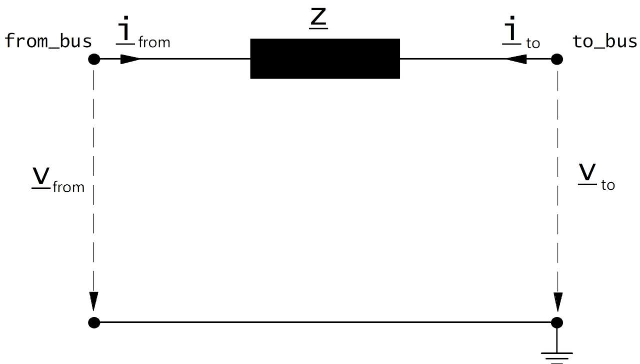
net.impedance

Parameter	Datatype	Value Range	Explanation
name	string		name of the impedance
from_bus*	integer		index of bus where the impedance starts
to_bus*	integer		index of bus where the impedance ends
rft_pu*	float	> 0	resistance of the impedance from ‘from’ to ‘to’ bus [p.u]
xft_pu*	float	> 0	reactance of the impedance from ‘from’ to ‘to’ bus [p.u]
rtf_pu*	float	> 0	resistance of the impedance from ‘to’ to ‘from’ bus [p.u]
xtf_pu*	float	> 0	reactance of the impedance from ‘to’ to ‘from’ bus [p.u]
sn_kva*	float	> 0	reference apparent power for the impedance per unit values [kVA]
in_service*	boolean	True / False	specifies if the imepdance is in service.

*necessary for executing a power flow calculation.

2.12.3 Electric Model

The impedance is modelled as a longitudinal per unit impedance with $\underline{z}_{ft} \neq \underline{z}_{tf}$:



The per unit values given in the parameter table are assumed to be relative to the rated voltage of from and to bus as well as to the apparent power given in the table. The per unit values are therefore transformed into the network per unit system:

$$\underline{z}_{ft} = (rft\_pu + j \cdot xft\_pu) \cdot \frac{S_N}{sn\_kva}$$

$$\underline{z}_{tf} = (rft\_pu + j \cdot xtf\_pu) \cdot \frac{S_N}{sn\_kva}$$

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

The asymetric impedance results in an asymetric nodal point admittance matrix: $\begin{bmatrix} \underline{Y}_{00} & \cdots & \cdots & \underline{Y}_{nn} \\ \vdots & \ddots & \underline{y}_{ft} & \vdots \\ \vdots & \underline{y}_{tf} & \ddots & \vdots \\ \underline{Y}_{n0} & \cdots & \cdots & \underline{y}_{nn} \end{bmatrix}$

2.12.4 Result Parameters

net.res_impedance

Parameter	Datatype	Explanation
p_from_kw	float	active power flow into the impedance at "from" bus [kW]
q_from_kvar	float	reactive power flow into the impedance at "from" bus [kVAr]
p_to_kw	float	active power flow into the impedance at "to" bus [kW]
q_to_kvar	float	reactive power flow into the impedance at "to" bus [kVAr]
pl_kw	float	active power losses of the impedance [kW]
ql_kvar	float	reactive power consumption of the impedance [kVar]
i_from_ka	float	current at from bus [kA]
i_to_ka	float	current at to bus [kA]

$$
\begin{aligned}
i\_from\_ka &= i_{from} \\
i\_to\_ka &= i_{to} \\
p\_from\_kw &= Re(\underline{v}_{from} \cdot \underline{i}^*_{from}) \\
q\_from\_kvar &= Im(\underline{v}_{from} \cdot \underline{i}^*_{from}) \\
p\_to\_kw &= Re(\underline{v}_{to} \cdot \underline{i}^*_{to}) \\
q\_to\_kvar &= Im(\underline{v}_{to} \cdot \underline{i}^*_{to}) \\
pl\_kw &= p\_from\_kw + p\_to\_kw \\
ql\_kvar &= q\_from\_kvar + q\_to\_kvar
\end{aligned}
$$

2.13 Ward

See also:

Unit Systems and Conventions

2.13.1 Create Function

`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

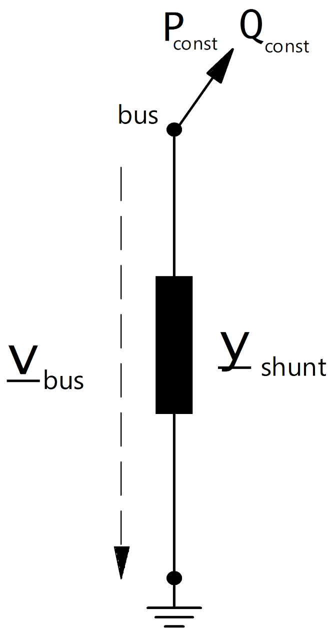
2.13.2 Input Parameters

net.ward

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

*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:

$$P_{const} = ps\_kw$$
$$Q_{const} = qs\_kvar$$

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

$$\underline{y}_{shunt} = \frac{pz\_kw + j \cdot qz\_kvar}{S_N}$$

2.13.4 Result Parameters

net.res_ward

Parameter	Datatype	Explanation
p_kw	float	active power demand of the ward equivalent [kW]
q_kvar	float	reactive power demand of the ward equivalent [kVar]
vm_pu	float	voltage at the ward bus [p.u]

$$vm\_pu = v_{bus}$$
$$p\_kw = P_{const} + Re(\frac{V_{bus}^2}{\underline{Y}_{shunt}})$$
$$q\_kvar = Q_{const} + Im(\frac{V_{bus}^2}{\underline{Y}_{shunt}})$$

2.14 Extended Ward

See also:

Unit Systems and Conventions

2.14.1 Create Function

`pandapower.`**`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

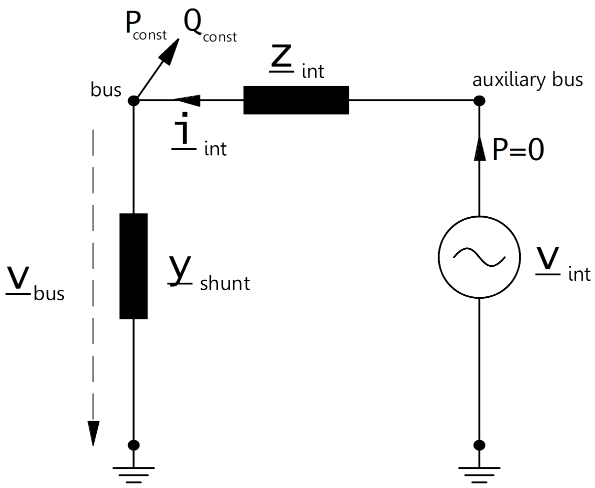
net.xward

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

*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:

$$P_{const} = ps\_kw$$
$$Q_{const} = qs\_kvar$$

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

$$\underline{y}_{shunt} = \frac{pz\_kw + j \cdot qz\_kvar}{S_N}$$

The internal resistance is defined as:

$$\underline{z}_{int} = r\_pu + j \cdot x\_pu$$

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

Parameter	Datatype	Explanation
p_kw	float	active power demand of the ward equivalent [kW]
q_kvar	float	reactive power demand of the ward equivalent [kVar]
vm_pu	float	voltage at the ward bus [p.u]

$$vm\_pu = v_{bus}$$
$$p\_kw = P_{const} + Re(\frac{\underline{V}_{bus}^2}{\underline{Y}_{shunt}}) + Re(\underline{I}_{int} \cdot \underline{V}_{bus})$$
$$q_kvar = Q_{const} + Im(\frac{\underline{V}_{bus}^2}{\underline{Y}_{shunt}} + Im(\underline{I}_{int} \cdot \underline{V}_{bus}))$$

2.15 DC Line

See also:

Unit Systems and Conventions

2.15.1 Create Function

`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

p_kw - (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.

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) - Force a specified ID if it is available

name (str, None) - A custom name for this dc line

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

OUTPUT: (int) Index of dc line

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

net.dcline

Parameter	Datatype	Value Range	Explanation
name	string		name of the generator
from_bus*	integer		Index of bus where the dc line starts
to_bus*	integer		Index of bus where the dc line ends
p_kw*	float	> 0	Active power transmitted from ‘from_bus’ to ‘to_bus’
loss_percent*	float	> 0	Relative transmission loss in percent of active power transmission
loss_kw*	float	> 0	Total transmission loss in kW
vm_from_pu*	float	> 0	Voltage setpoint at from bus
vm_to_pu*	float	> 0	Voltage setpoint at to bus
max_p_kw**	float	> 0	Maximum active power transmission
min_q_from_kvar**	float		Minimum reactive power at from bus
max_q_from_kvar**	float		Maximum reactive power at from bus
min_q_to_kvar**	float		Minimum reactive power at to bus
max_q_to_kvar**	float		Maximum reactive power at to bus
in_service*	bool	True/False	specifies if DC line is in service

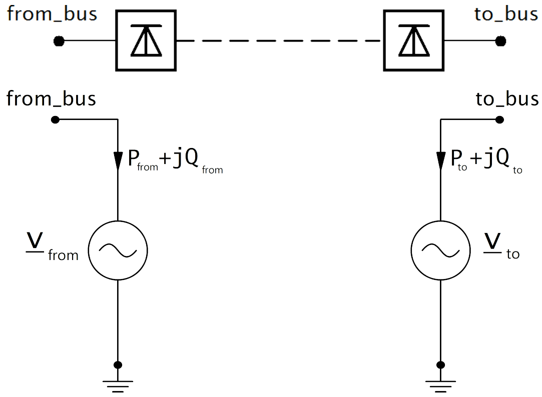
*necessary for executing a power flow calculation **optimal power flow parameter

---

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:



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

net.res_dcline

Parameter	Datatype	Explanation
p_from_kw	float	active power flow into the line at 'from_bus' [kW]
q_from_kvar	float	reactive power flow into the line at 'from_bus' [kVar]
p_to_kw	float	active power flow into the line at 'to_bus' [kW]
q_to_kvar	float	reactive power flow into the line at 'to_bus' [kVar]
pl_kw	float	active power losses of the line [kW]
vm_from_pu	float	voltage magnitude at 'from_bus' [p.u]
va_from_degree	float	voltage angle at 'from_bus' [degree]
vm_to_pu	float	voltage magnitude at 'to_bus' [p.u]
va_to_degree	float	voltage angle at 'to_bus' [degree]

$$
\begin{aligned}
p\_from\_kw &= P_{from} \\
p\_to\_kw &= P_{to} \\
pl\_kw &= p\_from\_kw + p\_to\_kw \\
q\_from\_kvar &= Q_{from} \\
q\_to\_kvar &= Q_{to} \\
va\_from\_degree &= \angle \underline{v}_{from} \\
va\_to\_degree &= \angle \underline{v}_{to} \\
vm\_from\_degree &= |\underline{v}_{from}| \\
vm\_to\_degree &= |\underline{v}_{to}|
\end{aligned}
$$

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

Parameter	Datatype	Value Range	Explanation
type	string	“p” “q” “i” “v”	Defines what physical quantity is measured

Parameter	Datatype	Value Range	Explanation
element_type	string	"bus" "line" "transformer"	Defines which element type is equipped with the measurement
value	float		Measurement value
std_dev	float		Standard deviation (same unit as measurement)
bus	int	must be in net.bus.index	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).
element	int	must be in net.line.index or net.trafo.index	If the element_type is "line" or "transformer", element is the index of the relevant element. For "bus" measurements it is None (default)
check_existing	bool		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
index	int		Defines a specific index for the new measurement (if possible)

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

	r_ohm_per_km	x_ohm_per_km	c_nf_per_km	max_i_ka	type	q_mm2
149-AL1/24-ST1A 10.0	0.194	0.315	11.25	0.47	ol	149
149-AL1/24-ST1A 110.0	0.194	0.41	8.75	0.47	ol	149
149-AL1/24-ST1A 20.0	0.194	0.337	10.5	0.47	ol	149
15-AL1/3-ST1A 0.4	1.8769	0.35	11	0.105	ol	16
184-AL1/30-ST1A 110.0	0.1571	0.4	8.8	0.535	ol	184
184-AL1/30-ST1A 20.0	0.1571	0.33	10.75	0.535	ol	184
24-AL1/4-ST1A 0.4	1.2012	0.335	11.25	0.14	ol	24
243-AL1/39-ST1A 110.0	0.1188	0.39	9	0.645	ol	243
243-AL1/39-ST1A 20.0	0.1188	0.32	11	0.645	ol	243
305-AL1/39-ST1A 110.0	0.0949	0.38	9.2	0.74	ol	305
48-AL1/8-ST1A 0.4	0.5939	0.3	12.2	0.21	ol	48
48-AL1/8-ST1A 10.0	0.5939	0.35	10.1	0.21	ol	48
48-AL1/8-ST1A 20.0	0.5939	0.372	9.5	0.21	ol	48
490-AL1/64-ST1A 220.0	0.059	0.285	10	0.96	ol	490
490-AL1/64-ST1A 380.0	0.059	0.253	11	0.96	ol	490
94-AL1/15-ST1A 0.4	0.306	0.29	13.2	0.35	ol	94
94-AL1/15-ST1A 10.0	0.306	0.33	10.75	0.35	ol	94
94-AL1/15-ST1A 20.0	0.306	0.35	10	0.35	ol	94
N2XS(FL)2Y 1x120 RM/35 64/110 kV	0.153	0.166	112	0.366	cs	120
N2XS(FL)2Y 1x185 RM/35 64/110 kV	0.099	0.156	125	0.457	cs	185
N2XS(FL)2Y 1x240 RM/35 64/110 kV	0.075	0.149	135	0.526	cs	240
N2XS(FL)2Y 1x300 RM/35 64/110 kV	0.06	0.144	144	0.588	cs	300
NA2XS2Y 1x185 RM/25 12/20 kV	0.161	0.117	273	0.362	cs	185
NA2XS2Y 1x240 RM/25 12/20 kV	0.122	0.112	304	0.421	cs	240
NA2XS2Y 1x95 RM/25 12/20 kV	0.313	0.132	216	0.252	cs	95
NAYY 4x120 SE	0.225	0.08	264	0.242	cs	120
NAYY 4x150 SE	0.208	0.08	261	0.27	cs	150
NAYY 4x50 SE	0.642	0.083	210	0.142	cs	50

3.1.2 Transformers

3.1.3 Three Winding 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	vscr_hv_percent
63/25/38 MVA 110/10/10 kV	63000	25000	38000	110	10	10	10.4	10.4	10.4	0.28
63/25/38 MVA 110/20/10 kV	63000	25000	38000	110	20	10	10.4	10.4	10.4	0.28

	vscr_mv_percent	vscr_lv_percent	pfe_kw	i0_percent	shift_mv_degree	shift_lv_degree	tp_side	tp_mid	tp_min	tp_max	tp_st_percent
63/25/38 MVA 110/10/10 kV	0.32	0.35	35	0.89	0	0	hv	0	-10	10	1.2
63/25/38 MVA 110/20/10 kV	0.32	0.35	35	0.89	0	0	hv	0	-10	10	1.2

3.2 Manage Standard Types

3.2.1 Show all Available Standard Types

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

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, vscr_hv_percent, vscr_mv_percent, vscr_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:

```
>>> 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}
>>> pandapower.create_std_type(net, line_data, "NAYY 4×50 SE", element='line')
```

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:

```
>>> 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 data base. 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 panadpower 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

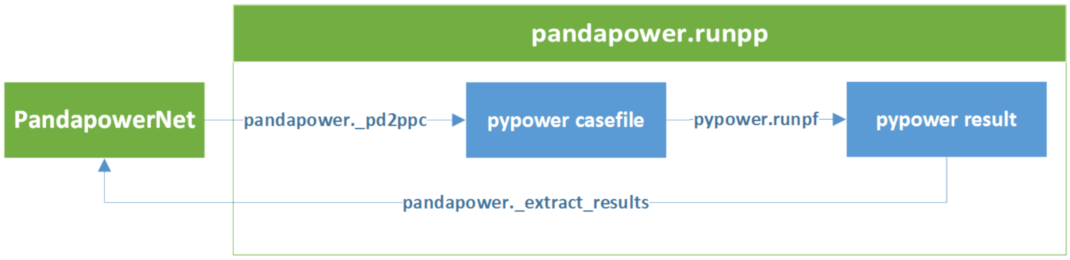
The power flow is the most import static network calculation operation. This section shows you how to run different power flows (AC, DC, OPF), what known problems and caveats there are and how you can identify problems using the pandapower diagnostic function.

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, **kwargs*)

Runs PANDAPOWER AC Flow

Note: May raise pandapower.api.run["load"]flowNotConverged

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. Thats 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:

- “flat”- flat start with voltage of 1.0pu and angle of 0° at all 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 valdiation 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, True) - 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.

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, 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 perfomed. 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.

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

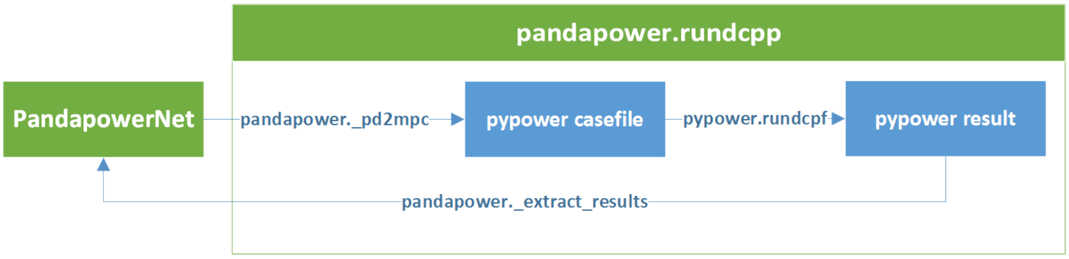
Warning: Neglecting voltage angles is only valid in radial networks! pandapower was developed for distribution networks, which is why omitting the voltage angles is the default. However be aware that voltage angle differences in networks with multiple galvanically coupled external grids lead to balancing 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 pandpower 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:



`pandapower.`**`rundcpp`**(*net*, *trafo_model='t'*, *trafo_loading='current'*, *recycle=None*, *check_connectivity=True*, *r_switch=0.0*, ***kwargs*)

Runs PANDAPOWER DC Flow

Note: May raise pandapower.api.run["load"]flowNotConverged

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 perfomed. 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 pandpower 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.

$$\min \sum_{i \;\epsilon\; gen,sgen,load,extgrid} P_i * f_i(P_i)$$

$$subject\ to$$

$$Loadflow\ equations$$

$$branch\ constraints$$

$$bus\ constraints$$

$$operational\ power\ constraints$$

Generator Flexibilities / Operational power constraints

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

Constraint	Defined in
$P_{min,i} \leq P_g \leq P_{max,g}, g \;\epsilon\; gen$	net.gen.min_p_kw / net.gen.max_p_kw
$Q_{min,g} \leq Q_g \leq Q_{max,g}, g \;\epsilon\; gen$	net.gen.min_q_kvar / net.gen.max_q_kvar
$P_{min,sg} \leq P_{sg} \leq P_{max,sg}, sg \;\epsilon\; sgen$	net.sgen.min_p_kw / net.sgen.max_p_kw
$Q_{min,sg} \leq Q_{sg} \leq Q_{max,sg}, sg \;\epsilon\; sgen$	net.sgen.min_q_kvar / net.sgen.max_q_kvar
$P_{max,g}, g \;\epsilon\; dcline$	net.dcline.max_p_kw
$Q_{min,g} \leq Q_g \leq Q_{max,g}, g \;\epsilon\; dcline$	net.dcline.min_q_from_kvar / net.dcline.max_q_from_kvar / net.dcline.min_q_to_kvar / net.dcline.max_q_to_kvar
$P_{min,eg} \leq P_{eg} \leq P_{max,eg}, eg \;\epsilon\; ext_grid$	net.ext_grid.min_p_kw / net.ext_grid.max_p_kw
$Q_{min,eg} \leq Q_{eg} \leq Q_{max,eg}, eg \;\epsilon\; ext_grid$	net.ext_grid.min_q_kvar / net.ext_grid.max_q_kvar

Network Constraints

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

Constraint	Defined in
$V_{min,j} \leq V_{g,i} \leq V_{min,i}, j \;\epsilon\; bus$	net.bus.min_vm_pu / net.bus.max_vm_pu
$L_k \leq L_{max,k}, k \;\epsilon\; trafo$	net.trafo.max_loading_percent
$L_l \leq L_{max,l}, l \;\epsilon\; line$	net.line.max_loading_percent
$L_l \leq L_{max,l}, l \;\epsilon\; trafo_{3w}$	net.trafo3w.max_loading_percent

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 \$n\$ data points:

$$f_{pwl}(p) = f_\alpha + (p - p_\alpha)\frac{f_{\alpha+1} - f_\alpha}{p_{\alpha+1} - p_\alpha}\ ,\ (p_\alpha, f_\alpha)\ = \begin{cases} (p_0, f_0)\ , & p_0 < p < p_1) \\ ... \\ (p_{n-1}, f_{n-1})\ , & p_{n-1} < p < p_n) \end{cases}$$

$$f_{pwl}(q) = f_\alpha + (q - q_\alpha)\frac{f_{\alpha+1} - f_\alpha}{q_{\alpha+1} - q_\alpha}\ ,\ (q_\alpha, f_\alpha)\ = \begin{cases} (q_0, f_0)\ , & q_0 < p < q_1) \\ ... \\ (q_{n-1}, f_{n-1})\ , & q_{n-1} < q < q_n) \end{cases}$$

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

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) - Force a specified ID if it is available

OUTPUT: (int) Index of 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:

$$f_{pol}(p) = c_n p^n + ... + c_1 p + c_0$$
$$f_{pol}(q) = c_2 q^2 + c_1 q + c_0$$

Polynomial cost functions can be speciefied using create_polynomial_cost():

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", "sgen", "ext_grid", "load", "dcline"] are possible

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

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

index (int) - Force a specified ID if it is available

OUTPUT: (int) Index of cost entry

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

Please note, that polynomial costs for reactive power can only be quadratic, linear or constant. Active and reactive power costs are calculted seperately. 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 optimisation with a valid loadflow solution. For optimiation of a timeseries, this warm start possibilty 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:

`pandapower.`**`runopp`**(*net*, *verbose=False*, *calculate_voltage_angles=False*, *check_connectivity=True*, *suppress_warnings=True*, *r_switch=0.0*, ***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.sgen.min_q_kvar / net.sgen.max_q_kvar
- net.gen.min_p_kw / net.gen.max_p_kw
- net.gen.min_q_kvar / net.gen.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

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:

- “On the Computation and Application of Multi-period Security-Constrained Optimal Power Flow for Real-time Electricity Market Operations”, Cornell University, May 2007.

- H. Wang, C. E. Murillo-Sanchez, R. D. Zimmerman, R. J. Thomas, "On Computational Issues of Market-Based Optimal Power Flow", IEEE Transactions on Power Systems, Vol. 22, No. 3, Aug. 2007, pp. 1185-1193.
- R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "MATPOWER: Steady-State Operations, Planning and Analysis Tools for Power Systems Research and Education," Power Systems, IEEE Transactions on, vol. 26, no. 1, pp. 12-19, Feb. 2011.

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.`**`rundcopp`**(*net*, *verbose=False*, *check_connectivity=True*, *suppress_warnings=True*, *r_switch=0.0*, ***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.sgen.min_q_kvar / net.sgen.max_q_kvar
- net.gen.min_p_kw / net.gen.max_p_kw
- net.gen.min_q_kvar / net.gen.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

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.

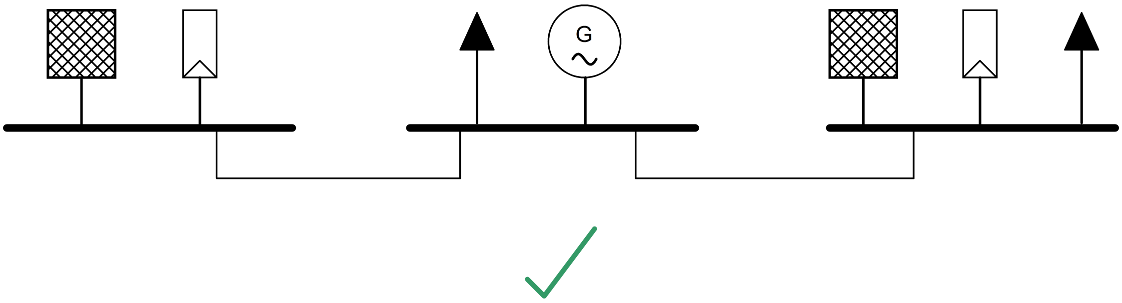
Flexibilities, costs and constraints (except voltage constraints) are handled as in the *Optimal Power Flow*. Voltage constraints are not considered in the DC OPF, since voltage magnitutes are not part of the linearized power flow equations.

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.

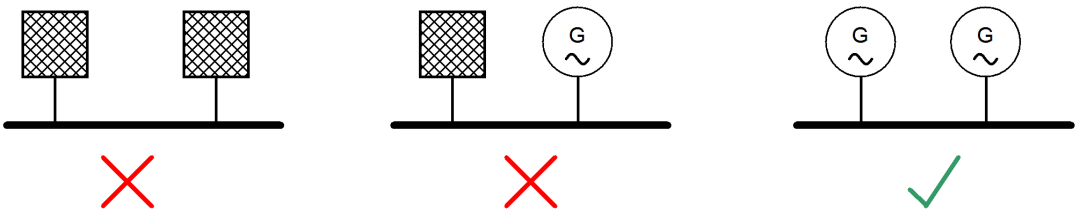
4.2 Known Problems and Caveats

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:

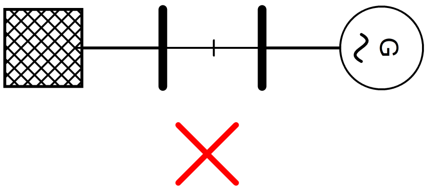


It is however not possible to connect multiple ext_grids and gens at one bus, since this would convergence problems in PYPOWER:



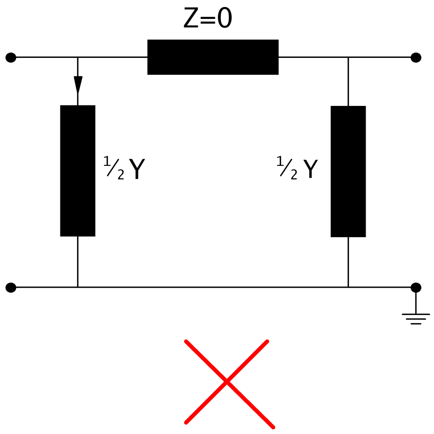
The pandapower API will prevent you from adding a second voltage controlling element to a bus, so you should not be able to build the networks pictured above through the pandapower API.

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*):



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.

`pandapower.`**`diagnostic`**(*net*, *report_style='detailed'*, *warnings_only=False*, *return_result_dict=True*, *overload_scaling_factor=0.001*, *lines_min_length_km=0*, *lines_min_z_ohm=0*, *nom_voltage_tolerance=0.3*, *numba_tolerance=1e-05*)

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 ouput in the console

  'detailled': 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:

<<< 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:

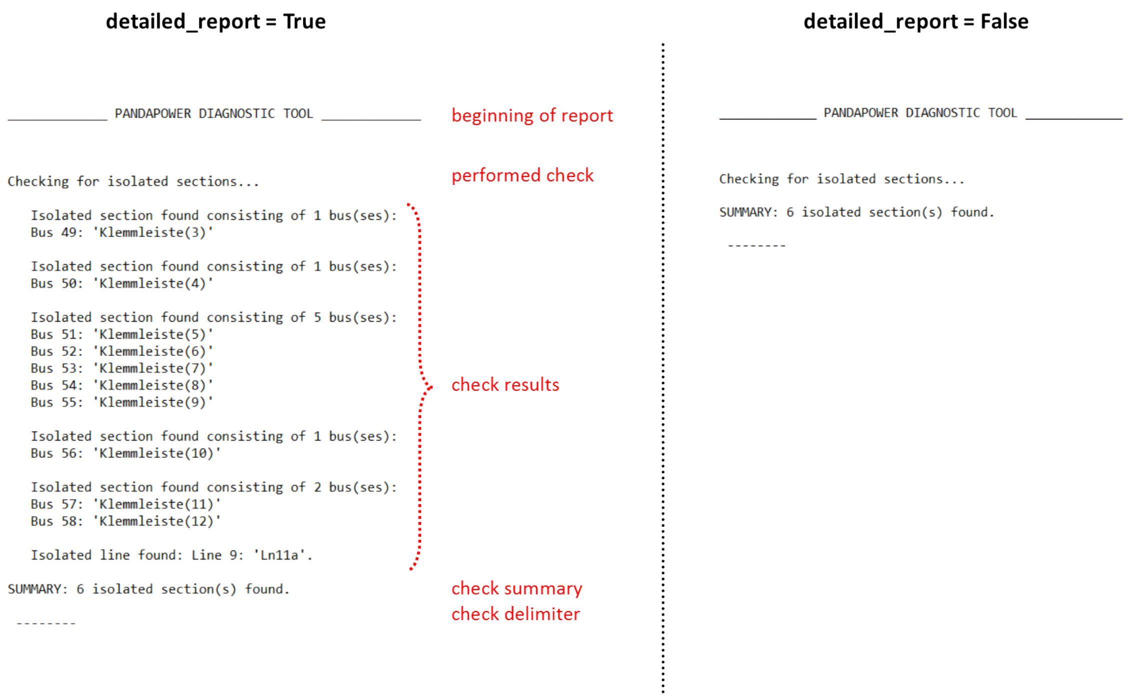
- invalid values (e.g. negative element indeces)
- check, if at least one external grid exists
- check, if there are buses with more than one gen and/or ext_grid
- overload: tries to run a power flow calculation with loads scaled down to 10%
- switch_configuration: tries to run a power flow calculation 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.



warnings_only = True/False

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.

Key	Type	Size	Value
busses_mult_gens_ext_grids	NoneType	1	None
deviating_nominal_voltages	dict	1	{'trafos': {'hv_bus': [], 'hv_lv_swapped': [1], 'lv_bus': []}}
ext_grid	NoneType	1	None
inconsistent_voltages	dict	2	{'lines': [5, 6, 7], 'switches': [284, 299, 300]}
invalid_values	dict	8	{'bus': [[16, 'un_kv', -20.0]], 'ext_grid': [], 'gen': [], 'line': [[0, 'length_km', -10.0], [8, 'length_km', -10.0], [9, 'length_km', 0.0]], 'load': [], 'sgen': [], 'switch': [[0, 'closed', 1.5]], 'trafo': []}
isolated_sections	dict	2	{'isolated_sections': [[49], [50], [51, 52, 53, 54, 55], [56], [57, 58]], 'lines_both_switches_open': [9]}
lines_with_impedance_zero	list	1	[9]
overload	NoneType	1	None
problematic_switches	NoneType	1	None
wrong_reference_system	NoneType	1	None
wrong_switch_configuration	NoneType	1	None

5 Short-Circuit

The shortcircuit 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:

`pandapower.shortcircuit.calc_sc`(*net*, *fault='3ph'*, *case='max'*, *lv_tol_percent=10*, *topology='auto'*, *ip=False*, *ith=False*, *tk_s=1.0*, *r_fault_ohm=0.0*, *x_fault_ohm=0.0*, *consider_sgens=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 olerance

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 Symmetric Three-Phase Current

5.2.2 Initial Short-Circuit Current

The general ohmic network equation is given as:

$$\begin{bmatrix} \underline{Y}_{11} & \dots & \dots & \underline{Y}_{n1} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \underline{Y}_{1n} & \dots & \dots & \underline{Y}_{nn} \end{bmatrix} \begin{bmatrix} \underline{V}_1 \\ \vdots \\ \vdots \\ \underline{V}_n \end{bmatrix} = \begin{bmatrix} \underline{I}_1 \\ \vdots \\ \vdots \\ \underline{I}_n \end{bmatrix}$$

For the short-circuit calculation with the equivalent voltage source, two assumptions are made:

1. All operational currents are neglected
2. The voltage at the fault bus is assumed to be $\frac{c \cdot \underline{V}_N}{\sqrt{3}}$

where V_N is the nominal voltage at the fault bus and c is the *voltage correction factor*.

For the calculation of a short-circuit at bus m, this yields the following equations:

$$\begin{bmatrix} \underline{Y}_{11} & \dots & \dots & \underline{Y}_{n1} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \underline{Y}_{1n} & \dots & \dots & \underline{Y}_{nn} \end{bmatrix} \begin{bmatrix} \underline{V}_1 \\ \vdots \\ \frac{c_m \cdot \underline{V}_{N,m}}{\sqrt{3}} \\ \vdots \\ \underline{V}_n \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ \underline{I}''_{k,m} \\ \vdots \\ 0 \end{bmatrix}$$

where $\underline{I}''_{k,m}$ is the short-circuit current at bus m and all other bus currents are assumed to be zero. The voltages at all non-fault buses and the current at the fault bus are unknown. To solve for $\underline{I}''_{k,m}$, we multipliy with the inverted nodal point admittance matrix (impedance matrix):

$$\begin{bmatrix} \underline{V}_1 \\ \vdots \\ \frac{c_k \cdot \underline{V}_{N,k}}{\sqrt{3}} \\ \vdots \\ \underline{V}_n \end{bmatrix} = \begin{bmatrix} \underline{Z}_{11} & \dots & \dots & \underline{Z}_{n1} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \underline{Z}_{1n} & \dots & \dots & \underline{Z}_{nn} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ \underline{I}''_{k,m} \\ \vdots \\ 0 \end{bmatrix}$$

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

$$I''_{k,m} = \frac{c \cdot V_{N,m}}{\sqrt{3} \cdot Z_{mm}}$$

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

$$\begin{bmatrix} \underline{V}_1 \\ \vdots \\ \frac{c_k \cdot \underline{V}_{N,k}}{\sqrt{3}} \\ \vdots \\ \underline{V}_n \end{bmatrix} = \begin{bmatrix} \underline{Z}_{11} & \dots & \dots & \underline{Z}_{n1} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \underline{Z}_{1n} & \dots & \dots & \underline{Z}_{nn} \end{bmatrix} \begin{bmatrix} \underline{I}_{k,0} & \dots & \dots & 0 \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \dots & \dots & \underline{I}_{k,n} \end{bmatrix}$$

which yields:

$$\begin{bmatrix} \underline{I}''_{k,0} \\ \vdots \\ \underline{I}''_{k,m} \end{bmatrix} = \begin{bmatrix} \frac{c_1 \cdot \underline{V}_{N,1}}{\sqrt{3} \cdot \underline{Z}_{11}} \\ \vdots \\ \frac{c_n \cdot \underline{V}_{N,n}}{\sqrt{3} \cdot \underline{Z}_{nn}} \end{bmatrix}$$

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:

$$\begin{bmatrix} \underline{I}''_{k,1} \\ \vdots \\ \underline{I}''_{k,m} \end{bmatrix} = \begin{bmatrix} \frac{c_1 \cdot \underline{V}_{N,1}}{\sqrt{3} \cdot (\underline{Z}_{11} + \underline{Z}_{fault})} \\ \vdots \\ \frac{c_n \cdot \underline{V}_{N,n}}{\sqrt{3} \cdot (\underline{Z}_{nn} + \underline{Z}_{fault})} \end{bmatrix}$$

5.2.3 Peak Short-Circuit Current

The peak short-circuit current is calculated as:

$$\begin{bmatrix} i_{p,1} \\ \vdots \\ i_{p,n} \end{bmatrix} = \sqrt{2} \begin{bmatrix} \kappa_1 \\ \vdots \\ \kappa_1 \end{bmatrix} \begin{bmatrix} \underline{I}''_{k,1} \\ \vdots \\ \underline{I}''_{k,n} \end{bmatrix}$$

where the factor κ is calculated for each bus as defined *here*.

5.2.4 Thermal Short-Circuit Current

The equivalent

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

where the factors m and n are calculated for each bus as defined *here*.

5.2.5 Unsymmetric Two-Phase Current

5.2.6 Initial Short-Circuit Current

The two-phase initial short-circuit current is calculated in the same way the three-phase current is calculated, only with a source voltage of $c \cdot \sqrt{2} \cdot V_N$ instead of $\frac{c \cdot V_N}{\sqrt{3}}$:

$$\begin{bmatrix} \underline{I}''_{k2,1} \\ \vdots \\ \underline{I}''_{k2,m} \end{bmatrix} = \begin{bmatrix} \frac{c_1 \cdot \sqrt{2} \cdot V_{N,1}}{Z_{11} + Z_{fault}} \\ \vdots \\ \frac{c_n \cdot \sqrt{2} \cdot V_{N,n}}{Z_{nn} + Z_{fault}} \end{bmatrix}$$

5.2.7 Peak Short-Circuit Current

The peak short-circuit current is calculated as:

$$\begin{bmatrix} i_{p2,1} \\ \vdots \\ i_{p2,n} \end{bmatrix} = \sqrt{2} \begin{bmatrix} \kappa_1 \\ \vdots \\ \kappa_1 \end{bmatrix} \begin{bmatrix} \underline{I}''_{k2,1} \\ \vdots \\ \underline{I}''_{k2,n} \end{bmatrix}$$

where the factor κ is calculated for each bus as defined *here*.

5.2.8 Thermal Short-Circuit Current

The equivalent

$$\begin{bmatrix} \underline{I}_{th2,1} \\ \vdots \\ \underline{I}_{th2,n} \end{bmatrix} = \begin{bmatrix} \sqrt{m_1 + n_1} \\ \vdots \\ \sqrt{m_n + n_n} \end{bmatrix} \begin{bmatrix} \underline{I}''_{k2,1} \\ \vdots \\ \underline{I}''_{k2,n} \end{bmatrix}$$

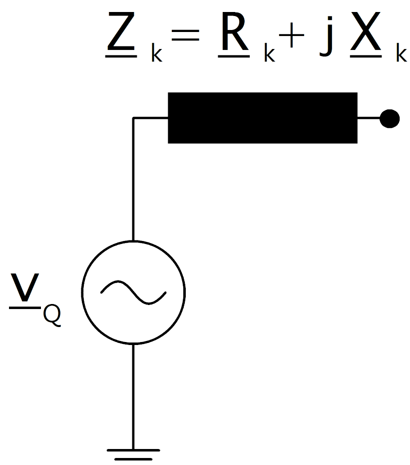
where the factors m and n are calculated for each bus as defined *here*.

5.3 Network Elements

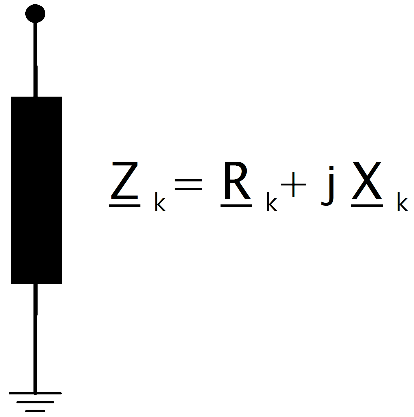
Correction factors for generator and branch elements are implemented as defined in the IEC 60909 standard. The results for all elements are tested against commercial software to ensure that correction factors are correctly applied.

5.3.1 Bus Elements

All bus elements are represented by their internal voltage source with an internal resistance Z_k:



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:



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:

$$\begin{aligned} z_{k,eg} &= \frac{c_{max}}{s\_sc\_max\_mva} \\ x_{k,sg} &= \frac{z_{sg}}{\sqrt{1-(rx\_max)^2}} \\ r_{k,sg} &= rx\_max \cdot x_{sg} \end{aligned}$$

where rx_max and $s\_sc\_max\_mva$ are parameters in the ext_grid table and c_{max} is the *voltage correction factor* of the external grid bus.

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

$$\begin{aligned} z_{k,eg} &= \frac{c_{min}}{s\_sc\_min\_mva} \\ x_{k,sg} &= \frac{z_{sg}}{\sqrt{1-(rx\_min)^2}} \\ r_{k,sg} &= rx\_min \cdot x_{sg} \end{aligned}$$

5.3.3 Static Generator

Not all inverter based elements contribute to the short-circuit current. That is why it can be chosen in the runsc function if static generators are to be considered or not. If they are considered, the short-circuit impedance is defined according to the standard as:

$$\begin{aligned} Z_{k,sg} &= \frac{1}{3} \cdot \frac{vn\_kv^2 \cdot 1000}{sn\_kva} \\ X_{k,sg} &= \frac{Z_{sg}}{\sqrt{1-0.1^2}} \\ R_{k,sg} &= 0.1 \cdot X_{sg} \end{aligned}$$

5.3.4 Synchronous Generator

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

$$\underline{Z}_{k,gen} = K_G \cdot (R''_d + jX''_d)$$

The short-circuit impedance is calculated as:

$$z_k = xdss$$

The generator correction factor K_G is given as:

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

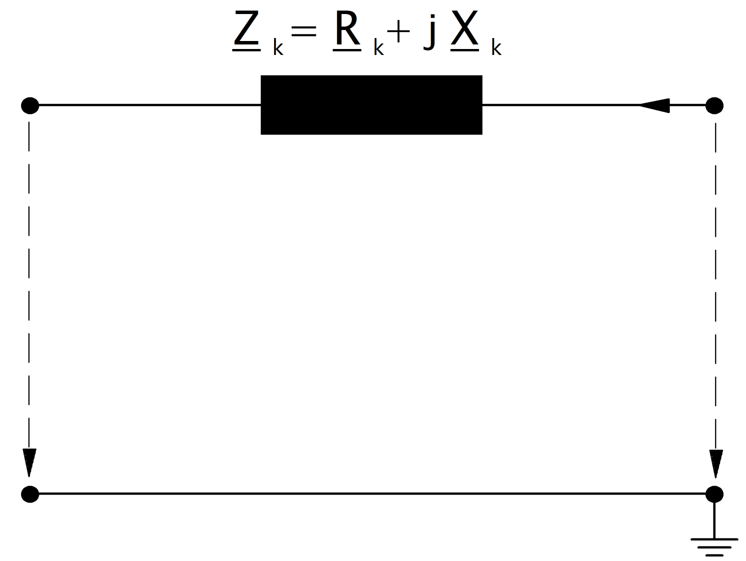
where $V_{N,bus}$ is the rated voltage of the bus the generator is connected to and $V_{N,gen}$ is the rated voltage of the generator which is defined by the parameter sn_kva in the gen table. The rated phasor angle φ is given as:

$$\varphi = arcos(cos\_phi)$$

where cos_phi is defined in the gen table.

5.3.5 Branch Elements

All branches are represented by a single short circuit impedance:



Shunt admittances are neglected for all branch elements.

5.3.6 Line

$$\underline{R}_k = r\_ohm\_per\_km \cdot \frac{length\_km}{parallel} \cdot K_L$$
$$\underline{X}_k = x\_ohm\_per\_km \cdot \frac{length\_km}{parallel}$$

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

$$K_L = \begin{cases} 1 & \text{for maximum short-circuit calculations} \\ 1 + 0.04K^{-1}(endtemp\_degree - 20C) & \text{for minimum short-circuit calculations} \end{cases}$$

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

5.3.7 Two-Winding Transformer

The short-circuit impedance is calculated as:

$$z_k = \frac{vsc\_percent}{100} \cdot \frac{1000}{sn\_kva} \cdot K_T$$
$$r_k = \frac{vscr\_percent}{100} \cdot \frac{1000}{sn\_kva} \cdot K_T$$
$$x_k = \sqrt{z^2 - r^2}$$

where the correction factor K_T is defined in the standard as:

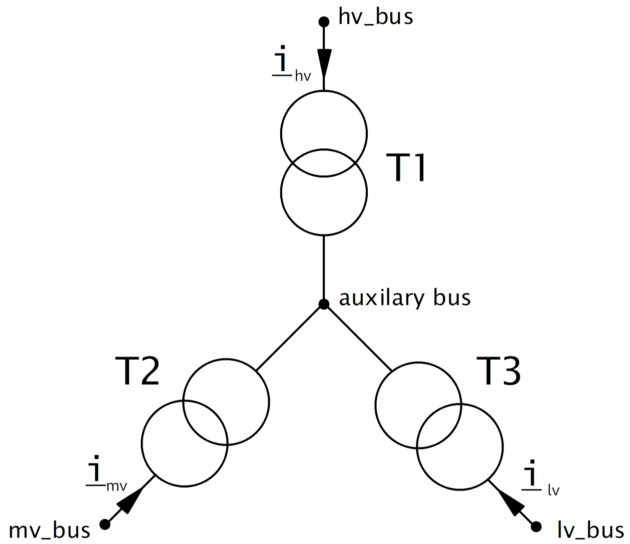
$$K_T = 0.95 \frac{c_{max}}{1 + 0.6 x_T}$$

where c_{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.

5.3.8 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:

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

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

5.3.9 Impedance

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

5.4 Correction Factors

5.4.1 Voltage Corection Factor c

The voltage correction factors c_{min} for minimum and c_{max} for maximum short-circuit currents are applied in calculating short-circuit impedances for some elements (transformer, ext_grid) as well as for the equivalent voltage source for the calculation of the initials short-circuit current I''_k.

It is 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}:

Voltage Level		c_{min}	c_{max}
< 1 kV	Tolerance 6%	0.95	1.05
	Tolerance 10%		1.10
> 1 kV		1.00	

5.4.2 Peak Factor κ

The factor κ is used for calculation of the peak short-circuit current i_p, thermal equivalent short-circuit current I_{th} and unsymmetrical short-circuit currents.

In radial networks, κ is given as:

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

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 definition of κ:

a) Uniform Ratio R/X b) Ratio R/X at short-circuit location c) Equivalent frequency

pandapower implements version b), in which the factor κ is given as:

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

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

Voltage Level	κ_{min}	κ_{max}
< 1 kV	1.0	1.8
> 1 kV		2.0

5.4.3 Thermal Factors m and n

The factors m and n are necessary for the calculation of the thermal equivalent short-circuit current I_{th}.

pandapower currently only implements short-circuit currents far from synchronous generators, where:

$$n = 1$$

and m is given as:

$$m = \frac{1}{2 \cdot f \cdot T_k \cdot ln(\kappa - 1)}[e^{4 \cdot f \cdot T_k \cdot ln(\kappa - 1)} - 1]$$

where κ is defined as above and T_k is the duration of the short-circuit current that can be defined as a parameter when running the short-circuit calculation.

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 *net.ext_grid.va_degree* entry (see *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:

- *Power System State Estimation: Theory and Implementation* by Ali Abur, Antonio Gómez Expósito, CRC Press, 2004.
- *State Estimation in Electric Power Systems - A Generalized Approach* by A. Monticelli, Springer, 1999.

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

	Element Type	Available Measurement Types		
	bus	v,	p,	q
	line	i,	p,	q
	transformer	i,	p,	q

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*)

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

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 identifys 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.

`pandapower.estimation.`**`remove_bad_data`**(*net*, *init='flat'*, *tolerance=1e-06*, *maximum_iterations=10*, *calculate_voltage_angles=True*, *rn_max_threshold=3.0*, *chi2_prob_false=0.05*)

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).

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

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.

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

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

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

See also:

- *Power System State Estimation: Theory and Implementation* by Ali Abur, Antonio Gómez Expósito, CRC Press, 2004.
- *Power Generation, Operation, and Control* by Allen J. Wood, Bruce Wollenberg, Wiley Interscience Publication, 1996.

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:

```
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..

```
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 in our state estimation, we use the following code:

```
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 (probabilty of fault can be adjusted), the Chi-squared test should be performed separatly. If the test detects the possibility of fault data, the value of the added class member variable *bad_data_present* would be *true*.

```
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)

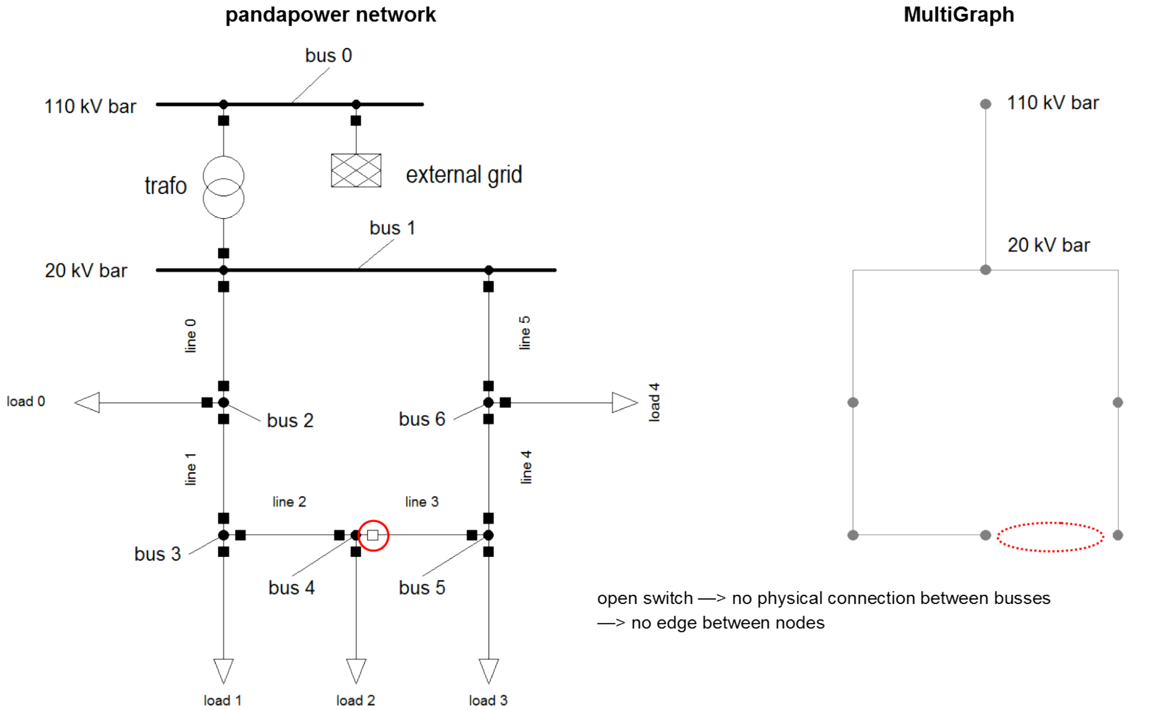
pandapower provides a function to translate pandapower networks into networkx graphs. Once the electric network is translated into an abstract networkx graph, all network operations that are available in networkx can be used to analyse the network. For example you can find the shortest path between two 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 specialiced 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 padapower 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 Multi-Graph's shape is completely arbitrary since MultiGraphs have no inherent shape unless geodata is provided.)



Nodes have the same indicees 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:

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:

`pandapower.topology.`**`create_nxgraph`**`(`*net*, *respect_switches=True*, *include_lines=True*, *include_trafos=True*, *nogobuses=None*, *notravbuses=None*, *multi=True*`)`

Converts a pandapower network into a NetworkX graph, which is a 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)

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

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)

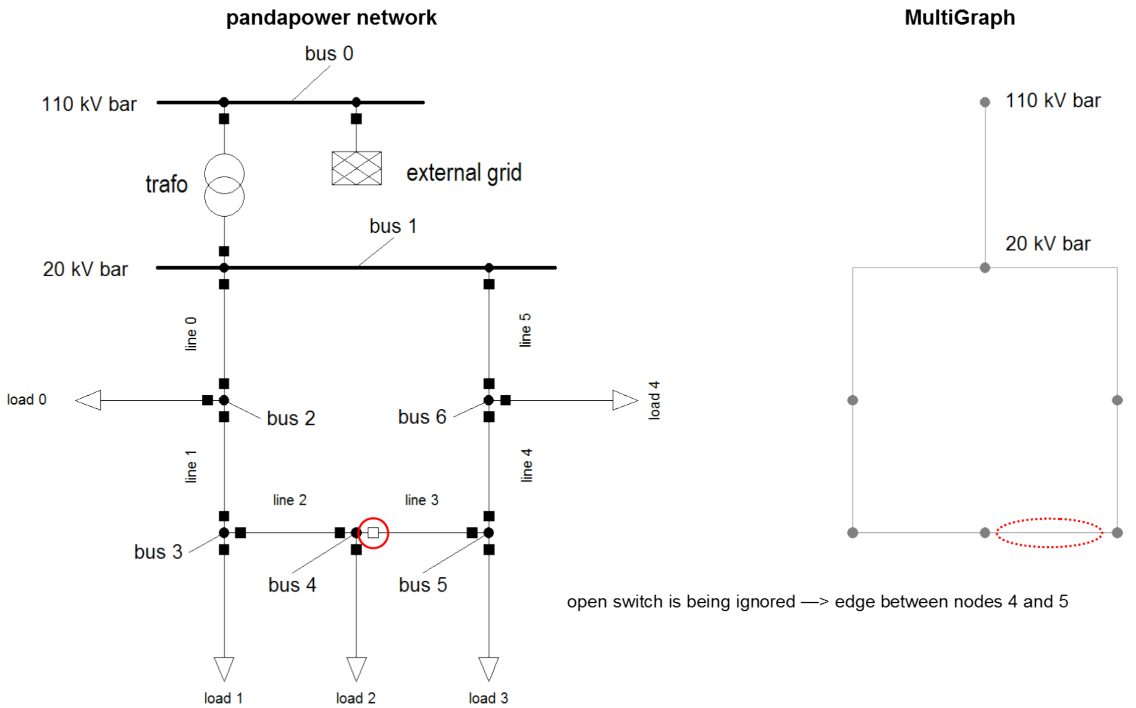
OUTPUT: **mg** - Returns the required NetworkX graph

EXAMPLE: import pandapower.topology as top

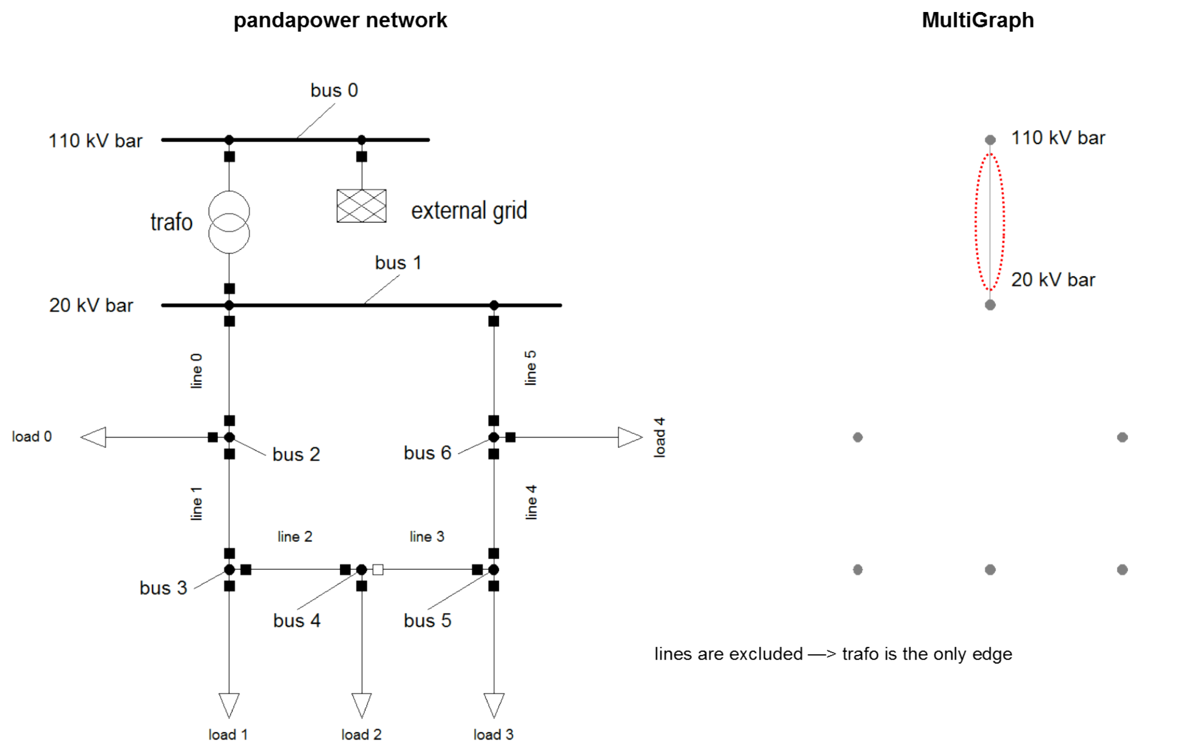
mg = top.create_nx_graph(net, respect_switches = False) # converts the pandapower network "net" to a MultiGraph. Open switches will be ignored.

Examples

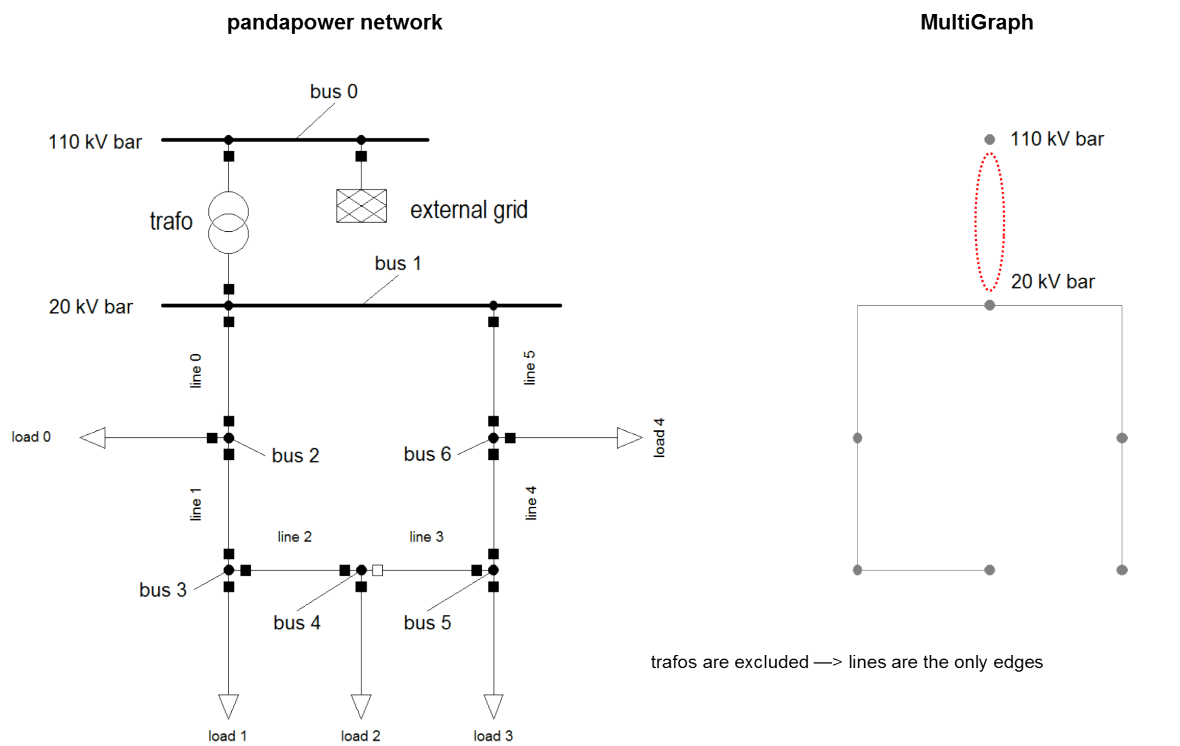
```
create_nxgraph(net, respect_switches = False)
```



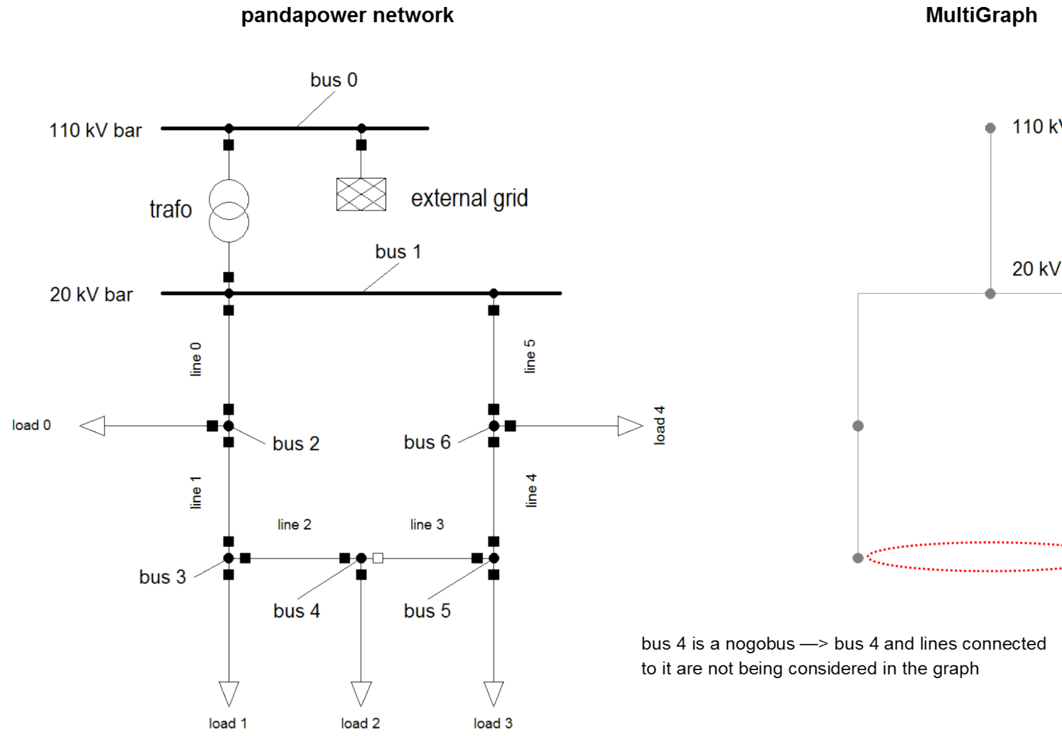

```
create_nxgraph(net, include_lines = False)
```



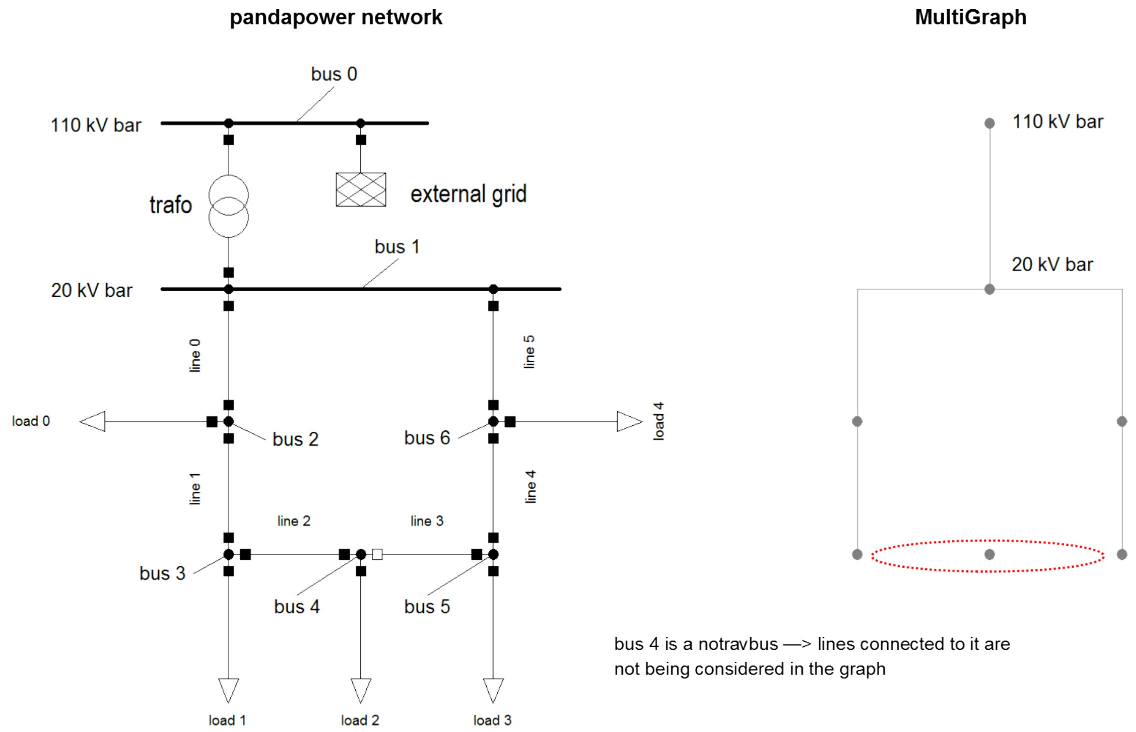

```
create_nxgraph(net, include_trafos = False)
```




```
create_nxgraph(net, nogobuses = [4])
```




```
create_nxgraph(net, notravbuses = [4])
```



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

`pandapower.topology.connected_component` (*mg*, *bus*, *notravbuses=[]*)

Finds all buses in a NetworkX graph that are connected to a certain bus.

INPUT: mg (NetworkX graph) - NetworkX Graph or MultiGraph that represents a pandapower network.

bus (integer) - Index of the bus at which the search for connected components originates

OPTIONAL:

notravbuses (list/set) - Indeces of notravbuses: lines connected to these buses are not being considered in the graph

OUTPUT: cc (generator) - Returns a generator that yields all buses connected to the input bus

EXAMPLE: import pandapower.topology as top

mg = top.create_nx_graph(net)

cc = top.connected_component(mg, 5)

7.2.3 connected_components

`pandapower.topology.connected_components` (*mg*, *notravbuses=set()*)

Clusters all buses in a NetworkX graph that are connected to each other.

INPUT: mg (NetworkX graph) - NetworkX Graph or MultiGraph that represents a pandapower network.

OPTIONAL: **notravbuses** (set) - Indeces of notravbuses: lines connected to these buses are

not being considered in the graph

OUTPUT:

cc (generator) - Returns a generator that yields all clusters of buses connected to each other.

EXAMPLE: import pandapower.topology as top

mg = top.create_nx_graph(net)

cc = top.connected_components(net, 5)

7.2.4 unsupplied_buses

`pandapower.topology.unsupplied_buses`(*net*, *mg=None*, *in_service_only=False*, *slacks=None*)

Finds buses, that are not connected to an external grid.

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

OPTIONAL: **mg** (NetworkX graph) - NetworkX Graph or MultiGraph that represents a pandapower network.

OUTPUT: **ub** (set) - unsupplied buses

EXAMPLE: import pandapower.topology as top

top.unsupplied_buses(net)

7.2.5 determine_stubs

`pandapower.topology.determine_stubs`(*net*, *roots=None*, *mg=None*)

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.

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

OPTIONAL:

roots (integer/list, None) - Indeces 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 availabe 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

```
import pandapower as pp

net = pp.create_empty_network()

pp.create_bus(net, name = "110 kV bar", vn_kv = 110, type = 'b')
pp.create_bus(net, name = "20 kV bar", vn_kv = 20, type = 'b')
pp.create_bus(net, name = "bus 2", vn_kv = 20, type = 'b')
pp.create_bus(net, name = "bus 3", vn_kv = 20, type = 'b')
pp.create_bus(net, name = "bus 4", vn_kv = 20, type = 'b')
pp.create_bus(net, name = "bus 5", vn_kv = 20, type = 'b')
pp.create_bus(net, name = "bus 6", vn_kv = 20, type = 'b')

pp.create_ext_grid(net, 0, vm_pu = 1)

pp.create_line(net, name = "line 0", from_bus = 1, to_bus = 2, length_km = 1, std_
↪type = "NAYY 150")
```

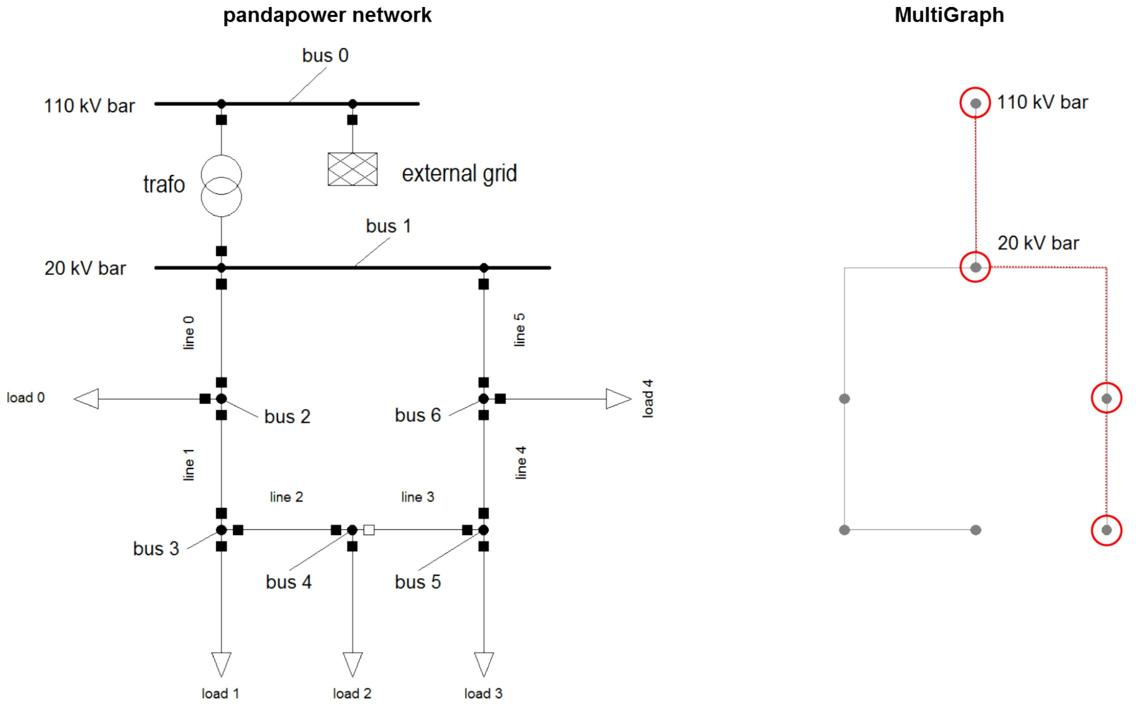
```
pp.create_line(net, name = "line 1", from_bus = 2, to_bus = 3, length_km = 1, std_
↪type = "NAYY 150")
pp.create_line(net, name = "line 2", from_bus = 3, to_bus = 4, length_km = 1, std_
↪type = "NAYY 150")
pp.create_line(net, name = "line 3", from_bus = 4, to_bus = 5, length_km = 1, std_
↪type = "NAYY 150")
pp.create_line(net, name = "line 4", from_bus = 5, to_bus = 6, length_km = 1, std_
↪type = "NAYY 150")
pp.create_line(net, name = "line 5", from_bus = 6, to_bus = 1, length_km = 1, std_
↪type = "NAYY 150")

pp.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)

pp.create_load(net, 2, p_kw = 1000, q_kvar = 200, name = "load 0")
pp.create_load(net, 3, p_kw = 1000, q_kvar = 200, name = "load 1")
pp.create_load(net, 4, p_kw = 1000, q_kvar = 200, name = "load 2")
pp.create_load(net, 5, p_kw = 1000, q_kvar = 200, name = "load 3")
pp.create_load(net, 6, p_kw = 1000, q_kvar = 200, name = "load 4")

pp.create_switch(net, bus = 1, element = 0, et = 'l')
pp.create_switch(net, bus = 2, element = 0, et = 'l')
pp.create_switch(net, bus = 2, element = 1, et = 'l')
pp.create_switch(net, bus = 3, element = 1, et = 'l')
pp.create_switch(net, bus = 3, element = 2, et = 'l')
pp.create_switch(net, bus = 4, element = 2, et = 'l')
pp.create_switch(net, bus = 4, element = 3, et = 'l', closed = 0)
pp.create_switch(net, bus = 5, element = 3, et = 'l')
pp.create_switch(net, bus = 5, element = 4, et = 'l')
pp.create_switch(net, bus = 6, element = 4, et = 'l')
pp.create_switch(net, bus = 6, element = 5, et = 'l')
pp.create_switch(net, bus = 1, element = 5, 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 spezialised topology functions. Finding the shortest path between two buses is a good example for that.

```
import pandapower.topology as top
import networkx as nx

mg = top.create_nxgraph(net)
nx.shortest_path(mg, 0, 5)
```

```
Out: [0, 1, 6, 5]
```

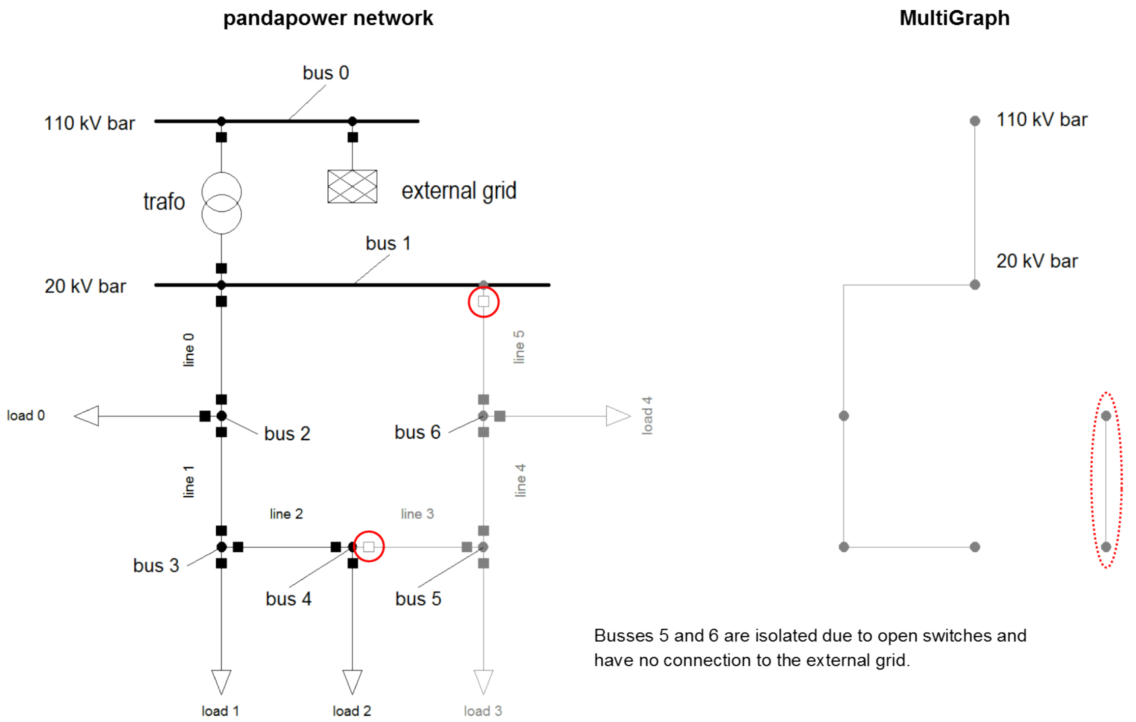


7.3.2 Find disconnected buses

With *unsupplied_buses* you can easily find buses that are not connected to an external grid.

```
import pandapower.topology as top

net.switch.closed.at[11] = 0
top.unsupplied_buses(net)
```

```
Out: {5, 6}
```



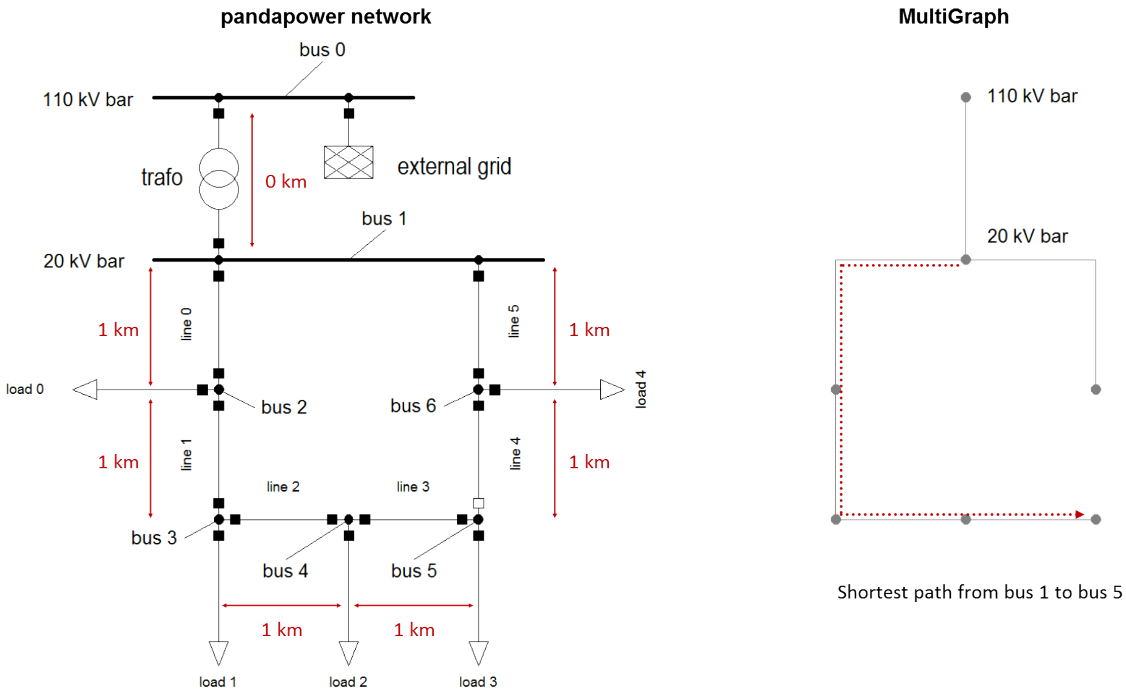
7.3.3 Calculate distances between buses

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)

```
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
```

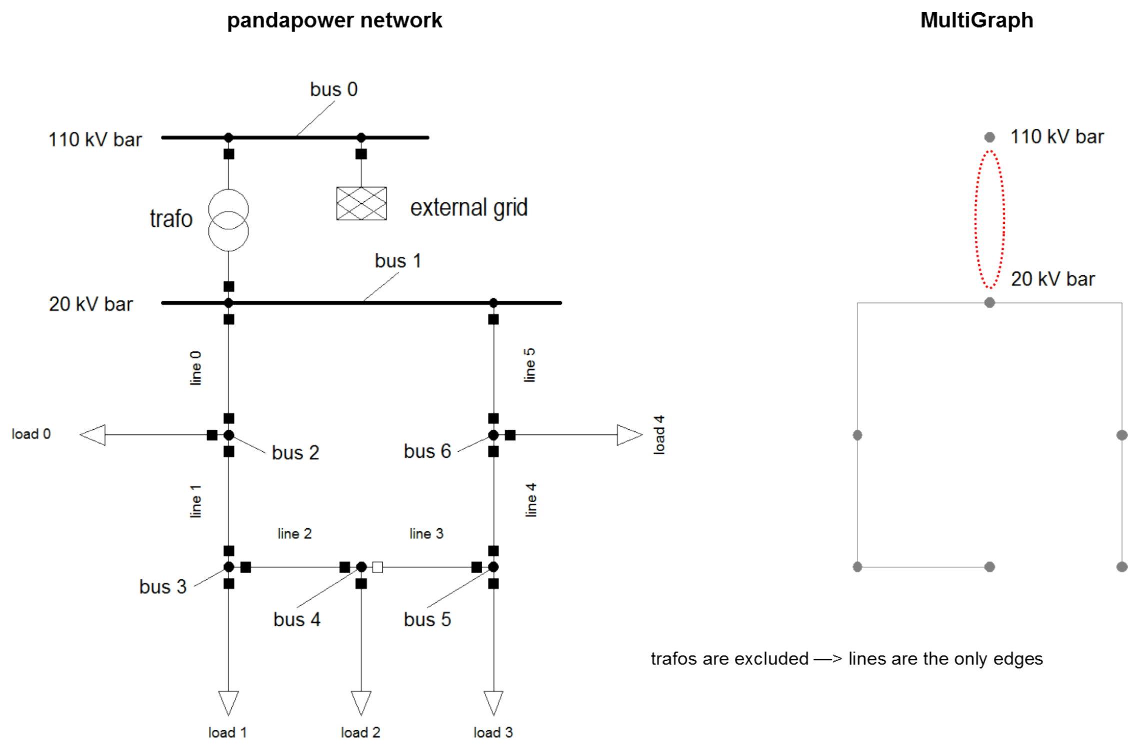
Interpretation: The distance between bus 1 and itself is 0 km. Bus 1 is also 0 km away from bus 0, since they are connected with a transformer. The shortest path between bus 1 and bus 5 is 4 km long.



7.3.4 Find connected buses with the same voltage level

```
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}
```

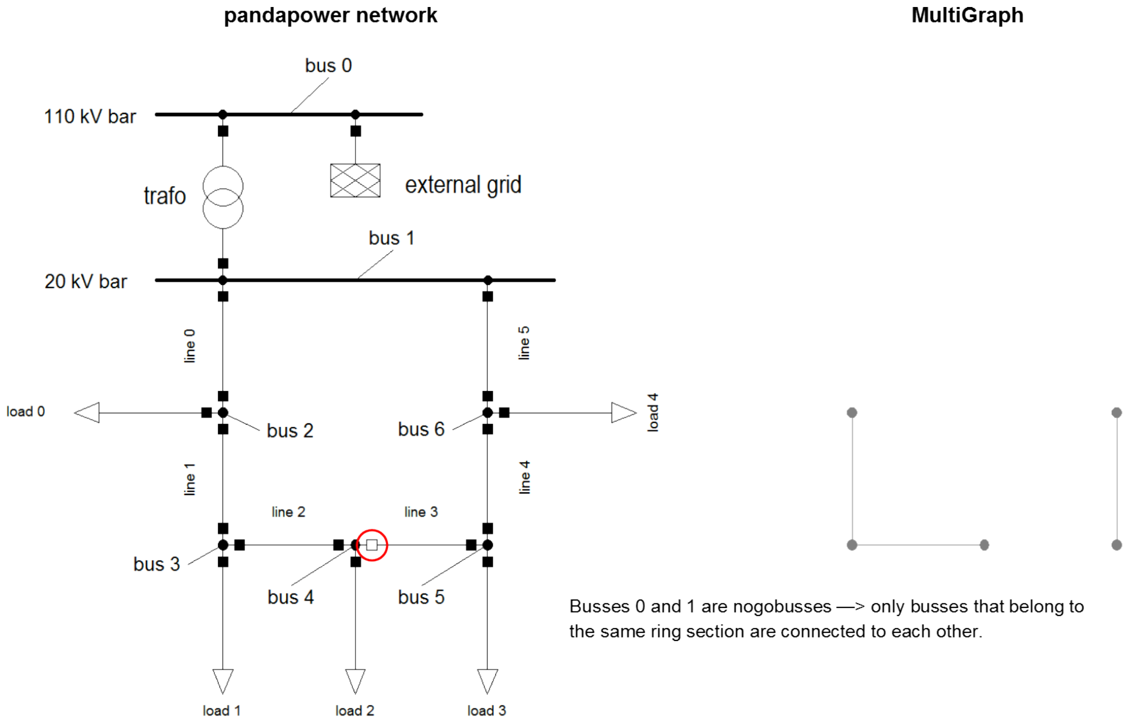


7.3.5 Find rings and ring sections

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.

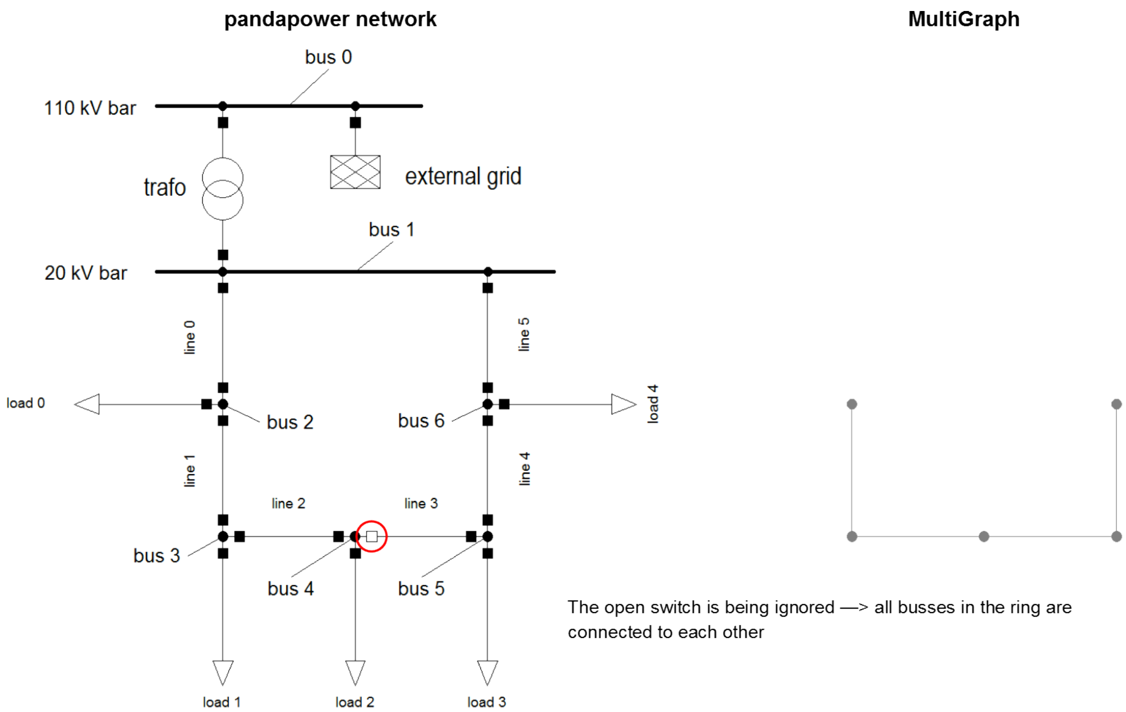
```
import pandapower.topology as top

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}
```




```
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

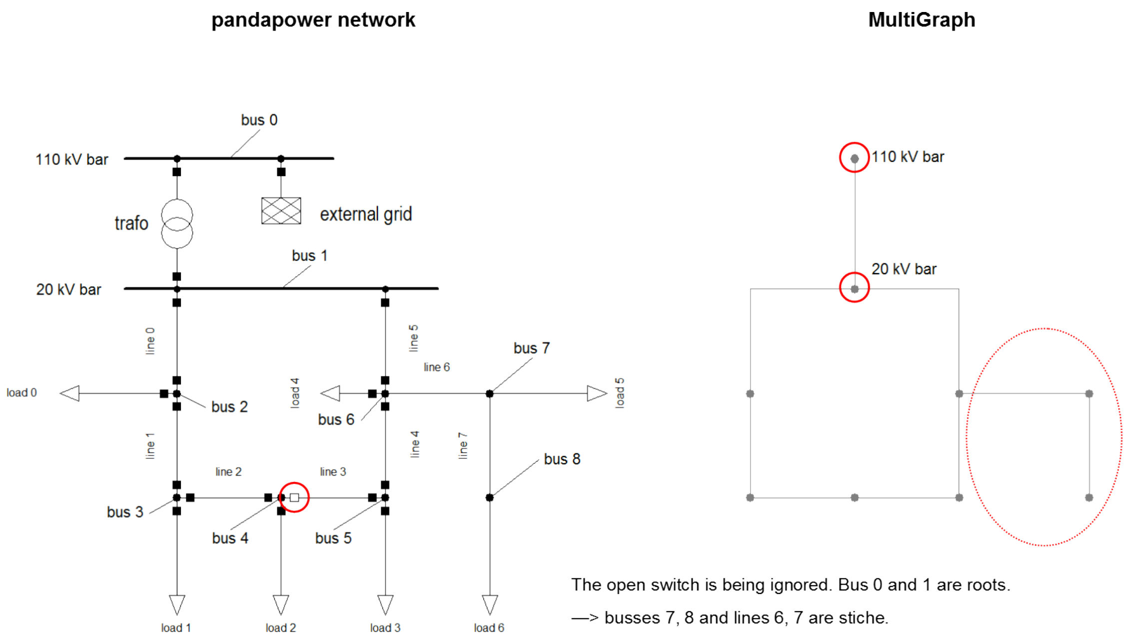
determine_stubs lets you identify buses and lines that are stubs. Open switches are being ignored. Busses that you want to exclude should be defined as roots. Ext_grid buses are roots by default.

This is a small extension for the example network:

```
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:
         name  vn_kv  min_vm_pu  max_vm_pu type  zone in_service auf_stich
0  110 kV bar    110        NaN        NaN    b  None       True     False
1   20 kV bar     20        NaN        NaN    b  None       True     False
2       bus 2     20        NaN        NaN    b  None       True     False
3       bus 3     20        NaN        NaN    b  None       True     False
4       bus 4     20        NaN        NaN    b  None       True     False
5       bus 5     20        NaN        NaN    b  None       True     False
6       bus 6     20        NaN        NaN    b  None       True     False
7       bus 7     20        NaN        NaN    b  None       True      True
8       bus 8     20        NaN        NaN    b  None       True      True

In: net.line

Out:
     name  std_type  from_bus  to_bus  length_km  r_ohm_per_km  x_ohm_per_km  c_nf_
↪per_km  max_i_ka  df type  in_service  is_stich
0  line 0  NAYY 150         1       2          1         0.206         0.091     ␣
↪     0     0.284   1   cs        True     False
1  line 1  NAYY 150         2       3          1         0.206         0.091     ␣
↪     0     0.284   1   cs        True     False
2  line 2  NAYY 150         3       4          1         0.206         0.091     ␣
↪     0     0.284   1   cs        True     False
3  line 3  NAYY 150         4       5          1         0.206         0.091     ␣
↪     0     0.284   1   cs        True     False
4  line 4  NAYY 150         5       6          1         0.206         0.091     ␣
↪     0     0.284   1   cs        True     False
5  line 5  NAYY 150         6       1          1         0.206         0.091     ␣
↪     0     0.284   1   cs        True     False
6  line 6  NAYY 150         6       7          1         0.206         0.091     ␣
↪     0     0.284   1   cs        True      True
7  line 7  NAYY 150         7       8          1         0.206         0.091     ␣
↪     0     0.284   1   cs        True      True
```


pandapower network
MultiGraph
bus 0
110 kV bar
trafo
external grid
bus 1
20 kV bar
line 0
line 5
line 6
bus 7
load 0
bus 2
load 4
bus 6
load 5
line 1
line 4
line 7
bus 8
line 2
line 3
bus 3
bus 4
bus 5
load 1
load 2
load 3
load 6
110 kV bar
20 kV bar
The open switch is being ignored. Bus 0 and 1 are roots.
—> busses 7, 8 and lines 6, 7 are stiche.

8 Generic Networks

Besides creating your own grids through the pandapower API pandapower provides generic networks through the networks module. The pandapower networks modul contains simple test networks, randomly generated networks, CIGRE test networks, IEEE case files and generic networks from the dissertation of Georg Kerber.

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.

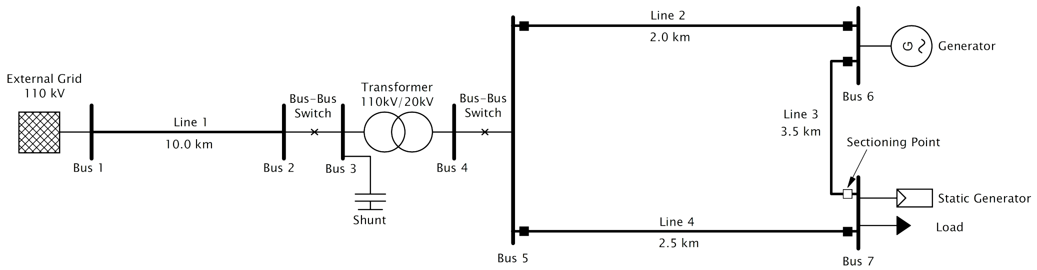
`pandapower.networks.`**`example_simple`**`()`

Returns the simple example network from the pandapower tutorials.

OUTPUT: net - simple example network

EXAMPLE:

```
>>> 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.

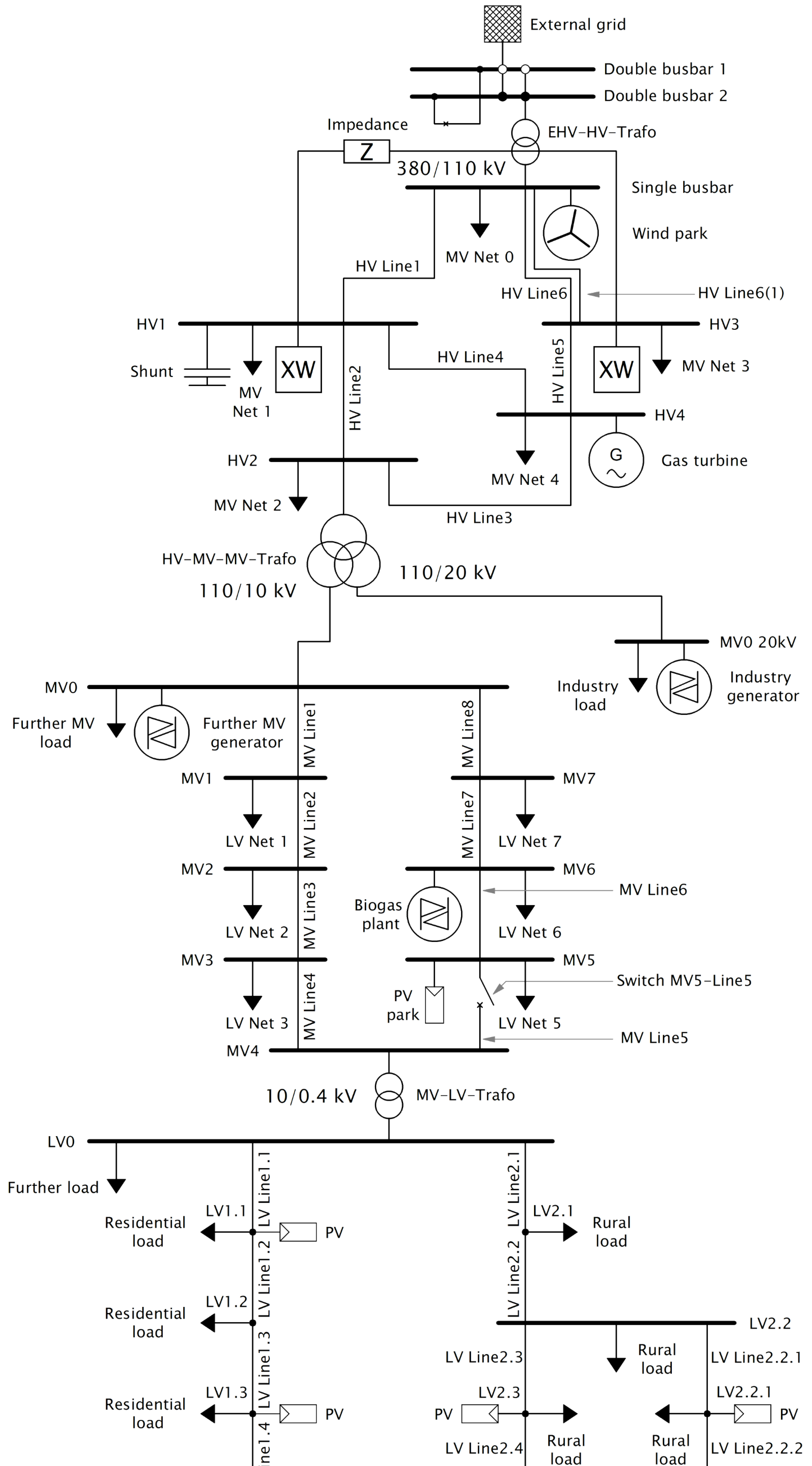
`pandapower.networks.`**`example_multivoltage`**`()`

Returns the multivoltage example network from the pandapower tutorials.

OUTPUT: net - multivoltage example network

EXAMPLE:

```
>>> import pandapower.networks
>>> net = pandapower.networks.example_multivoltage()
```


External grid
Double busbar 1
Double busbar 2
Impedance
Z
EHV-HV-Trafo
380/110 kV
Single busbar
Wind park
MV Net 0
HV Line1
HV Line6
HV Line6(1)
HV1
HV3
Shunt
MV
Net 1
XW
HV Line2
HV Line4
HV Line5
XW
MV Net 3
HV4
HV2
G
Gas turbine
MV Net 4
MV Net 2
HV Line3
HV-MV-MV-Trafo
110/10 kV
110/20 kV
MV0 20kV
Industry
load
Industry
generator
MV0
Further MV
load
Further MV
generator
MV Line1
MV Line8
MV1
MV7
MV Line2
MV Line7
LV Net 1
LV Net 7
MV2
MV6
MV Line6
Biogas
plant
MV Line3
LV Net 2
LV Net 6
MV3
MV5
Switch MV5-Line5
PV
park
MV Line4
LV Net 3
LV Net 5
MV Line5
MV4
10/0.4 kV
MV-LV-Trafo
LV0
Further load
LV Line1.1
LV Line2.1
LV1.1
Residential
load
PV
LV2.1
Rural
load
LV Line1.2
LV Line2.2
LV1.2
Residential
load
LV2.2
LV Line1.3
LV Line2.3
Rural
load
LV Line2.2.1
LV1.3
Residential
load
PV
LV2.3
PV
LV2.2.1
PV
LV Line1.4
LV Line2.4
Rural
load
Rural
load
LV Line2.2.2

The stepwise creation of this network is shown in the pandapower tutorials.

8.2 Simple pandapower test networks

8.2.1 Four load branch

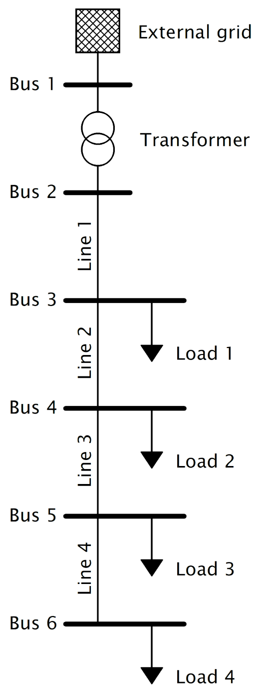
`pandapower.networks.`**`panda_four_load_branch`**`()`

This function creates a simple six bus system with four radial low voltage nodes connected to a medium valtage slack bus. At every low voltage node the same load is connected.

OUTPUT: **net** - Returns the required four load system

EXAMPLE: import pandapower.networks as pn

net_four_load = pn.panda_four_load_branch()



8.2.2 Four loads with branches out

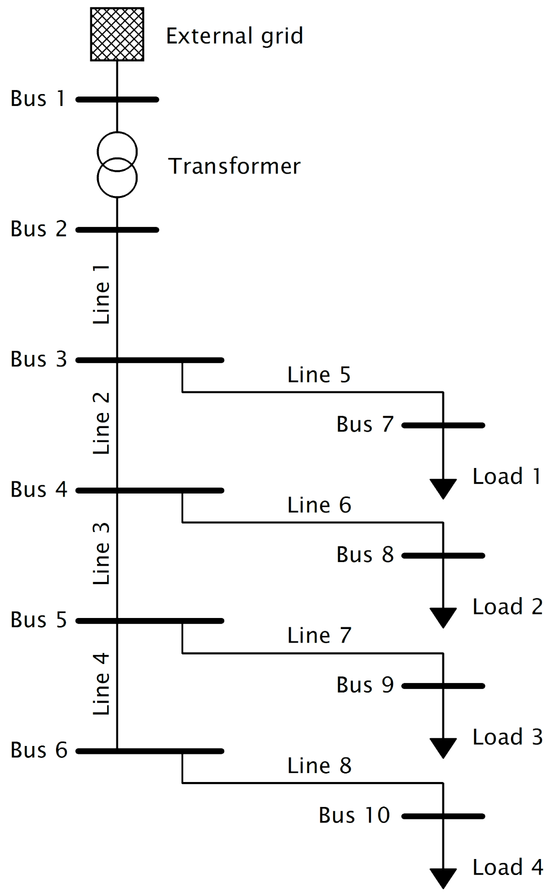
`pandapower.networks.`**`four_loads_with_branches_out`**`()`

This function creates a simple ten bus system with four radial low voltage nodes connected to a medium valtage 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: import pandapower.networks as pn

net_four_load_with_branches = pn.four_loads_with_branches_out()



8.2.3 Four bus system

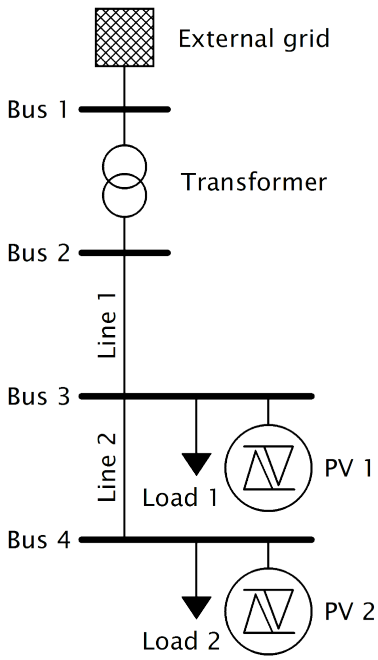
pandapower.networks.**simple_four_bus_system**()

This function creates a simple four bus system with two radial low voltage nodes connected to a medium valtage 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

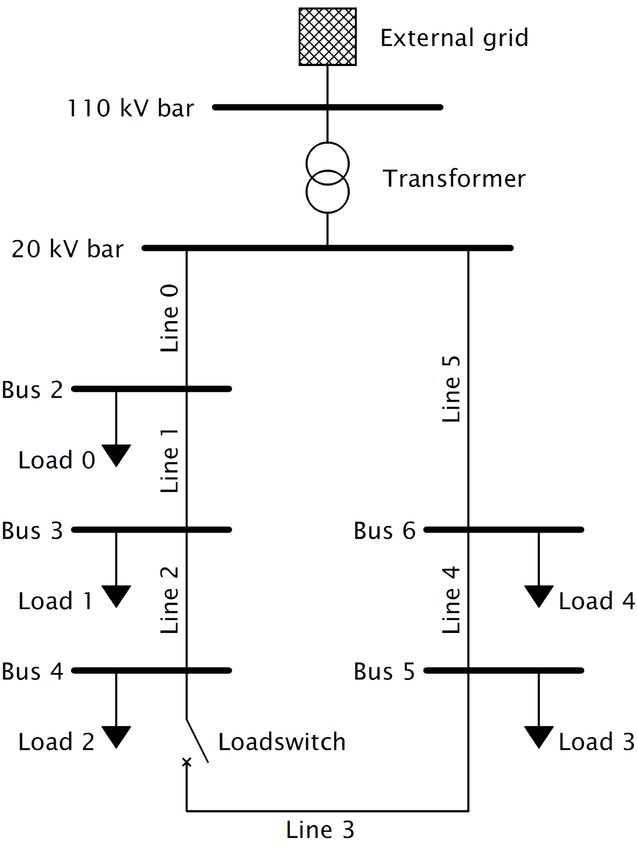
`pandapower.networks.simple_mv_open_ring_net()`

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".

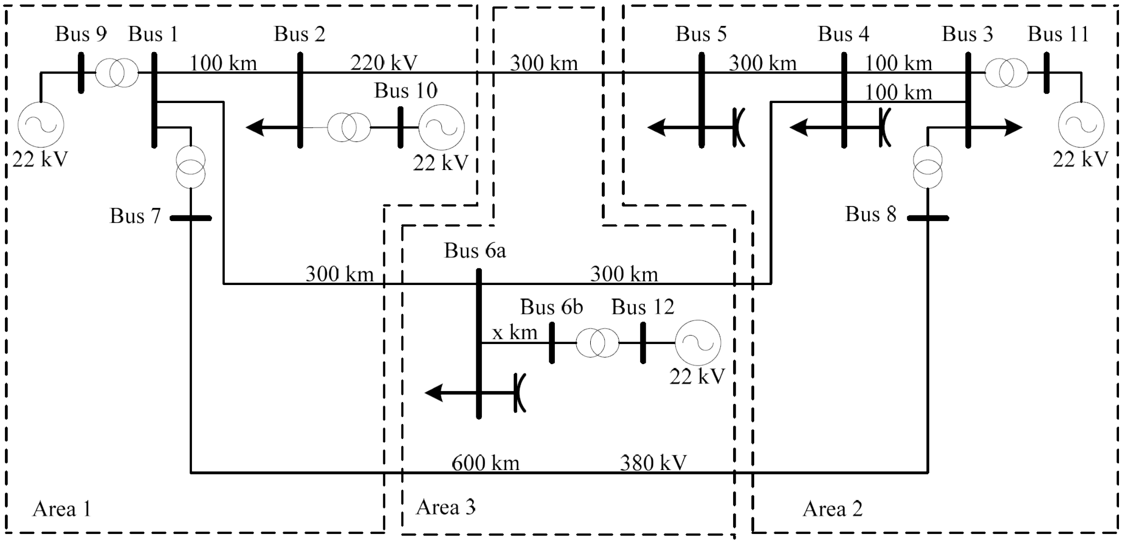
Note: Source for this network is the final Report of Task Force C6.04.02: "Benchmark Systems for Network Integration of Renewable and Distributed Energy Resources", 2014

See also a correlating Paper with tiny changed network parameters: K. Rudion, A. Orths, Z. A. Styczynski and K. Strunz, Design of benchmark of medium voltage distribution network for investigation of DG integration 2006 IEEE Power Engineering Society General Meeting, Montreal, 2006

8.3.1 High voltage transmission network

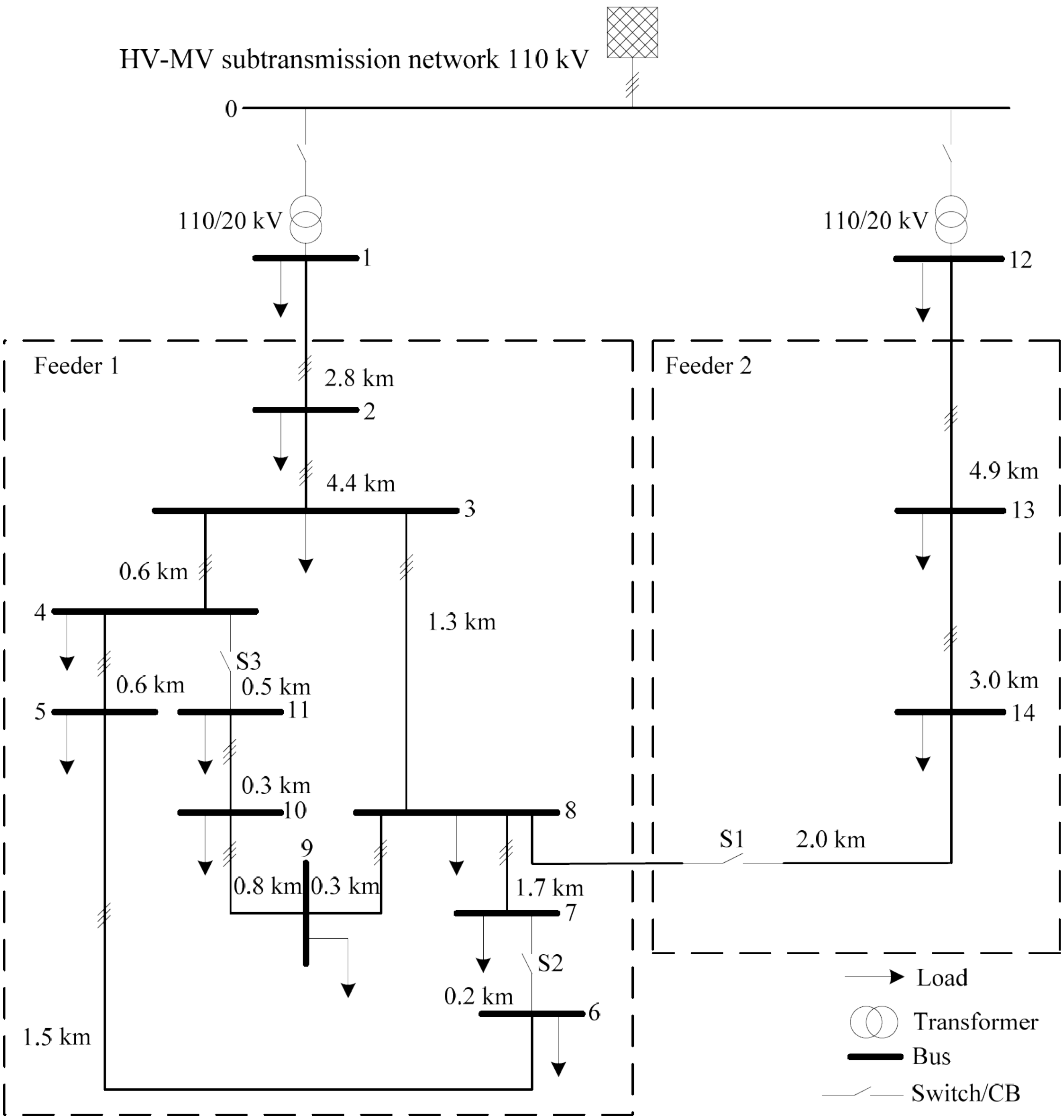
```
import pandapower.networks as pn

# 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)
  - bus (13 elements)
```

```
  - line (9 elements)
  - load (5 elements)
  - ext_grid (1 elements)
  - gen (3 elements)
'''
```



8.3.2 Medium voltage distribution network

```
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)
'''
```



8.3.3 Medium voltage distribution network with PV and Wind DER

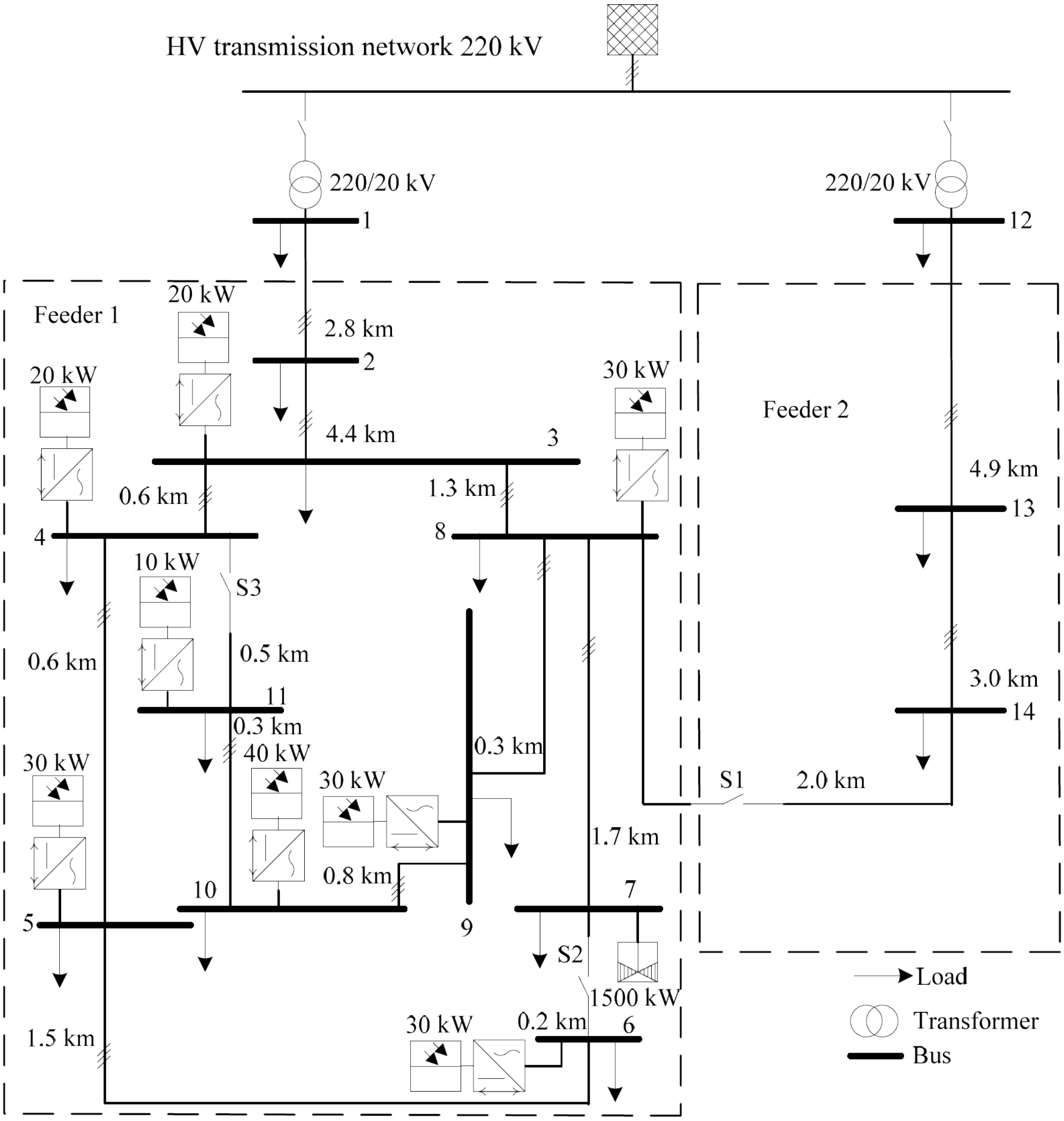
Note: This network contains additional 9 distributed energy resources compared to medium voltage distribution network:

- 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.

```
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 elements)
  - sgen (9 elements)
  - line (15 elements)
  - trafo (2 elements)
  - bus (15 elements)
'''
```



8.3.4 Medium voltage distribution network with all DER

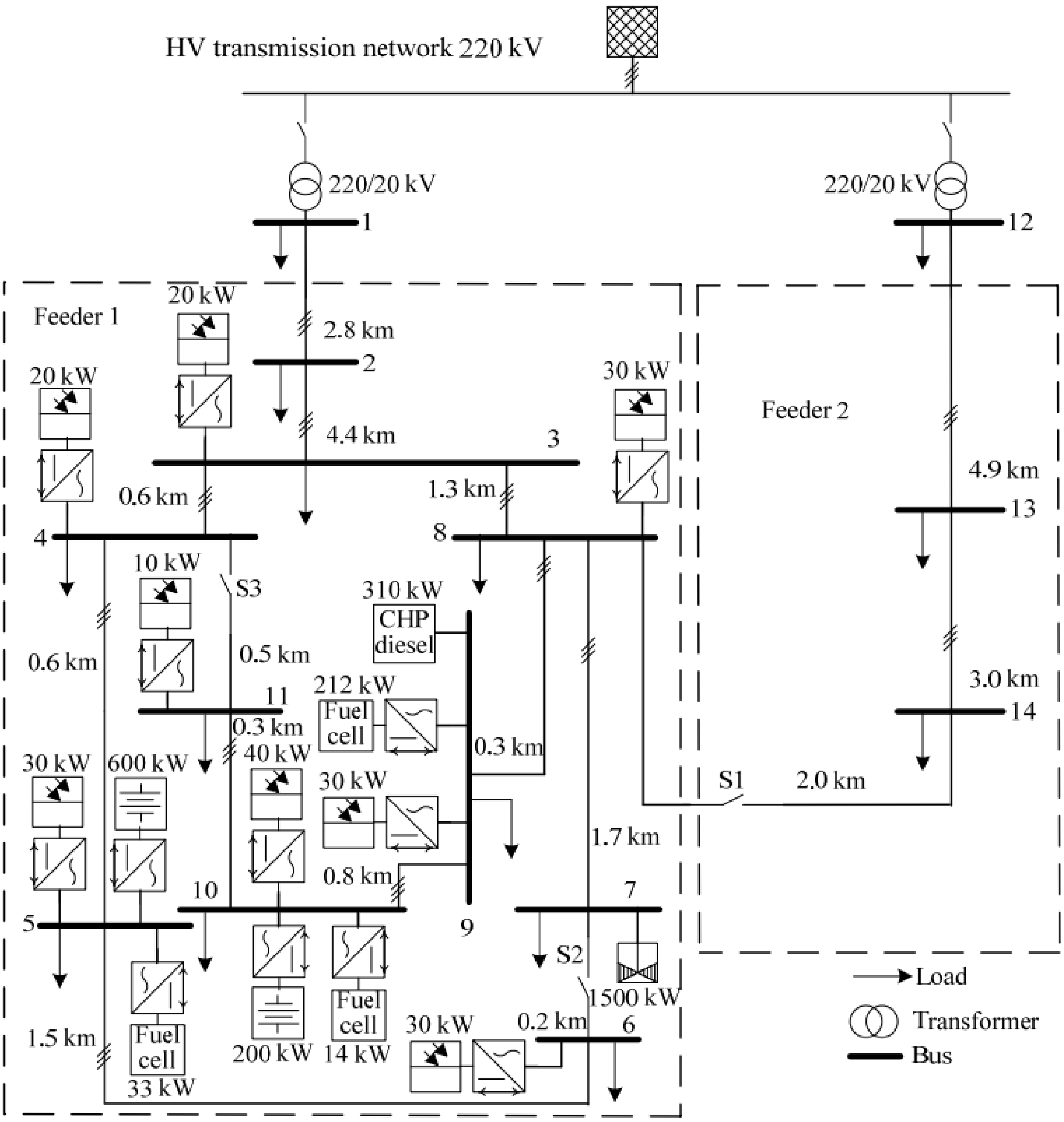
Note: This network contains additional 15 distributed energy resources compared to medium voltage distribution network:

- 8 photovoltaic generators
- 1 wind turbine
- 2 Batteries

- 2 residential fuel cells
- 1 CHP diesel
- 1 CHP fuel cell

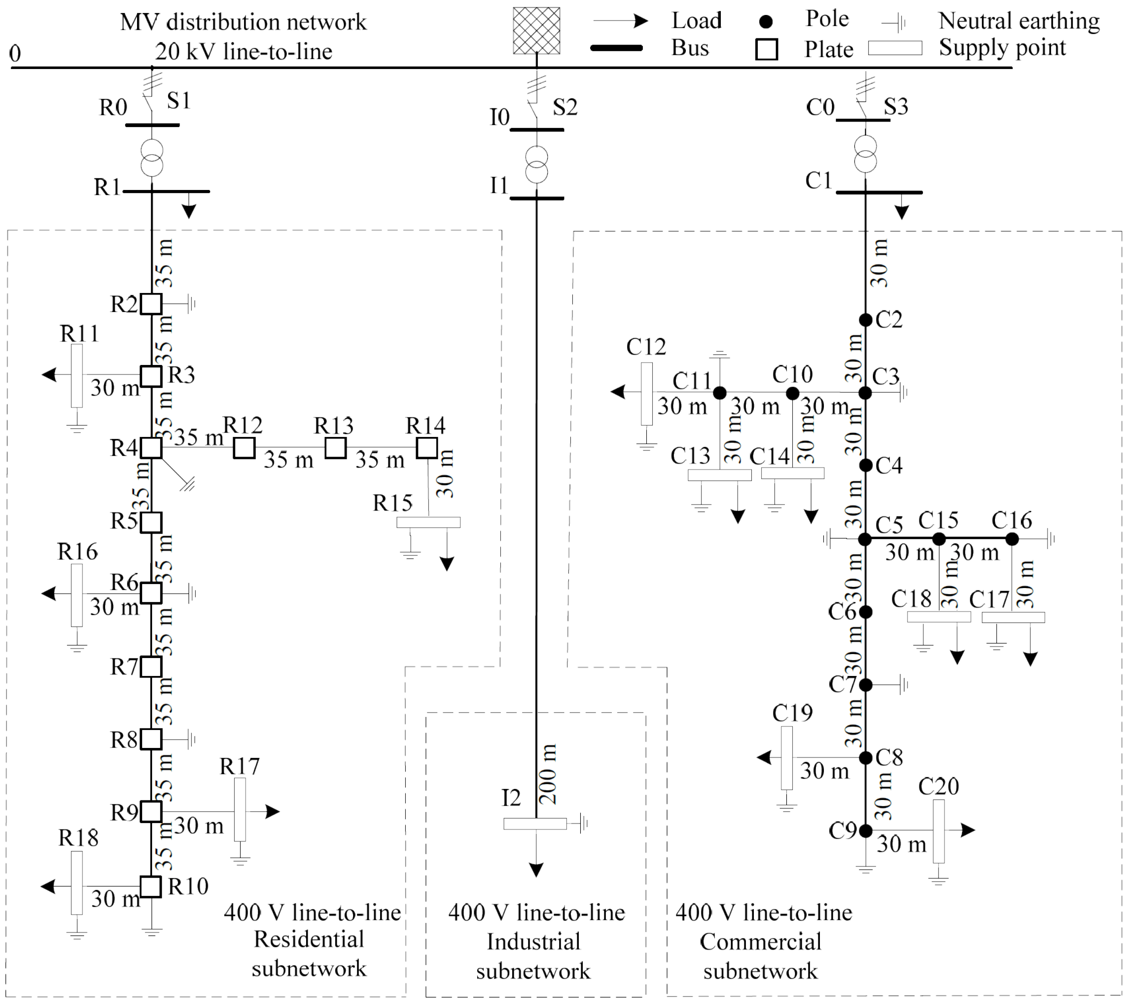
Compared to the case study of CIGRE Task Force C6.04.02 paper all distributed energy resources are considered. Although the case study mentions the High Voltage as 220 kV, we assume 110 kV again because of no given 220 kV-Trafo data.

```
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)
'''
```



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)
'''
```



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

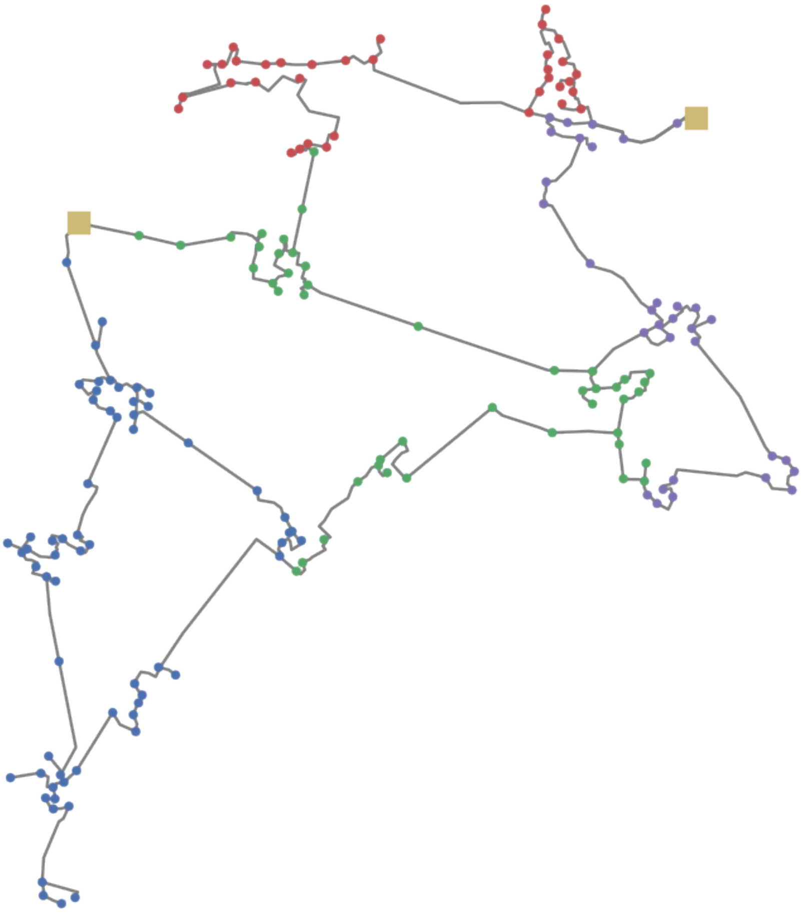
include_substations - (bool, False): if True, the transformers of the MV/LV level are modelled, otherwise the loads representing the LV networks are connected directly to the MV node

OUTPUT: **net** - pandapower network

EXAMPLE:

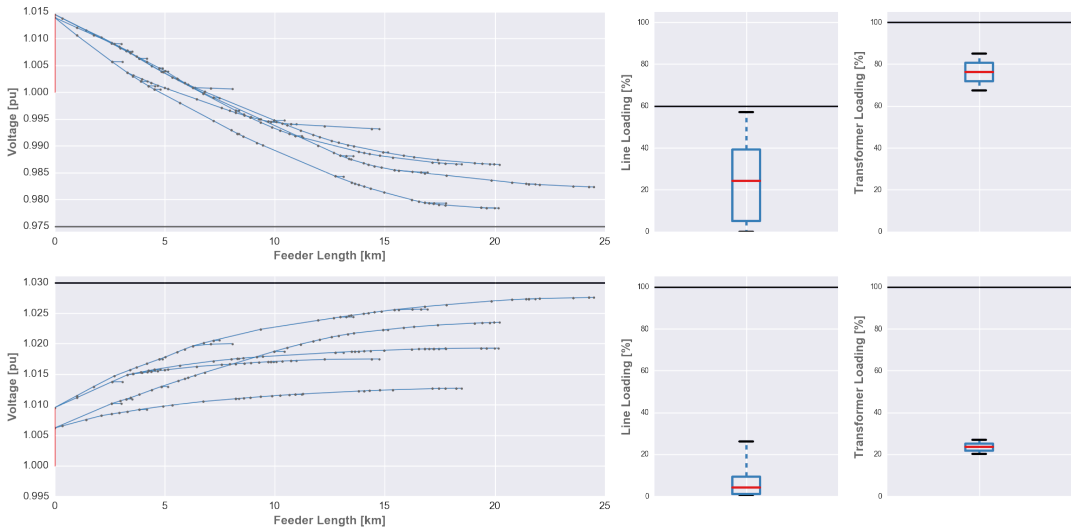
```
>>> import pandapower.networks
>>> net = pandapower.networks.mv_oberrhein("generation")
```

The geographical representation of the network looks like this:



The different colors of the MV/LV stations indicate the feeders which are galvanically seperated 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:



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

`pandapower.networks.case4gs()`

Calls the pickle file case4gs.p which data origin is PYPOWER.

OUTPUT: **net** - Returns the required ieee network case4gs

EXAMPLE: import pandapower.networks as pn

net = pn.case4gs()

8.5.2 Case 6ww

`pandapower.networks.case6ww()`

Calls the pickle file case6ww.p which data origin is PYPOWER.

OUTPUT: **net** - Returns the required ieee network case6ww

EXAMPLE: import pandapower.networks as pn

net = pn.case6ww()

8.5.3 Case 9

`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

net = pn.case9()

8.5.4 Case 14

`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()

Case 24_ieee_rts

`pandapower.networks.case24_ieee_rts()`

Calls the pickle file case24_ieee_rts.p which data origin is PYPOWER. Some more information about this network are given by Illinois University case 24.

OUTPUT: **net** - Returns the required ieee network case24

EXAMPLE: import pandapower.networks as pn

net = pn.case24_ieee_rts()

8.5.5 Case 30

`pandapower.networks.case30()`

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. 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 Pypower 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 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()

8.5.10 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()

8.5.11 Case 1354pegase

`pandapower.networks.case1354pegase()`

Calls the pickle file case1354pegase.p which 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.

OUTPUT: **net** - Returns the required ieee network case1354pegase

EXAMPLE: import pandapower.networks as pn

net = pn.case1354pegase()

8.5.12 Case 2869pegase

`pandapower.networks.case2869pegase()`

Calls the pickle file case33bw.p which 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.

OUTPUT: **net** - Returns the required ieee network case2869pegase

EXAMPLE: import pandapower.networks as pn

net = pn.case300()

8.5.13 Case 9241pegase

`pandapower.networks.case9241pegase()`

Calls the pickle file case33bw.p which 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.

OUTPUT: **net** - Returns the required ieee network case9241pegase

EXAMPLE: import pandapower.networks as pn

net = pn.case9241pegase()

8.5.14 Case GB network

`pandapower.networks.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: import pandapower.networks as pn

net = pn.GBnetwork()

8.5.15 Case GB reduced network

`pandapower.networks.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: import pandapower.networks as pn

net = pn.GBreducednetwork()

8.5.16 Case iceland

`pandapower.networks.iceland()`

Calls the pickle file iceland.p which data is provided by W. A. Bukhsh, Ken McKinnon, Network data of real 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: 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 Niederspannungsverteilnetzen für die Einspeisung aus Photvoltaikanlagen" (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 Photovoltaikkleinanlagen, Dissertation

- Georg Kerber, Statistische Analyse von NS-Verteilungsnetzen und Modellierung von Referenznetzen

	Lines	Total Length	Loads	Installed Power
Kerber Landnetze				
Freileitung 1	13	0.273 km	13	104 kW
Freileitung 2	8	0.390 km	8	64 kW
Kabel 1	16	1.046 km	8	64 kW
Kabel 2	28	1.343 km	14	112 kW
Kerber Dorfnetz	114	3.412 km	57	342 kW
Kerber Vorstadtnetze				
Kabel 1	292	4.476 km	146	292 kW
Kabel 2	288	4.689 km	144	288 kW

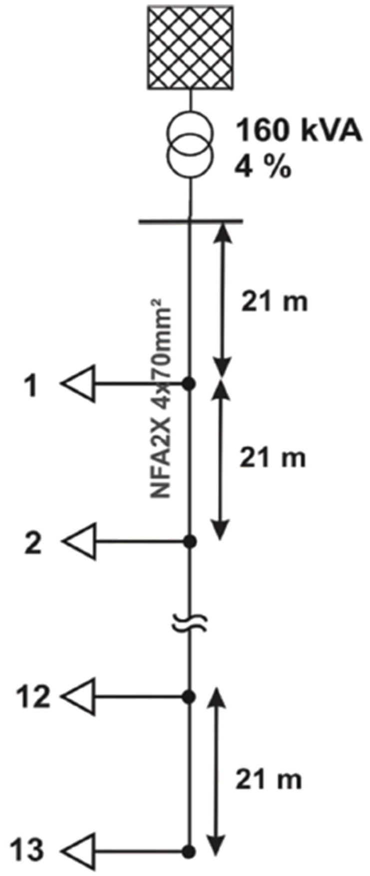
You can include the kerber networks by simply using:

```
import pandapower.networks as pn

net1 = pn.create_kerber_net()
```

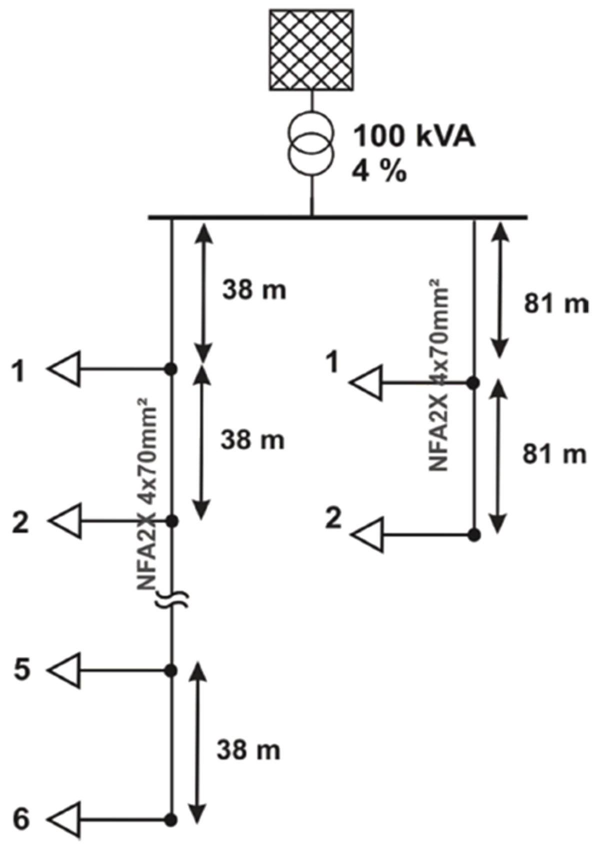
8.6.2 Kerber Landnetze

```
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)
'''
```

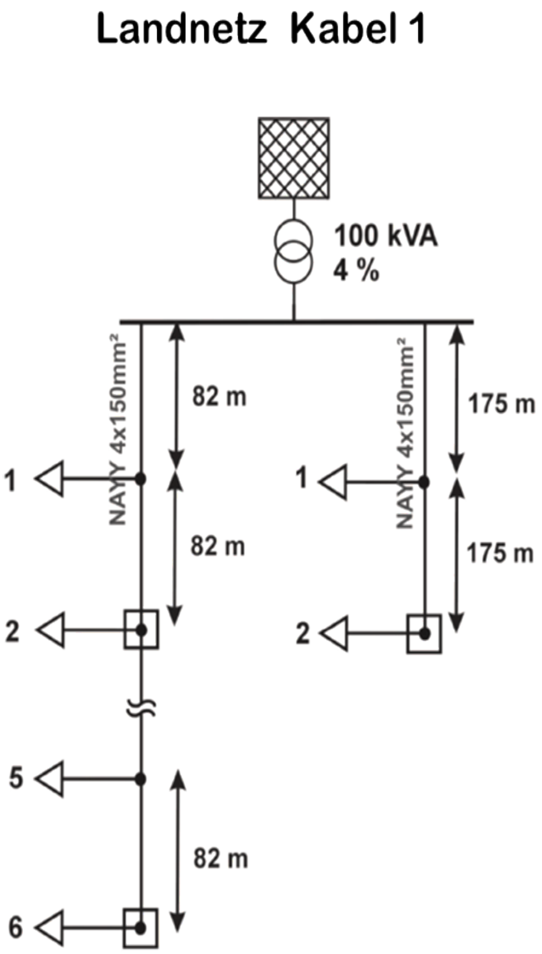
Landnetz Freileitung 1

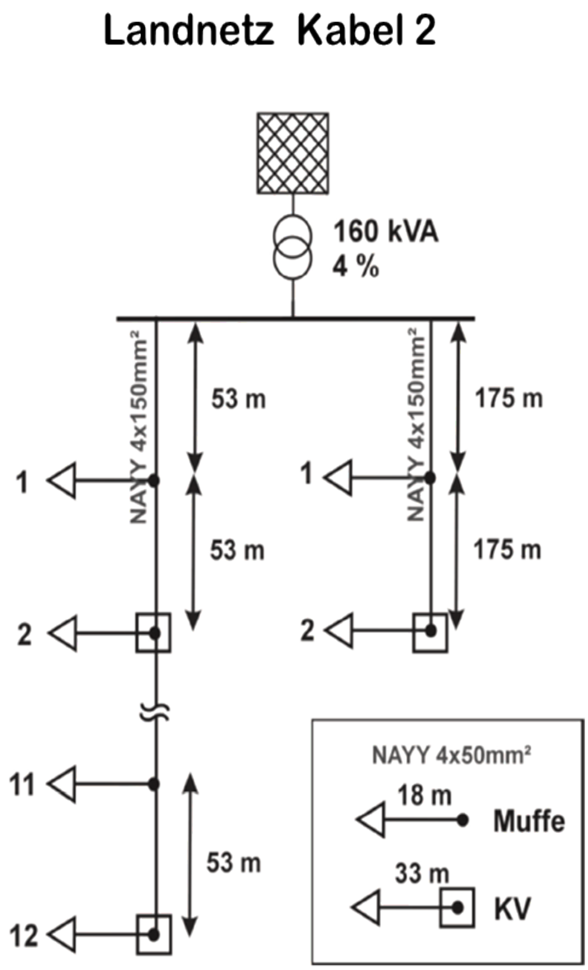


Landnetz Freileitung 2



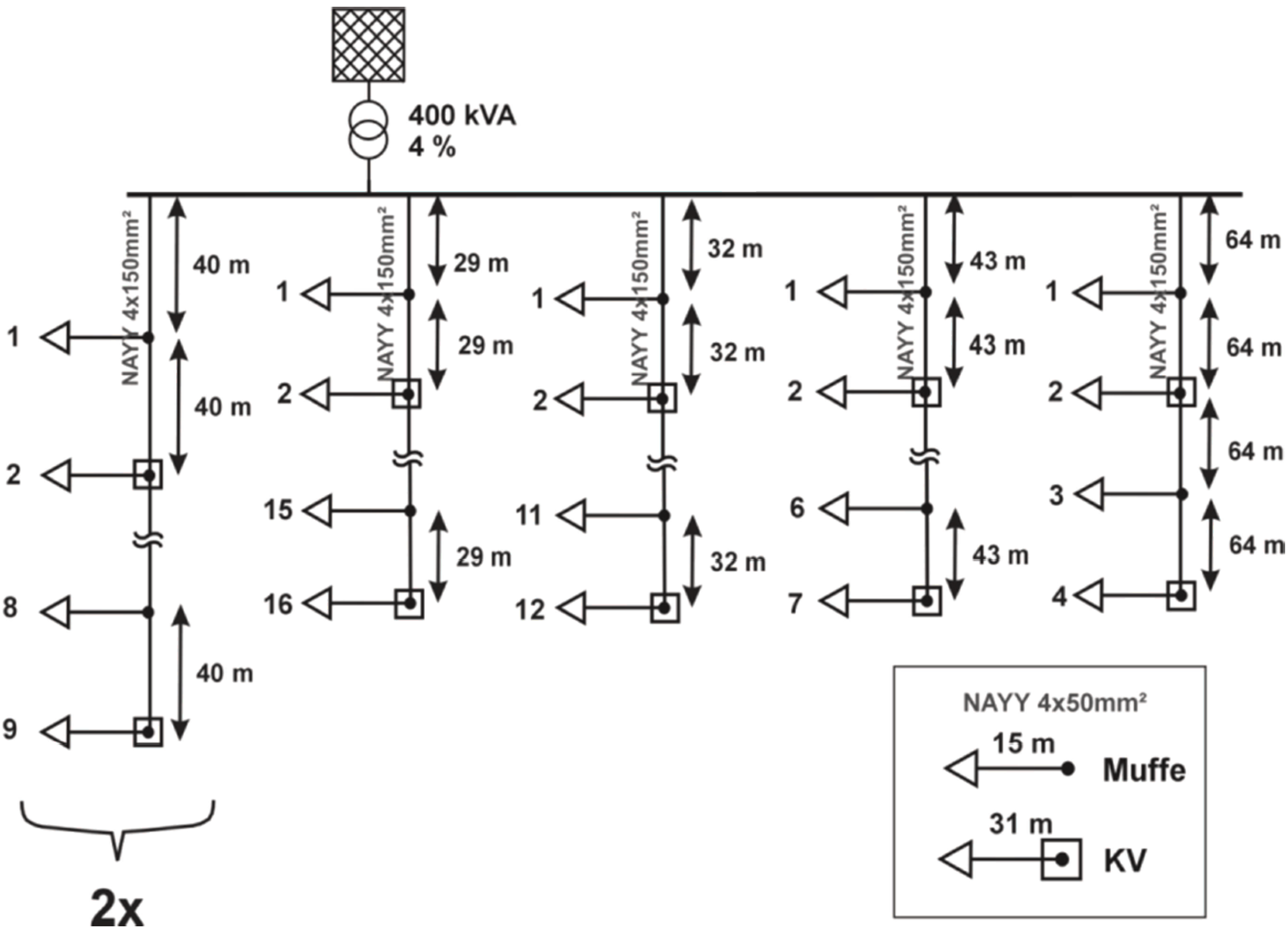

```
import pandapower.networks as pn

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)
'''
```





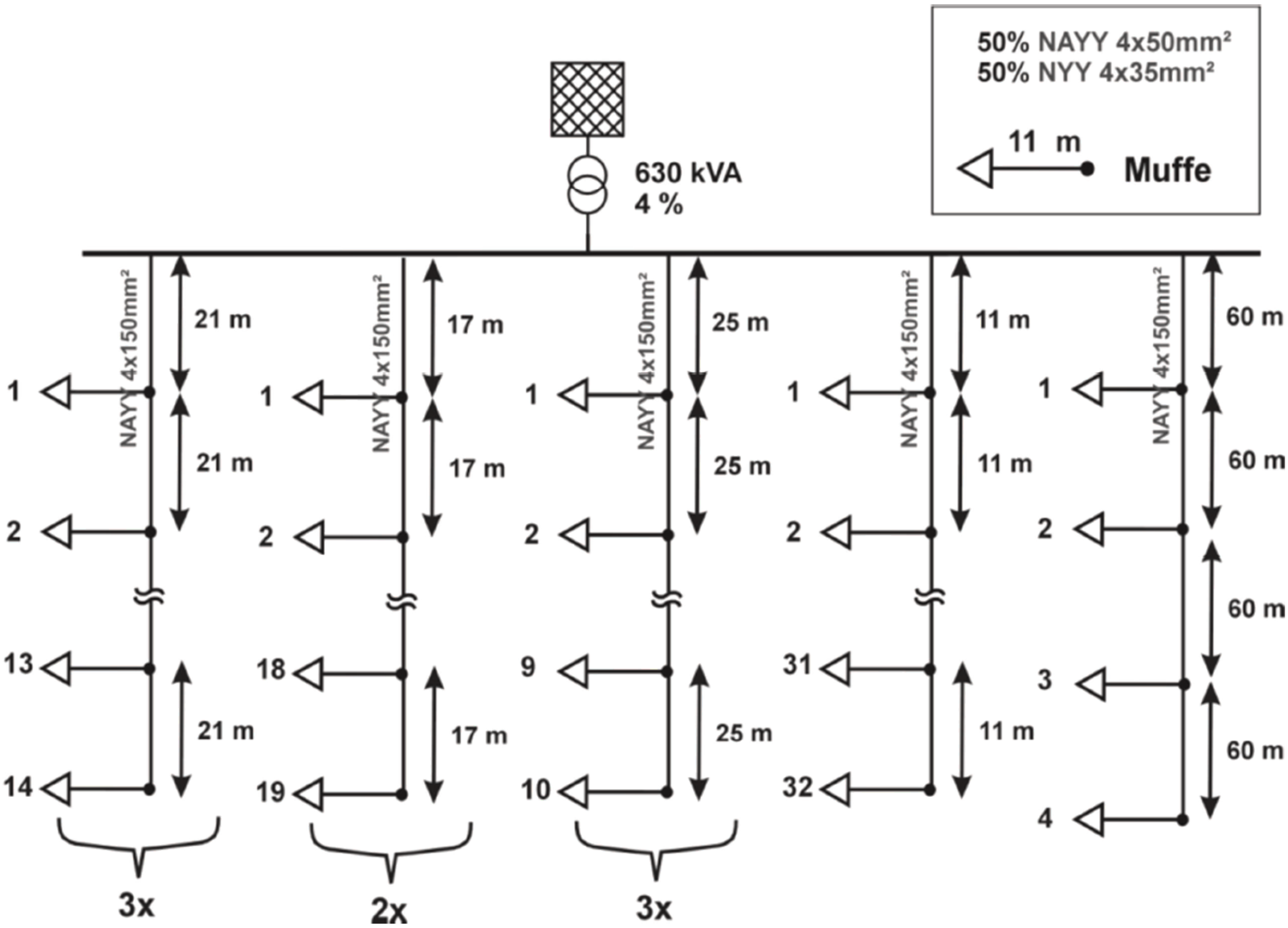
8.6.3 Kerber Dorfnetz

```
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)
'''
```

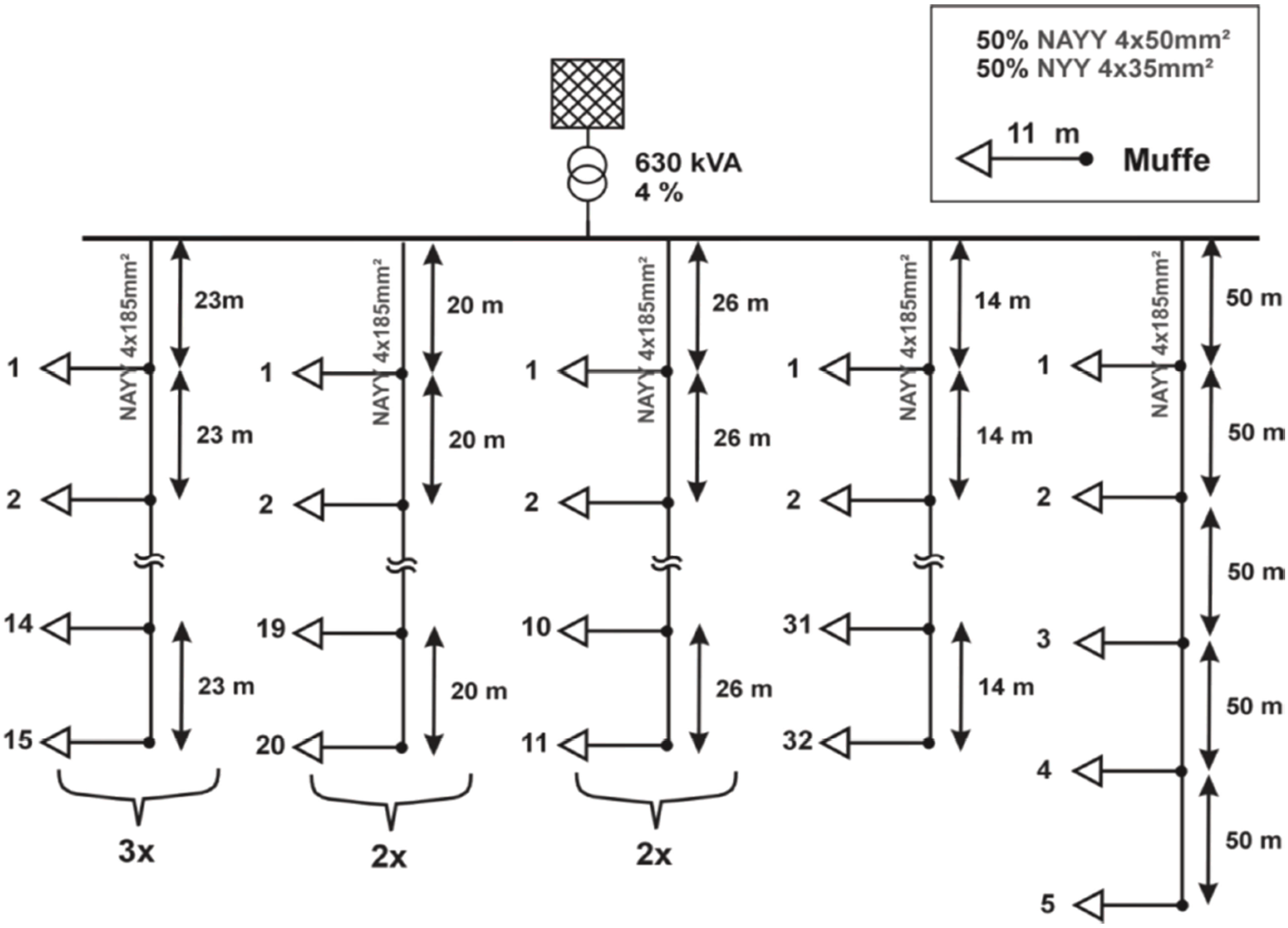


8.6.4 Kerber Vorstadtnetze

```
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_load_in_kw=2,  q_load_in_kw=0
  - bus (290 elements)
  - line (288 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)
'''
```



8.6.5 Extreme Kerber networks

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

	Lines	Total Length	Loads	Installed Power
Kerber Landnetze				
Freileitung 1	26	0.312 km	26	208 kW
Freileitung 2	27	0.348 km	27	216 kW
Kabel 1	52	1.339 km	26	208 kW
Kabel 2	54	1.435 km	27	216 kW

	Lines	Total Length	Loads	Installed Power
Kerber Dorfnetze				
Kabel 1	116	3.088 km	58	348 kW
Kabel 2	234	6.094 km	117	702 kW
Vorstadtnetze				
Kabel_a Type 1	290	3.296 km	145	290 kW
Kabel_b Type 1	290	4.019 km	145	290 kW
Kabel_c Type 2	382	5.256 km	191	382 kW
Kabel_d Type 2	384	5.329 km	192	384 kW

The Kerber extreme networks are categorized into two groups:

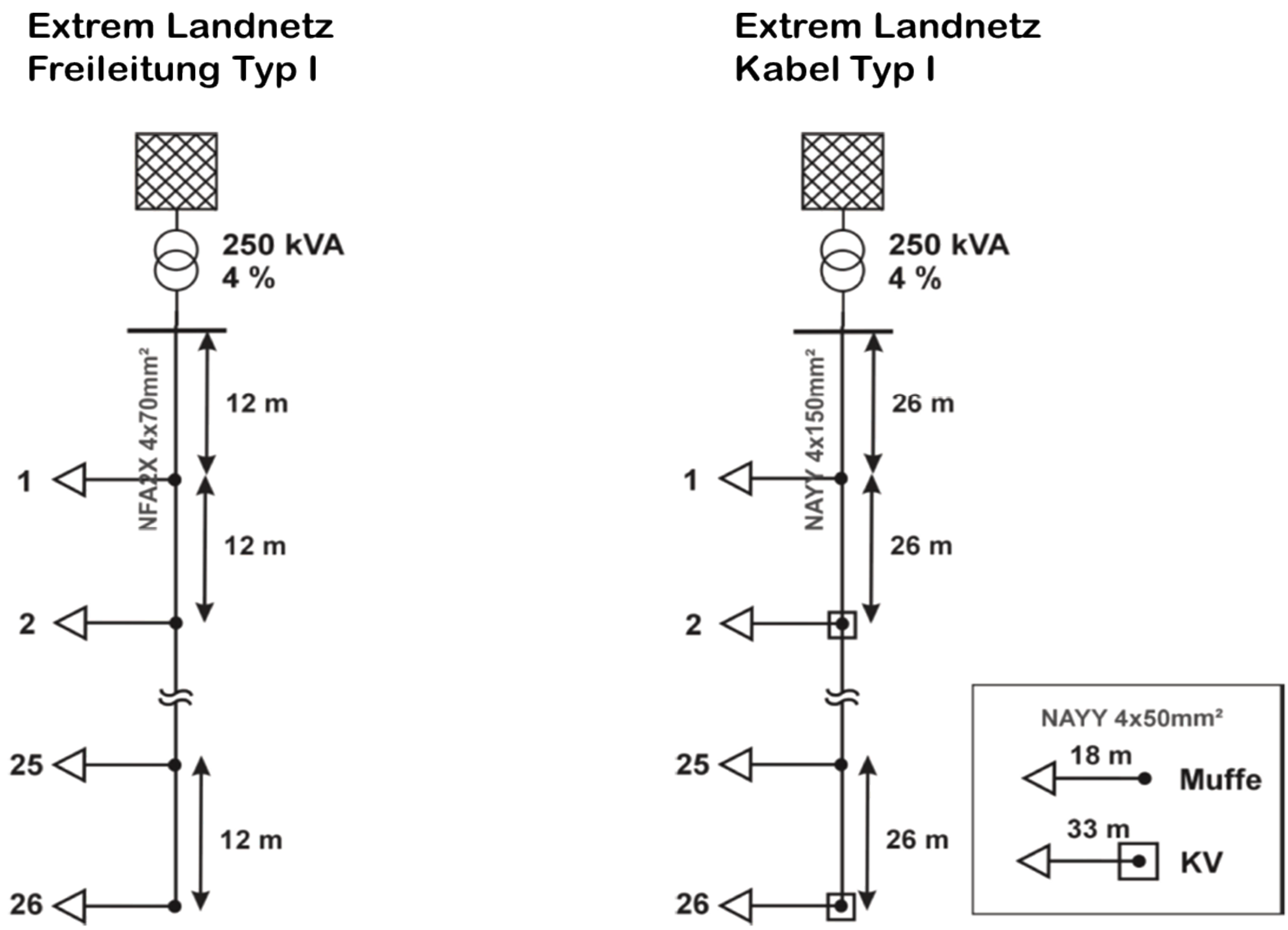
Type I: Kerber networks with extreme lines

Type II: Kerber networks with extreme lines and high loaded transformer

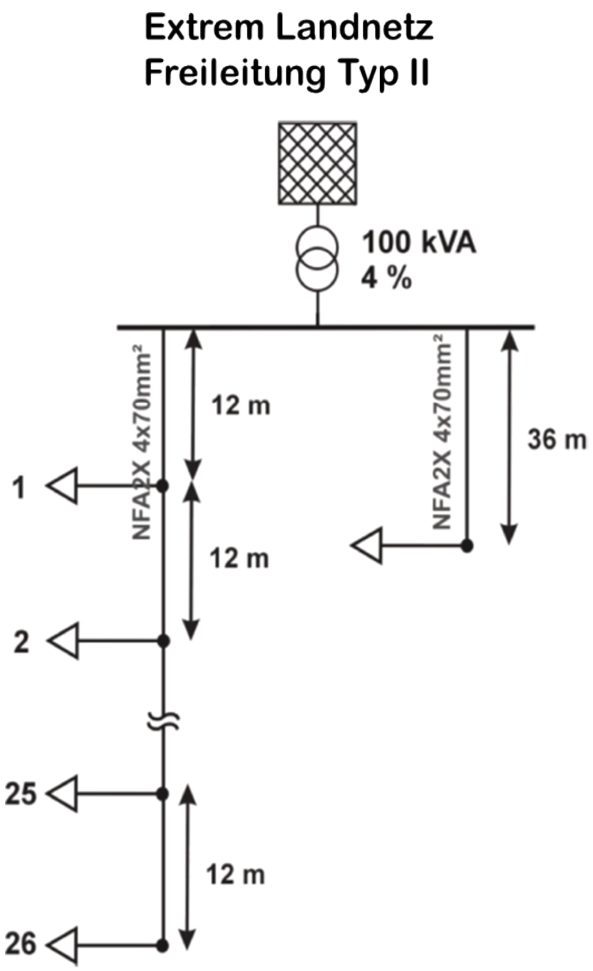
Note: Note that all Kerber exteme networks (no matter what type / territory) consist of various branches, linetypes or line length.

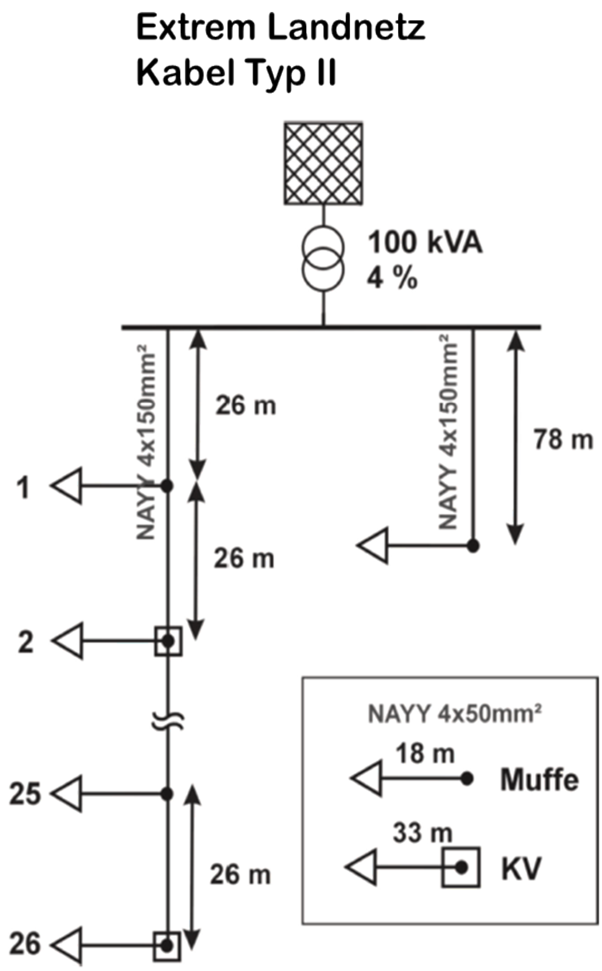
8.6.6 Extreme Kerber Landnetze

```
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()
```



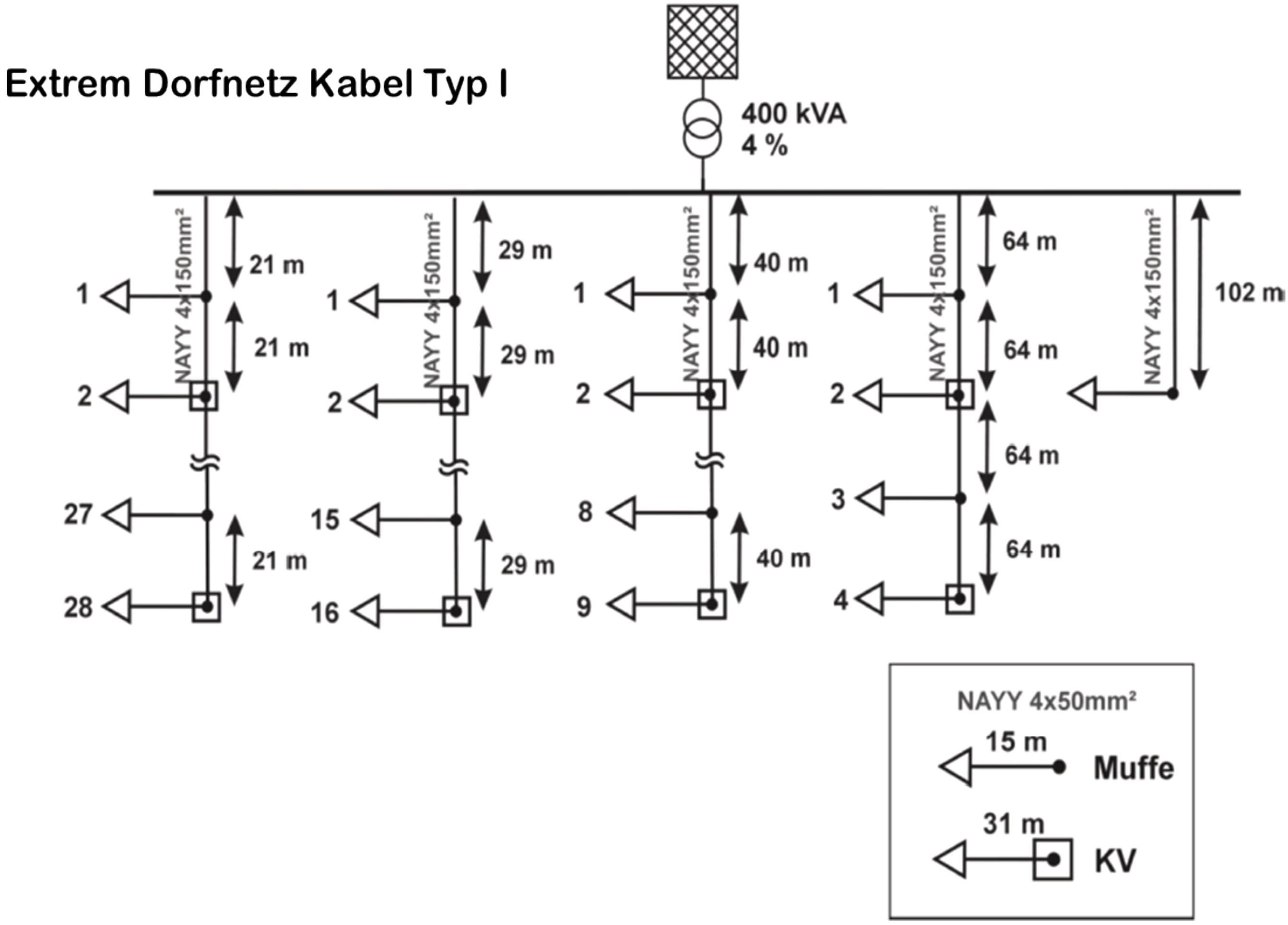

```
import pandapower.networks as pn

'''Extrem Landnetz Freileitung Typ II'''
net = pn.kb_extrem_landnetz_freileitung_trafo()


'''Extrem Landnetz Kabel Typ II'''
net = pn.kb_extrem_landnetz_kabel_trafo()
```



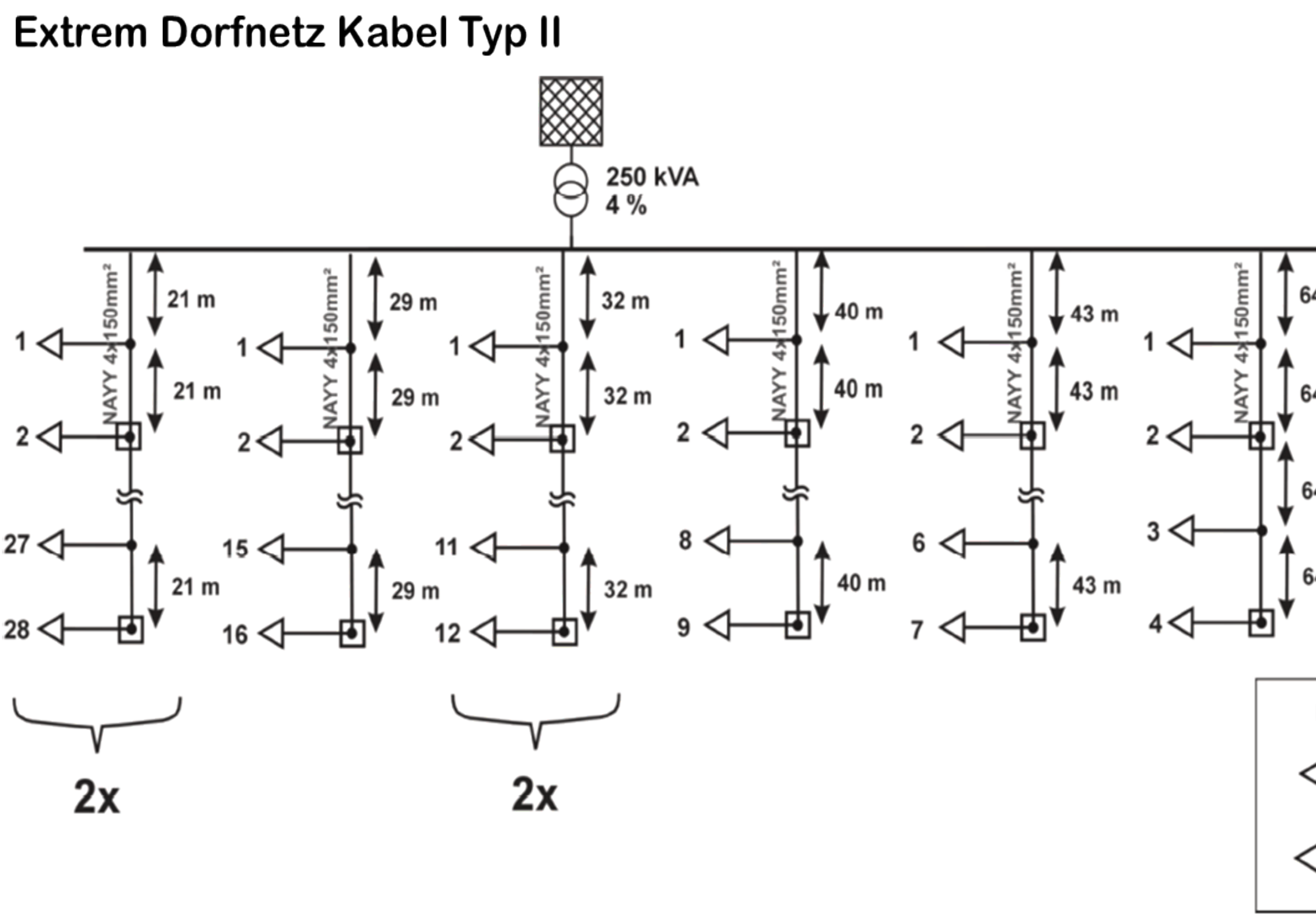


8.6.7 Extreme Kerber Dorfnetze

```
import pandapower.networks as pn

'''Extrem Dorfnetz Kabel Typ I'''
net = pn.kb_extrem_dorfnetz()
```



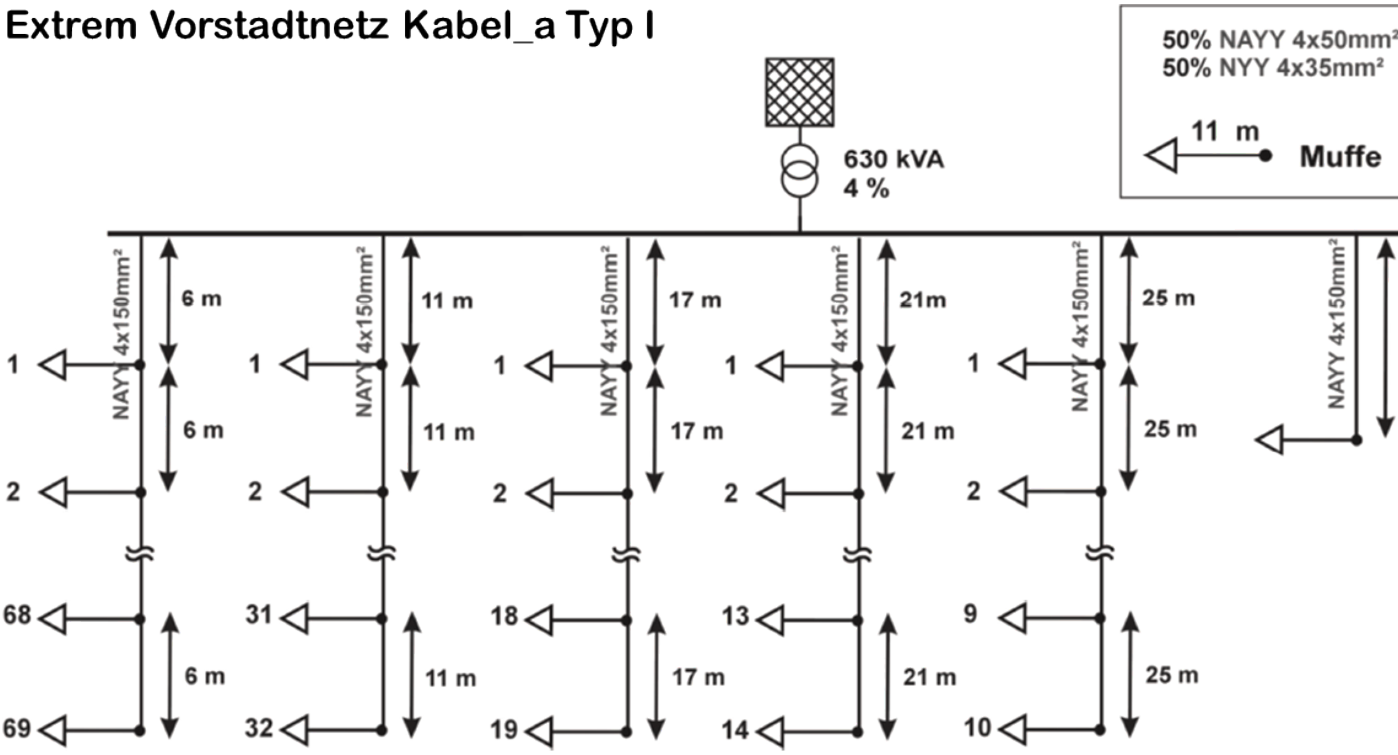

```python
import pandapower.networks as pn

'''Extrem Dorfnetz Kabel Typ II'''
net = pn.kb_extrem_dorfnetz_trafo()
```



8.6.8 Extreme Kerber Vorstadtnetze

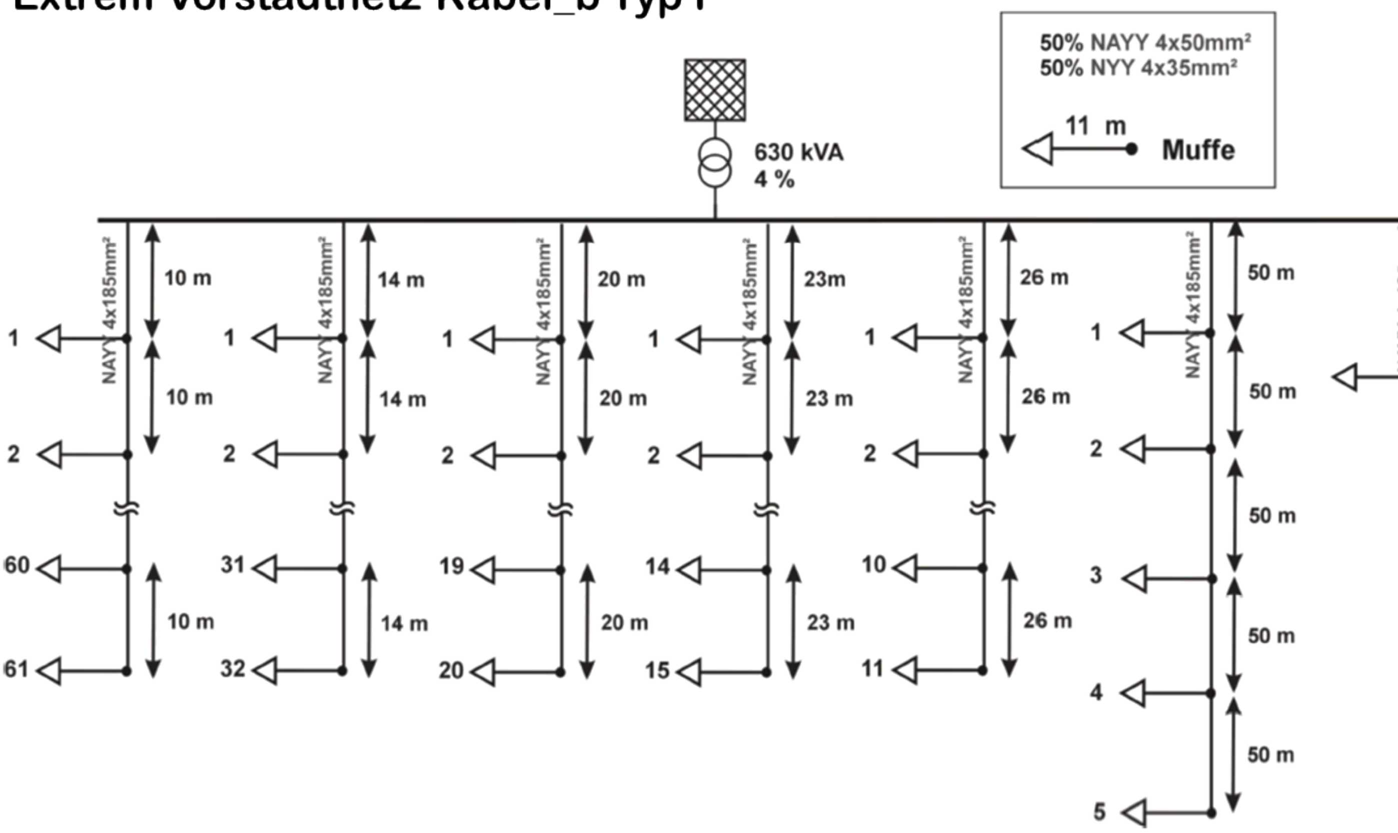
```
import pandapower.networks as pn

'''Extrem Vorstadtnetz Kabel_a Typ I'''
net = pn.kb_extrem_vorstadtnetz_1()
```

Extrem Vorstadtnetz Kabel_a Typ I

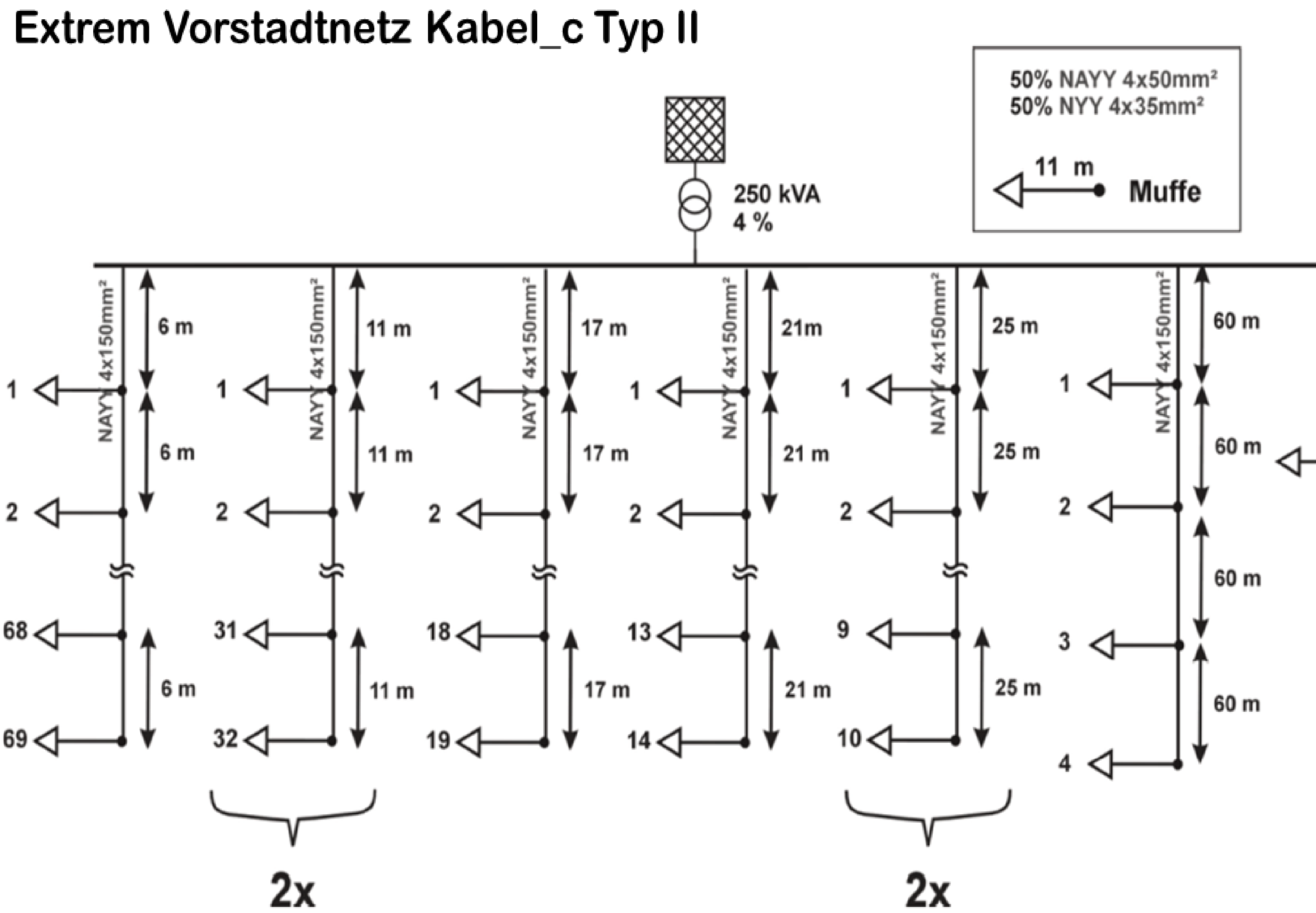



```python
import pandapower.networks as pn

'''Extrem Vorstadtnetz Kabel_b Typ I'''
net = pn.kb_extrem_vorstadtnetz_2()
```

Extrem Vorstadtnetz Kabel_b Typ I

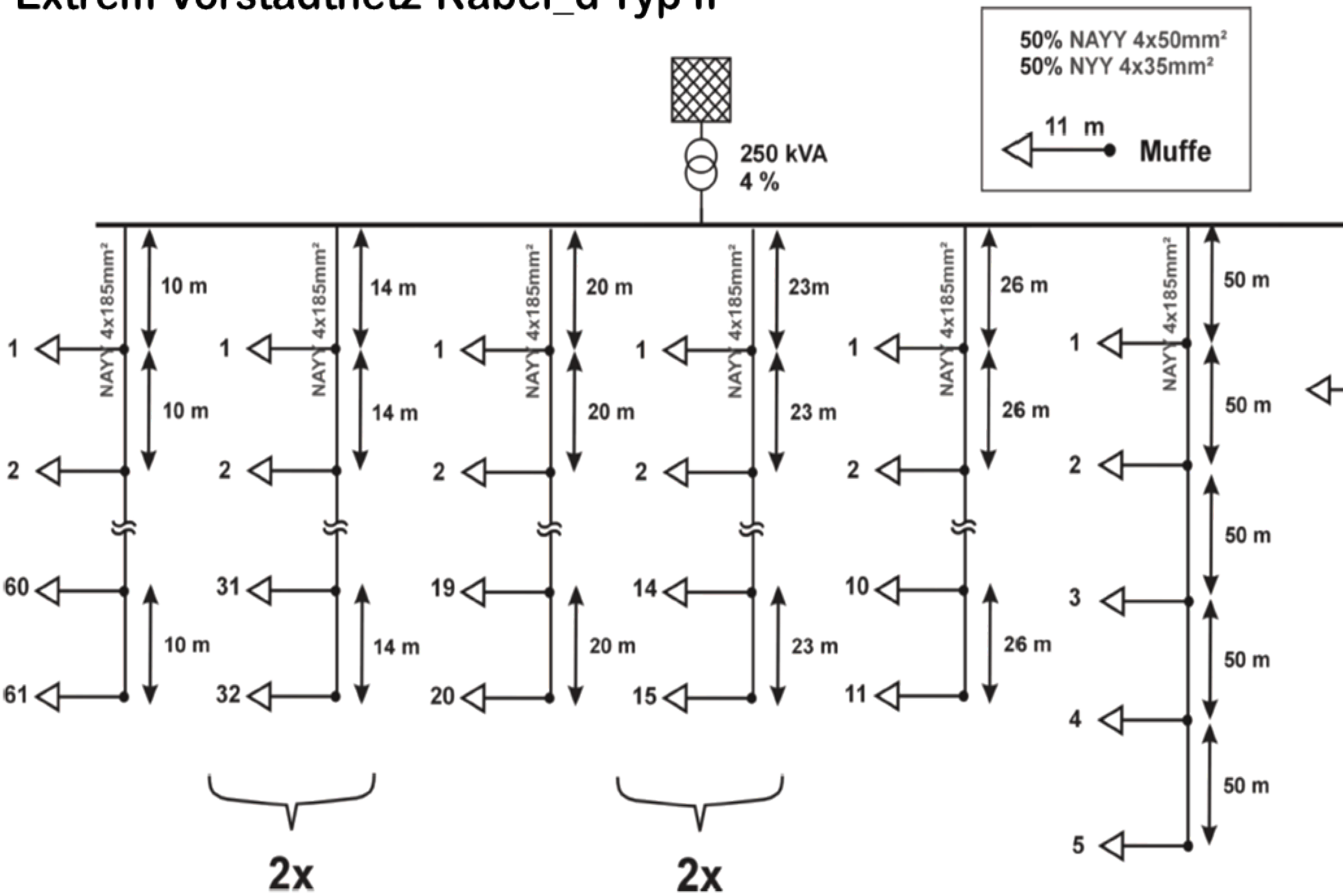


```python
import pandapower.networks as pn

'''Extrem Vorstadtnetz Kabel_c Typ II'''
net = pn.kb_extrem_vorstadtnetz_trafo_1()
```




```python
import pandapower.networks as pn

'''Extrem Vorstadtnetz Kabel_d Typ II'''
net = pn.kb_extrem_vorstadtnetz_trafo_2()
```

Extrem Vorstadtnetz Kabel_d Typ II



9 Plotting Networks

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 Simple Plotting

The function simple_plot() can be used for simple plotting. For advanced possibilities see the tutorials

`pandapower.plotting.`**`simple_plot`**(*net=None, respect_switches=False, line_width=1.0, bus_size=1.0, ext_grid_size=1.0, bus_color=(0.2980392156862745, 0.4470588235294118, 0.6901960784313725), line_color='grey', trafo_color='g', ext_grid_color='y'*)

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. If none is provided, mv_oberrhein() will be plotted as an example

OPTIONAL: respect_switches (bool, False) - Respect switches when 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 geoocord and the min divided by 200. mean_distance_between_buses = sum((net['bus_geodata'].max() - 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

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

9.2 Create Collections

Matplotlib collections can be created from pandapower networks with the following functions:

9.2.1 Bus Collections

`pandapower.plotting.`**`create_bus_collection`**(*net, buses=None, size=5, marker='o', patch_type='circle', colors=None, cmap=None, norm=None, infofunc=None, picker=False, **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

picker - picker argument passed to the patch collection

**kwargs - key word arguments are passed to the patch function

9.2.2 Branch Collections

`pandapower.plotting.`**`create_line_collection`**(*net*, *lines=None*, *use_line_geodata=True*, *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.

*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)

infofunc (function, None) - infofunction for the patch element

**kwargs - key word arguments are passed to the patch function

`pandapower.plotting.`**`create_trafo_collection`**(*net*, *trafos=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

9.3 Create Colormaps

9.3.1 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:

```
>>> from pandapower.plotting import cmap_discrete, create_line_collection,␣
↪draw_collections
>>> cmap_list = [((20, 50), "green"), ((50, 70), "yellow"), ((70, 100),
↪"red")]
>>> cmap, norm = cmap_discrete(cmap_list)
>>> lc = create_line_collection(net, cmap=cmap, norm=norm)
>>> draw_collections([lc])
```

9.3.2 Continous

pandapower.plotting.**cmap_continous**(*cmap_list*)
Can be used to create a continous 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:

```
>>> from pandapower.plotting import cmap_continous, create_bus_collection,␣
↪draw_collections
>>> cmap_list = [(0.97, "blue"), (1.0, "green"), (1.03, "red")]
>>> cmap, norm = cmap_continous(cmap_list)
>>> bc = create_bus_collection(net, cmap=cmap, norm=norm)
>>> draw_collections([bc])
```

9.4 Draw Collections

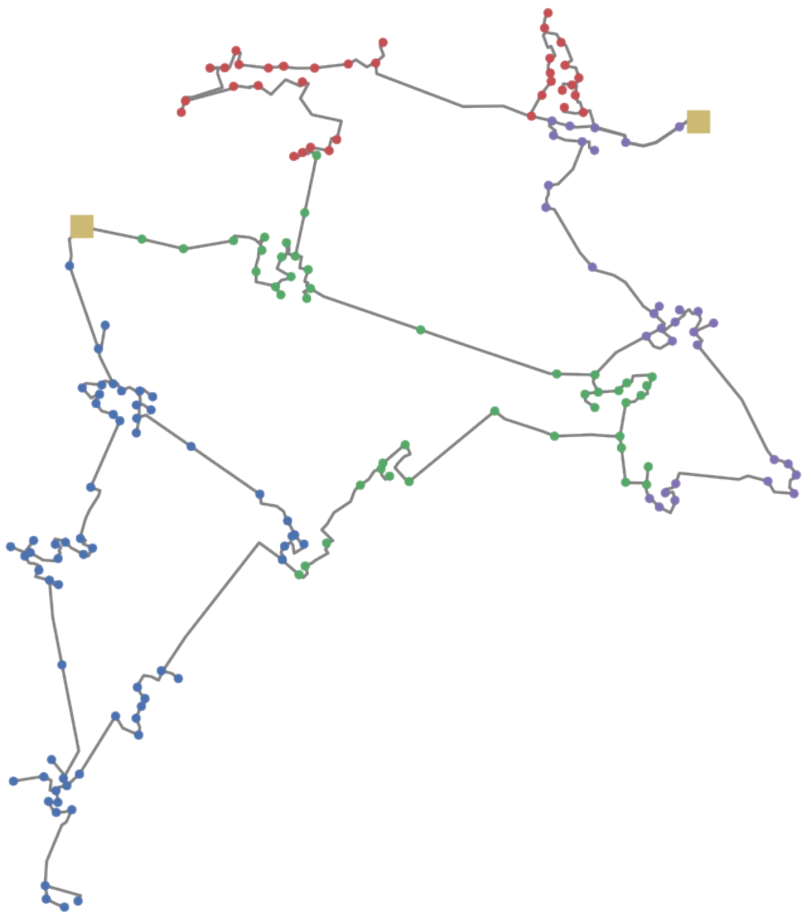
pandapower.plotting.**draw_collections**(*collections*, *figsize=(10, 8)*, *ax=None*, *plot_colorbars=True*)
Draws matplotlib collections which can be created with the create collection functions.

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

Example plot with mv_oberrhein network from the pandapower.networks package:



9.5 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

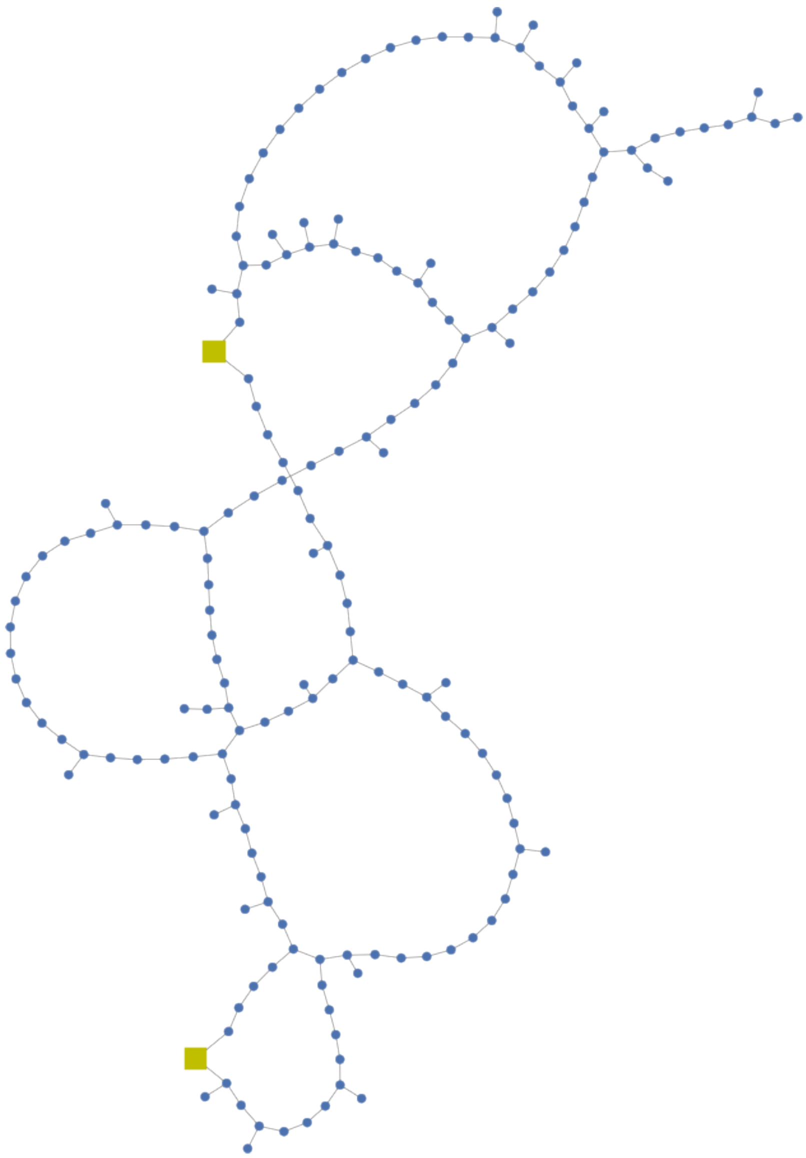
OPTIONAL: mg - Existing networkx multigraph, if available. Convenience to save computation time.

library - "igraph" to use igraph package or "networkx" to use networkx package

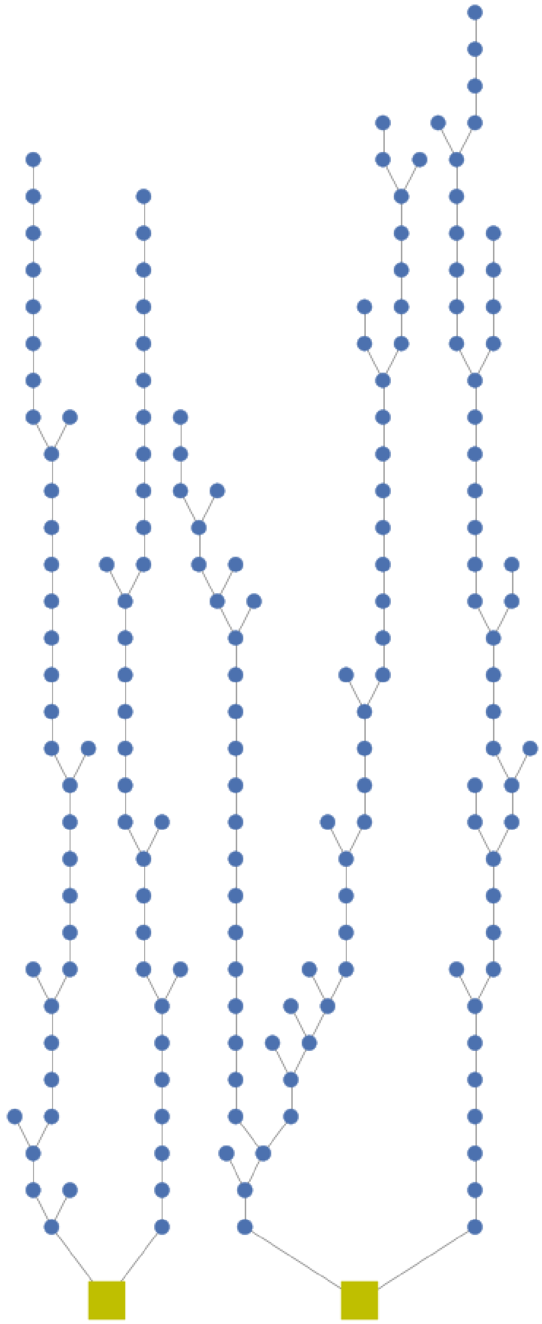
OUTPUT: net - pandapower network with added geo coordinates for the buses

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):



10 Save and Load Networks

pandapower networks can be saved and loaded using the pickle library or with an excel file.

pickle painlessly stores all datatypes, which is why the network will be exactly the same after saving/loading a network with the pickle library.

Excel has the upside that it provides a human readable format. However since Excel only accepts table-type inputs, some data mangling is necessary to save and load pandapower network through excel. Even though the relevant information is conserved, the process is not as robust as saving networks with pickle.

Important: Always use the pickle format unless you need a human readable file as output!

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 input file.

EXAMPLE:

```
>>> 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) - The absolute or relative path to the input file.

OUTPUT: **net** (dict) - The pandapower format network

EXAMPLE:

```
>>> 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 input 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:

```
>>> 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

net (dict) - The pandapower format network

EXAMPLE:

```
>>> 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*)

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) - The absolute or relative path to the input file.

EXAMPLE:

```
>>> pp.to_pickle(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) - The absolute or relative path to the input file.

OUTPUT: **convert** (bool) - use the convert format function to

net (dict) - The pandapower format network

EXAMPLE:

```
>>> net = pp.from_json("example.json")
```

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.

`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 - The frequency of the network.

OUTPUT:

net

EXAMPLE:

import pandapower.converter as pc

from pypower import case4gs

ppc_net = case4gs.case4gs()

pp_net = cv.from_ppc(ppc_net, f_hz=60)

`pandapower.converter.`**`validate_from_ppc`**(*ppc_net*, *pp_net*, *max_diff_values={'p_branch_kw': 0.001, 'q_branch_kvar': 0.001, 'p_gen_kw': 0.001, 'q_gen_kvar': 0.001, 'vm_pu': 1e-06, 'va_degree': 1e-05}*)

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.

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 power flow or if the maximum difference values (max_diff_values) cannot be hold.

EXAMPLE:

import pandapower.converter as pc

from pypower import case4gs

ppc_net = case4gs.case4gs()

pp_net = cv.from_ppc(ppc_net, f_hz=60)

cv.validate_from_ppc(ppc_net, pp_net)

`pandapower.converter.to_ppc(net, calculate_voltage_angles=False, trafo_model='t', r_switch=0, check_connectivity=True)`

This function converts a pandapower net to a pypower case file.

INPUT:

net - The pandapower net.

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.pp2ppc(net)

11.2 MATPOWER

To communicate to MATPOWER to exchange network data these functions are available.

`pandapower.converter.from_mpc(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. Note: python is 0-based while Matlab is 1-based.

INPUT:

mpc_file - path to a matpower case file (.mat).

OPTIONAL:

f_hz (int, 50) - The frequency of the network.

casename_mpc_file (str, 'mpc') - If mpc_file does not contain the arrays "gen", "branch" and "bus" it will use th
sub-struct casename_mpc_file

OUTPUT:

net - The pandapower network

EXAMPLE:

import pandapower.converter as pc

pp_net = cv.from_ppc('case9.mat', f_hz=60)

`pandapower.converter.to_mpc(net, filename=None, init='results', calculate_voltage_angles=False, trafo_model='t')`

This function converts a pandapower net to a matpower case files (.mat) version 2. Note: python is 0-based while Matlab is 1-based.

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:

- “flat”- flat start with voltage of 1.0pu and angle of 0° at all buses as initial solution
- “results” - voltage vector of last loadflow from net.res_bus is copied to the mpc

calculate_voltage_angles (bool, False) - copy the voltage angles from pandapower to the mpc

If True, voltage angles are copied from pandapower to the mpc. In some cases with large differences in voltage angles (for example in case of transformers with high voltage shift), the difference between starting and end angle value is very large. In this case, the loadflow might be slow or it might not converge at all. That is why the possibility of neglecting the voltage angles of transformers and ext_grids is provided to allow and/or accelarate convergence for networks where calculation of voltage angles is not necessary.

The default value is False because pandapower was developed for distribution networks. Please be aware that this parameter has to be set to True in meshed network for correct results!

trafo_model (str, “t”) - transformer equivalent circuit model pandapower provides two equivalent circuit models for the transformer:

- “t” - transformer is modelled 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 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:

import pandapower.converter as pc

import pandapower.networks as pn

net = pn.case9()

pc.pp2mpc(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 inforamtion 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*)

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}

Parameters

- **net** –

- **scalings** – A dictionary containing a mapping from element type to
- **scale_load** –

  param 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_trafos**(*net*, *trafos*)
: Deletes all trafos 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: **cl** (set) - Returns connected lines.

`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 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

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 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.