TECHNICAL REPORT

# pandapower

## - Convenient Power System Modelling and Analysis based on PYPOWER and pandas -

Fraunhofer IWES
Universität Kassel

November 14, 2016

Version 1.0.1

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*

*Main Authors:*
Leon Thurner
Alexander Scheidler
Julian Dollichon
Friederike Meier

*Contributors:*
Tobias Deß
Bastian Junker
Jannis Kupka
Lothar Löwer
Steffen Meinecke
Jan-Hendrik Menke
Jan Ulffers
Florian Schäfer

*Supervision:*
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

**pandapower does not yet support:**

- static short circuit calculation (currently in development)
- unbalanced power flow problems (planned, but not currently in development)
- RMS simulation (theoretically possible, but not currently in development)

**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 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.

**Electric Network Representation**

A common representation for electric models in open source power system software is the bus/branch model, which models the network as a bunch of nodes that are connected by branches. Loads or shunts are then modelled as an attribute or admittances of a certain bus. In reality however, the power demand is not an attribute of the bus, but of seperate electric elements (such as loads, pv generators or capacitor banks), which are connected to the buses. In pandapower, we model each electric bus element instead of considering summed values of power demand and shunt admittance for each bus.

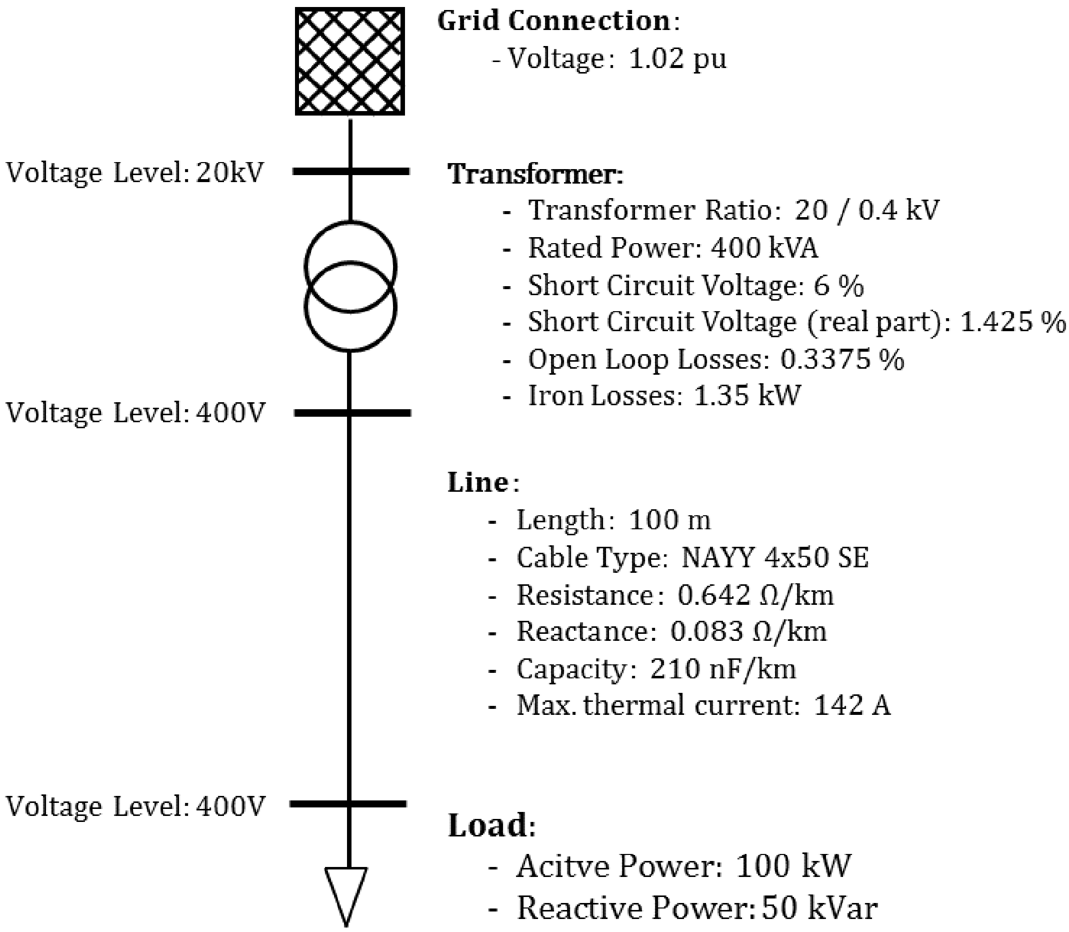
Branch/bus models also make no distinction between modeling different kind of branches, even though the electric models and behaviour for lines and transformers are very different. pandapower includes seperate electric models for lines, transformers and three winding transformers that allow a defintion of each element with common nameplate parameters.

Finally, pandapower includes a comprehensive switch model, which allows explicit modeling of bus/bus and bus/line or bus/transformer switches.

**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 network with three buses:

```
import pandapower as pp
net = pp.create_empty_network()
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 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, imax_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 |

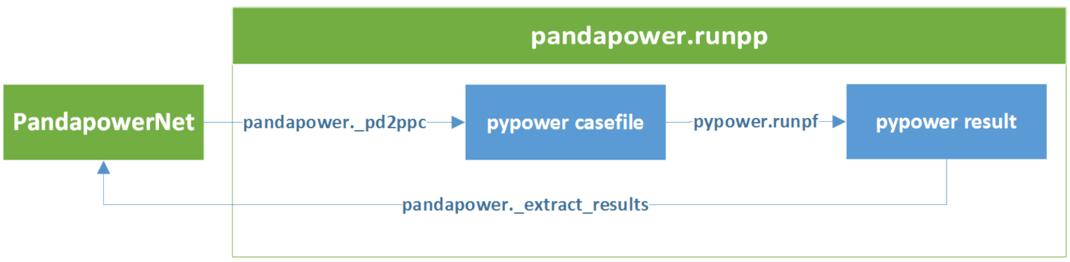
This is the version where transformer and line have been created through the standard type libraries, which is why the line has a specified type (cs for cable system) and the transformer has a tap changer, both of which are defined in the type data.

**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.

## 1.3 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

## 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 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*)

This function initializes the pandapower datastructure.

OPTIONAL:

**f_hz** (float, 50.) - power system frequency in hertz

**name** (string, None) - name for the network

RETURN:

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

- bus
- ext_grid
- gen
- impedance
- line
- load
- sgen
- shunt
- trafo
- trafo3w
- ward
- xward

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, default 0.4) - The grid voltage level.

**busgeodata** ((x,y)-tuple, default None) - coordinates used for plotting

**type** (string, default k) - 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*

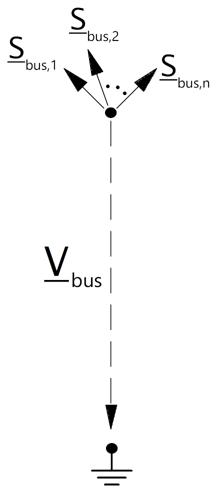
| Parameter | Datatype | Value Range | Explanation |
|---|---|---|---|
| name | string | | name of the bus |
| vn_kv* | float | $>0$ | rated voltage of the bus [kV] |
| type | string | naming conventions:<br>*"n"* - node<br>*"b"* - busbar<br>*"m"* - muff | type variable to classify buses |
| zone | string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

*necessary for executing a power flow calculation.

*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:

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

**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.

### 2.2.5 Optimal Power Flow Parameters

The voltage limits for the optimal power flow can be set bus wise in the bus tables:

| Parameter | Datatype | Explanation |
|---|---|---|
| max_vm_pu | float | Maximum voltage |
| min_vm_pu | float | Minimum voltage |

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

**Note:** $max_{vm,pu} > min_{vm,pu}$ is a necessary condition

## 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** (np.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

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*, *imax_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

**imax_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** (np.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

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, imax_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 |
| to_bus* | integer | | Index of bus where the line ends |
| length_km* | float | $> 0$ | length of the line [km] |
| r_ohm_per_km* | float | $\geq 0$ | resistance of the line [Ohm per km] |
| x_ohm_per_km* | float | $\geq 0$ | inductance of the line [Ohm per km] |
| c_nf_per_km* | float | $\geq 0$ | capacitance of the line [nF per km] |
| imax_ka* | float | $> 0$ | maximal thermal current [kA] |
| parallel* | integer | $\geq 1$ | number of parallel line systems |
| df* | float | 0...1 | derating factor (scaling) for imax_ka |

| Parameter | Datatype | Value Range | Explanation |
|---|---|---|---|
| type | string | Naming conventions:<br><br>*"ol"* - overhead line<br>*"cs"* - underground cable system | type of line |
| in_service* | boolean | True / False | specifies if the line is in service. |

*necessary for executing a power flow calculation.

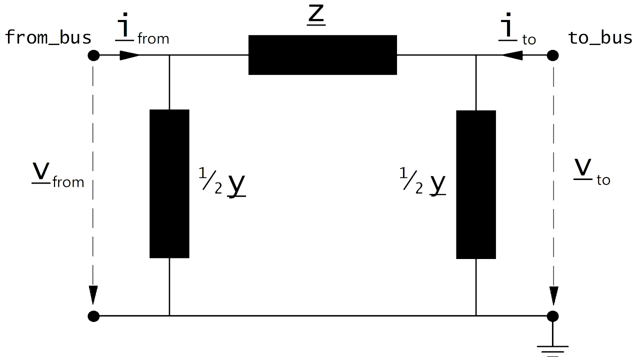
**Note:** Defining a line with length zero leads to a division by zero in the power flow and is therefore not allowed. Lines with a very low impedance might lead to convergence problems in the power flow for the same reason. If you want to directly connect two buses, please use the switch element instead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

$$
\begin{aligned}
p\_from\_kw &= Re(\underline{v}_{from} \cdot \underline{i}_{from}) \\
q\_from\_kvar &= Im(\underline{v}_{from} \cdot \underline{i}_{from}) \\
p\_to\_kw &= Re(\underline{v}_{to} \cdot \underline{i}_{to}) \\
q\_to\_kvar &= Im(\underline{v}_{to} \cdot \underline{i}_{to}) \\
pl\_kw &= p\_from\_kw + p\_to\_kw \\
ql\_kvar &= q\_from\_kvar + q\_to\_kvar \\
i\_from\_ka &= i_{from} \\
i\_to\_ka &= i_{to} \\
i\_ka &= max(i_{from}, i_{to}) \\
loading\_percent &= \frac{i\_ka}{imax\_ka \cdot df \cdot parallel} \cdot 100
\end{aligned}
$$

### 2.3.5 Optimal Power Flow Parameters

The line loading constraint is an upper constraint for loading_percent.

| Parameter | Datatype | Explanation |
|---|---|---|
| max_loading_percent | float | Maximum loading of the line with respect to i_max_ka |

## 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"

**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 |
| element* | integer | | index of the element the switch is connected to:<br>- bus index if et = *"b"*<br>- line index if et = *"l"*<br>- trafo index if et = *"t"* |
| et* | string | *"b"* - bus-bus switch<br>*"l"* - bus-line switch<br>*"t"* - bus-trafo switch | element type the switch connects to |

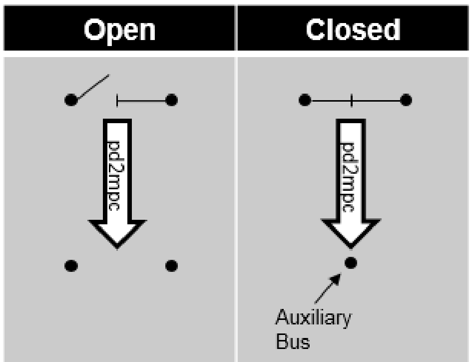
| Parameter | Datatype | Value Range | Explanation |
|---|---|---|---|
| type | string | naming conventions:<br>*"CB"* - circuit breaker<br>*"LS"* - load switch<br>*"LBS"* - load break switch<br>*"DS"* - disconnecting switch | type of switch |
| closed* | boolean | True / False | signals the switching 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:

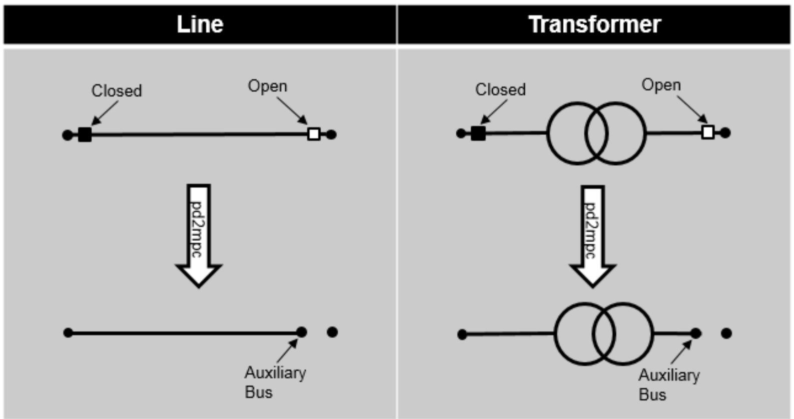
- there is no voltage drop over the switch (ideal switch)
- no convergence problems due to small impedances / large admittances
- less buses in the admittance matrix

*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*)

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

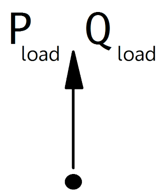
*net.load*

| Parameter | Datatype | Value Range | Explanation |
|---|---|---|---|
| name | string | | name of the load |
| bus * | integer | | index of connected bus |
| p_kw* | float | $\geq 0$ | 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 | $\geq 0$ | scaling factor for active and reactive power |
| in_service* | boolean | True / False | specifies if the load is in service. |

*necessary for executing a power flow calculation.

### 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 $Q_{load}$.

### 2.5.5 Optimal Power Flow Parameters

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

## 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*, *cost_per_kw=nan*, *cost_per_kvar=nan*, *controllable=False*)

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

OUTPUT:

**index** - The unique id of the created sgen

EXAMPLE:

create_sgen(net, 1, p_kw = -120)

### 2.6.2 Input Parameters

*net.sgen*

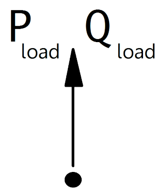
| Parameter | Datatype | Value Range | Explanation |
|---|---|---|---|

| Parameter | Datatype | Value Range | Explanation |
|---|---|---|---|
| name | string | | name of the static generator |
| type | string | naming conventions:<br>*"PV"* - photovoltaic system<br>*"WP"* - wind power system<br>*"CHP"* - combined heating and power system | type of generator |
| bus* | integer | | index of connected bus |
| p_kw* | float | $\leq 0$ | 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 | $\geq 0$ | scaling factor for the active and reactive power |
| in_service* | boolean | True / False | specifies if the generator is in service. |

*necessary for executing a power flow calculation.

### 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:

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

---

**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 $P_{load}$ and $Q_{load}$.

### 2.6.5 Optimal Power Flow Parameters

The cost and constraints of a generator can be definded using the following parameters:

| Parameter | Datatype | Explanation |
|---|---|---|
| max_p_kw | float | Maximum active ower |
| min_p_kw | float | Minimum active power |
| max_q_kvar | float | Maximum reactive ower |
| 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 |
| cost_per_kw | float | Cost per kW |
| cost_per_kvar | float | Cost per kvar |

## 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, index=None, cost_per_kw=nan, cost_per_kvar=nan*)

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] |
| in_service* | boolean | True / False | specifies if the external grid is in service. |

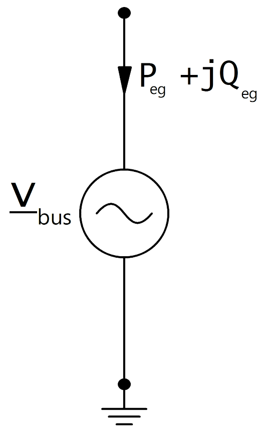
*necessary for executing a power flow calculation.

*Optional Parameters*:

| Parameter | Datatype | Value Range | Explanation |
|---|---|---|---|
| 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 |

### 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.

---

### 2.7.5 Optimal Power Flow Parameters

An external grid is not considered as a flexibility for the optimal power flow. The voltage setpoint can not be changed by the optimal power flow. Still, costs can be defined for an external grid:

| Parameter | Datatype | Explanation |
|---|---|---|
| cost_per_kw | float | Cost per kW |
| cost_per_kvar | float | Cost per kvar |

## 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*)

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

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_pos=nan*, *in_service=True*, *name=None*, *index=None*, *max_loading_percent=nan*, ***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

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*

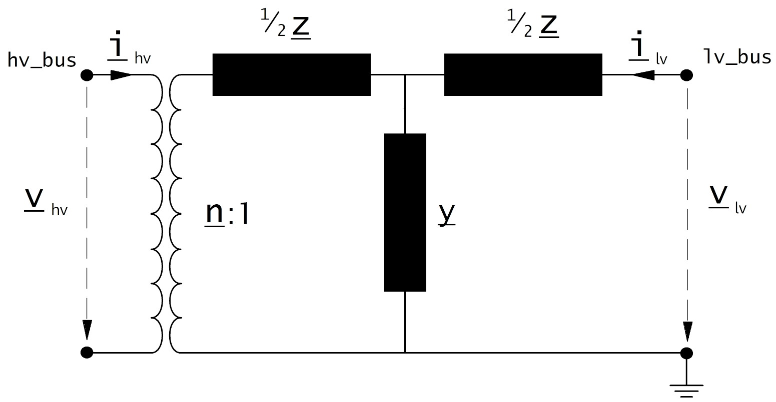
| Parameter | Datatype | Value Range | Explanation |
|---|---|---|---|
| name | string | | name of the transformer |
| std_type | string | | transformer standard type name |
| hv_bus* | integer | | high voltage bus index of the transformer |
| lv_bus* | integer | | low voltage bus index of the transformer |
| 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 | $\geq 0$ | real component of short circuit voltage [%] |
| pfe_kw* | float | $\geq 0$ | iron losses [kW] |
| i0_percent* | float | $\geq 0$ | 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 |
| 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.

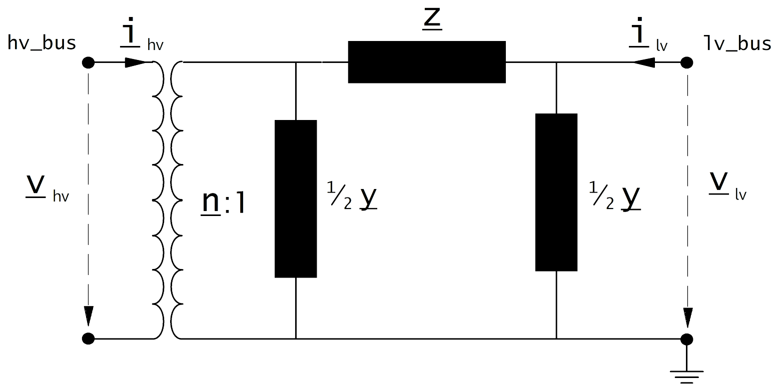
### 2.8.3 Electric Model

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:

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

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 [%] |

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

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:

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

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

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

### 2.8.5 Optimal Power Flow Parameters

The transformer loading constraint for the optimal power flow corresponds to the option trafo_loading=”current”(see above):

## 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*)

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

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*)

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, np.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.

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*

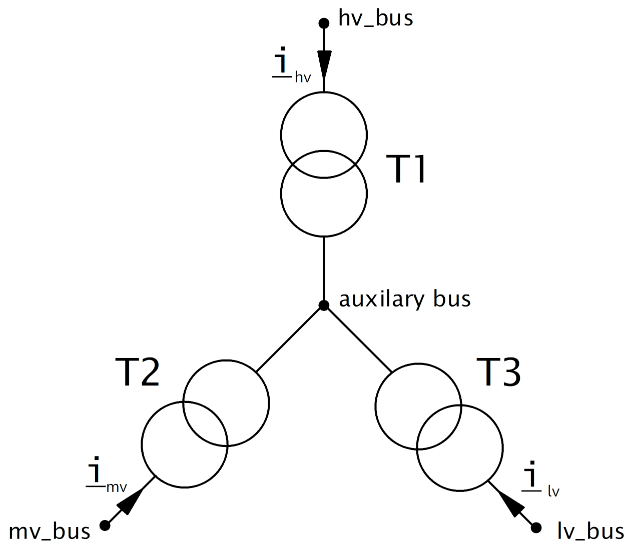
| Parameter | Datatype | Value Range | Explanation |
|---|---|---|---|
| name | string | | name of the transformer |
| hv_bus* | integer | | high voltage bus index of the transformer |
| mv_bus | integer | | medium voltage bus index of the transformer |
| lv_bus* | integer | | low voltage bus index of the transformer |
| 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 | $\geq 0$ | real part of short circuit voltage from high to medium voltage [%] |

| Parameter | Datatype | Value Range | Explanation |
|---|---|---|---|
| vscr_mv_percent* | float | $\geq 0$ | real part of short circuit voltage from medium to low voltage [%] |
| vscr_lv_percent* | float | $\geq 0$ | real part of short circuit voltage from high to low voltage [%] |
| pfe_kw* | float | $\geq 0$ | iron losses [kW] |
| i0_percent* | float | $\geq 0$ | 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.

### 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:

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

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

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

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

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

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:

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

### 2.9.5 Optimal Power Flow Parameters

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

## 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, in_service=True, cost_per_kw=nan, cost_per_kvar=nan, controllable=False*)

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

**in_service** (boolean) - 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:<br>*"sync"* - synchronous generator<br>*"async"* - asynchronous generator | type variable to classify generators |
| bus* | integer | | index of connected bus |
| p_kw* | float | $\leq 0$ | 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 | $\leq 0$ | scaling factor for the active power |
| in_service* | boolean | True / False | specifies if the generator is in service. |

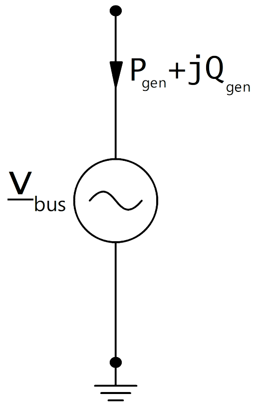
*necessary for executing a power flow calculation.

---

**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 anlge [degree] |
| vm_pu | float | voltage at the generator [p.u] |

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

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

**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.

### 2.10.5 Optimal Power Flow Parameters

The cost and constraints of a generator can be definded using the following parameters:

| Parameter | Datatype | Explanation |
|---|---|---|
| max_p_kw | float | Maximum active ower |
| min_p_kw | float | Minimum active power |
| max_q_kvar | float | Maximum reactive ower |
| min_q_kvar | float | Minimum reactive power |
| controllable | bool | States if a gen is controllable or not. Currently gens must be controllable, because there is no metho |
| cost_per_kw | float | Cost per kW |
| cost_per_kvar | float | Cost per kvar |

## 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)

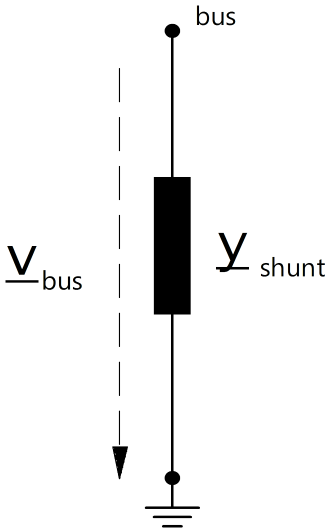
### 2.11.2 Input Parameters

*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 | $\geq 0$ | 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:

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

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] |

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

### 2.11.5 Optimal Power Flow Parameters

Shunts are not yet respected by the optimal power flow as a flexibility.

## 2.12 Impedance

**See also:**

*Unit Systems and Conventions*

### 2.12.1 Create Function

pandapower.**create_impedance**(*net*, *from_bus*, *to_bus*, *r_pu*, *x_pu*, *sn_kva*, *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

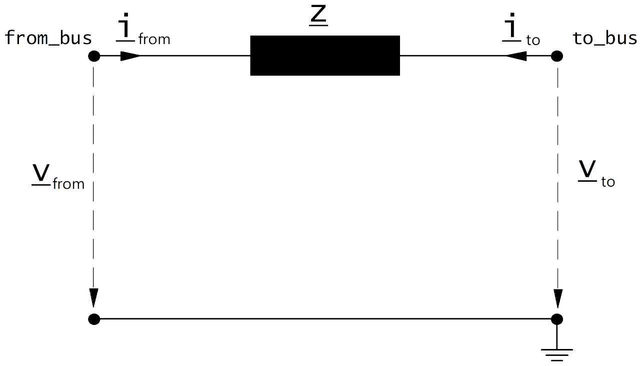
*net.impedance*

| 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 |
| r_pu* | float | $>0$ | resistance of the impedance [p.u] |
| x_pu* | float | $>0$ | reactance of the impedance [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 simple longitudinal per unit impedance:




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:

$$\underline{z}_{impedance} = r\_pu + j \cdot x_{pu}$$
$$\underline{z} = \underline{z}_{impedance} \frac{S_N}{sn\_kva}$$

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

### 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] |

$$\begin{aligned}
i_from_ka &= i_{from} \\
i_to_ka &= i_{to} \\
p_from_kw &= Re(\underline{v}_{from} \cdot \underline{i}_{from}) \\
q_from_kvar &= Im(\underline{v}_{from} \cdot \underline{i}_{from}) \\
p_to_kw &= Re(\underline{v}_{to} \cdot \underline{i}_{to}) \\
q_to_kvar &= Im(\underline{v}_{to} \cdot \underline{i}_{to}) \\
pl_kw &= p_from_kw + p_to_kw \\
ql_kvar &= q_from_kvar + q_to_kvar
\end{aligned}$$

### 2.12.5 Optimal Power Flow Parameters

Impedances are not yet respected in the optimal power flow. No constraints can be formulated.

## 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

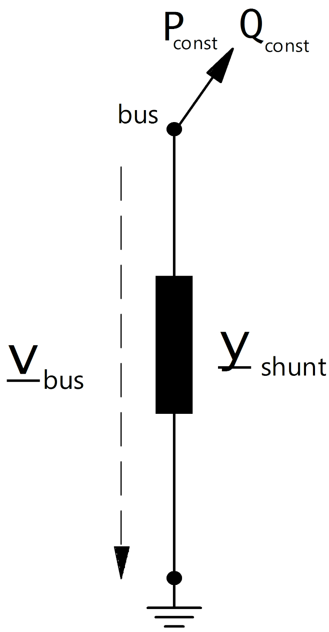
### 2.13.2 Input Parameters

*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] |

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

### 2.13.5 Optimal Power Flow Parameters

Ward equivalents are not yet respected by the optimal power flow.

## 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

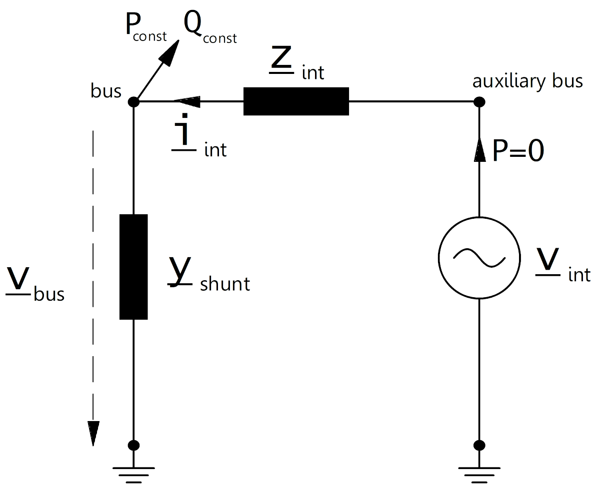
*net.xward*

| 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] |

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

### 2.14.5 Optimal Power Flow Parameters

Extended Ward equivalents are not yet respected by the optimal power flow.

# 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 | imax_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

| | sn_kva | vn_hv_kv | vn_lv_kv | vsc_percent | vscr_percent | pfe_kw | i0_percent | shift_degree | tp_side | tp_mid | tp_min | tp_max | tp_st_percent |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.25 MVA 10/0.4 kV | 250 | 10 | 0.4 | 4 | 1.2 | 0.6 | 0.24 | 150 | hv | 0 | -2 | 2 | 2.5 |
| 0.25 MVA 20/0.4 kV | 250 | 20 | 0.4 | 6 | 1.44 | 0.8 | 0.32 | 150 | hv | 0 | -2 | 2 | 2.5 |
| 0.4 MVA 10/0.4 kV | 400 | 10 | 0.4 | 4 | 1.325 | 0.95 | 0.2375 | 150 | hv | 0 | -2 | 2 | 2.5 |
| 0.4 MVA 20/0.4 kV | 400 | 20 | 0.4 | 6 | 1.425 | 1.35 | 0.3375 | 150 | hv | 0 | -2 | 2 | 2.5 |
| 0.63 MVA 10/0.4 kV | 630 | 10 | 0.4 | 4 | 1.0794 | 1.18 | 0.1873 | 150 | hv | 0 | -2 | 2 | 2.5 |
| 0.63 MVA 20/0.4 kV | 630 | 20 | 0.4 | 6 | 1.206 | 1.65 | 0.2619 | 150 | hv | 0 | -2 | 2 | 2.5 |

| | sn_kva | vn_hv_kv | vn_lv_kv | vsc_percent | vscr_percent | pfe_kw | i0_percent | shift_degree | tp_side | tp_mid | tp_min | tp_max | tp_st_percent |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 100 MVA 220/110 kV | 100000.0 | 220.0 | 110.0 | 12.0 | 0.26 | 55 | 0.06 | 0 | hv | 0 | -9 | 9 | 1.5 |
| 160 MVA 380/110 kV | 160000.0 | 380.0 | 110.0 | 12.2 | 0.25 | 60 | 0.06 | 0 | hv | 0 | -9 | 9 | 1.5 |
| 25 MVA 110/10 kV | 25000 | 110 | 10 | 10.04 | 0.276 | 28.51 | 0.073 | 150 | hv | 0 | -9 | 9 | 1.5 |
| 25 MVA 110/20 kV | 25000 | 110.0 | 20.0 | 11.2 | 0.282 | 29 | 0.071 | 150 | hv | 0 | -9 | 9 | 1.5 |
| 40 MVA 110/10 kV | 40000 | 110 | 10 | 10.04 | 0.295 | 30.45 | 0.076 | 150 | hv | 0 | -9 | 9 | 1.5 |
| 40 MVA 110/20 kV | 40000 | 110.0 | 20.0 | 11.2 | 0.302 | 31 | 0.08 | 150 | hv | 0 | -9 | 9 | 1.5 |
| 63 MVA 110/10 kV | 63000 | 110 | 10 | 10.04 | 0.31 | 31.51 | 0.078 | 150 | hv | 0 | -9 | 9 | 1.5 |
| 63 MVA 110/20 kV | 63000 | 110.0 | 20.0 | 11.2 | 0.322 | 33 | 0.086 | 150 | hv | 0 | -9 | 9 | 1.5 |

### 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 the 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 imax_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, "imax_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, "imax_ka": 0.4, "type": "cs"},
>>>              "typ2": {"r_ohm_per_km": 0.015, "x_ohm_per_km": 0.01, "c_nf_
↪per_km": 30, "imax_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

**element** - type of element ("line" or "trafo")

### 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

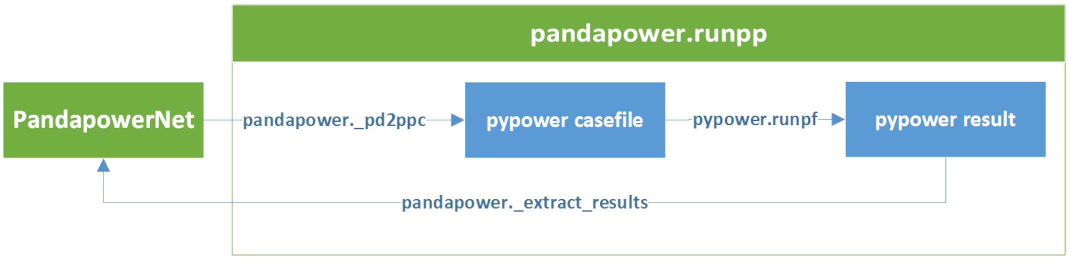
The power flow is the most import static network calculation operation. This section shows you how to run different power flows (AC, DC, OPF), what known problems and caveats there are and how you can identify problems using the pandapower diagnostic function.

## 4.1 Run a Power Flow

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

### 4.1.1 AC Power Flow

pandapower uses PYPOWER to solve the AC power flow problem:




`pandapower.`**`runpp`**(*net*, *init='flat'*, *calculate_voltage_angles=False*, *tolerance_kva=1e-05*, *trafo_model='t'*, *trafo_loading='current'*, *enforce_q_lims=False*, *suppress_warnings=True*, *Numba=True*, *recycle=None*, ***kwargs*)

Runs PANDAPOWER AC Flow

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

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

Optional:

**init** (str, "flat") - initialization method of the loadflow Pandapower supports three 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.

**calculate_voltage_angles** (bool, False) - consider voltage angles in loadflow calculation

If True, voltage angles are considered in the loadflow calculation. 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. Note that if calculate_voltage_angles is True the loadflow is initialized with a DC power flow (init = "dc")

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!

**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 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.

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

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

- "current"- transformer loading is given as ratio of current flow and rated current of the transformer. This is the recommended setting, since thermal as well as magnetic effects in the transformer depend on the current.
- "power" - transformer loading is given as ratio of apparent power flow to the rated apparent power of the transformer.

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

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

**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 also suppressed.

**Numba** (bool, True) - Usage Numba JIT compiler

If set to True, the Numba JIT compiler is used to generate matrices for the powerflow. Massive speed improvements are likely.

**recycle** (dict, none) - Reuse of internal powerflow variables

Contains a dict with the following parameters: is_elems: If True in service elements are not filtered again and are taken from the last result in net["_is_elems"] ppc: If True the ppc (PYPOWER case file) is taken from net["_ppc"] and gets updated instead of regenerated entirely bus_lookup: If True the bus_lookup variable (Indices Pandapower -> ppc) is taken from net["_bus_lookup"] Ybus: If True the admittance matrix (Ybus, Yf, Yt) is taken from ppc["internal"] and not regenerated

****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!

---

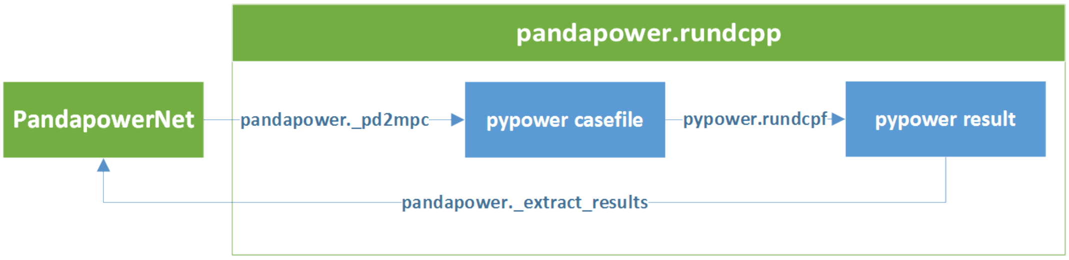
**Note:** If you are interested in the pypower casefile that pandapower is using for power flow, you can find it in net["_ppc"]. However all necessary informations are written into the pandpower format net, so the pandapower user should not usually have to deal with pypower.

---

### 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', suppress_warnings=True, recycle=None, **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 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.

**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.

**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 also suppressed.

**Numba** (bool, True) - Usage Numba JIT compiler

If set to True, the Numba JIT compiler is used to generate matrices for the powerflow. Massive speed improvements are likely.

**recycle** (dict, none) - Reuse of internal powerflow variables

Contains a dict with the following parameters: is_elems: If True in service elements are not filtered again and are taken from the last result in net["_is_elems"] ppc: If True the ppc (PYPOWER case file) is taken from net["_ppc"] and gets updated instead of regenerated entirely bus_lookup: If True the bus_lookup variable (Indices Pandapower -> ppc) is taken from net["_bus_lookup"] Ybus: If True the admittance matrix (Ybus, Yf, Yt) is taken from ppc["internal"] and not regenerated

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

---

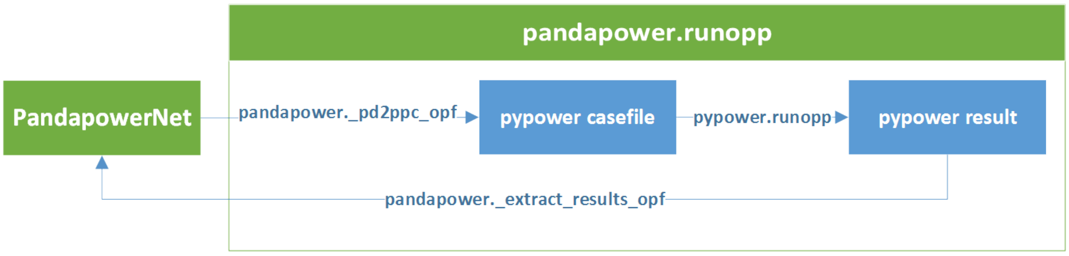
**Note:** If you are interested in the pypower casefile that pandapower is using for power flow, you can find it in net["_ppc"]. However all necessary informations are written into the pandpower format net, so the pandapower user should not usually have to deal with pypower.

---

### 4.1.3 Optimal Power Flow

The pandapower optimal power flow is a tool for optimizing the grid state. It offers two cost function options, that are fitting special use cases. In addition to the equality constraints in the problem formulations below, the full set of nonlinear real and reactive power balance equations is always considered in the pandapower optimal power flow. To understand the usage, the OPF tutorial is recommended (see tutorial).

pandapower uses PYPOWER to solve the optimal power flow problem:




pandapower.**runopp**(*net*, *cost_function='linear'*, *verbose=False*, *suppress_warnings=True*, ***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

**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

**Costs can be assigned to generation units in the following columns:**

- net.gen.cost_per_kw
- net.sgen.cost_per_kw

- net.ext_grid.cost_per_kw

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

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

**OPTIONAL:**

**cost_function (str,"linear")- cost function**

- "linear" - minimizes weighted generator costs
- "linear_minloss" - minimizes weighted generator cost and line losses

**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.

---

**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.

---

### Generator Flexibilities

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

| Constraint | Defined in |
|---|---|
| $P_{max,i} \leq P_g \leq P_{min,g}, g \ \epsilon \ gen$ | net.gen.min_p_kw / net.gen.max_p_kw |
| $Q_{max,g} \leq Q_g \leq Q_{min,g}, g \ \epsilon \ gen$ | net.gen.min_q_kvar / net.gen.max_q_kvar |
| $P_{max,sg} \leq P_{sg} \leq P_{min,sg}, sg \ \epsilon \ sgen$ | net.sgen.min_p_kw / net.sgen.max_p_kw |
| $Q_{max,sg} \leq Q_{sg} \leq Q_{min,sg}, sg \ \epsilon \ sgen$ | net.sgen.min_q_kvar / net.sgen.max_q_kvar |

If you do want to have flexible active but fixed reactive power, you can set the maximum and minimum reactive power constraints to the same value (might cause convergence problems, in which case relax the constraint a little bit). If you want to set active and reactive power without flexibility for a specific generator, set the controllable flag (net.gen.controllable / net.sgen.controllable) to False .

---

**Note:** Only (static) generators with controllable = True are considered as flexible. If controllable == False, the (static) generators is considered fixed as in a regular power flow.

---

### Network Constraints

Constraints can be defined for bus voltages, line and transformer loading.

| 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 |

### Cost Functions

The default cost function summs up the weighted generator costs:

$$min \sum_{i \ \epsilon \ gen} P_{g,i} * w_{g,i}$$

The generator costs are defined in the "cost_per_km" column in net.gen, net.sgen and net.ext_grid respectively. If no costs are specified, the costs default to 1, which results in a minimization of overall power injection.

By specifying objectivetype='linear_minloss' in the OPF, the cost function is evaluated as a summation of the weighted generator costs and the sum of the line losses:

$$min \sum_{i\ \epsilon\ gen} P_{g,i} * w_{g,i} + \lambda \sum_{l\ \epsilon\ line} P_{loss,l}$$

The weighting factor $\lambda$ can be specified as the keyword argument lambda_opf for runopp().

---

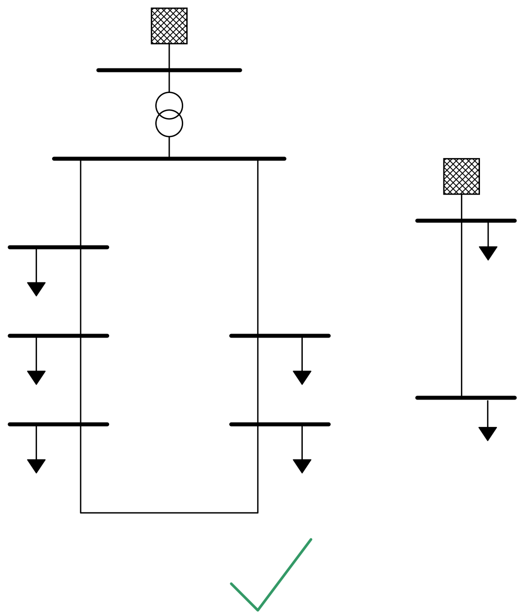
**Note:** The pandapower optimal power flow is a recent development. In the future, this module will be developed further in order to allow more flexible opf calculations.

---

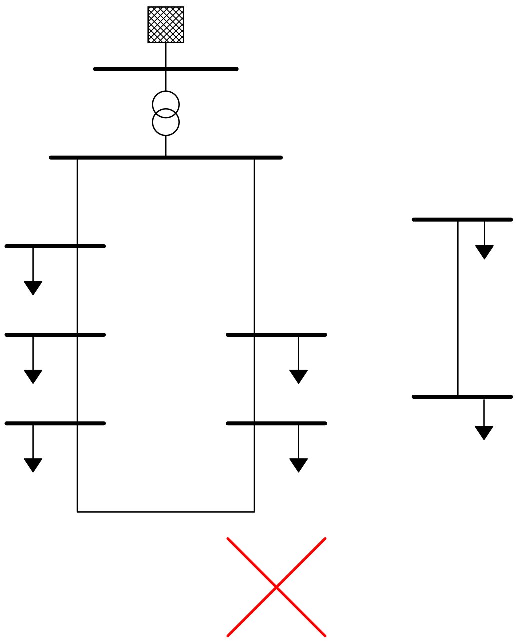
## 4.2 Known Problems and Caveats

### 4.2.1 Disconnected Buses

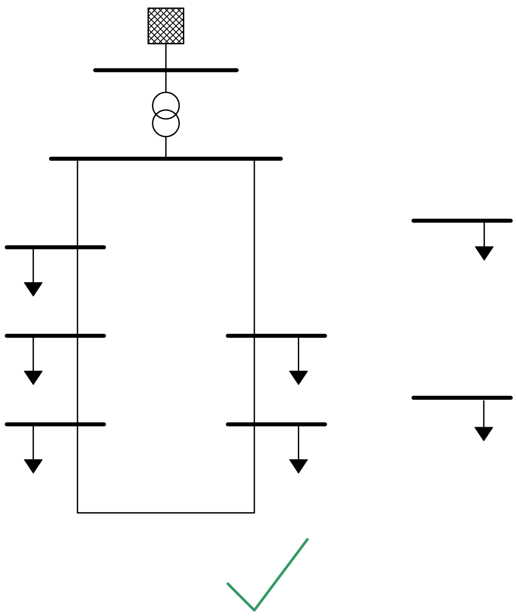
It is possible to have multiple galvanically seperated network in one pandapower network:



However, if there are in service buses that are not connected to any slack bus, the power flow will not converge:



The only exception are buses which are not connected to any branch. These buses are ignored in the power flow and therefore do not cause convergence problems:

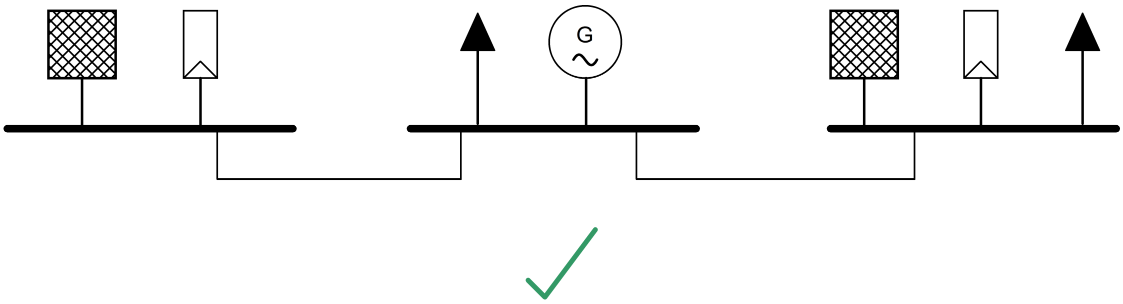


The reason is that buses which are not connected to any branch can be easily identified, while buses which form an isolated group can only be found with a topology search. The pandapower topology package includes a function to find disconnected buses. For the power flow to converge, they have to be set out of service:

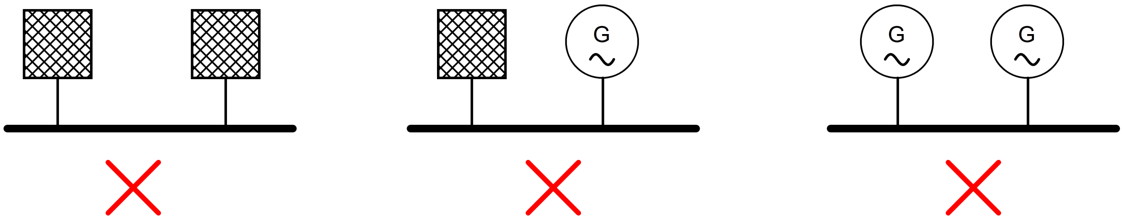
```
import pandapower.topology as topology
unsupplied_buses = top.unsupplied_buses(net)
net.bus.loc[unsupplied_buses, "in_service"] = False
pp.runpp(net)
```

### 4.2.2 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:

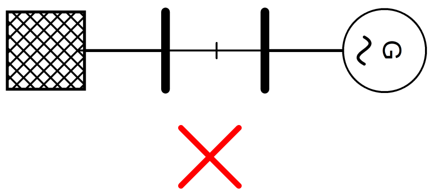



It is however not possible to connect multiple voltage controllable element (ext_grid, gen) at one bus, since this would convergence problems in PYPOWER:



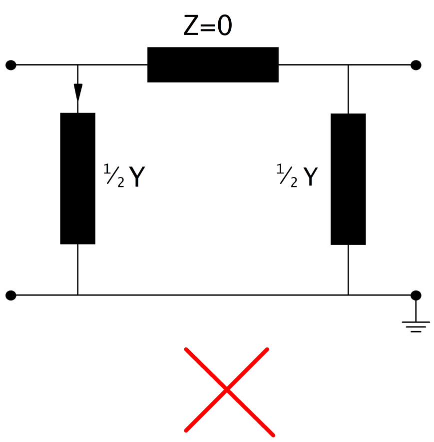

The pandapower API will prevent you from adding a second voltage controlling element to a bus, so you should not be able to build the networks pictured above through the pandapower API.

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*):




### 4.2.3 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*)

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

**RETURN:**

- **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:

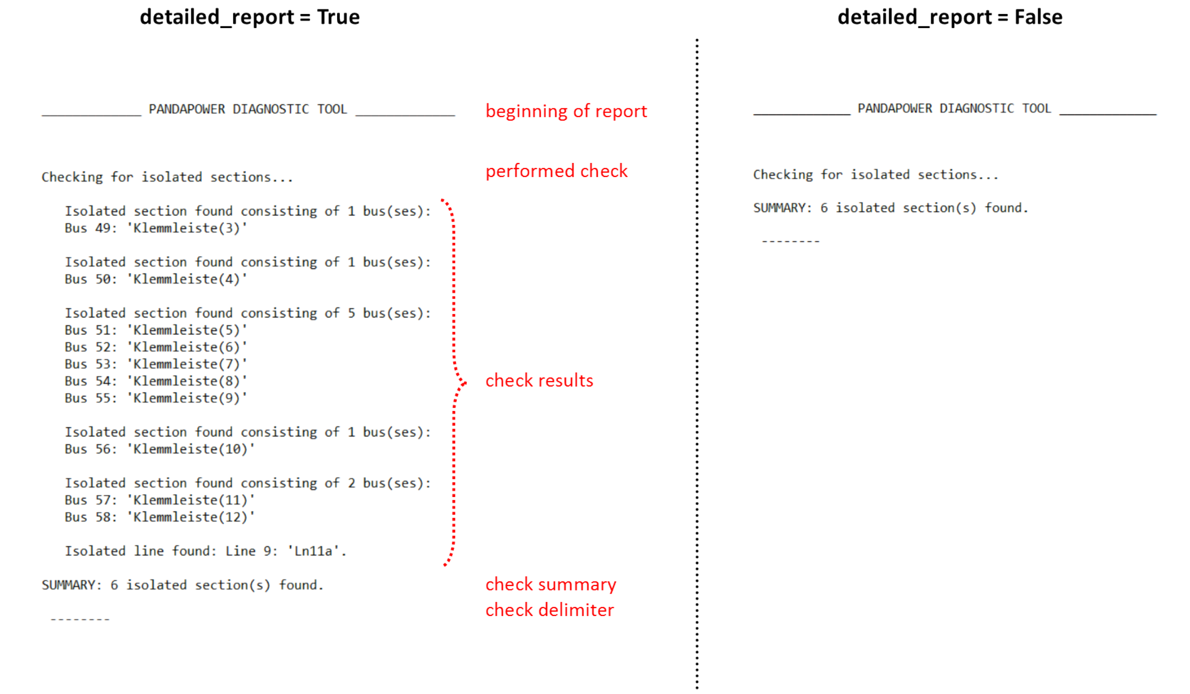
- invalid values (e.g. negative element indeces)
- check, if at least one external grid exists
- check, if there are buses with more than one gen and/or ext_grid
- overload: tries to run a power flow calculation with loads scaled down to 10%
- switch_configuration: tries to run a power flow calculation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

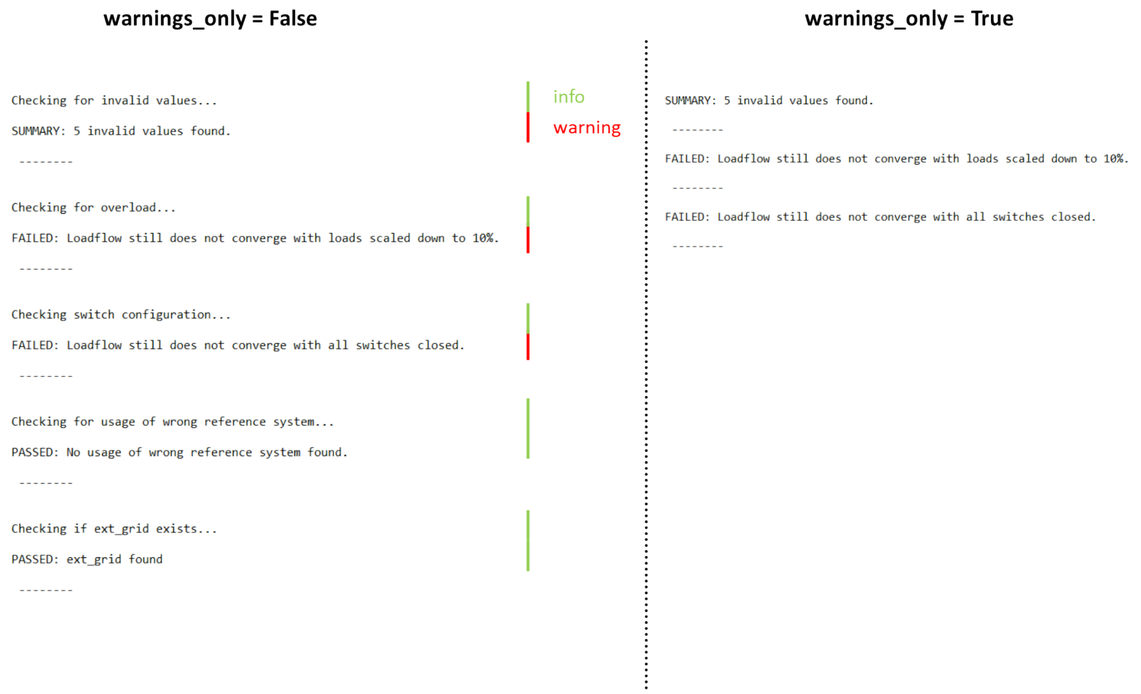
Here are a few examples of what logging output looks like:

**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 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 )

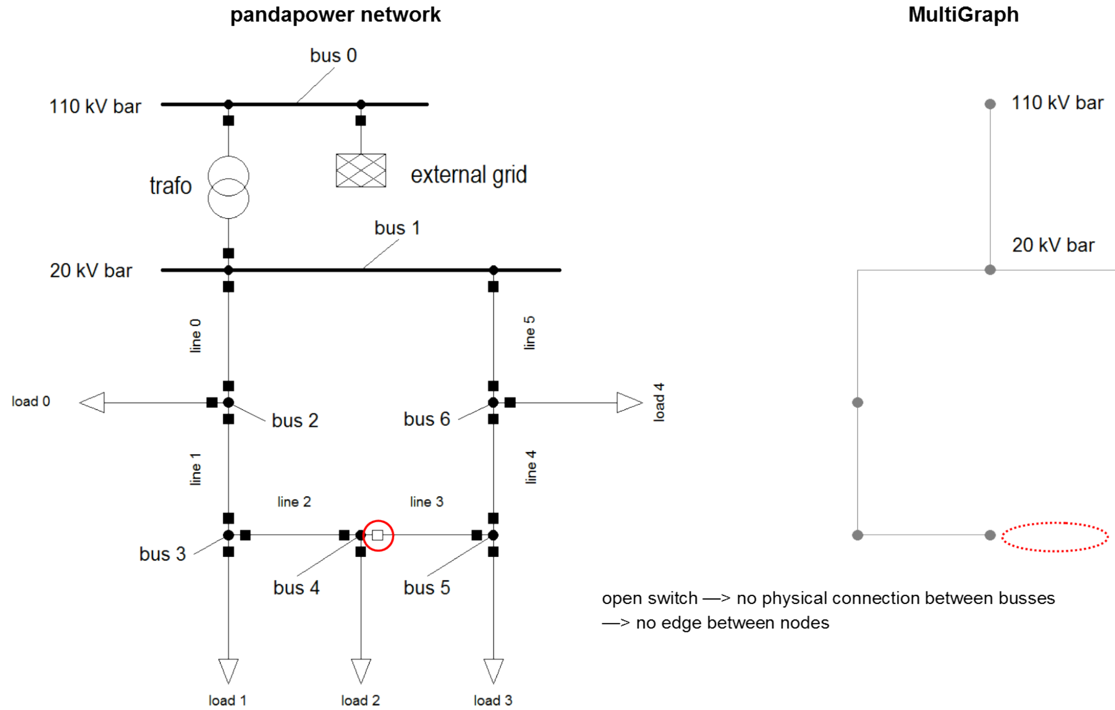
pandapower provides a function to translate pandapower networks into networkx graphs. Once the electric network is translated into an abstract networkx graph, all network operations that are available in networkx can be used to analyse the network. For example you can find the shortest path between to 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.

## 5.1 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:

| lines | trafos |
|---|---|

| lines | trafos |
|---|---|
| • from_bus<br>• to_bus<br>• length_km<br>• index<br>• in_service<br>• imax_ka | • hv_bus<br>• lv_bus<br>• index<br>• in_service |

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)

RETURN:

**mg** - Returns the required NetworkX graph

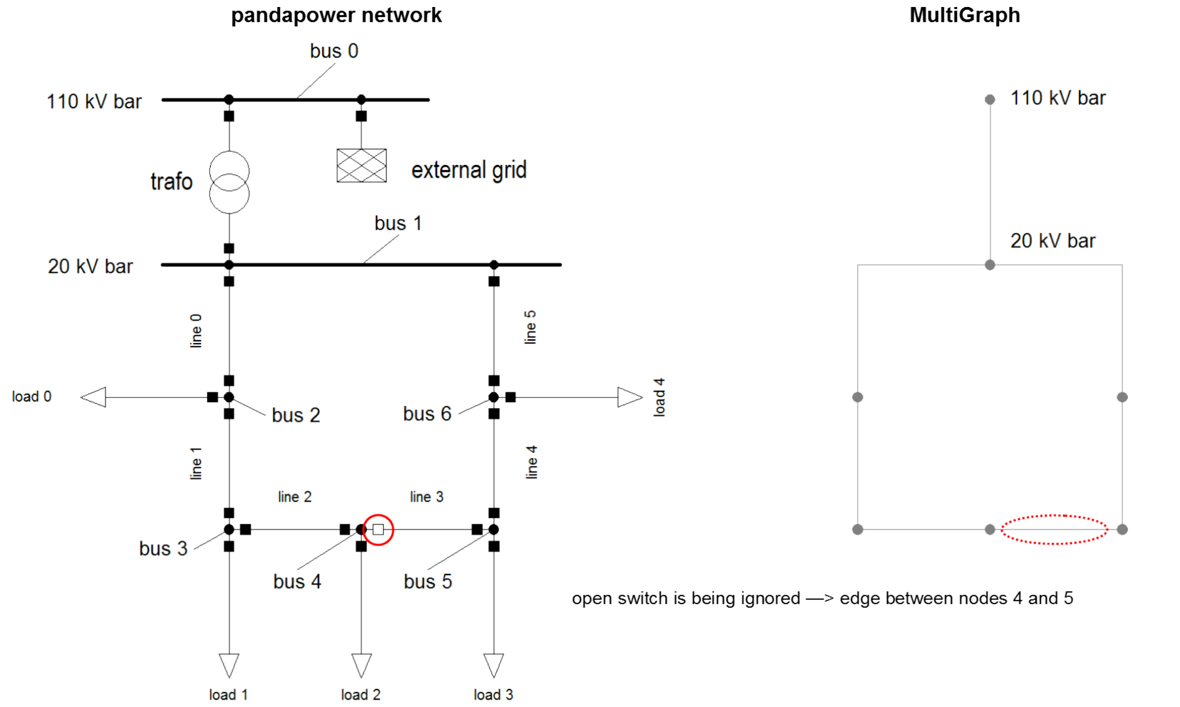
EXAMPLE:

import pandapower.topology as top

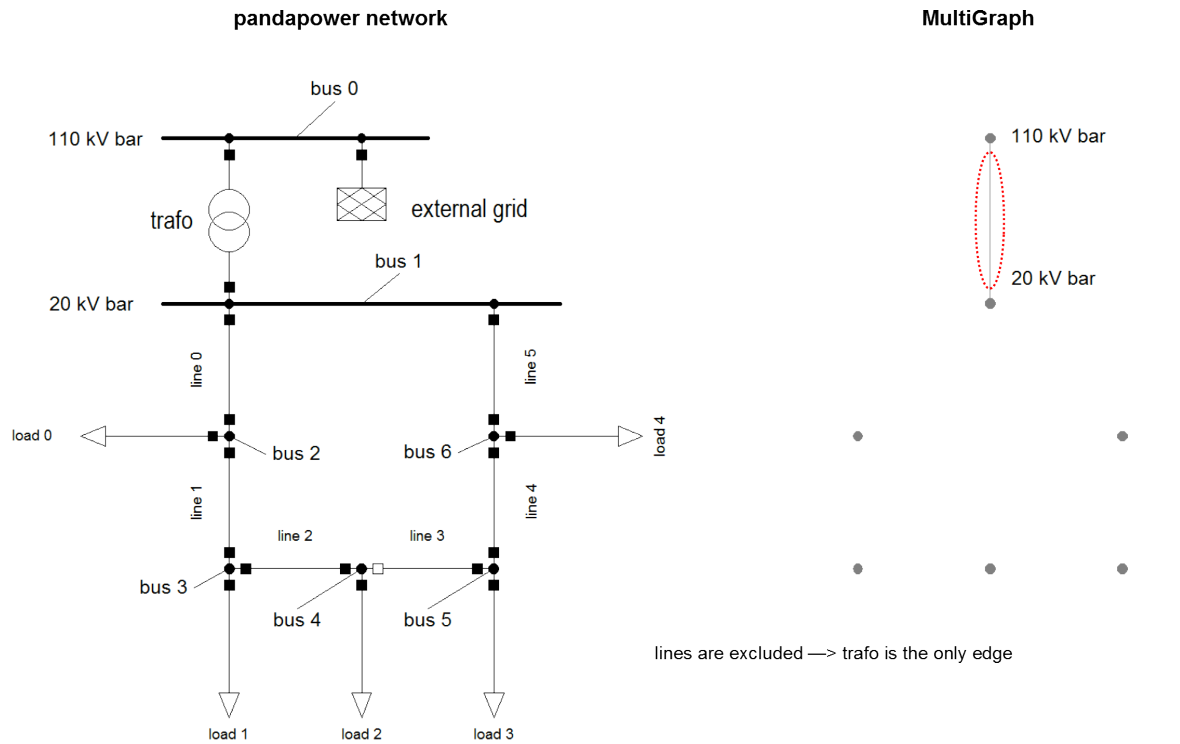
mg = top.create_nx_graph(net, respect_switches = False) # converts the pandapower network "net" to a MultiGraph. Open switches will be ignored.

**Examples**

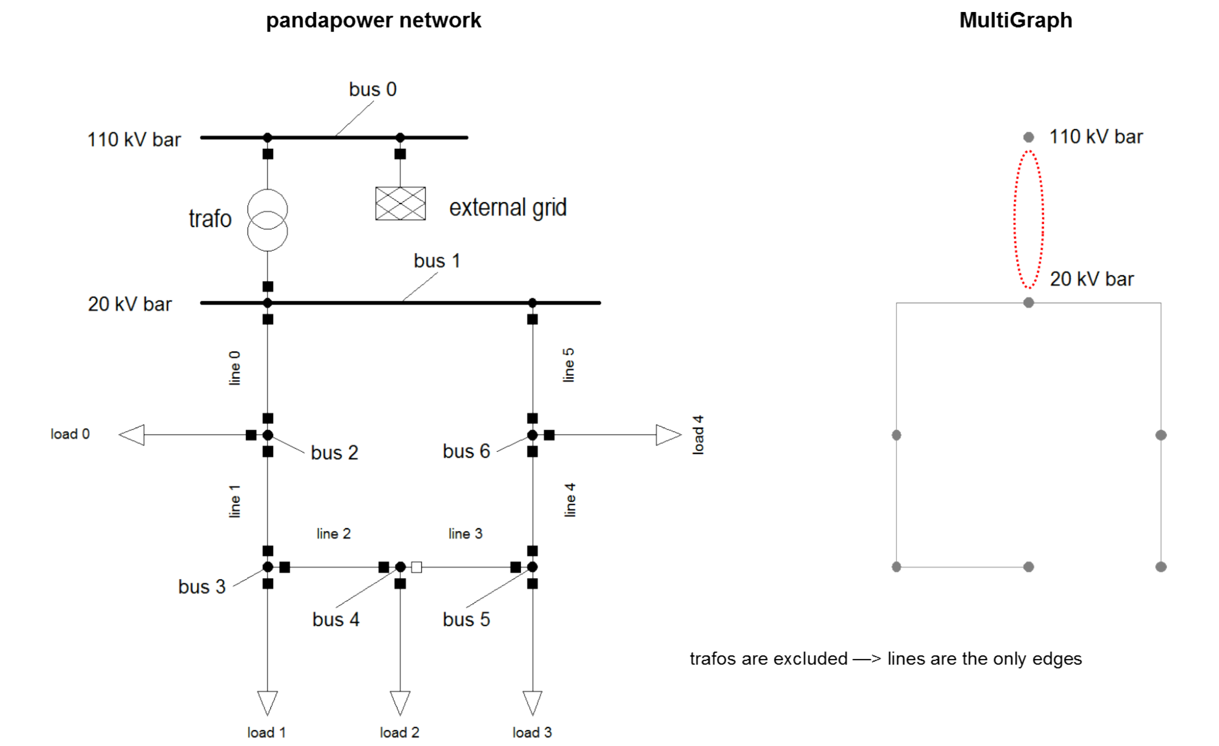
```
create_nxgraph(net, respect_switches = False)
```



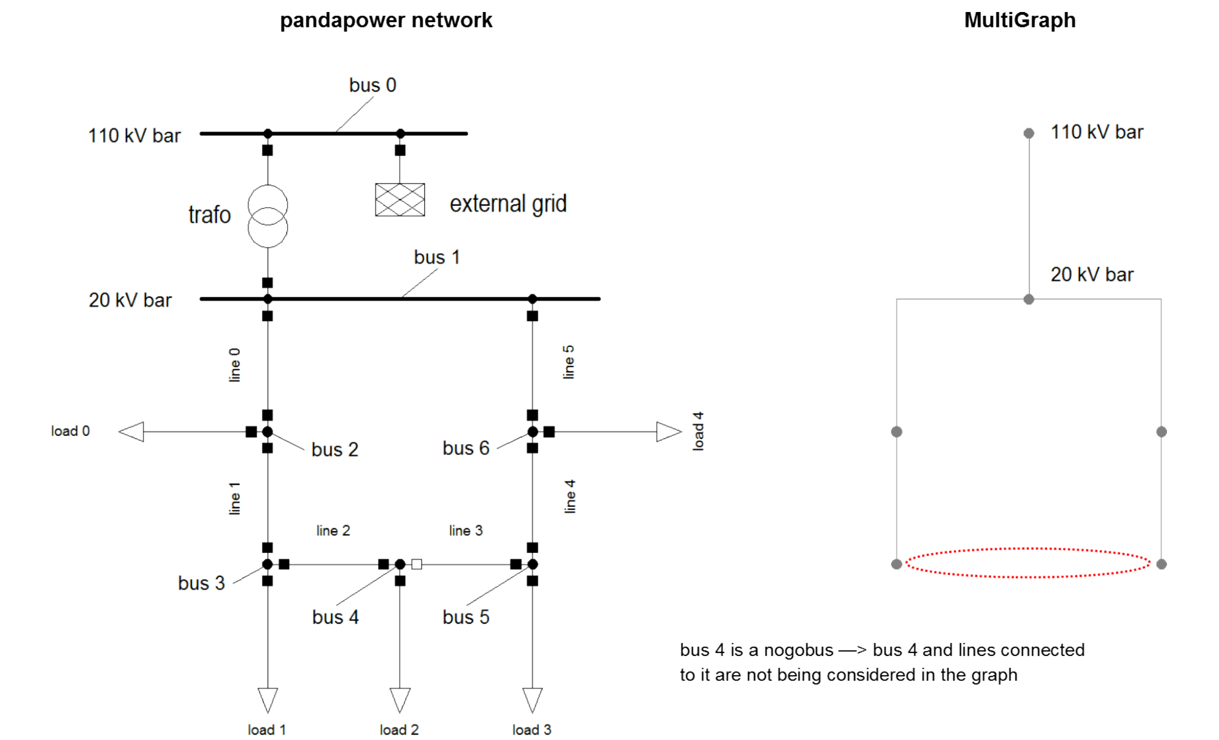

```
create_nxgraph(net, include_lines = False)
```



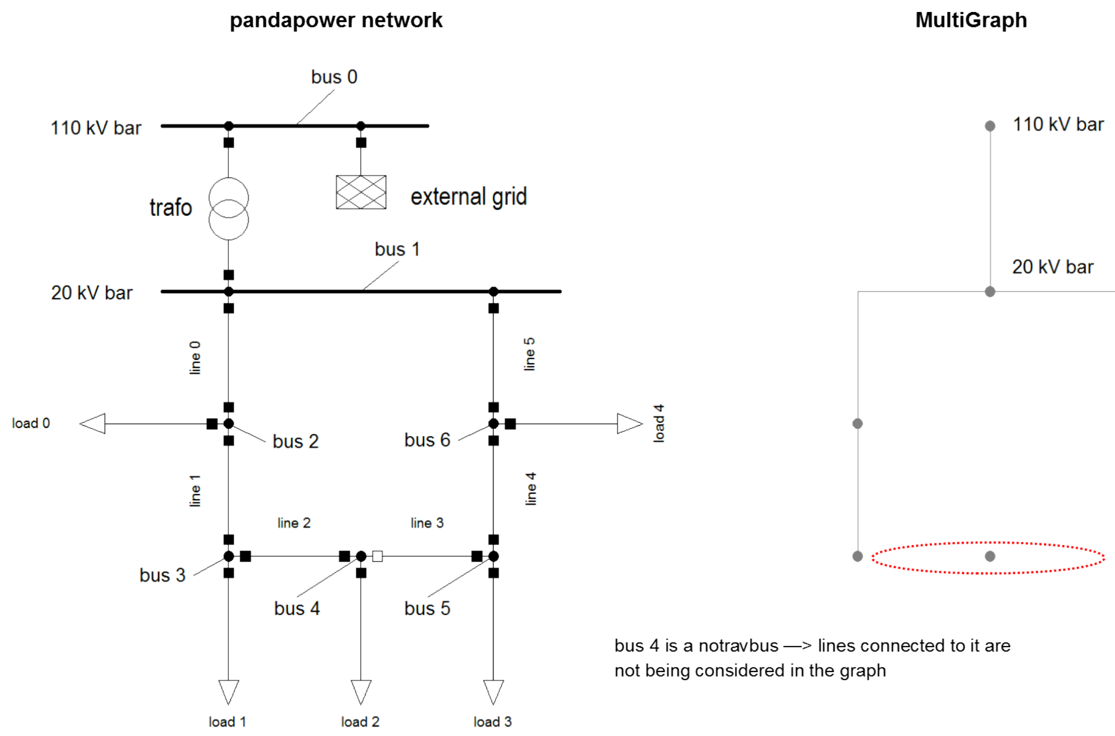

```
create_nxgraph(net, include_trafos = False)
```




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




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




## 5.2 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.

### 5.2.1 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

RETURN:

**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)

### 5.2.2 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

RETURN:

**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)

### 5.2.3 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

RETURN:

**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)

### 5.2.4 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.

RETURN:

**ub** (set) - unsupplied buses

EXAMPLE:

import pandapower.topology as top

top.unsupplied_buses(net)

### 5.2.5 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)

RETURN:

None

EXAMPLE:

import pandapower.topology as top

top.determine_stubs(net, roots = [0, 1])

## 5.3 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")
```

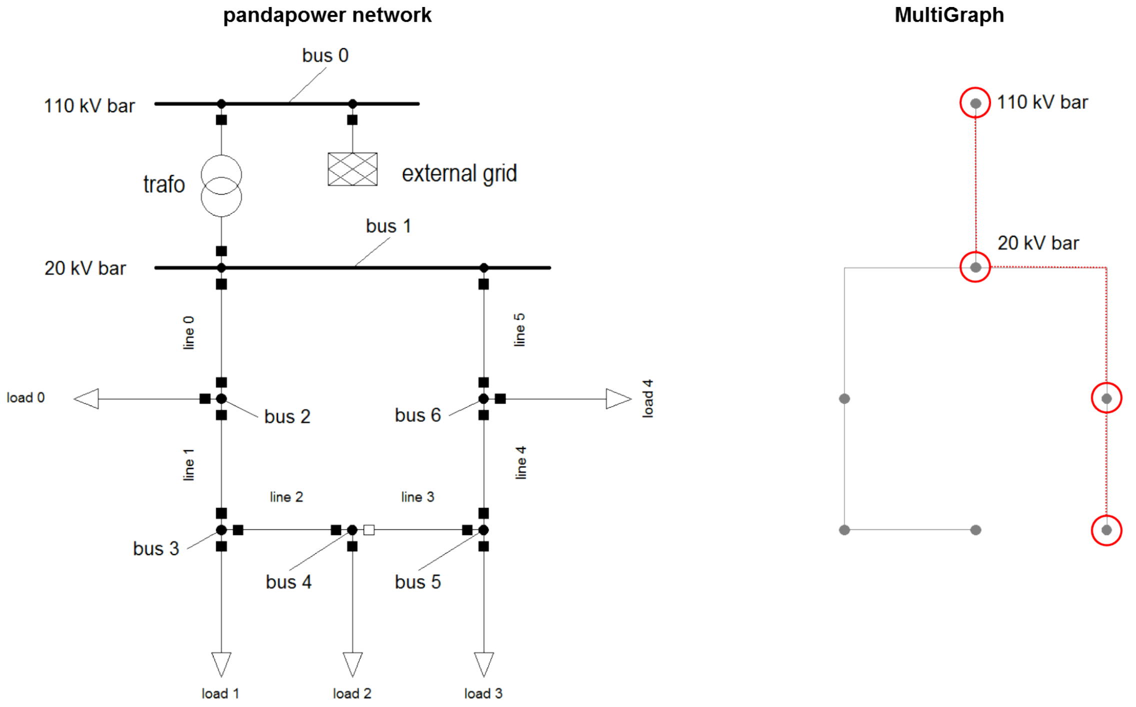
```
pp.create_line(net, name = "line 5", from_bus = 6, to_bus = 1, length_km = 1, std_
↪type = "NAYY 150")

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')
```

### 5.3.1 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]
```

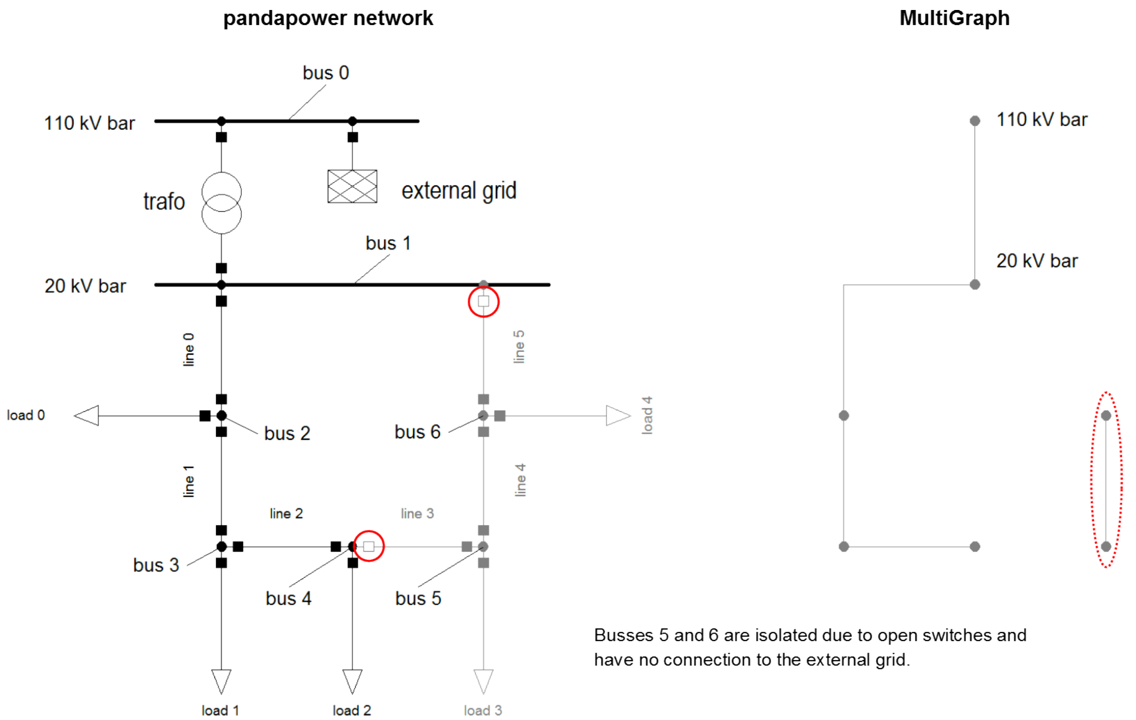



### 5.3.2 Find disconnected buses

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}
```



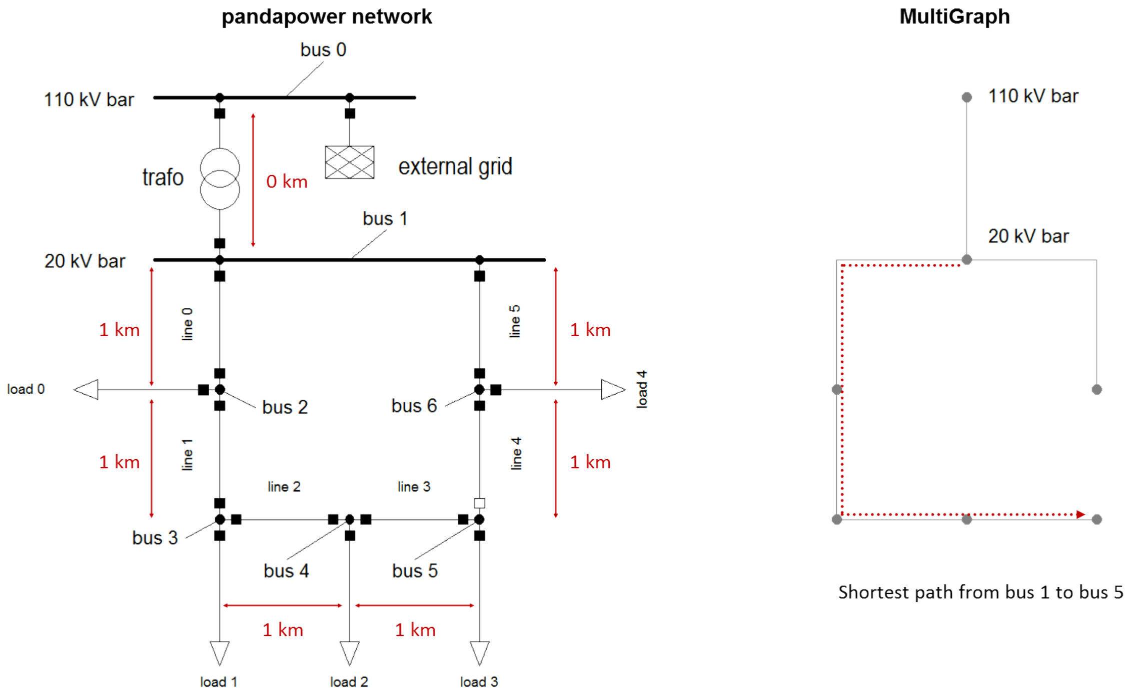
### 5.3.3 Calculate distances between buses

*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.

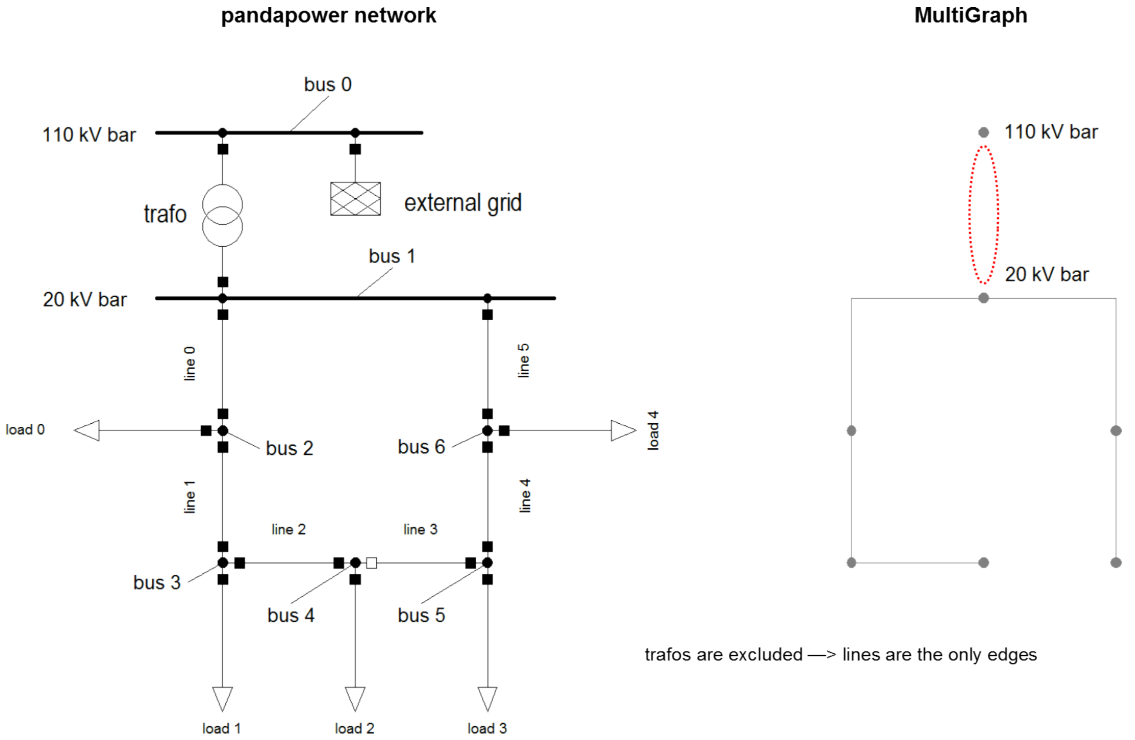



Shortest path from bus 1 to bus 5

### 5.3.4 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}
```

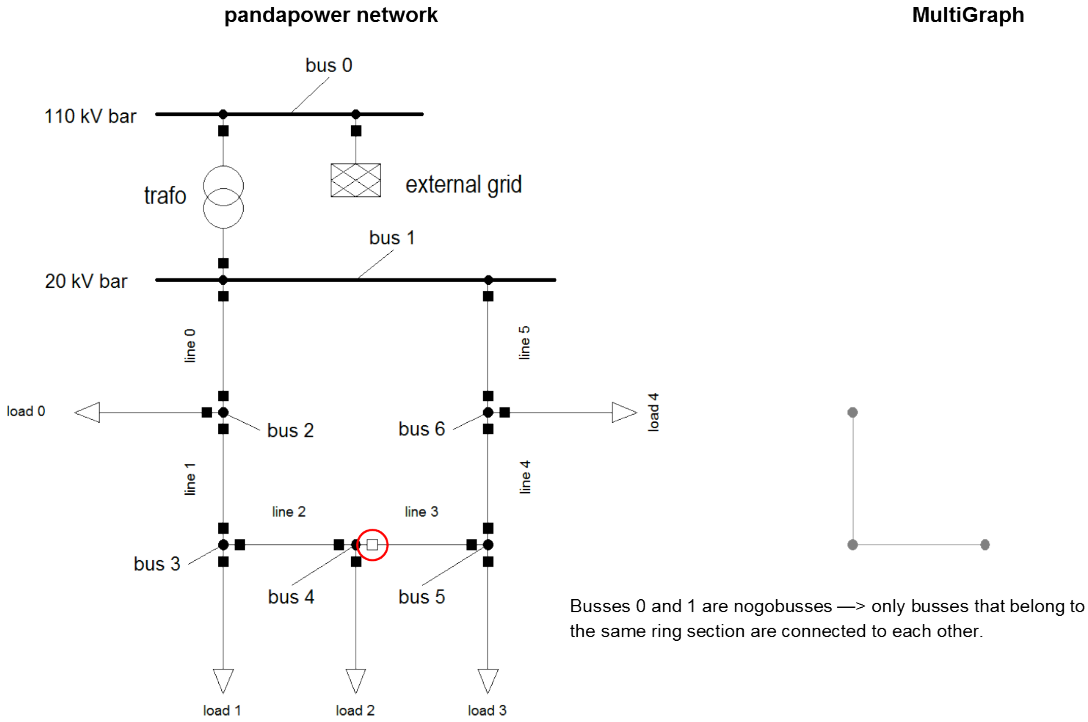



### 5.3.5 Find rings and ring sections

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.

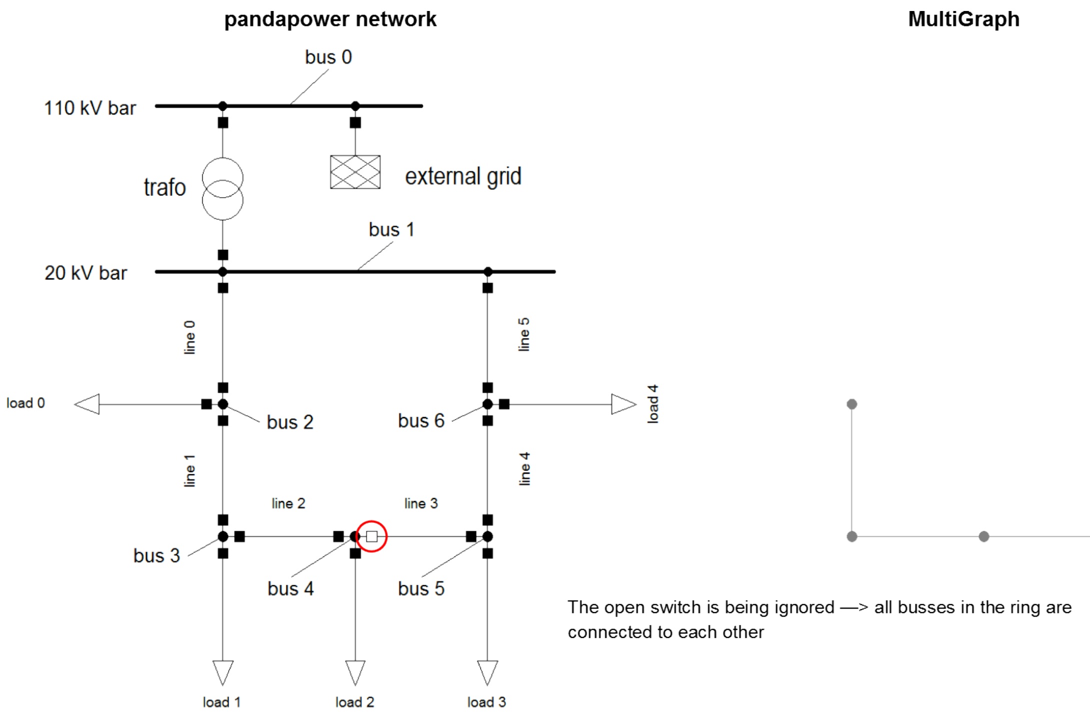
```
import pandapower.topology as top

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}
```




```python
import pandapower.topology as top

mg_ring = top.create_nxgraph(net, respect_switches = False, nogobuses = [0,1])
cc_ring = top.connected_components(mg_ring)
```

```
In      : next(cc_ring)
Out     : {2, 3, 4, 5, 6}
```




### 5.3.6 Find stubs

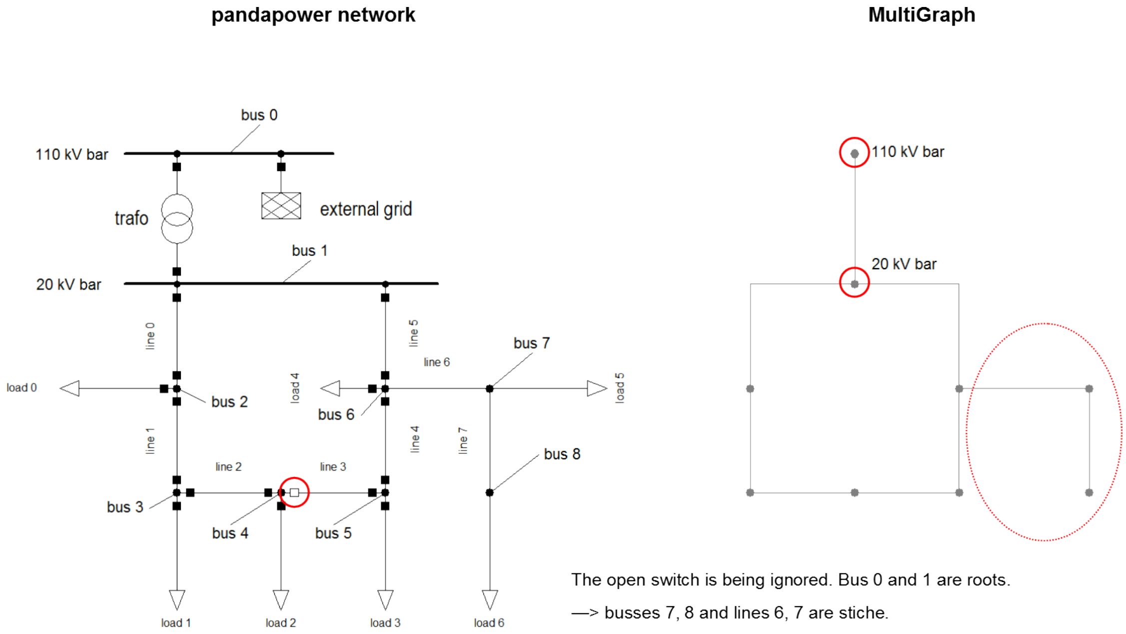
*determine_stubs* lets you identify buses and lines that are stubs. Open switches are being ignored. Busses that you want to exclude should be defined as roots. Ext_grid buses are roots by default.

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  imax_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.

# 6 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:

## 6.1 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.

### 6.1.1 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`**`()`

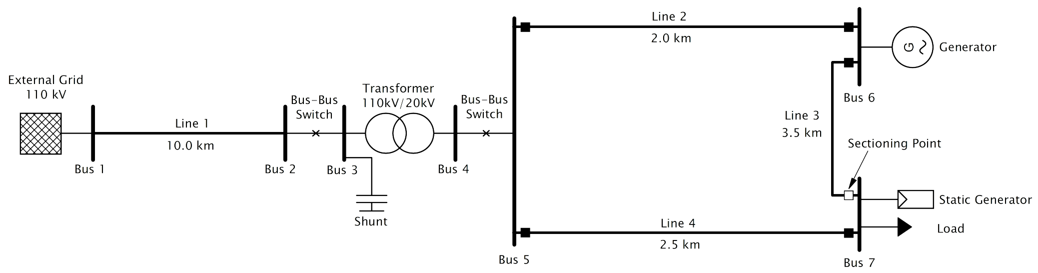
Returns the simple example network from the pandapower tutorials.

RETURN:

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.

### 6.1.2 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`**`()`

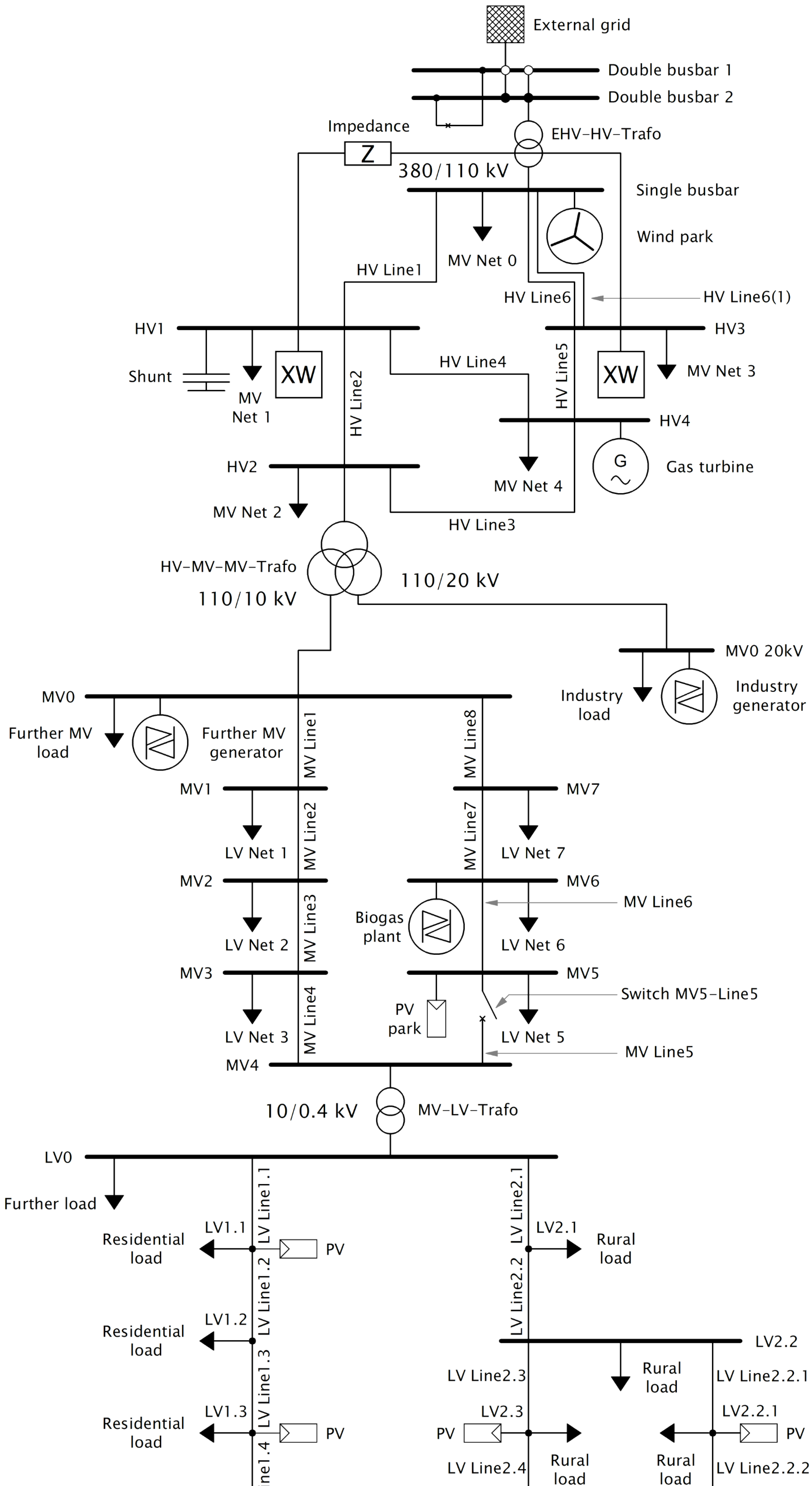
Returns the multivoltage example network from the pandapower tutorials.

RETURN:

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
G
Gas turbine
HV2
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
LV Net 1
MV Line2
MV Line7
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
LV2.1
Residential
load
PV
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
LV2.3
LV2.2.1
Residential
load
PV
PV
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.

## 6.2 Simple pandapower test networks

### 6.2.1 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.

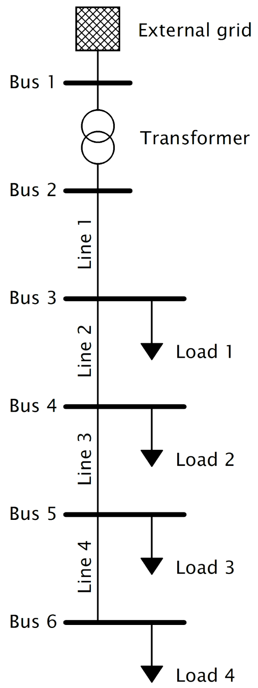
RETURN:

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

EXAMPLE:

import pandapower.networks as pn

net_four_load = pn.panda_four_load_branch()




### 6.2.2 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.

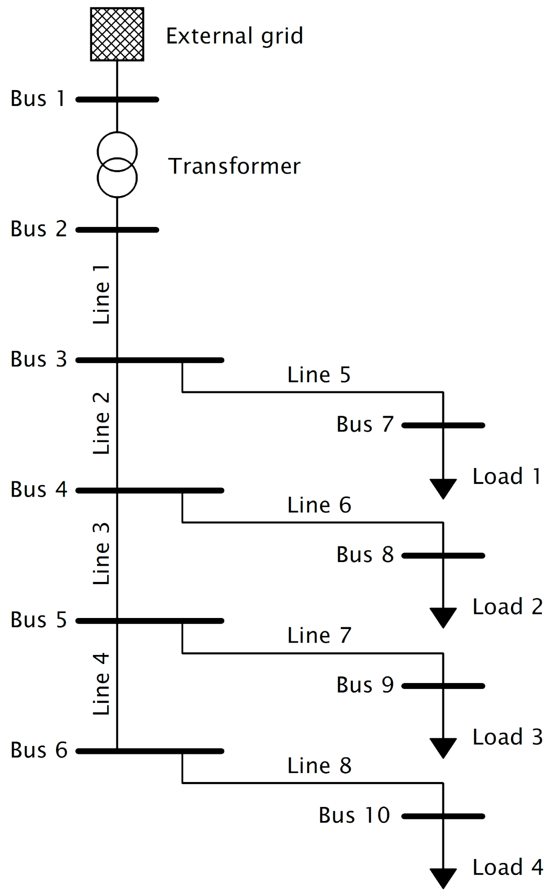
RETURN:

**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()




### 6.2.3 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.

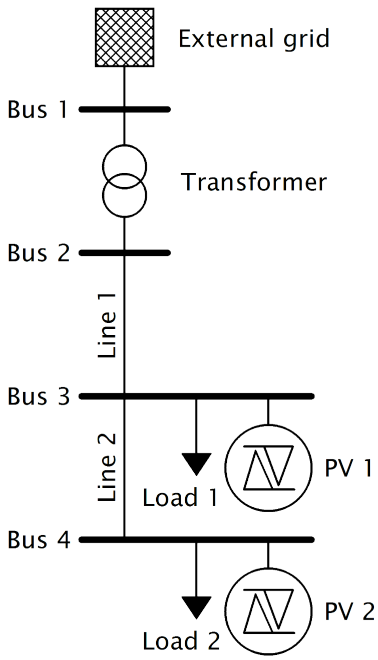
RETURN:

**net** - Returns the required four bus system

EXAMPLE:

import pandapower.networks as pn

net_simple_four_bus = pn.simple_four_bus_system()




### 6.2.4 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.

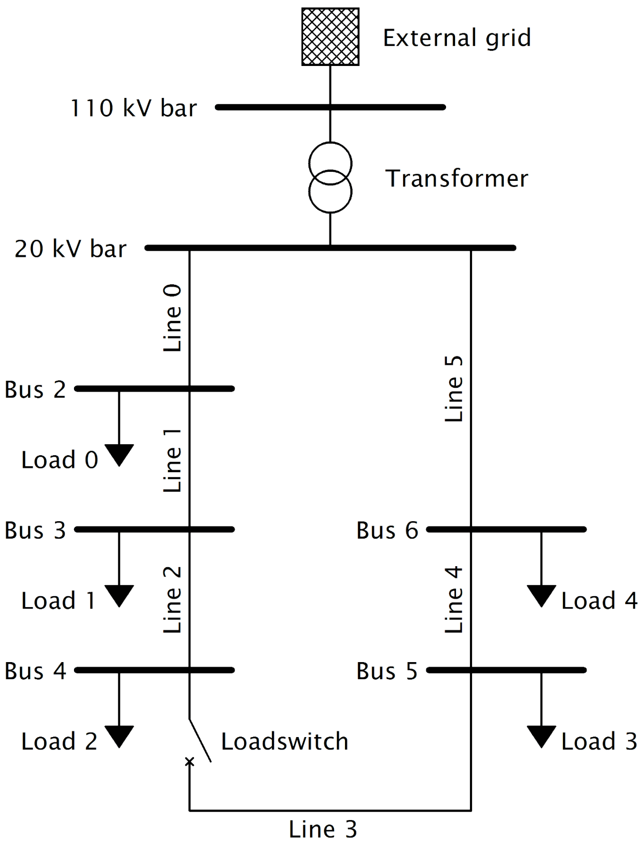
RETURN:

**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()




## 6.3 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".

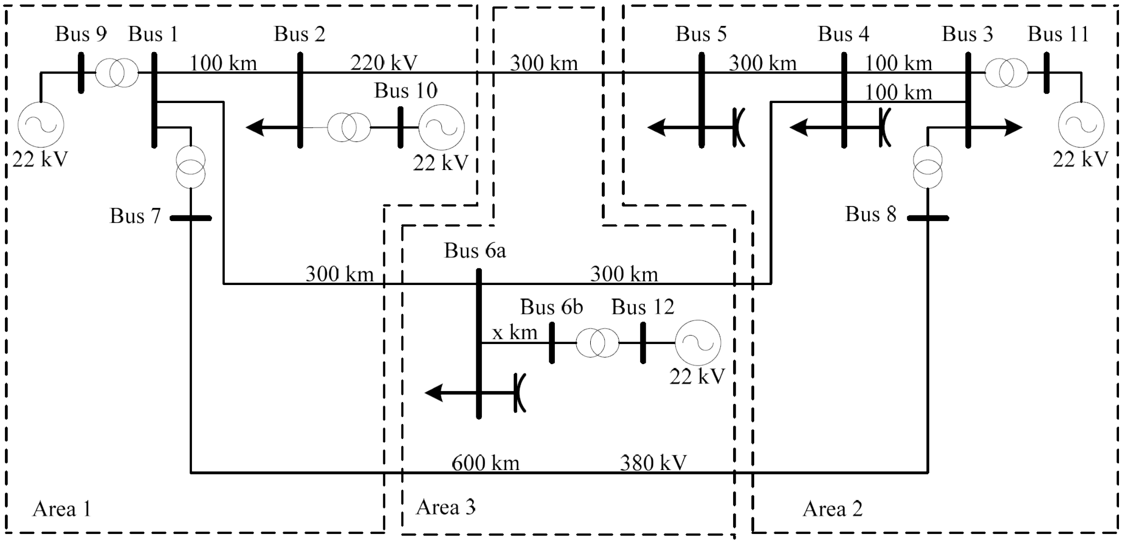
**Note:** Source for this network is the final Report of Task Force C6.04.02: "Benchmark Systems for Network Integration of Renewable and Distributed Energy Resources", 2014

See also: 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

### 6.3.1 High voltage transmission network

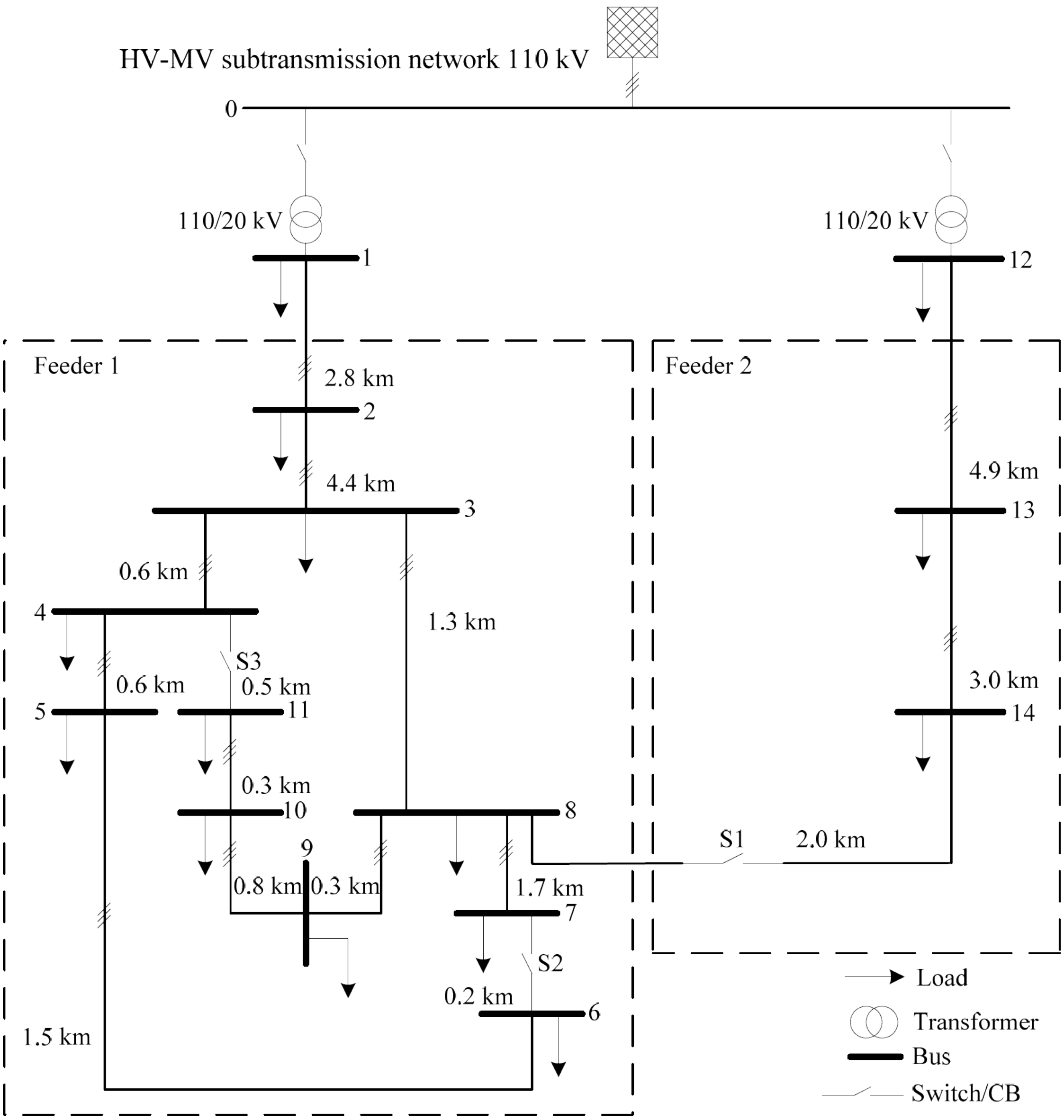
```
import pandapower.networks as pn

# 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)
'''
```




### 6.3.2 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)
'''
```




### 6.3.3 Medium voltage distribution network with DER

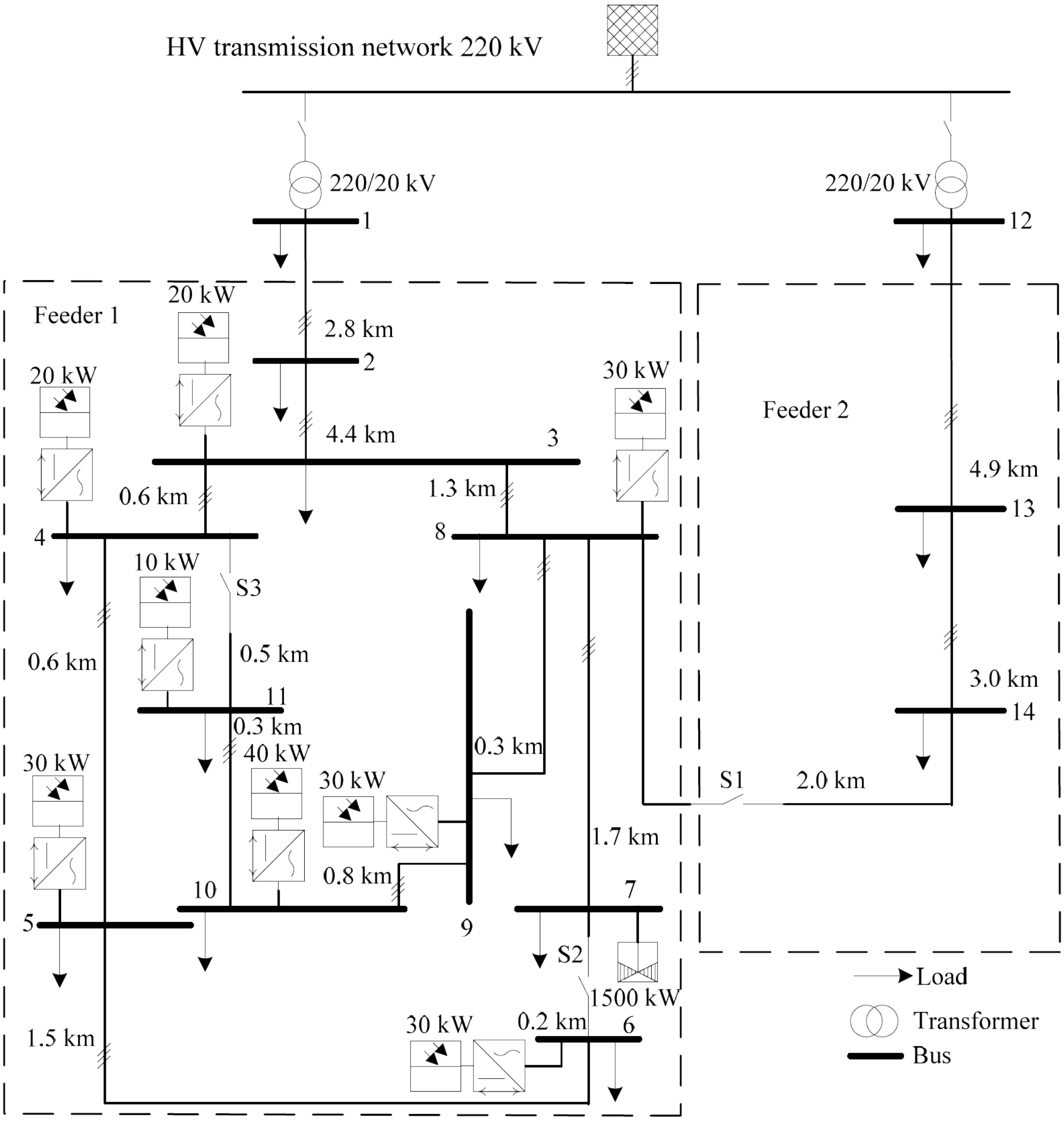
**Note:** This network contains additional 9 distributed energy resources compared to medium voltage distribution network:

- 8 photovoltaic generators
- 1 wind turbine

Compared to the CIGRE Task Force C6.04.02 paper 2 Batteries, 2 residential fuel cells, 1 CHP diesel and 1 CHP fuel cell are neglected.

```
import pandapower.networks as pn

net = pn.create_cigre_network_mv(with_der=True)

'''
This pandapower network includes the following parameter tables:
  - switch (8 elements)
```

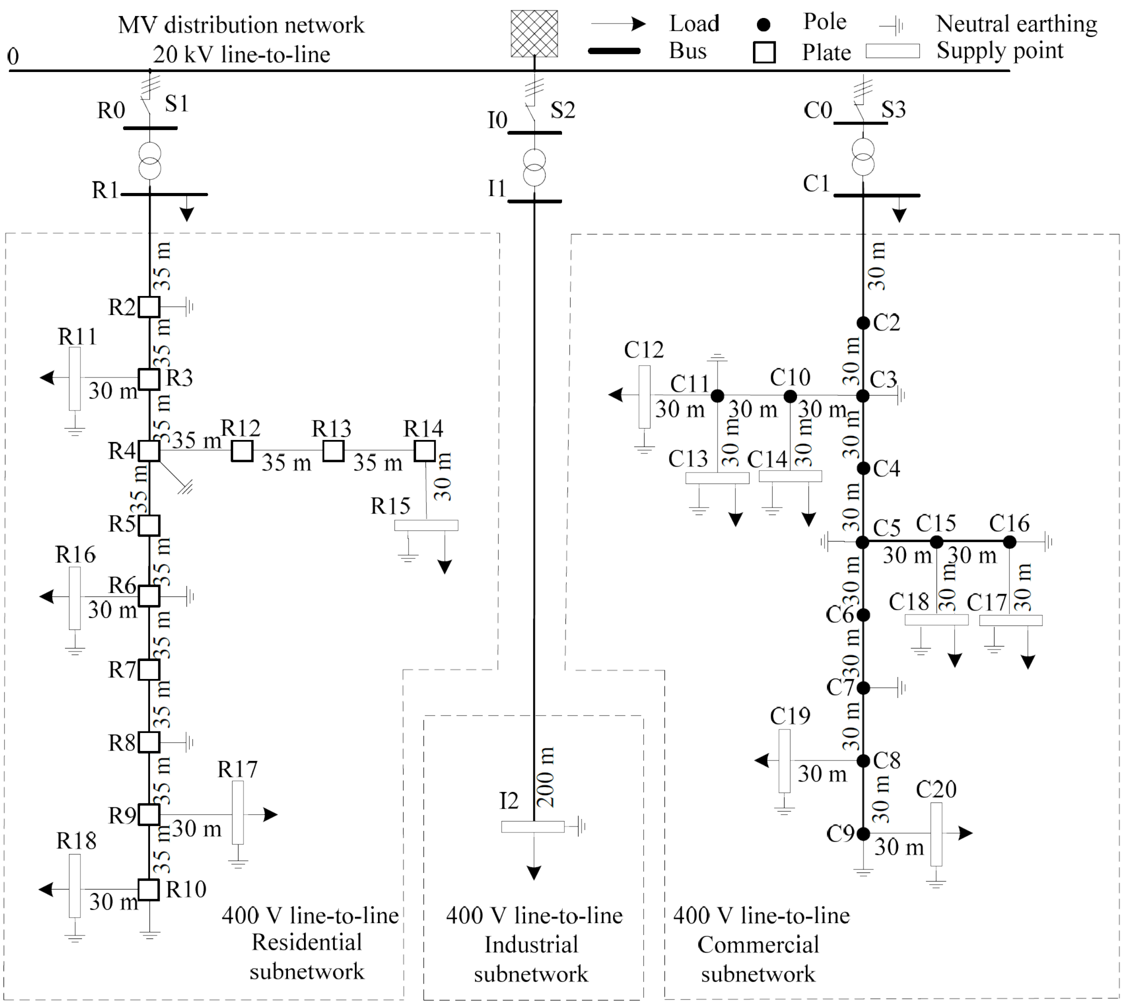
```
  - load (18 elements)
  - ext_grid (1 elements)
  - sgen (9 elements)
  - line (15 elements)
  - trafo (2 elements)
  - bus (15 elements)
'''
```




### 6.3.4 Low voltage distribution network

```
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)
'''
```




## 6.4 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

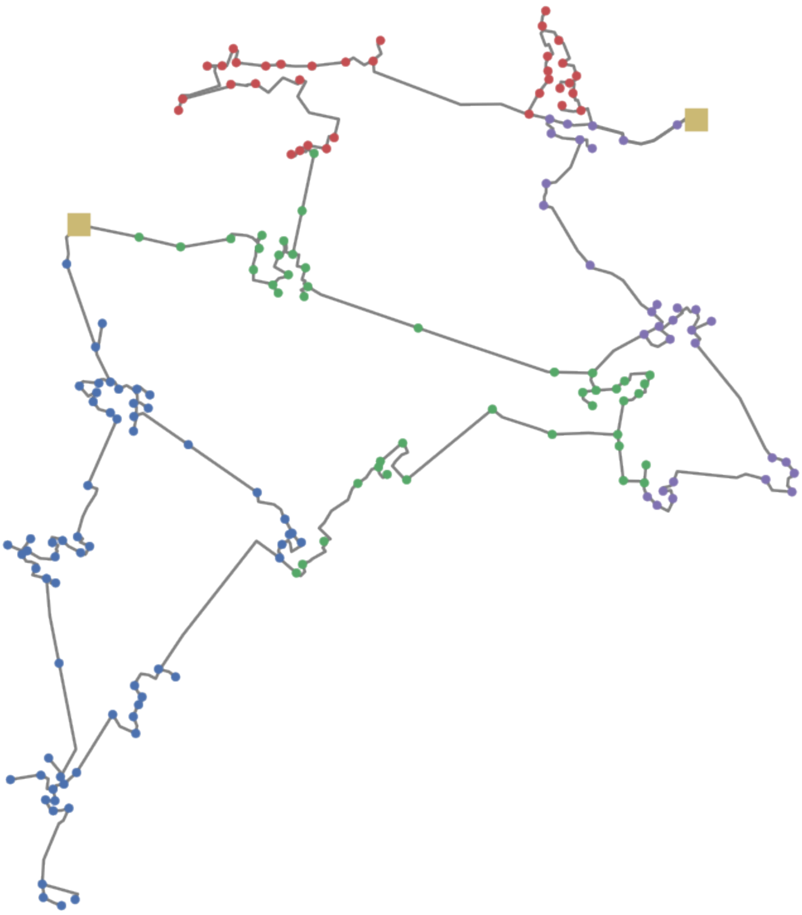
RETURN:

**net** - pandapower network

EXAMPLE:

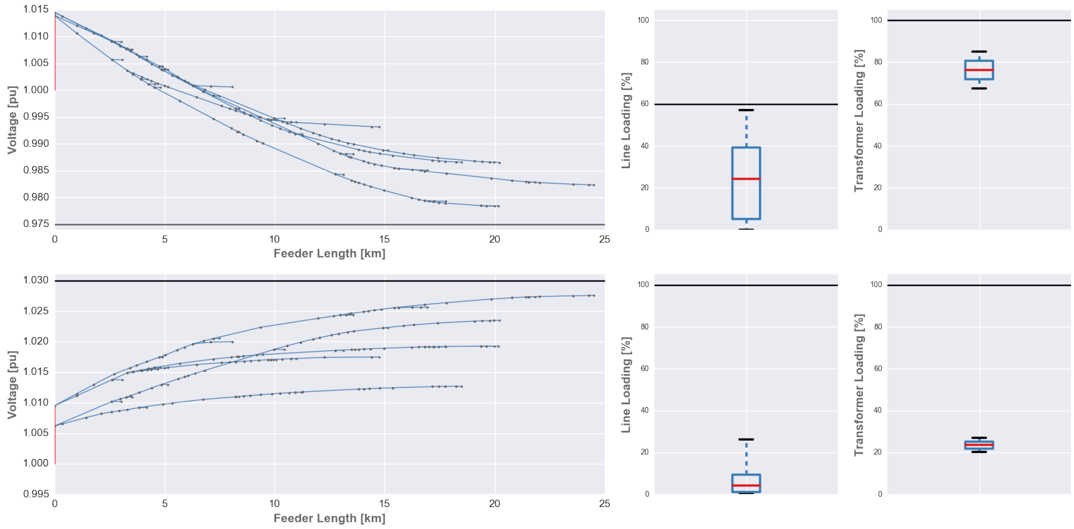
```
>>> import pandapower.networks
>>> net = pandapower.networks.mv_oberrhein("generation")
```

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.

## 6.5 IEEE cases

**Note:** All IEEE case files were converted from PYPOWER

### 6.5.1 Case 4gs

`pandapower.networks.case4gs()`

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

RETURN:

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

EXAMPLE:

import pandapower.networks as pn

net = pn.case4gs()

### 6.5.2 Case 6ww

`pandapower.networks.case6ww()`

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

RETURN:

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

EXAMPLE:

import pandapower.networks as pn

net = pn.case6ww()

### 6.5.3 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.

RETURN:

**net** - Returns the required ieee network case9

EXAMPLE:

import pandapower.networks as pn

net = pn.case9()

### 6.5.4 Case 9Q

`pandapower.networks.case9Q()`

Calls the pickle file case9Q.p which data origin is PYPOWER. This network is highly correlated to case9.

RETURN:

**net** - Returns the required ieee network case9Q

EXAMPLE:

import pandapower.networks as pn

net = pn.case9Q()

### 6.5.5 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 and Illinois University.

RETURN:

**net** - Returns the required ieee network case30

EXAMPLE:

import pandapower.networks as pn

net = pn.case30()

### 6.5.6 Case 30pwl

`pandapower.networks.case30pwl()`

Calls the pickle file case30pwl.p which data origin is PYPOWER. This network is highly correlated to case30.

RETURN:

**net** - Returns the required ieee network case30pwl

EXAMPLE:

import pandapower.networks as pn

net = pn.case30pwl()

### 6.5.7 Case 30Q

`pandapower.networks.case30Q()`

Calls the pickle file case30Q.p which data origin is PYPOWER. This network is highly correlated to case30.

RETURN:

**net** - Returns the required ieee network case30Q

EXAMPLE:

import pandapower.networks as pn

net = pn.case30Q()

## 6.6 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)

### 6.6.1 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 |

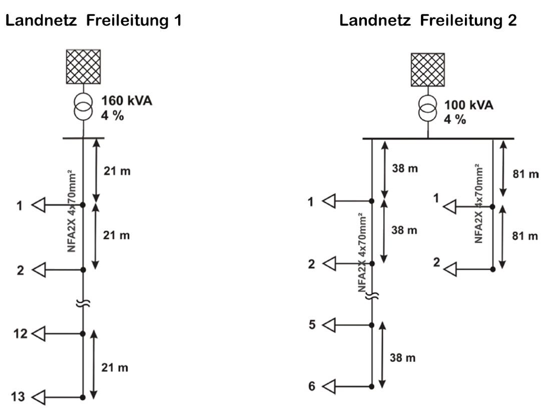
| | Lines | Total Length | Loads | Installed Power |
|---|---|---|---|---|
| Freileitung 2 | 8 | 0.390 km | 8 | 64 kW |
| Kabel 1 | 16 | 1.046 km | 8 | 64 kW |
| Kabel 2 | 28 | 1.343 km | 14 | 112 kW |
| | | | | |
| **Kerber Dorfnetz** | 114 | 3.412 km | 57 | 342 kW |
| | | | | |
| **Kerber Vorstadtnetze** | | | | |
| 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:

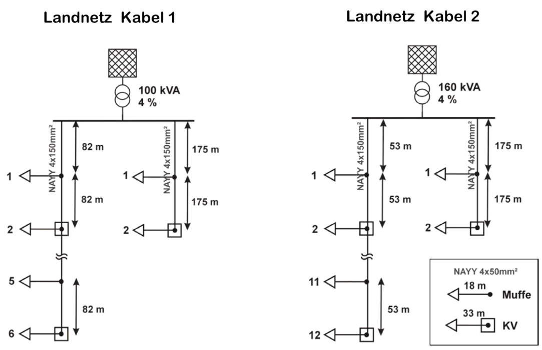
```
import pandapower.networks as pn

net1 = pn.create_kerber_net()
```

### 6.6.2 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)
'''
```

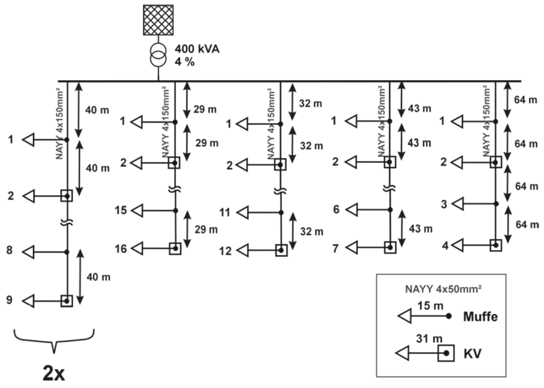


```
import pandapower.networks as pn

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)
'''
```



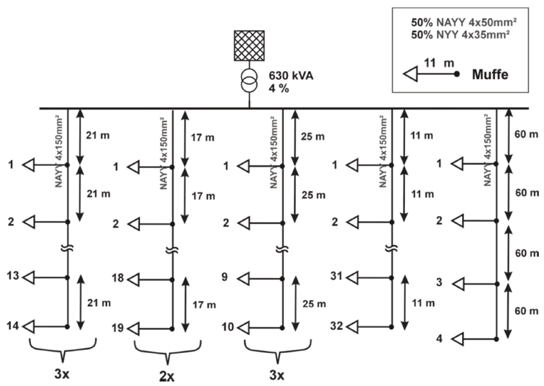

### 6.6.3 Kerber Dorfnetz

```
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)
'''
```

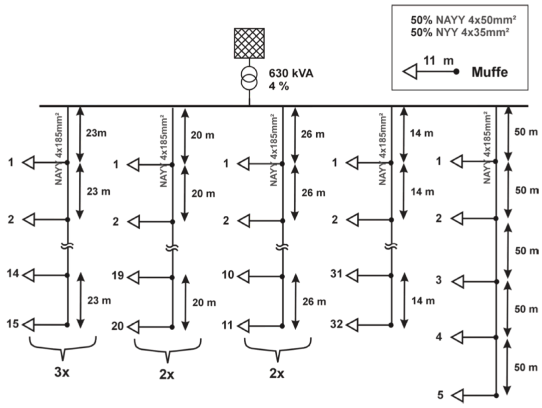



### 6.6.4 Kerber Vorstadtnetze

```
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)
'''
```




### 6.6.5 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 |
| | | | | |
| **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

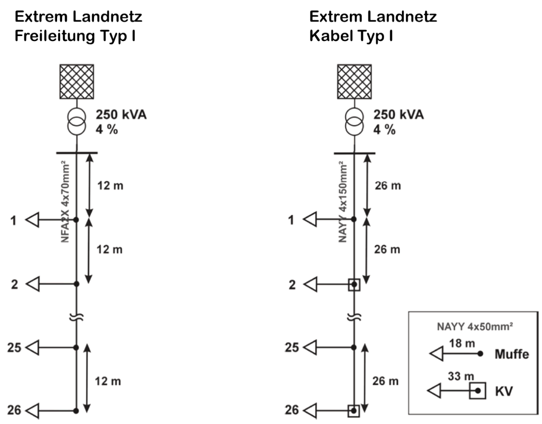
---

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

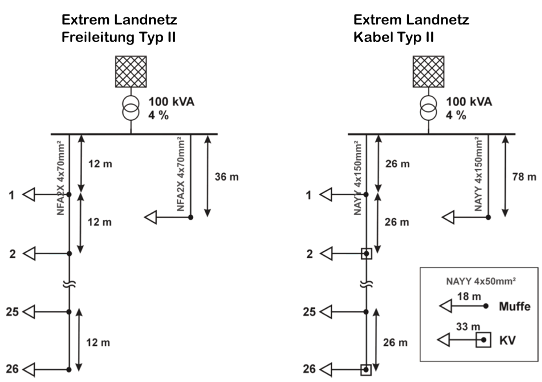
---

### 6.6.6 Extreme Kerber Landnetze

```python
import pandapower.networks as pn

'''Extrem Landnetz Freileitung Typ I'''
net = pn.kb_extrem_landnetz_freileitung()


'''Extrem Landnetz Kabel Typ I'''
net = pn.kb_extrem_landnetz_kabel()
```

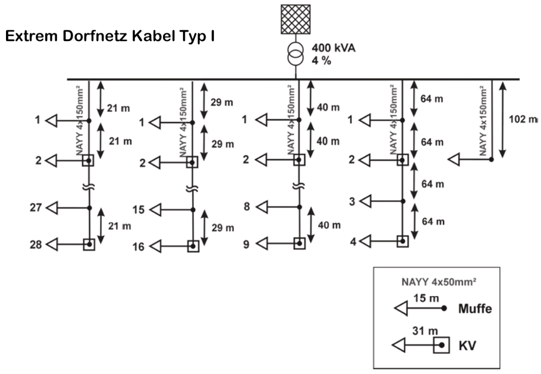



```python
import pandapower.networks as pn

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


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

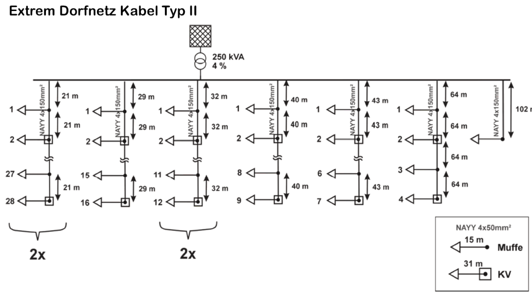



### 6.6.7 Extreme Kerber Dorfnetze

```python
import pandapower.networks as pn

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

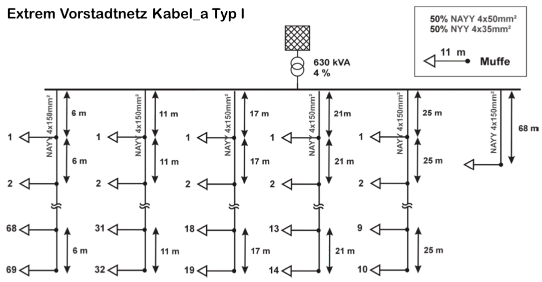



```
import pandapower.networks as pn

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

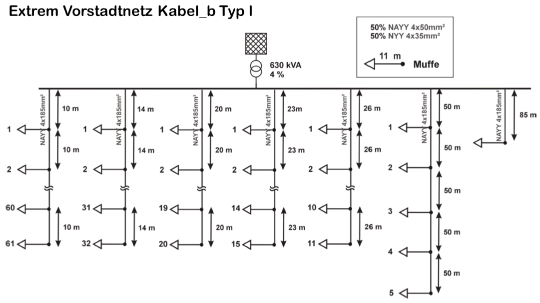



### 6.6.8 Extreme Kerber Vorstadtnetze

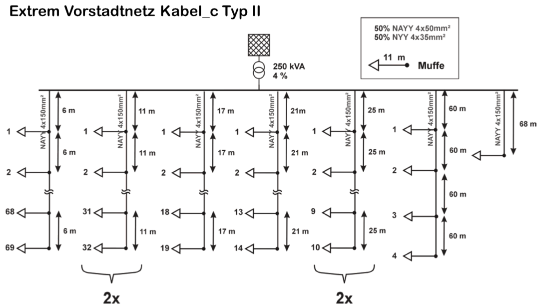
```
import pandapower.networks as pn

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



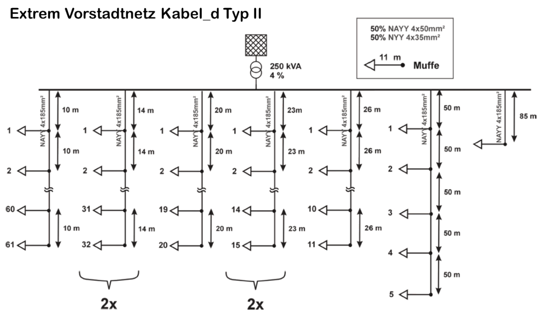

```
import pandapower.networks as pn

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




```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()
```



# 7 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.

## 7.1 Create Collections

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

### 7.1.1 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*, ***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 rectanlge
- "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

### 7.1.2 Branch Collections

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

## 7.2 Create Colormaps

### 7.2.1 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).

Return:

- 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])
```

### 7.2.2 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.

Return:

- 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])
```

## 7.3 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:

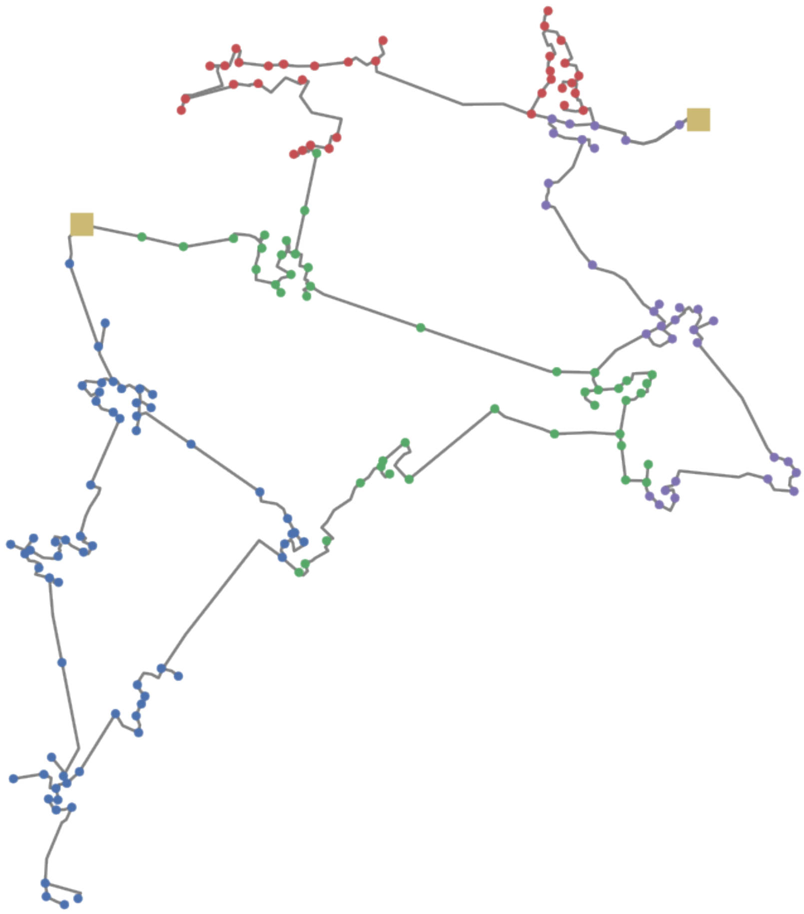
**collections** (list) - iterable of collection objects

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:



## 7.4 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

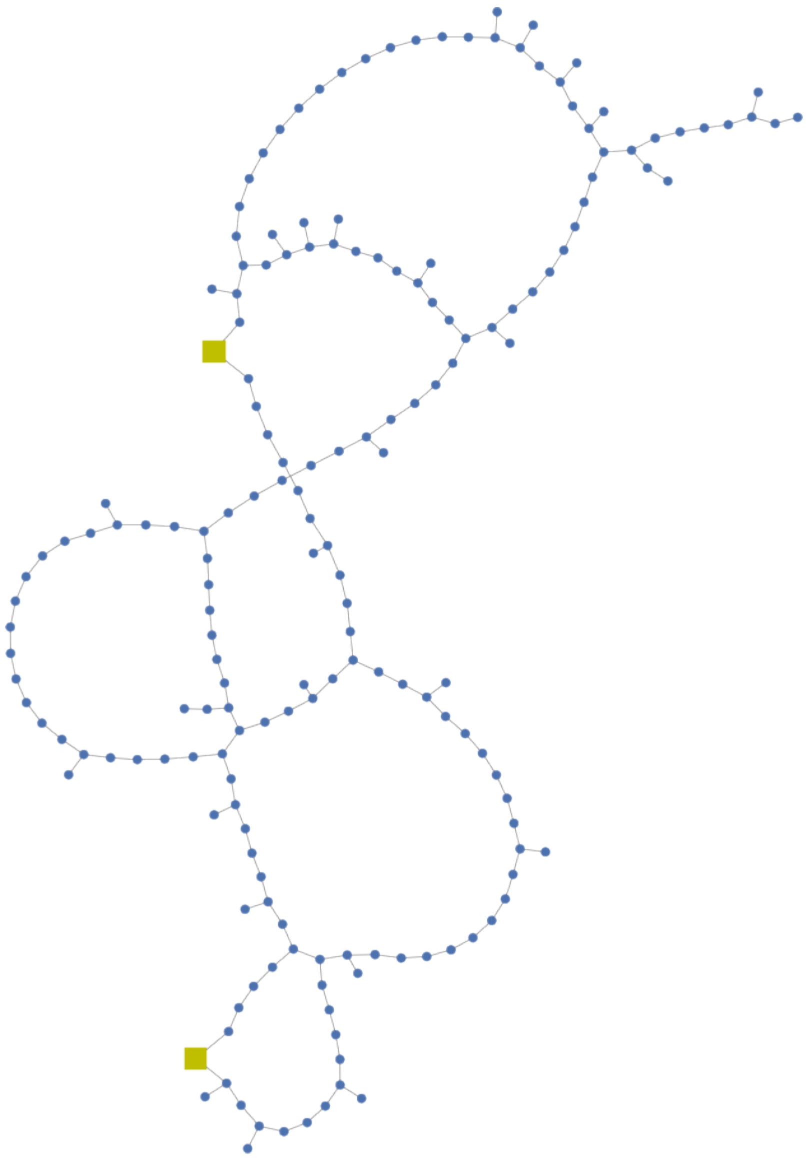
Output:

**net** - Pandapower network with added geo coordinates for the buses

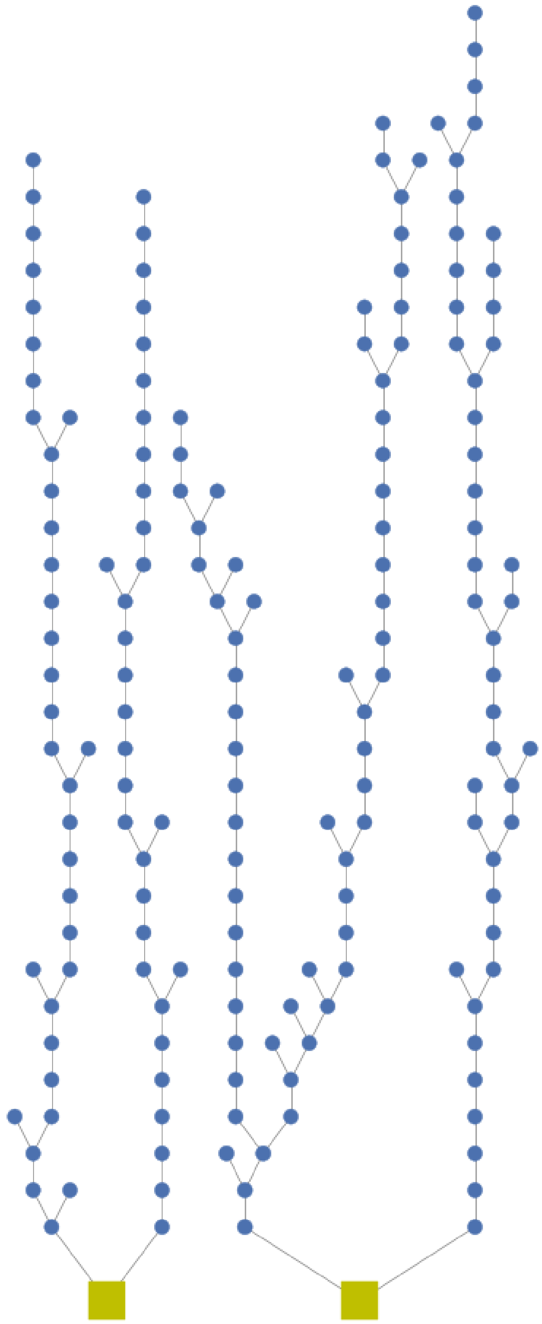
Example:

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):



# 8 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!

## 8.1 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.

RETURN:

**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
```

## 8.2 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.

RETURN:

**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
```

# 9 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.

---

## 9.1 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)

`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*, *u_min*, *u_max*)
Returns the results for all buses with if vm_pu is not within u_max and u_min or None, if there are none.

`pandapower.`**`equal_nets`**(*x*, *y*, *check_only_results=False*, *tol=1e-14*)
Compares 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 '_'

## 9.2 Simulation Setup and Preparation

`pandapower.`**`convert_format`**(*net*)
Converts old nets to new format to ensure consistency. 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** –
- **scaling** – A dictionary containing a mapping from element type to

scaling factor. :return:

## 9.3 Topology Modification

`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.

`pandapower.drop_inactive_elements(net)`
Drops any elements not in service AND any elements connected to inactive buses.

`pandapower.drop_buses(net, buses, also_drop_elements=True)`
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 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.

## 9.4 Item/Element Selection

`pandapower.get_element_index(net, element, name, exact_match=True, regex=False)`
Returns the element identified by the element string and a name.

**INPUT:** **net** - pandapower network

**element** - line indices of lines that are considered. If None, all lines in the network are considered.

**name** - if True, switches are also considered as connected elements.

OUTPUT:

**connections** - pandapower Series with number of connected elements for each bus. If a selection of lines is given, only buses connected to these lines are in the returned series.

`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

RETURN:

**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

RETURN:

**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

RETURN:

**cl** (set) - Returns connected buses.