SMART GRIDS: FROM TRADITIONAL TO MODERNIZED RESILIENT SYSTEMS

Víctor Escala García

Josep Fanals Batllori

Pol Heredia Julbe

Roger Izquierdo Toro

Palina Nicolas

SMART GRIDS

Master's degree in Energy Engineering

Master's degree in Electric Power Systems and Drives

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Tècnica Superior d'Enginyeria
Industrial de Barcelona

CONTENTS

1. INTRODUCTION

2. PHASE 1

A system such as the one displayed in Figure 1 is analyzed. The network operates at the transmission level and feds dispersed demand points that symbolyze distribution grids. The grid has an interconnection with a transmission grid, and at the same time, some power is provided by the nuclear power plant. Initially, there are no renewable power plants nor storage systems, which compromises the security of the grid.

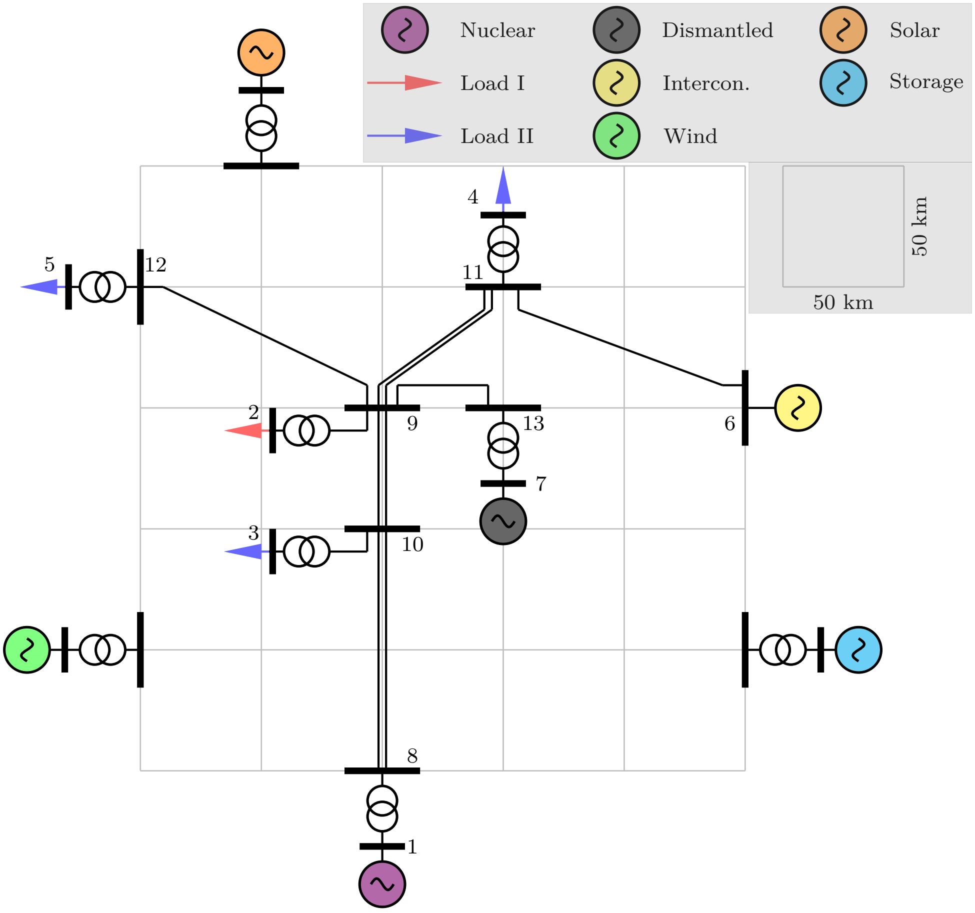

Figure 1. Overview of the network

The first step to analyze the system is to know the demand and the generation profile. In order to model them, the hourly demand and generation data of Spain have been collected [1]. To obtain a typical working day, a statistical analysis has been performed taking into account only the days from the 1st of January to the 31st of March, from Tuesday to Thursday and removing the national holidays. The result then has been normalized. analyze the system is This way, the consumption profile is obtained by from the product of the normalized demand and the peak power consumption of 375 MW for load type I and 140 MW for type II. For the generation profile, for simplicity, it has been assumed that the nuclear power plant follows the demand curve, i.e., it is not acting as a constant generator.

Hour \ Bus	1	2	3	4	5	6	8	9	10	11	12
0	1.050	0.963	0.989	0.972	0.937	1.000	1.031	0.986	1.000	0.984	0.950
1	1.050	0.974	0.998	0.982	0.952	1.000	1.035	0.995	1.008	0.993	0.963
2	1.050	0.981	1.003	0.988	0.961	1.000	1.037	1.001	1.013	0.998	0.972
3	1.050	0.984	1.005	0.990	0.965	1.000	1.038	1.003	1.015	1.000	0.975
4	1.050	0.985	1.006	0.991	0.966	1.000	1.038	1.004	1.015	1.001	0.976
5	1.050	0.981	1.003	0.988	0.961	1.000	1.037	1.001	1.013	0.998	0.972
6	1.050	0.967	0.992	0.975	0.942	1.000	1.032	0.989	1.003	0.987	0.955
7	1.050	0.938	0.969	0.950	0.905	1.000	1.022	0.965	0.983	0.964	0.920
8	1.050	0.915	0.952	0.931	0.876	1.000	1.014	0.947	0.967	0.946	0.893
9	1.050	0.904	0.944	0.921	0.862	1.000	1.010	0.938	0.960	0.938	0.880
10	1.050	0.900	0.941	0.918	0.856	1.000	1.009	0.935	0.957	0.935	0.875
11	1.050	0.901	0.941	0.919	0.857	1.000	1.009	0.936	0.958	0.935	0.876
12	1.050	0.904	0.944	0.921	0.861	1.000	1.010	0.938	0.960	0.938	0.879
13	1.050	0.906	0.945	0.923	0.864	1.000	1.011	0.939	0.961	0.939	0.882
14	1.050	0.916	0.953	0.931	0.877	1.000	1.014	0.948	0.968	0.947	0.894
15	1.050	0.922	0.957	0.936	0.884	1.000	1.016	0.953	0.972	0.952	0.901
16	1.050	0.925	0.960	0.939	0.889	1.000	1.018	0.955	0.974	0.954	0.905
17	1.050	0.926	0.961	0.940	0.890	1.000	1.018	0.956	0.975	0.955	0.906
18	1.050	0.921	0.957	0.936	0.884	1.000	1.016	0.952	0.972	0.951	0.900
19	1.050	0.903	0.943	0.920	0.860	1.000	1.010	0.937	0.959	0.937	0.878
20	1.050	0.891	0.933	0.910	0.844	1.000	1.005	0.927	0.950	0.927	0.863
21	1.050	0.897	0.939	0.915	0.853	1.000	1.008	0.933	0.955	0.932	0.871
22	1.050	0.921	0.957	0.936	0.884	1.000	1.016	0.952	0.972	0.951	0.900
23	1.050	0.946	0.975	0.957	0.915	1.000	1.025	0.972	0.988	0.970	0.929

Table 1. Voltage profile, in pu, for 24 hours

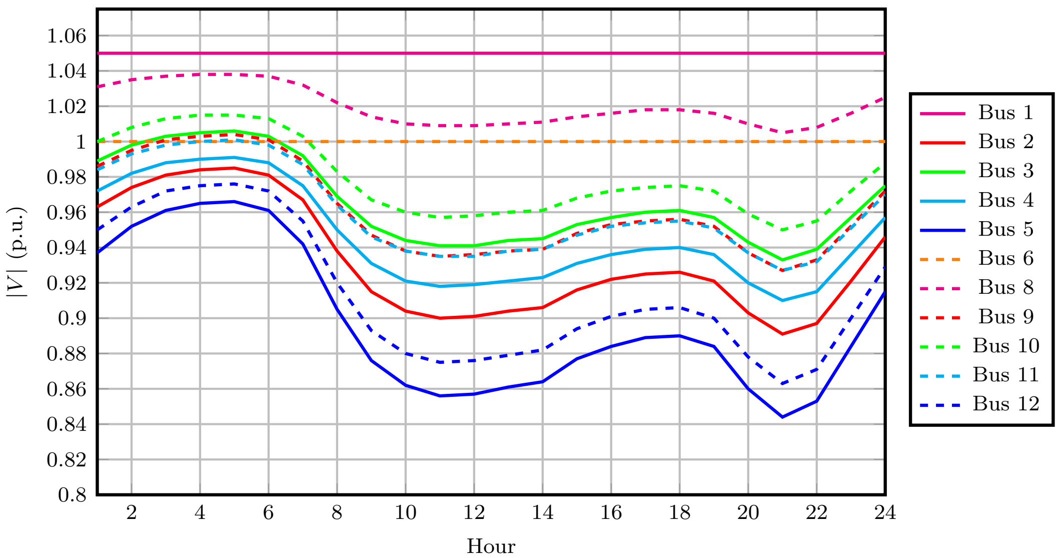

Figure 2. Voltage profile during 24 hours for the initial grid. The low-voltage buses are plotted in solid lines; the high-voltage ones are in dashed lines.

Hour \ Load	8-10	10-9	9-11	9-12	11-6
0	26.423	19.999	9.388	30.774	63.562
1	24.385	18.464	8.850	28.312	59.190
2	23.089	17.491	8.517	26.761	56.451
3	22.549	17.087	8.381	26.117	55.319
4	22.458	17.019	8.358	26.009	55.130
5	23.082	17.486	8.515	26.752	56.435
6	25.729	19.475	9.203	29.932	62.065
7	30.615	23.176	10.549	35.928	72.752
8	34.100	25.846	11.559	40.331	80.765
9	35.690	27.073	12.031	42.383	84.443
10	36.296	27.542	12.213	43.173	85.847
11	36.166	27.441	12.174	43.002	85.544
12	35.740	27.111	12.046	42.448	84.558
13	35.480	26.910	11.969	42.110	83.956
14	33.997	25.766	11.528	40.198	80.526
15	33.090	25.069	11.262	39.042	78.435
16	32.569	24.669	11.110	38.381	77.234
17	32.392	24.534	11.059	38.158	76.828
18	33.187	25.143	11.290	39.165	78.657
19	35.865	27.208	12.084	42.610	84.847
20	37.641	28.588	12.620	44.945	88.969
21	36.689	27.847	12.332	43.688	86.759
22	33.143	25.110	11.278	39.109	78.557
23	29.319	22.190	10.183	34.319	69.877

Table 2. Percentual loading of the lines for a full day operation

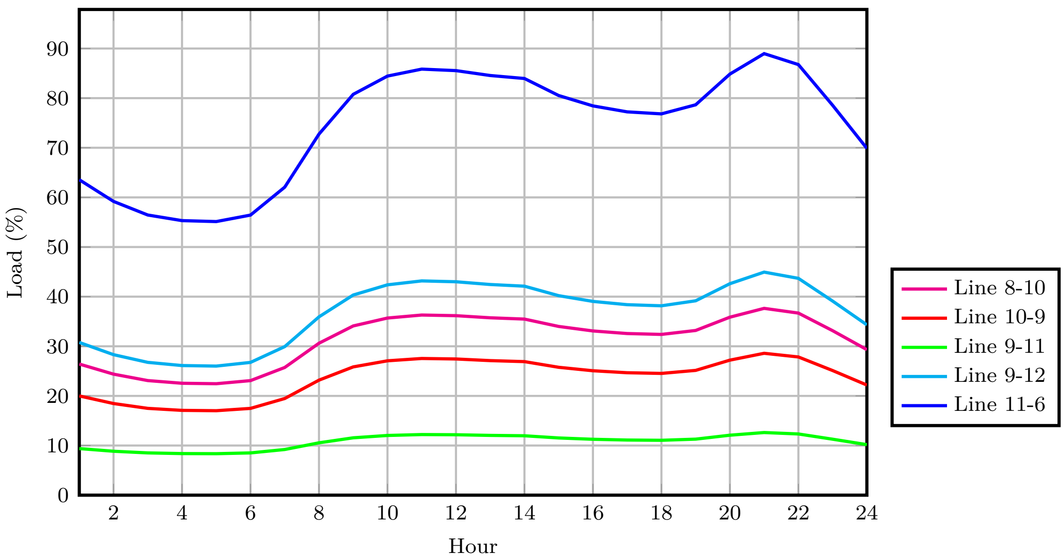

Figure 3. Representation of the percentual loading of the lines during 24 hours

2.1. Operating costs

Regarding the operating costs, some estimations are made in order to asses the influence of importing energy and the impact of faults on lines and transformers.

First, the cost of importing energy depends on the time zone: valley, flat or peak. The analysis that follows considers a working day, which is precisely the date for which the voltages and loading profiles have been shown in Figures 2 and 3 respectively. The cost of importing the energy is mathematically expressed as:

$$C_{imp} = \sum_{k=1}^{n=24} P_{s,k}c(k), \tag{1}$$

where C_{imp} stands for the cost of importing energy for a full day, k denotes the index of a given hour, n the total number of hours in a day, $P_{s,k}$ the energy provided by the slack bus (interconnection point) in MWh at hour k, and $c(k)$ the cost at a certain hour in €/MWh. This last term is equal to 45 €/MWh from 0 to 8 hours, 65 €/MWh from 8 to 10, 14 to 18 and 22 to 24 hours, and 90 €/MWh from 10 to 14 and 18 to 22 hours.

Equation 1 can be treated as a weighting sum. With the generation data obtained from the timeseries power flow, the total importing cost of importing energy becomes 418753.03 €/day, or about 152.84 M€ in a full year. It is important to note that the study related to the cost of importing energy is decoupled from the fault analysis. This is not a hundred percent realistic, because it could be that a switch trips and hence a line or a transformer are disconnected. Then, it could happen that the interconnection has to provide more power. However, since the probabilities are extremely low, they are discarded when computing this cost.

On the other hand, there are the costs due to faults in transformers or lines. About 0.05 failures per km and year are expected in lines, while transformers are meant to fail 0.15 times a year. The penalty for not providing energy is 180 €/MWh. Given that the length of the lines has an impact on its probability of failure, Table 3 shows the length and the subsequent failures per year.

Line	Length (km)	Failures/year
8-10	100.00	5.00
10-9	50.00	2.50
9-11	70.71	3.54
9-12	111.80	5.59
11-6	111.80	5.59

Table 3. Length and failures per year of all active lines

Figure 3 shows that the line connected to the interconnection point operates at a high load. It is critical to note that if line 8-10 fails, the slack should provide all power, but this would result in exceeding the thermal capacity of the line. Thus, if line 8-10 fails, no power can reach the loads.

In the case of line failures, there is a total disconnection time of 2.5 hours; instead, for transformers it is 8 hours. The expected time that an element will be disconnected in a year is found by multiplying the aforementioned disconnection time by the number of failures that take place during a year. Table 4 displays the yearly disconnection time and explains the consequences spotted by running the power flow. This will allow to estimate the penalties due disconnection.

Element	Disconnection time (h)	Consequences
Line 8-10	12.50	No load served - divergence
Line 9-10	6.25	No load served - divergence
Line 9-11	8.85	Loads at buses 2, 3 and 5 unserved
Line 9-12	13.98	Load at bus 5 unserved
Line 11-6	13.98	No load served
Trafo 1-8	1.20	No load served - divergence
Trafo 2-9	1.20	Load at bus 2 unserved
Trafo 3-10	1.20	Load at bus 3 unserved
Trafo 4-11	1.20	Load at bus 4 unserved
Trafo 5-12	1.20	Load at bus 5 unserved

Table 4. Disconnection time and consequences of losing each element

Once the unserved loads and the associated disconnection times are known, the next step has to do with applying the penalty as follows:

$$C_{discon} \approx \sum_{i=1}^{10} \overline{P}_{uns,i} t_{discon,i} C_p, \tag{2}$$

where C_{discon} is the total disconnection cost, i represents the index of the line or transformer with a total of 10 elements prone to be disconnected (see Table 4), $\overline{P}_{uns,i}$ is the mean unserved power, $t_{discon,i}$ the disconnection time, and C_p the penalty cost to apply. Equation 2 is an approximation in the sense that the unserved power varies according to the time of the day. To not overcomplicate the problem, it has been decided to pick a representative value such as the average.

The application of Equation 2 yields a total yearly penalty cost of 4.99 M€. For the most part, it is due to the disconnection of lines. Meshing more the system would decrease this cost, but on the other side, it would increase the investment cost. Hence, there is a trade-off between cost and reliability.

All the calculations related to costs have not set an inferior limit to the voltages. However, some of them are likely to be unacceptable in reality. The project will proceed to discuss solutions to this issue in the following phases.

2.2. Problem identification

The network modeled presents some serious issues. First, in case of fault, the demand cannot be covered. The network is a ramified line but does not have any interconnection within the system. If a fault occurs, the two branches are not connected and have to support the demand of the remaining part on its own. In the case of the nuclear power plant, it cannot produce enough energy to fulfill all the demand and in the case of the interconnection, if it was to cover all, it would be overloaded.

This leads to the second problem the network faces, there is a risk of overloading. This may happen in case or fault or if the the nuclear power plant shuts down because there is no other source of generation. This could lead to burning hence security and material damage issue. A third drawback is the high impact of the interruptions. As many line are single lines and there are no multiple connections, only ramifications, a fault has a high chance to directly disconnect the network. Finally, the voltage cannot be kept constant enough. It is usually accepted to fluctuate 10% around the nominal 1 p.u. while in the current transmission network, the voltage reaches almost 0.8 p.u..

2.3. Solution suggestion

As some of the lines present some overloading and demand coverage problems, we suggest improving lines to better ones which are able to transport more power. In order to do so, the critical lines could be changed from single to double lines and/or even change the conductors to thicker cables which allow a larger amount of power flow. Another approach would be to add more lines to the grid to overcome the demand coverage but we must also be aware that when adding new lines to the system we are also increasing the possibility of line failures which may affect interruptibility and cause economic losses to the system.

On the other hand, adding generation points to the system would also help overcome the stated problems in the previous point. If power is more accessible in different locations, the demand can be fulfilled from various points without saturating the most critical lines while evenly distributing the generation. Finally, another solution could be to change the 220 kV existing lines to 400 kV ones in order to allow these to transport higher amounts of power. With this change, only the amount of power transported would be around three times higher than the current one. However, it has to be taken into account that there would have to be an additional transformer to adapt the 220 kV from the interconnection to 400 kV, and the rest of the transformers would have to be replaced to match nominal voltages.

3. CODE

```
import pandapower as pp
import pandas as pd
import numpy as np
import pandapower.control as control
import pandapower.networks as nw
import pandapower.timeseries as timeseries
from pandapower.timeseries.data_sources.frame_data import DFData
from pandapower.plotting import simple_plot

from line_param_calc import calc_line



pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

def initialize_net(path_bus, path_geodata, path_line, path_demand, path_busload, path_generation
     , path_busgen, path_trafo):
    """
    initialize the grid from the .csv files

    :param path_bus: path to the bus .csv file
    :param geodata: path to the geodata .csv file
    :param path_line: path to the line .csv file
    :param path_demand: path to the normalized demand .csv file
    :param busload: path to the bus-load look up table .csv file
    :param path_generation: path to the normalized generation .csv file
    :param busgen: path to the bus-generator look up table .csv file
    :param trafo: path to the trafo .csv file
    :return: the net class
    """

    def create_bus(path_bus, path_geodata):
        """
        adapts the data from the bus file (if needed)

        :param path_bus:
        :param path_geodata:
        :return: the net with the buses added
        """

        df_bus = pd.read_csv(path_bus)
        df_geodata = pd.read_csv(path_geodata)

        net.bus = df_bus

        # adapt geodata
        for ll in range(len(df_geodata)):
            indx_bus = pp.get_element_index(net, "bus", df_geodata['name'][ll])
            df_geodata['name'][ll] = indx_bus

        net.bus_geodata = df_geodata

        return net


    def create_line(path_line):
        """
        adapts the data from the line file

        :param path_line:
        :return: the net with the lines added
        """

        df_line = pd.read_csv(path_line)
```

```
        for _, line in df_line.iterrows():
            from_bus = pp.get_element_index(net, "bus", line.from_bus)
            to_bus = pp.get_element_index(net, "bus", line.to_bus)

            rr, xx, cc, imax = calc_line(line.a,
                                         line.b,
                                         line.c,
                                         line.d,
                                         line.e,
                                         line.max_i,
                                         int(line.parallel))

            pp.create_line_from_parameters(net,
                                           from_bus,
                                           to_bus,
                                           length_km=line.length,
                                           r_ohm_per_km=rr,
                                           x_ohm_per_km=xx,
                                           c_nf_per_km=cc,
                                           max_i_ka=imax,
                                           name=line.name_l,
                                           parallel=line.parallel)

        return net


    def create_load(path_demand, path_busload, path_bus):
        """
        adapts the load files

        :param path_demand:
        :param path_busload:
        :param path_bus:
        :return: the net with the loads added
        """

        df_demand = pd.read_csv(path_demand)
        df_busload = pd.read_csv(path_busload)
        df_bus = pd.read_csv(path_bus)

        # create basic load dataframe
        # find the bus index of each load
        load_indx = []
        for _, load in df_busload.iterrows():
            bus_load = pp.get_element_index(net, "bus", load.bus)
            load_indx.append(bus_load)

        load_indx = pd.DataFrame(load_indx)
        load_indx = load_indx.rename(columns={0: "bus"})

        # load name and peak power
        load_name = df_busload['bus']
        load_pmw = df_busload['p_mw']
        load_qmvar = df_busload['q_mvar']

        # merge in a full dataframe
        headers = ["name", "bus", "p_mw", "q_mvar"]
        df_load = pd.concat([load_name, load_indx, load_pmw, load_qmvar], axis=1)
        df_load.columns.values[0] = "name"

        # create time series from the basic load df
        Nt = len(df_demand)
        Nl = len(df_load)
        pmw_ts = np.zeros((Nt, Nl), dtype=float)
        qmvar_ts = np.zeros((Nt, Nl), dtype=float)
        for i in range(Nt):  # number of time periods
            pmw_ts[i,:] = df_load['p_mw'][:] * df_demand['norm'][i]
```

```
        qmvar_ts[i,:] = df_load['q_mvar'][:] * df_demand['norm'][i]

    # form loads as a static picture (initial time)
    for ll in range(len(df_busload)):
        pp.create_load(net, bus=load_indx['bus'][ll], p_mw=pmw_ts[0, ll], q_mvar=qmvar_ts[0, ll], name=load_name[ll], index=int(ll))

    # timeseries
    df_pload_ts = pd.DataFrame(pmw_ts, index=list(range(Nt)), columns=net.load.index)
    df_qload_ts = pd.DataFrame(qmvar_ts, index=list(range(Nt)), columns=net.load.index)
    ds_pload_ts = DFData(df_pload_ts)
    ds_qload_ts = DFData(df_qload_ts)
    const_load = control.ConstControl(net, element='load', element_index=net.load.index, variable='p_mw', data_source=ds_pload_ts, profile_name=net.load.index)
    const_load = control.ConstControl(net, element='load', element_index=net.load.index, variable='q_mvar', data_source=ds_qload_ts, profile_name=net.load.index)  # add the reactive like this?

    return net


def create_generator(path_generation, path_busgen, path_bus):
    """
    adapts the generation files

    :param path_generation:
    :param path_busgenerator:
    :param path_bus:
    :return: the net with the generators added
    """

    df_generation = pd.read_csv(path_generation)
    df_busgen = pd.read_csv(path_busgen)
    df_bus = pd.read_csv(path_bus)

    # create basic generator dataframe
    # find the bus index of each gen
    gen_indx = []
    for _, gen in df_busgen.iterrows():
        bus_gen = pp.get_element_index(net, "bus", gen.bus)
        gen_indx.append(bus_gen)

    gen_indx = pd.DataFrame(gen_indx)
    gen_indx = gen_indx.rename(columns={0: "bus"})

    # load name and peak power
    gen_name = df_busgen['bus']
    gen_pmw = df_busgen['p_mw']
    gen_vpu = df_busgen['vm_pu']

    # merge in a full dataframe
    headers = ["name", "bus", "p_mw", "vm_pu"]
    df_gen = pd.concat([gen_name, gen_indx, gen_pmw, gen_vpu], axis=1)
    df_gen.columns.values[0] = "name"

    # create time series from the basic load df
    Nt = len(df_generation)
    Ng = len(df_gen)
    pmw_ts = np.zeros((Nt, Ng), dtype=float)
    for i in range(Nt):  # number of time periods
        pmw_ts[i,:] = df_gen['p_mw'][:] * df_generation['norm'][i]

    # gen structure for 1 t
    for ll in range(len(df_busgen)):
        pp.create_gen(net, bus=gen_indx['bus'][ll], p_mw=pmw_ts[0, ll], vm_pu=gen_vpu[ll], name=gen_name[ll], index=int(ll))

```

```python

    # timeseries
    df_gen_ts = pd.DataFrame(pmw_ts, index=list(range(Nt)), columns=net.gen.index)
    ds_gen_ts = DFData(df_gen_ts)
    const_gen = control.ConstControl(net, element='gen', element_index=net.gen.index, variable='p_mw', data_source=ds_gen_ts, profile_name=net.gen.index)

    return net


def create_intercon(path_bus):
    """
    defines the interconnection (slack bus)

    :param path_bus:
    :return: the net with the interconnection added
    """

    df_bus = pd.read_csv(path_bus)

    # find the slack index
    slack_indx = 0
    for ll in range(len(df_bus)):
        # slack_indx = pp.get_element_index(net, "bus", bb.name)
        if df_bus['name'][ll] == 'intercon':
            slack_indx = pp.get_element_index(net, "bus", df_bus['name'][ll])

    pp.create_ext_grid(net, slack_indx, vm_pu=1.0, va_degree=0)

    return net


def create_trafo(path_trafo):
    """
    defines the transformers

    :param path_trafo:
    :return: the net with the transformers added
    """

    df_trafo = pd.read_csv(path_trafo)

    # for trafo in df_trafo:
    for _, trafo in df_trafo.iterrows():
        hv_bus = pp.get_element_index(net, "bus", trafo.hv_bus)
        lv_bus = pp.get_element_index(net, "bus", trafo.lv_bus)

        pp.create_transformer_from_parameters(net,
                                              hv_bus,
                                              lv_bus,
                                              trafo.sn_mva,
                                              trafo.vn_hv_kv,
                                              trafo.vn_lv_kv,
                                              trafo.vkr_percent,
                                              trafo.vk_percent,
                                              trafo.pfe_kw,
                                              trafo.i0_percent)

    return net






# create empty network
net = pp.create_empty_network()
```

```

    # buses
    net = create_bus(path_bus, path_geodata)

    # lines
    net = create_line(path_line)

    # loads
    net = create_load(path_demand, path_busload, path_bus)

    # gens
    net = create_generator(path_generation, path_busgen, path_bus)

    # interconnection
    net = create_intercon(path_bus)

    # trafos
    net = create_trafo(path_trafo)



    return net


if __name__ == "__main__":
    # load paths
    path_bus = 'Datafiles/bus1.csv'
    path_geodata = 'Datafiles/geodata1.csv'
    path_line = 'Datafiles/line1.csv'
    path_demand = 'Datafiles/demand1.csv'
    path_busload = 'Datafiles/bus_load1.csv'
    path_generation = 'Datafiles/generation1.csv'
    path_busgen = 'Datafiles/bus_gen1.csv'
    path_trafo = 'Datafiles/trafo1.csv'

    # define net
    net = initialize_net(path_bus, path_geodata, path_line, path_demand, path_busload,
     path_generation, path_busgen, path_trafo)

    # run timeseries
    ow = timeseries.OutputWriter(net, output_path="./Results/", output_file_type=".xlsx")
    ow.log_variable('res_bus', 'vm_pu')
    ow.log_variable('res_line', 'loading_percent')
    timeseries.run_timeseries(net)

    # run diagnostic
    # pp.diagnostic(net)
    print(net.bus)

    # plot
    # pp.plotting.simple_plot(net)
    # simple_plot(net)
```

Listing 3.1. Main code in Python with the Pandapower library

```
import numpy as np

def calc_line(a, b, c, d, e, immax, npar, Rca, Dext, kgg):
    """
    calculate r, x, c, and return also Imax

    :param a: horizontal distance between A1 and C2
    :param b: horizontal distance between B1 and B2
    :param c: horizontal distance between C1 and A2
    :param d: vertical distance between A1 and B1
    :param e: vertical distance between B1 and C1
    :param immax: max current in A
```

```
    :param npar: number of parallel lines (1 or 2)
    :param Rca: ac resistance in ohm/km
    :param Dext: external diameter in mm
    :param kg: factor of roughly 0.8
    :return: r, x, c, imax
    """

    def single_line(a, b, immax, Rca, Dext, kgg):
        """
        calculate the R, X, C parameters, also return Imax

        :param a: horizontal distance between A and C
        :param b: vertical distance between A and B
        :param immax: max current in A
        :param Rca: ac resistance in ohm/km
        :param Dext: external diameter in mm
        :param kg: factor of roughly 0.8

        :return: R, X, C, Imax, in the units desired by pandapower
        """

        # cardinal: https://www.elandcables.com/media/38193/acsr-astm-b-aluminium-conductor-
     steel-reinforced.pdf
        # 54 Al + 7 St, Imax = 888.98 A

        w = 2 * np.pi * 50  # rad / s
        Imax = immax * 1e-3  # kA
        # Stot = 547.3 * 1e-6  # m2, the total section
        # R_ac_75 = 0.07316 * 1e-3  # ohm / m
        # kg = 0.809  # from the slides in a 54 + 7

        R_ac_75 = Rca * 1e-3  # ohm / m, should we correct by temperatures?
        Stot = np.pi * Dext ** 2 / 4 * 1e-6  # m2, the total section
        kg = kgg

        r = np.sqrt(Stot / np.pi)  # considering the total section

        dab = np.sqrt((a / 2) ** 2 + b ** 2)
        dbc = np.sqrt((a / 2) ** 2 + b ** 2)
        dca = a

        GMD = (dab * dbc * dca) ** (1 / 3)
        GMR = kg * r
        RMG = r

        L = 4 * np.pi * 1e-7 / (2 * np.pi) * np.log(GMD / GMR)  # H / m

        C = 2 * np.pi * 1e-9 / (36 * np.pi) / np.log(GMD / RMG)  # F / m

        # in the units pandapower wants
        R_km = R_ac_75 * 1e3  # ohm / km
        X_km = L * w * 1e3  # ohm / km
        C_km = C * 1e9 * 1e3  # nF / km

        return R_km, X_km, C_km, Imax


    def double_line(a, b, c, d, e, immax, Rca, Dext, kgg):
        """
        calculate the R, X, C parameters, also return Imax

        :param a: horizontal distance between A1 and C2
        :param b: horizontal distance between B1 and B2
        :param c: horizontal distance between C1 and A2
        :param d: vertical distance between A1 and B1
        :param e: vertical distance between B1 and C1
        :param immax: max current in A
```

```
        :param Rca: ac resistance in ohm/km
        :param Dext: external diameter in mm
        :param kgg: factor of roughly 0.8
        :return: R, X, C, Imax, in the units desired by pandapower
        """

        # cardinal: https://www.elandcables.com/media/38193/acsr-astm-b-aluminium-conductor-
    steel-reinforced.pdf
        # 54 Al + 7 St, Imax = 888.98 A

        w = 2 * np.pi * 50  # rad / s
        Imax = immax * 1e-3 * 2  # kA, for the full line, x2
        # Stot = 547.3 * 1e-6  # m2, the total section
        # R_ac_75 = 0.07316 * 1e-3  # ohm / m
        # kg = 0.809  # from the slides in a 54 + 7

        R_ac_75 = Rca * 1e-3  # ohm / m, should we correct by temperatures?
        Stot = np.pi * Dext ** 2 / 4 * 1e-6  # m2, the total section
        kg = kgg



        r = np.sqrt(Stot / np.pi)  # considering the total section

        da1b1 = np.sqrt((b / 2 - a / 2) ** 2 + d ** 2)
        da1b2 = np.sqrt((a / 2 + b / 2) ** 2 + d ** 2)
        da2b1 = np.sqrt((c / 2 + b / 2) ** 2 + e ** 2)
        da2b2 = np.sqrt((b / 2 - c / 2) ** 2 + e ** 2)

        db1c1 = np.sqrt((b / 2 - c / 2) ** 2 + e ** 2)
        db1c2 = np.sqrt((b / 2 + a / 2) ** 2 + d ** 2)
        db2c1 = np.sqrt((b / 2 + c / 2) ** 2 + e ** 2)
        db2c2 = np.sqrt((b / 2 - a / 2) ** 2 + d ** 2)

        dc1a1 = np.sqrt((a / 2 - c / 2) ** 2 + (d + e) ** 2)
        dc1a2 = c
        dc2a1 = a
        dc2a2 = np.sqrt((a / 2 - c / 2) ** 2 + (d + e) ** 2)

        dab = (da1b1 * da1b2 * da2b1 * da2b2) ** (1 / 4)
        dbc = (db1c1 * db1c2 * db2c1 * db2c2) ** (1 / 4)
        dca = (dc1a1 * dc1a2 * dc2a1 * dc2a2) ** (1 / 4)

        rp = kg * r

        da1a2 = np.sqrt((a / 2 + c / 2) ** 2 + (d + e) ** 2)
        db1b2 = b
        dc1c2 = np.sqrt((c / 2 + a / 2) ** 2 + (d + e) ** 2)

        drap = np.sqrt(rp * da1a2)
        drbp = np.sqrt(rp * db1b2)
        drcp = np.sqrt(rp * dc1c2)

        dra = np.sqrt(r * da1a2)
        drb = np.sqrt(r * db1b2)
        drc = np.sqrt(r * dc1c2)

        GMD = (dab * dbc * dca) ** (1 / 3)
        GMR = (drap * drbp * drcp) ** (1 / 3)
        RMG = (dra * drb * drc) ** (1 / 3)

        L = 4 * np.pi * 1e-7 / (2 * np.pi) * np.log(GMD / GMR)  # H / m

        C = 2 * np.pi * 1e-9 / (36 * np.pi) / np.log(GMD / RMG)  # F / m

        # in the units pandapower wants
        R_km = R_ac_75 / 2 * 1e3  # ohm / km, like 2 resistances in parallel
```

```
        X_km = L * w * 1e3  # ohm / km
        C_km = C * 1e9 * 1e3  # nF / km

        return R_km, X_km, C_km, Imax

    if npar == 1:
        rr, xx, cc, imm = single_line(a, b, immax, Rca, Dext, kgg)
    elif npar == 2:
        rr, xx, cc, imm = double_line(a, b, c, d, e, immax, Rca, Dext, kgg)
    else:
        print('Error: number of parallel lines is not 1 nor 2')

    return rr, xx, cc, imm

# rr, xx, cc, ii = double_line(11, 2, 4, 5, 6, 1000)
# print(rr, xx, cc, ii)
```

Listing 3.2. Code for the calculation of lines

BIBLIOGRAPHY

[1] Red Eléctrica de España, *Sistema de información del operador del sistema (esios). perfiles de demanda y generación.* https://www.esios.ree.es/es?locale=en, Accessed: 2021-10-25.