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

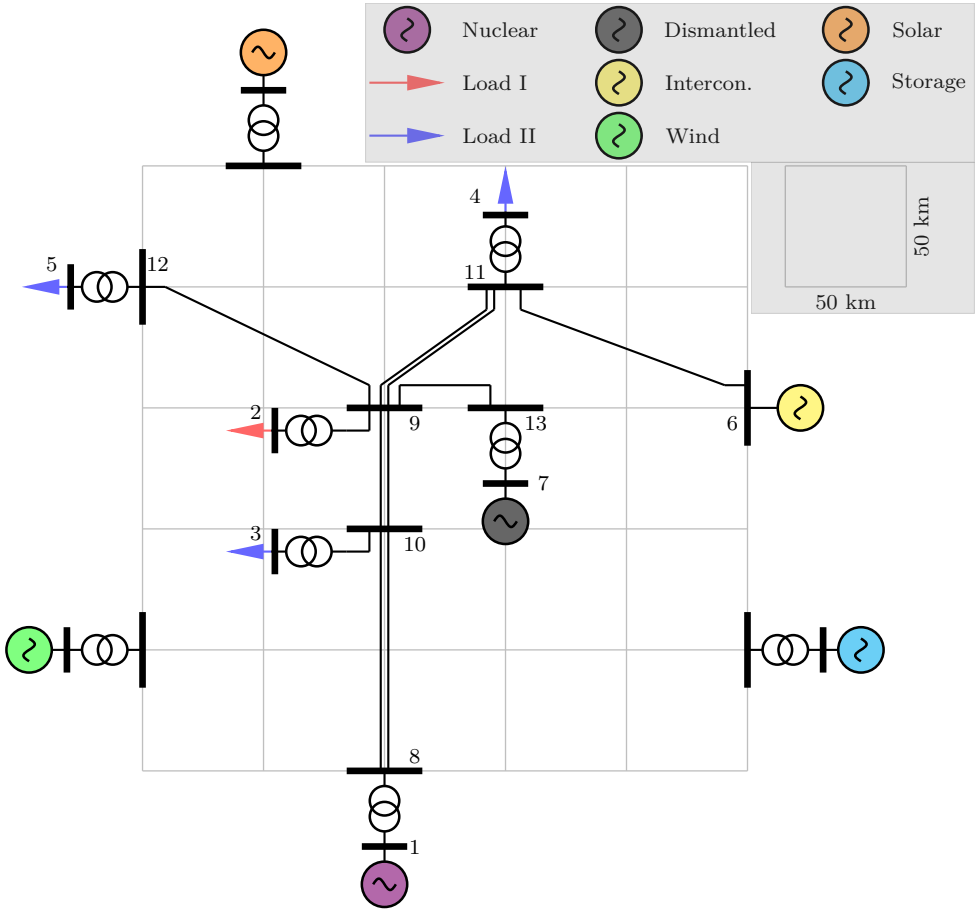

Figure 1. Overview of the network

Bus / Hour	1	2	3	4	5	6	8	9	10	11	12
0	1.050	0.958	0.986	0.967	0.929	1.000	1.030	0.982	0.997	0.979	0.942
1	1.050	0.971	0.995	0.978	0.945	1.000	1.034	0.992	1.006	0.989	0.957
2	1.050	0.978	1.001	0.985	0.955	1.000	1.037	0.998	1.011	0.995	0.966
3	1.050	0.981	1.003	0.987	0.959	1.000	1.038	1.001	1.013	0.997	0.969
4	1.050	0.982	1.003	0.988	0.960	1.000	1.038	1.001	1.013	0.998	0.970
5	1.050	0.978	1.001	0.985	0.955	1.000	1.037	0.998	1.011	0.995	0.966
6	1.050	0.963	0.989	0.971	0.935	1.000	1.032	0.985	1.000	0.983	0.947
7	1.050	0.932	0.965	0.944	0.894	1.000	1.021	0.960	0.979	0.958	0.909
8	1.050	0.907	0.946	0.923	0.862	1.000	1.013	0.940	0.962	0.939	0.880
9	1.050	0.896	0.937	0.913	0.847	1.000	1.008	0.930	0.954	0.930	0.865
10	1.050	0.891	0.934	0.909	0.840	1.000	1.007	0.926	0.950	0.926	0.859
11	1.050	0.892	0.935	0.910	0.842	1.000	1.007	0.927	0.951	0.927	0.861
12	1.050	0.895	0.937	0.913	0.846	1.000	1.008	0.930	0.953	0.929	0.865
13	1.050	0.897	0.939	0.914	0.849	1.000	1.009	0.931	0.955	0.931	0.867
14	1.050	0.908	0.947	0.924	0.863	1.000	1.013	0.940	0.962	0.940	0.880
15	1.050	0.915	0.952	0.929	0.872	1.000	1.015	0.946	0.967	0.945	0.888
16	1.050	0.918	0.955	0.933	0.877	1.000	1.016	0.949	0.969	0.948	0.893
17	1.050	0.920	0.956	0.934	0.878	1.000	1.017	0.950	0.970	0.949	0.894
18	1.050	0.914	0.951	0.929	0.871	1.000	1.015	0.945	0.966	0.944	0.888
19	1.050	0.894	0.936	0.912	0.845	1.000	1.008	0.929	0.953	0.929	0.863
20	1.050	0.881	0.926	0.900	0.826	1.000	1.003	0.918	0.943	0.918	0.846
21	1.050	0.888	0.932	0.906	0.836	1.000	1.006	0.924	0.948	0.924	0.856
22	1.050	0.914	0.952	0.929	0.871	1.000	1.015	0.945	0.966	0.944	0.888
23	1.050	0.940	0.971	0.952	0.905	1.000	1.024	0.967	0.984	0.965	0.920

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

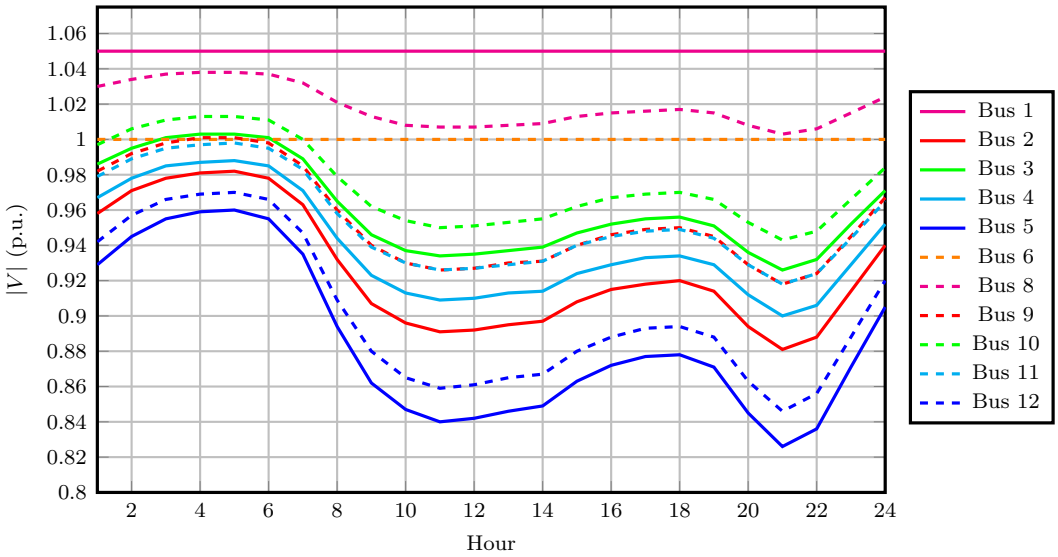

Figure 2. Voltage profile during 24 hours for the initial grid. The low-voltage buses are plotted in solid lines; the high-voltage ones are in dashed lines.

Load / Hour	8-10	10-9	9-11	9-12	11-6
0	26.455	19.983	9.608	31.041	64.359
1	24.390	18.423	9.046	28.520	59.872
2	23.081	17.437	8.701	26.936	57.069
3	22.536	17.027	8.559	26.280	55.912
4	22.445	16.958	8.536	26.171	55.719
5	23.074	17.431	8.699	26.928	57.053
6	25.751	19.450	9.415	30.178	62.821
7	30.722	23.232	10.829	36.361	73.837
8	34.303	25.990	11.904	40.967	82.183
9	35.951	27.271	12.414	43.140	86.055
10	36.582	27.764	12.611	43.983	87.541
11	36.445	27.657	12.568	43.800	87.220
12	36.002	27.311	12.430	43.209	86.176
13	35.732	27.101	12.346	42.850	85.540
14	34.196	25.907	11.872	40.827	81.932
15	33.261	25.184	11.587	39.611	79.743
16	32.725	24.771	11.425	38.918	78.490
17	32.543	24.631	11.370	38.685	78.066
18	33.360	25.261	11.617	39.739	79.975
19	36.132	27.413	12.470	43.382	86.482
20	37.988	28.867	13.054	45.886	90.862
21	36.992	28.085	12.739	44.534	88.508
22	33.316	25.226	11.604	39.681	79.870
23	29.399	22.221	10.443	34.693	70.872

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

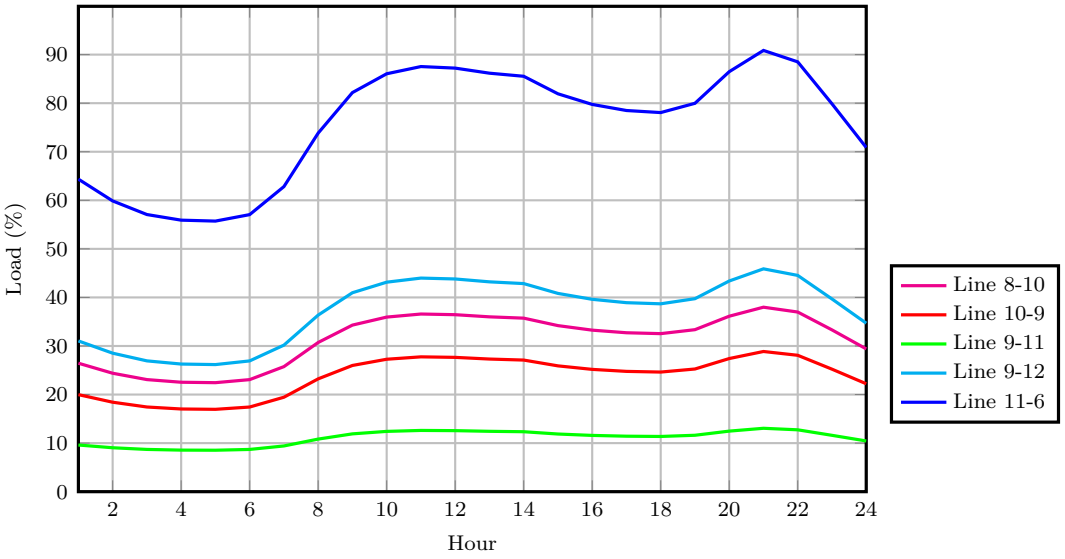

Figure 3. Representation of the percentual loading of the lines during 24 hours

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. This will allow to estimate the penalties due disconnection.

Element	Disconnection time (h)	Consequences
Line 8-10	12.50	?
Line 9-10	6.25	?
Line 9-11	8.85	?
Line 9-12	13.98	?
Line 11-6	13.98	?
Trafo 1-8	1.20	No load served
Trafo 2-9	1.20	Load 2 unserved
Trafo 3-10	1.20	Load 3 unserved
Trafo 4-11	1.20	Load 4 unserved
Trafo 5-12	1.20	Load 5 unserved

Table 4. Disconnection time and consequences of losing each element

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.

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)
```

```
66          for _, line in df_line.iterrows():
67              from_bus = pp.get_element_index(net, "bus", line.from_bus)
68              to_bus = pp.get_element_index(net, "bus", line.to_bus)
69
70              rr, xx, cc, imax = calc_line(line.a,
71                                           line.b,
72                                           line.c,
73                                           line.d,
74                                           line.e,
75                                           line.max_i,
76                                           int(line.parallel))
77
78              pp.create_line_from_parameters(net,
79                                             from_bus,
80                                             to_bus,
81                                             length_km=line.length,
82                                             r_ohm_per_km=rr,
83                                             x_ohm_per_km=xx,
84                                             c_nf_per_km=cc,
85                                             max_i_ka=imax,
86                                             name=line.name_l,
87                                             parallel=line.parallel)
88
89          return net
90
91
92      def create_load(path_demand, path_busload, path_bus):
93          """
94          adapts the load files
95
96          :param path_demand:
97          :param path_busload:
98          :param path_bus:
99          :return: the net with the loads added
100         """
101
102         df_demand = pd.read_csv(path_demand)
103         df_busload = pd.read_csv(path_busload)
104         df_bus = pd.read_csv(path_bus)
105
106         # create basic load dataframe
107         # find the bus index of each load
108         load_indx = []
109         for _, load in df_busload.iterrows():
110             bus_load = pp.get_element_index(net, "bus", load.bus)
111             load_indx.append(bus_load)
112
113         load_indx = pd.DataFrame(load_indx)
114         load_indx = load_indx.rename(columns={0: "bus"})
115
116         # load name and peak power
117         load_name = df_busload['bus']
118         load_pmw = df_busload['p_mw']
119         load_qmvar = df_busload['q_mvar']
120
121         # merge in a full dataframe
122         headers = ["name", "bus", "p_mw", "q_mvar"]
123         df_load = pd.concat([load_name, load_indx, load_pmw, load_qmvar], axis=1)
124         df_load.columns.values[0] = "name"
125
126         # create time series from the basic load df
127         Nt = len(df_demand)
128         Nl = len(df_load)
129         pmw_ts = np.zeros((Nt, Nl), dtype=float)
130         qmvar_ts = np.zeros((Nt, Nl), dtype=float)
131         for i in range(Nt):  # number of time periods
132             pmw_ts[i,:] = df_load['p_mw'][:] * df_demand['norm'][i]
```

```
133             qmvar_ts[i,:] = df_load['q_mvar'][:] * df_demand['norm'][i]
134
135         # form loads as a static picture (initial time)
136         for ll in range(len(df_busload)):
137             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))
138
139         # timeseries
140         df_pload_ts = pd.DataFrame(pmw_ts, index=list(range(Nt)), columns=net.load.index)
141         df_qload_ts = pd.DataFrame(qmvar_ts, index=list(range(Nt)), columns=net.load.index)
142         ds_pload_ts = DFData(df_pload_ts)
143         ds_qload_ts = DFData(df_qload_ts)
144         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)
145         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?
146
147         return net
148
149
150     def create_generator(path_generation, path_busgen, path_bus):
151         """
152         adapts the generation files
153
154         :param path_generation:
155         :param path_busgenerator:
156         :param path_bus:
157         :return: the net with the generators added
158         """
159
160         df_generation = pd.read_csv(path_generation)
161         df_busgen = pd.read_csv(path_busgen)
162         df_bus = pd.read_csv(path_bus)
163
164         # create basic generator dataframe
165         # find the bus index of each gen
166         gen_indx = []
167         for _, gen in df_busgen.iterrows():
168             bus_gen = pp.get_element_index(net, "bus", gen.bus)
169             gen_indx.append(bus_gen)
170
171         gen_indx = pd.DataFrame(gen_indx)
172         gen_indx = gen_indx.rename(columns={0: "bus"})
173
174         # load name and peak power
175         gen_name = df_busgen['bus']
176         gen_pmw = df_busgen['p_mw']
177         gen_vpu = df_busgen['vm_pu']
178
179         # merge in a full dataframe
180         headers = ["name", "bus", "p_mw", "vm_pu"]
181         df_gen = pd.concat([gen_name, gen_indx, gen_pmw, gen_vpu], axis=1)
182         df_gen.columns.values[0] = "name"
183
184         # create time series from the basic load df
185         Nt = len(df_generation)
186         Ng = len(df_gen)
187         pmw_ts = np.zeros((Nt, Ng), dtype=float)
188         for i in range(Nt):  # number of time periods
189             pmw_ts[i,:] = df_gen['p_mw'][:] * df_generation['norm'][i]
190
191         # gen structure for 1 t
192         for ll in range(len(df_busgen)):
193             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))
194
```

```
195
196         # timeseries
197         df_gen_ts = pd.DataFrame(pmw_ts, index=list(range(Nt)), columns=net.gen.index)
198         ds_gen_ts = DFData(df_gen_ts)
199         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)
200
201         return net
202
203
204     def create_intercon(path_bus):
205         """
206         defines the interconnection (slack bus)
207
208         :param path_bus:
209         :return: the net with the interconnection added
210         """
211
212         df_bus = pd.read_csv(path_bus)
213
214         # find the slack index
215         slack_indx = 0
216         for ll in range(len(df_bus)):
217             # slack_indx = pp.get_element_index(net, "bus", bb.name)
218             if df_bus['name'][ll] == 'intercon':
219                 slack_indx = pp.get_element_index(net, "bus", df_bus['name'][ll])
220
221         pp.create_ext_grid(net, slack_indx, vm_pu=1.0, va_degree=0)
222
223         return net
224
225
226     def create_trafo(path_trafo):
227         """
228         defines the transformers
229
230         :param path_trafo:
231         :return: the net with the transformers added
232         """
233
234         df_trafo = pd.read_csv(path_trafo)
235
236         # for trafo in df_trafo:
237         for _, trafo in df_trafo.iterrows():
238             hv_bus = pp.get_element_index(net, "bus", trafo.hv_bus)
239             lv_bus = pp.get_element_index(net, "bus", trafo.lv_bus)
240
241             pp.create_transformer_from_parameters(net,
242                                                   hv_bus,
243                                                   lv_bus,
244                                                   trafo.sn_mva,
245                                                   trafo.vn_hv_kv,
246                                                   trafo.vn_lv_kv,
247                                                   trafo.vkr_percent,
248                                                   trafo.vk_percent,
249                                                   trafo.pfe_kw,
250                                                   trafo.i0_percent)
251
252         return net
253
254
255
256
257
258
259     # create empty network
260     net = pp.create_empty_network()
```

```
261
262     # buses
263     net = create_bus(path_bus, path_geodata)
264
265     # lines
266     net = create_line(path_line)
267
268     # loads
269     net = create_load(path_demand, path_busload, path_bus)
270
271     # gens
272     net = create_generator(path_generation, path_busgen, path_bus)
273
274     # interconnection
275     net = create_intercon(path_bus)
276
277     # trafos
278     net = create_trafo(path_trafo)
279
280
281
282     return net
283
284
285 if __name__ == "__main__":
286     # load paths
287     path_bus = 'Datafiles/bus1.csv'
288     path_geodata = 'Datafiles/geodata1.csv'
289     path_line = 'Datafiles/line1.csv'
290     path_demand = 'Datafiles/demand1.csv'
291     path_busload = 'Datafiles/bus_load1.csv'
292     path_generation = 'Datafiles/generation1.csv'
293     path_busgen = 'Datafiles/bus_gen1.csv'
294     path_trafo = 'Datafiles/trafo1.csv'
295
296     # define net
297     net = initialize_net(path_bus, path_geodata, path_line, path_demand, path_busload,
        path_generation, path_busgen, path_trafo)
298
299     # run timeseries
300     ow = timeseries.OutputWriter(net, output_path="./Results/", output_file_type=".xlsx")
301     ow.log_variable('res_bus', 'vm_pu')
302     ow.log_variable('res_line', 'loading_percent')
303     timeseries.run_timeseries(net)
304
305     # run diagnostic
306     # pp.diagnostic(net)
307     print(net.bus)
308
309     # plot
310     # pp.plotting.simple_plot(net)
311     # simple_plot(net)
```

Listing 3.1. Main code in Python with the Pandapower library