

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
2	Phase 1	3
2.1	Operating costs	6
2.2	Problem identification	7
2.3	Solution suggestion	8
3	Phase 2	9
3.1	Addition of lines	9
3.1.1	Formulation	9
3.1.2	Results	11
3.2	Rising the voltage level	11
3.2.1	Description	11
3.2.2	Results	11
4	Code	12
	Bibliography	20

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 feeds dispersed demand points that symbolize 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.

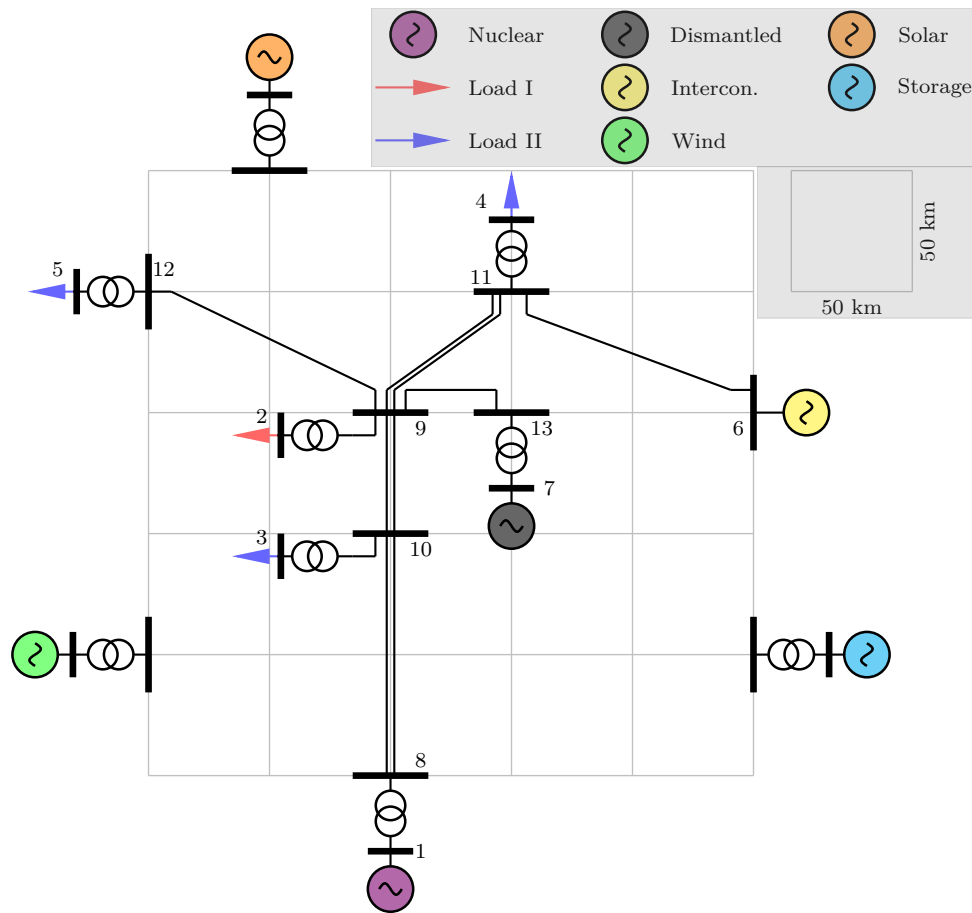


Figure 1. Overview of the network

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.

Bus Hour	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

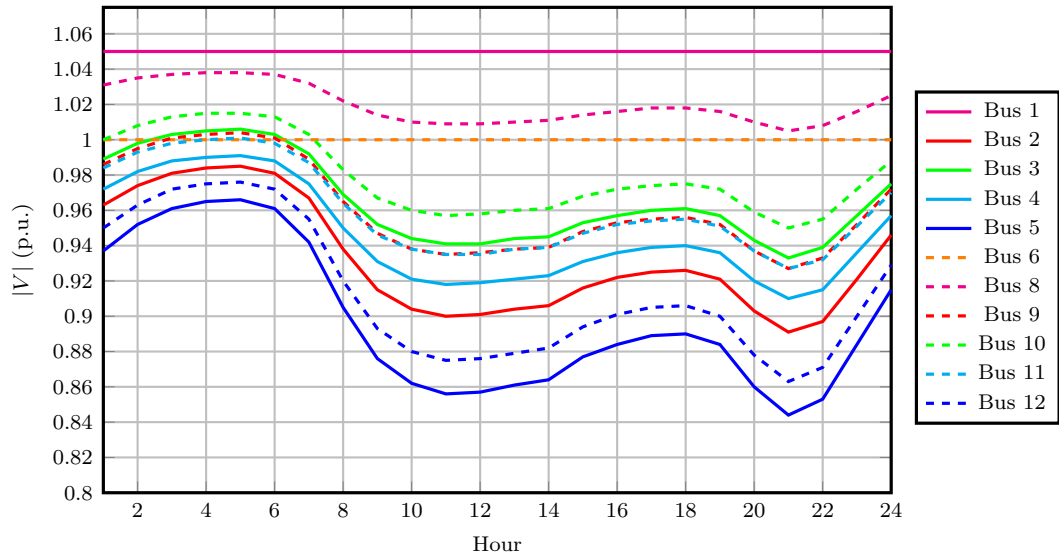


Figure 2. Voltage profile during 24 hours for the initial grid. The low-voltage buses are plotted in solid lines; the high-voltage ones are in dashed lines.

Load Hour	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

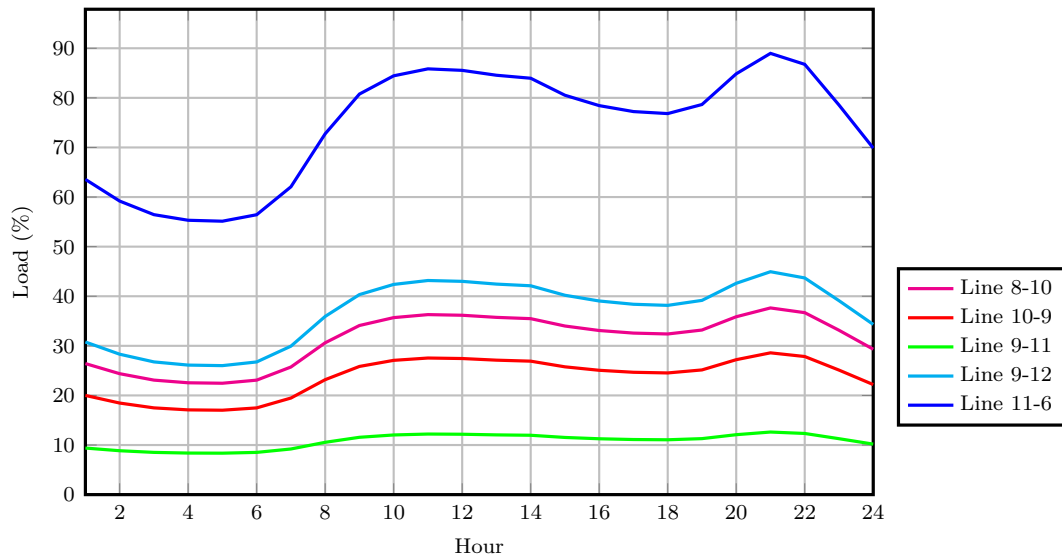


Figure 3. Representation of the percentual loading of the lines during 24 hours

2.1. Operating costs

Regarding the operating costs, some estimations are made in order to assess 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:

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

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:

$$C_{discon} \approx \sum_{i=1}^{10} \bar{P}_{uns,i} t_{discon,i} C_p, \quad (2)$$

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), $\bar{P}_{uns,i}$ 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 of 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. PHASE 2

This chapter considers upgrading the system by means of installing additional lines. In the previous analysis, it was visible that the disconnection of specific lines resulted in an unsolvable problem. Often, no solution was obtained as a result of an exceedingly high demand compared to the available generation. The criteria $N - 1$ was therefore not met, yet it is usually treated as a requirement by control center operators [2], [3].

The central aim of this chapter is reaching a robust grid, where a single failure does not compromise the full system. Several solutions were proposed, such as meshing the network, rising the voltage level, etc. This chapter is devoted to exploring the implications of such proposals. First, a more theoretical explanation describing the formulation is presented, and then, the analysis of results takes place. These results include electrical magnitudes as well as economic data. An optimal configuration is finally provided.

3.1. Addition of lines

Recall that the original system depicted in Figure 1 was for the most part a radial system, in the sense that there was no redundancy in the connection of lines. The operating costs are non-negligible since a significant amount of energy cannot be delivered during faults. To combat this issue, the installation of various new lines (in red) is contemplated, as shown in Figure 4.

By installing new lines, an additional installation cost has to be considered, but a reduction on the operating costs is expected. It is worth mentioning that not all contemplated new lines ought to be installed. Rather, by adding one or two of them, the system could already be operating under the desired conditions. There is a cost-benefit tradeoff, so only a few specific new lines will be selected.

3.1.1. Formulation

The formulation of the topology deserves special attention. Mathematically, it is defined as:

$$\{\mathcal{N} \in \mathcal{P}([n]) \mid |\mathcal{N}| = k\}, \quad (3)$$

where \mathcal{N} denotes a subset, that is, possible topology out of the full set of configurations $\mathcal{P}([n])$, $[n] = \{\mathbf{1}_{\sigma_1}, \mathbf{1}_{\sigma_2}, \dots, \mathbf{1}_{\sigma_n}\}$ represents the set of elements that cause variations on the topology, and k symbolizes the total amount of possibilities. In the contingency analysis, the set $[n]$ is a function of the state of the elements (mainly, lines). These can be in service or out of service, so can be regarded as boolean variables. The employed notation, of the form $\mathbf{1}_{\sigma_i}$, is 1 if the indicator variable σ_i holds true; otherwise, it is 0. The indicator variable, in the case under study, is simply the state of each particular line.

Note that the whole set \mathcal{P} can be further divided into the set composed of the original lines \mathcal{A} , and the set of new lines \mathcal{B} . Let j indicate the number of lines in the original set \mathcal{A} , and $n - j$ the amount of lines in the new set \mathcal{B} . In the particularized case shown in Figure 4, $j = 6$ and $n - j = 8$.

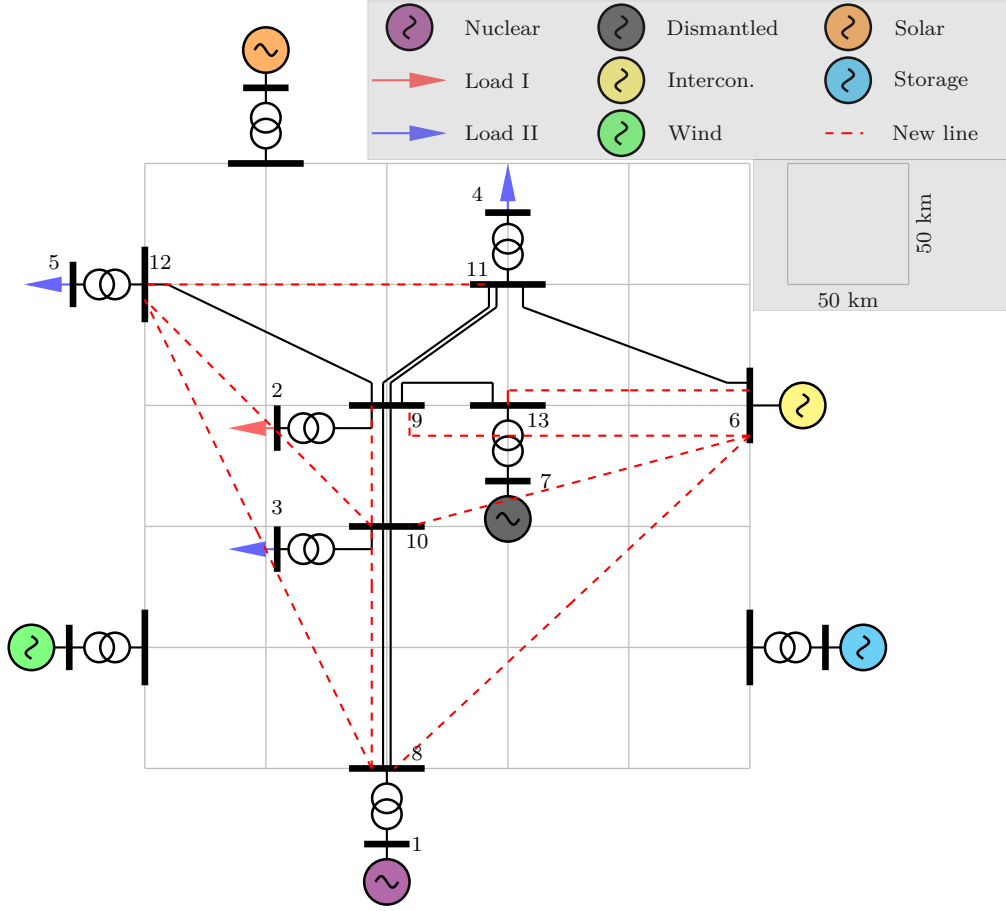


Figure 4. Overview of the network with the addition of lines

Thus, these subsets are expanded and explicitly become:

$$\begin{cases} \mathcal{A} = \{\mathbf{1}_{\sigma_1}, \mathbf{1}_{\sigma_2}, \dots, \mathbf{1}_{\sigma_j}\}, \\ \mathcal{B} = \{\mathbf{1}_{\sigma_{j+1}}, \mathbf{1}_{\sigma_{j+2}}, \dots, \mathbf{1}_{\sigma_n}\}. \end{cases} \quad (4)$$

To perform the contingency analysis, we consider that one of the original lines is faulted. For instance, if line i ends up disconnected, $\sigma_i \leftarrow \text{false}$ while the rest of original lines $\sigma_r \leftarrow \text{true}$ for $r = \{1, 2, \dots, i-1, i+1, \dots, j\}$.

On the other hand, in the subset \mathcal{B} it is not straightforward to estimate which lines should be connected or disconnected. Perhaps installing a single additional line is enough to meet the $N-1$ criteria, or maybe more lines are required. Hence, it has been decided that all permutations have to be analyzed. Since σ are boolean variables, the total number k of contingencies to simulate are:

$$k = j2^{(n-j)}, \quad (5)$$

so if $j = 6$ and $n - j = 8$, this results in $k = 1536$ different topologies. Thus, a total of 1536 timeseries power flows are computed.

Security analysis in power systems has been traditionally based on the usage of the DC power flow [4], [5]. It offers the advantage of being non-iterative (and consequently, fast), and allows to determine the influence of faults with the usage of transmission factors. However, its solution is

just an approximation. Since the amount of situations remains reasonable, it has been decided to solve them with the typical Newton-Raphson up to the default precision of $1 \cdot 10^{-8}$ MVA [6].

The general procedure to perform the contingency analysis is shown in Algorithm 1.

Algorithm 1: Pseudocode to solve the contingencies

Input: net initialized class, j , n

Output: stored results

```

1 Generate permutations  $\forall \sigma_g$  where  $g = [1, 2, \dots, 2^{(n-j)}]$ 
2 for  $i = [1, 2, \dots, j]$  do
3    $\sigma_i \leftarrow \text{false}$ 
4    $\sigma_r \leftarrow \text{true}$ , where  $r \neq i$  and  $r \leq j$ 
5    $\mathcal{A} \leftarrow \{\mathbf{1}_{\sigma_1}, \mathbf{1}_{\sigma_2}, \dots, \mathbf{1}_{\sigma_j}\}$ 
6   for  $g = [1, 2, \dots, 2^{(n-j)}]$  do
7      $[\sigma_{j+1}, \sigma_{j+2}, \dots, \sigma_n] \leftarrow \sigma_g$ 
8      $\mathcal{B} \leftarrow \{\mathbf{1}_{\sigma_{j+1}}, \mathbf{1}_{\sigma_{j+2}}, \dots, \mathbf{1}_{\sigma_n}\}$ 
9      $\mathcal{N} \leftarrow \mathcal{A} \cup \mathcal{B}$ 
10    pandapower.timeseries.run_timeseries( $\mathcal{N}$ , net)
11    Store results

```

3.1.2. Results

3.2. Rising the voltage level

3.2.1. Description

3.2.2. Results

4. CODE

```

1 import pandapower as pp
2 import pandas as pd
3 import numpy as np
4 import pandapower.control as control
5 import pandapower.networks as nw
6 import pandapower.timeseries as timeseries
7 from pandapower.timeseries.data_sources.frame_data import DFData
8 from pandapower.plotting import simple_plot
9
10 from line_param_calc import calc_line
11
12
13
14 pd.set_option('display.max_rows', 500)
15 pd.set_option('display.max_columns', 500)
16 pd.set_option('display.width', 1000)
17
18 def initialize_net(path_bus, path_geodata, path_line, path_demand, path_busload, path_generation
    , path_busgen, path_trafo):
19     """
20     initialize the grid from the .csv files
21
22     :param path_bus: path to the bus .csv file
23     :param geodata: path to the geodata .csv file
24     :param path_line: path to the line .csv file
25     :param path_demand: path to the normalized demand .csv file
26     :param busload: path to the bus-load look up table .csv file
27     :param path_generation: path to the normalized generation .csv file
28     :param busgen: path to the bus-generator look up table .csv file
29     :param trafo: path to the trafo .csv file
30     :return: the net class
31     """
32
33     def create_bus(path_bus, path_geodata):
34         """
35         adapts the data from the bus file (if needed)
36
37         :param path_bus:
38         :param path_geodata:
39         :return: the net with the buses added
40         """
41
42         df_bus = pd.read_csv(path_bus)
43         df_geodata = pd.read_csv(path_geodata)
44
45         net.bus = df_bus
46
47         # adapt geodata
48         for ll in range(len(df_geodata)):
49             indx_bus = pp.get_element_index(net, "bus", df_geodata['name'][ll])
50             df_geodata['name'][ll] = indx_bus
51
52         net.bus_geodata = df_geodata
53
54         return net
55
56
57     def create_line(path_line):
58         """
59         adapts the data from the line file
60
61         :param path_line:
62         :return: the net with the lines added
63         """
64
65         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'][:,i] * 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,
138         ll], name=load_name[ll], index=int(ll))
139
140     # timeseries
141     df_pload_ts = pd.DataFrame(pmw_ts, index=list(range(Nt)), columns=net.load.index)
142     df_qload_ts = pd.DataFrame(qmvar_ts, index=list(range(Nt)), columns=net.load.index)
143     ds_pload_ts = DFData(df_pload_ts)
144     ds_qload_ts = DFData(df_qload_ts)
145     const_load = control.ConstControl(net, element='load', element_index=net.load.index,
146     variable='p_mw', data_source=ds_pload_ts, profile_name=net.load.index)
147     const_load = control.ConstControl(net, element='load', element_index=net.load.index,
148     variable='q_mvar', data_source=ds_qload_ts, profile_name=net.load.index) # add the
149     reactive like this?
150
151     return net
152
153 def create_generator(path_generation, path_busgen, path_bus):
154     """
155     adapts the generation files
156
157     :param path_generation:
158     :param path_busgenerator:
159     :param path_bus:
160     :return: the net with the generators added
161     """
162
163     df_generation = pd.read_csv(path_generation)
164     df_busgen = pd.read_csv(path_busgen)
165     df_bus = pd.read_csv(path_bus)
166
167     # create basic generator dataframe
168     # find the bus index of each gen
169     gen_indx = []
170     for _, gen in df_busgen.iterrows():
171         bus_gen = pp.get_element_index(net, "bus", gen.bus)
172         gen_indx.append(bus_gen)
173
174     gen_indx = pd.DataFrame(gen_indx)
175     gen_indx = gen_indx.rename(columns={0: "bus"})
176
177     # load name and peak power
178     gen_name = df_busgen['bus']
179     gen_pmw = df_busgen['p_mw']
180     gen_vpu = df_busgen['vm_pu']
181
182     # merge in a full dataframe
183     headers = ["name", "bus", "p_mw", "vm_pu"]
184     df_gen = pd.concat([gen_name, gen_indx, gen_pmw, gen_vpu], axis=1)
185     df_gen.columns.values[0] = "name"
186
187     # create time series from the basic load df
188     Nt = len(df_generation)
189     Ng = len(df_gen)
190     pmw_ts = np.zeros((Nt, Ng), dtype=float)
191     for i in range(Nt): # number of time periods
192         pmw_ts[i,:] = df_gen['p_mw'][:] * df_generation['norm'][i]
193
194     # gen structure for 1 t
195     for ll in range(len(df_busgen)):
196         pp.create_gen(net, bus=gen_indx['bus'][ll], p_mw=pmw_ts[0, ll], vm_pu=gen_vpu[ll],
197         name=gen_name[ll], index=int(ll))

```

```

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,
200     variable='p_mw', data_source=ds_gen_ts, profile_name=net.gen.index)
201
202     return net
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,
298                          path_generation, path_busgen, path_trafo)
299
300     # run timeseries
301     ow = timeseries.OutputWriter(net, output_path="./Results/", output_file_type=".xlsx")
302     ow.log_variable('res_bus', 'vm_pu')
303     ow.log_variable('res_line', 'loading_percent')
304     timeseries.run_timeseries(net)
305
306     # run diagnostic
307     # pp.diagnostic(net)
308     print(net.bus)
309
310     # plot
311     # pp.plotting.simple_plot(net)
312     # simple_plot(net)

```

Listing 4.1. Main code in Python with the Pandapower library

```

1 import numpy as np
2
3 def calc_line(a, b, c, d, e, immax, npar, RCA, Dext, kgg):
4     """
5     calculate r, x, c, and return also Imax
6
7     :param a: horizontal distance between A1 and C2
8     :param b: horizontal distance between B1 and B2
9     :param c: horizontal distance between C1 and A2
10    :param d: vertical distance between A1 and B1
11    :param e: vertical distance between B1 and C1
12    :param immax: max current in A

```

```

13 :param npar: number of parallel lines (1 or 2)
14 :param Rca: ac resistance in ohm/km
15 :param Dext: external diameter in mm
16 :param kg: factor of roughly 0.8
17 :return: r, x, c, imax
18 """
19
20 def single_line(a, b, immax, Rca, Dext, kgg):
21     """
22     calculate the R, X, C parameters, also return Imax
23
24     :param a: horizontal distance between A and C
25     :param b: vertical distance between A and B
26     :param immax: max current in A
27     :param Rca: ac resistance in ohm/km
28     :param Dext: external diameter in mm
29     :param kg: factor of roughly 0.8
30
31     :return: R, X, C, Imax, in the units desired by pandapower
32     """
33
34     # cardinal: https://www.elandcables.com/media/38193/acsr-astm-b-aluminium-conductor-steel-reinforced.pdf
35     # 54 Al + 7 St, Imax = 888.98 A
36
37     w = 2 * np.pi * 50 # rad / s
38     Imax = immax * 1e-3 # kA
39     # Stot = 547.3 * 1e-6 # m2, the total section
40     # R_ac_75 = 0.07316 * 1e-3 # ohm / m
41     # kg = 0.809 # from the slides in a 54 + 7
42
43     R_ac_75 = Rca * 1e-3 # ohm / m, should we correct by temperatures?
44     Stot = np.pi * Dext ** 2 / 4 * 1e-6 # m2, the total section
45     kg = kgg
46
47     r = np.sqrt(Stot / np.pi) # considering the total section
48
49     dab = np.sqrt((a / 2) ** 2 + b ** 2)
50     dbc = np.sqrt((a / 2) ** 2 + b ** 2)
51     dca = a
52
53     GMD = (dab * dbc * dca) ** (1 / 3)
54     GMR = kg * r
55     RMG = r
56
57     L = 4 * np.pi * 1e-7 / (2 * np.pi) * np.log(GMD / GMR) # H / m
58
59     C = 2 * np.pi * 1e-9 / (36 * np.pi) / np.log(GMD / RMG) # F / m
60
61     # in the units pandapower wants
62     R_km = R_ac_75 * 1e3 # ohm / km
63     X_km = L * w * 1e3 # ohm / km
64     C_km = C * 1e9 * 1e3 # nF / km
65
66     return R_km, X_km, C_km, Imax
67
68
69 def double_line(a, b, c, d, e, immax, Rca, Dext, kgg):
70     """
71     calculate the R, X, C parameters, also return Imax
72
73     :param a: horizontal distance between A1 and C2
74     :param b: horizontal distance between B1 and B2
75     :param c: horizontal distance between C1 and A2
76     :param d: vertical distance between A1 and B1
77     :param e: vertical distance between B1 and C1
78     :param immax: max current in A

```

```

79     :param Rca: ac resistance in ohm/km
80     :param Dext: external diameter in mm
81     :param kgg: factor of roughly 0.8
82     :return: R, X, C, Imax, in the units desired by pandapower
83     """
84
85     # cardinal: https://www.elandcables.com/media/38193/acsr-astm-b-aluminium-conductor-steel-reinforced.pdf
86     # 54 Al + 7 St, Imax = 888.98 A
87
88     w = 2 * np.pi * 50 # rad / s
89     Imax = immax * 1e-3 * 2 # kA, for the full line, x2
90     # Stot = 547.3 * 1e-6 # m2, the total section
91     # Rac_75 = 0.07316 * 1e-3 # ohm / m
92     # kg = 0.809 # from the slides in a 54 + 7
93
94     Rac_75 = Rca * 1e-3 # ohm / m, should we correct by temperatures?
95     Stot = np.pi * Dext ** 2 / 4 * 1e-6 # m2, the total section
96     kg = kgg
97
98
99
100    r = np.sqrt(Stot / np.pi) # considering the total section
101
102    da1b1 = np.sqrt((b / 2 - a / 2) ** 2 + d ** 2)
103    da1b2 = np.sqrt((a / 2 + b / 2) ** 2 + d ** 2)
104    da2b1 = np.sqrt((c / 2 + b / 2) ** 2 + e ** 2)
105    da2b2 = np.sqrt((b / 2 - c / 2) ** 2 + e ** 2)
106
107    db1c1 = np.sqrt((b / 2 - c / 2) ** 2 + e ** 2)
108    db1c2 = np.sqrt((b / 2 + a / 2) ** 2 + d ** 2)
109    db2c1 = np.sqrt((b / 2 + c / 2) ** 2 + e ** 2)
110    db2c2 = np.sqrt((b / 2 - a / 2) ** 2 + d ** 2)
111
112    dc1a1 = np.sqrt((a / 2 - c / 2) ** 2 + (d + e) ** 2)
113    dc1a2 = c
114    dc2a1 = a
115    dc2a2 = np.sqrt((a / 2 - c / 2) ** 2 + (d + e) ** 2)
116
117    dab = (da1b1 * da1b2 * da2b1 * da2b2) ** (1 / 4)
118    dbc = (db1c1 * db1c2 * db2c1 * db2c2) ** (1 / 4)
119    dca = (dc1a1 * dc1a2 * dc2a1 * dc2a2) ** (1 / 4)
120
121    rp = kg * r
122
123    da1a2 = np.sqrt((a / 2 + c / 2) ** 2 + (d + e) ** 2)
124    db1b2 = b
125    dc1c2 = np.sqrt((c / 2 + a / 2) ** 2 + (d + e) ** 2)
126
127    drap = np.sqrt(rp * da1a2)
128    drbp = np.sqrt(rp * db1b2)
129    drcp = np.sqrt(rp * dc1c2)
130
131    dra = np.sqrt(r * da1a2)
132    drb = np.sqrt(r * db1b2)
133    drc = np.sqrt(r * dc1c2)
134
135    GMD = (dab * dbc * dca) ** (1 / 3)
136    GMR = (drap * drbp * drcp) ** (1 / 3)
137    RMG = (dra * drb * drc) ** (1 / 3)
138
139    L = 4 * np.pi * 1e-7 / (2 * np.pi) * np.log(GMD / GMR) # H / m
140
141    C = 2 * np.pi * 1e-9 / (36 * np.pi) / np.log(GMD / RMG) # F / m
142
143    # in the units pandapower wants
144    Rkm = Rac_75 / 2 * 1e3 # ohm / km, like 2 resistances in parallel

```

```
145     X_km = L * w * 1e3 # ohm / km
146     C_km = C * 1e9 * 1e3 # nF / km
147
148     return R_km, X_km, C_km, Imax
149
150     if npar == 1:
151         rr, xx, cc, imm = single_line(a, b, immax, Rca, Dext, kgg)
152     elif npar == 2:
153         rr, xx, cc, imm = double_line(a, b, c, d, e, immax, Rca, Dext, kgg)
154     else:
155         print('Error: number of parallel lines is not 1 nor 2')
156
157     return rr, xx, cc, imm
158
159 # rr, xx, cc, ii = double_line(11, 2, 4, 5, 6, 1000)
160 # print(rr, xx, cc, ii)
```

Listing 4.2. 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.
- [2] Q. Wang and J. D. McCalley, “Risk and “n-1” criteria coordination for real-time operations,” *IEEE Transactions on Power Systems*, vol. 28, no. 3, pp. 3505–3506, 2013.
- [3] A. Gourtani, H. Xu, D. Pozo, and T.-D. Nguyen, “Robust unit commitment with n-1 security criteria,” *Mathematical Methods of Operations Research*, vol. 83, no. 3, pp. 373–408, 2016.
- [4] B. Stott, J. Jardim, and O. Alsac, “Dc power flow revisited,” *IEEE Transactions on Power Systems*, vol. 24, no. 3, pp. 1290–1300, 2009.
- [5] F. Capitanescu, J. M. Ramos, P. Panciatici, D. Kirschen, A. M. Marcolini, L. Platbrood, and L. Wehenkel, “State-of-the-art, challenges, and future trends in security constrained optimal power flow,” *Electric Power Systems Research*, vol. 81, no. 8, pp. 1731–1741, 2011.
- [6] L. Thurner, A. Scheidler, F. Schafer, J. H. Menke, J. Dollichon, F. Meier, S. Meinecke, and M. Braun, “Pandapower - an open source python tool for convenient modeling, analysis and optimization of electric power systems,” *IEEE Transactions on Power Systems*, 2018, ISSN: 0885-8950. DOI: 10.1109/TPWRS.2018.2829021. [Online]. Available: <https://arxiv.org/abs/1709.06743>.