9bus-example-analyses

April 22, 2024

Analyses of simulation

This notebook provides an analyses of the results. These results can be demonstrated in a UI to provide insight on why N-1 security-constrained optimization is useful.

The only input to this notebook is a set of csv files. These csv files are the exact inputs to the UI application as well.

In this notebook, we are mostly interested in timeseries plots

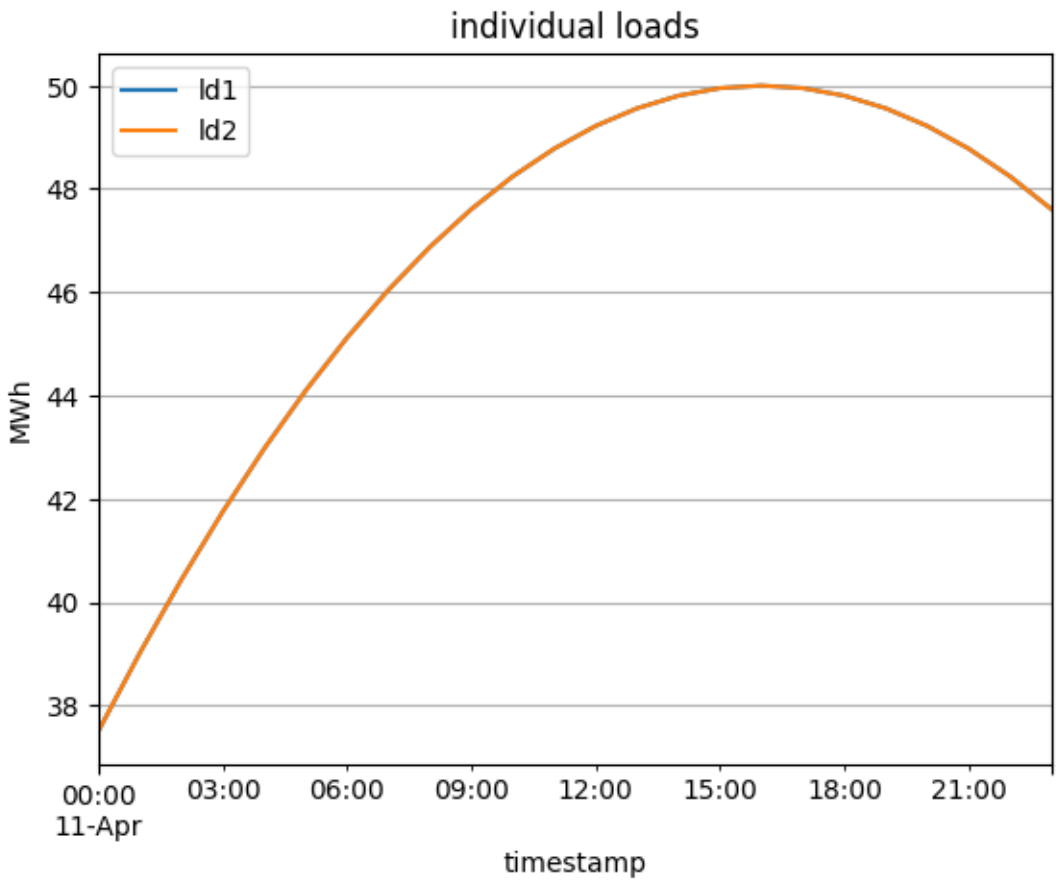
[ ]:
```
from typing import NamedTuple
import pandas as pd

wo_sc_path = "./opt_wo_security"
w_sc_path = "./opt_w_security"

def _make_snapshot_timestamp(dfm: pd.DataFrame) -> pd.DataFrame:

    dfm.snapshot = pd.to_datetime(dfm.snapshot)
    return dfm

class Result(NamedTuple):
    dfm_load: pd.DataFrame
    dfm_gen: pd.DataFrame
    dfm_line: pd.DataFrame
    dfm_post_ctg: pd.DataFrame


    @classmethod
    def read_from_path(cls, path: str) -> "Result":
        dfm_load = pd.read_csv(f"{path}/loads_timeseries.csv",␣
 ↪parse_dates=["snapshot"])
        dfm_gen = pd.read_csv(f"{path}/generation_production_timeseries.csv",␣
 ↪parse_dates=["snapshot"])
        dfm_lines = pd.read_csv(f"{path}/lines_from_timeseries.csv",␣
 ↪parse_dates=["snapshot"])
        dfm_post_ctg = pd.read_csv(f"{path}/post-ctg-flow.csv",␣
 ↪parse_dates=["snapshot"])
```

```
        result = cls(dfm_load=dfm_load, dfm_gen=dfm_gen, dfm_line=dfm_lines,␣
↪dfm_post_ctg=dfm_post_ctg)
        return result
```

[ ]:
```
result_wo_sc = Result.read_from_path(path=wo_sc_path)
result_w_sc = Result.read_from_path(path=w_sc_path)
```

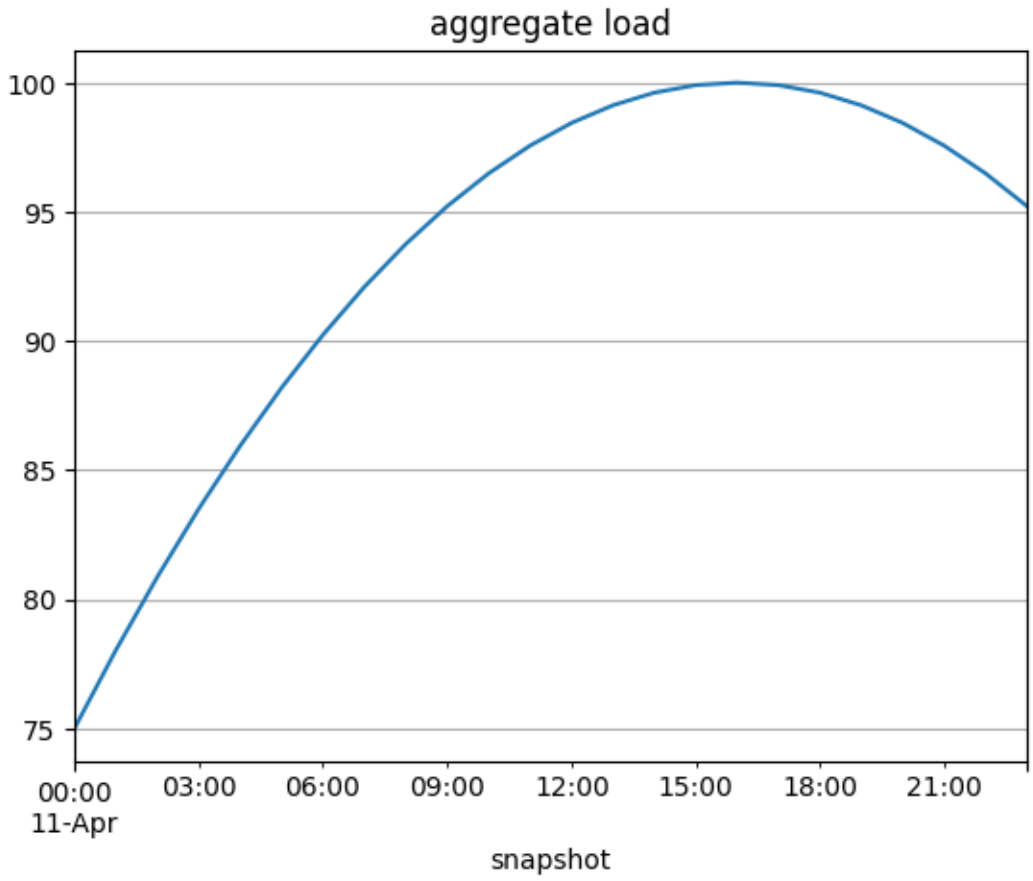
[ ]:
```
# let's plot what the total load looks like for the network
# In both result_wo_sc and result_w_sc the load profiles are the same
dfm_load = result_wo_sc.dfm_load
ax = dfm_load.set_index("snapshot").plot(grid=True, title="individual loads")
ax.set_xlabel("timestamp")
ax.set_ylabel("MWh")
```

[ ]:
```
Text(0, 0.5, 'MWh')
```



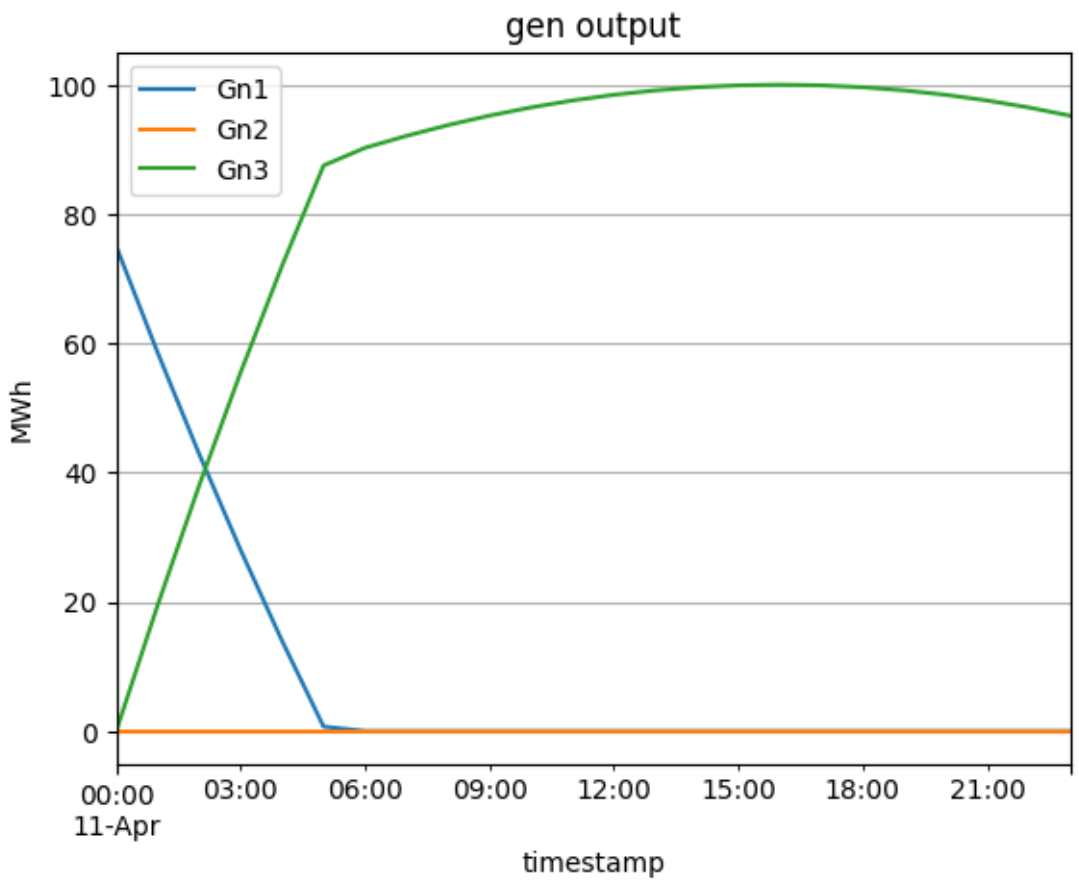
[ ]:
```
# here's aggregate load (sum of both loads at two load buses 5 and 6)
dfm_load.set_index("snapshot").sum(axis=1).plot(grid=True, title="aggregate␣
↪load")
```

[ ]:
```
<Axes: title={'center': 'aggregate load'}, xlabel='snapshot'>
```



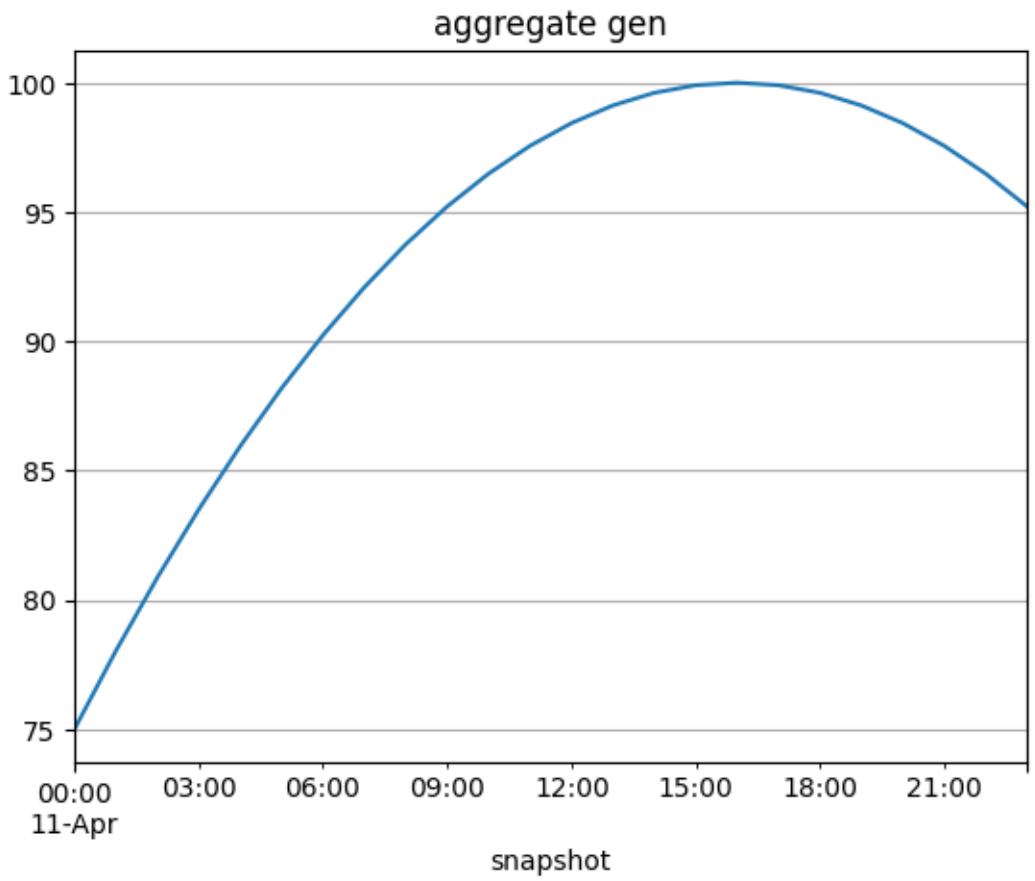
[ ]:
```
# Now let's plot how generators meet the load without n-1 security
# In result_wo_sc, wind (Gn3) meets the load during peak hours
# (it makes sense because it's much cheaper than oil)
dfm_gen = result_wo_sc.dfm_gen
ax = dfm_gen.set_index("snapshot").plot(grid=True, title="gen output")
ax.set_xlabel("timestamp")
ax.set_ylabel("MWh")
```

[ ]:
```
Text(0, 0.5, 'MWh')
```



[ ]:
```
# notic that aggregate generator is the same as the aggregate load
dfm_gen.set_index("snapshot").sum(axis=1).plot(grid=True, title="aggregate gen")
```

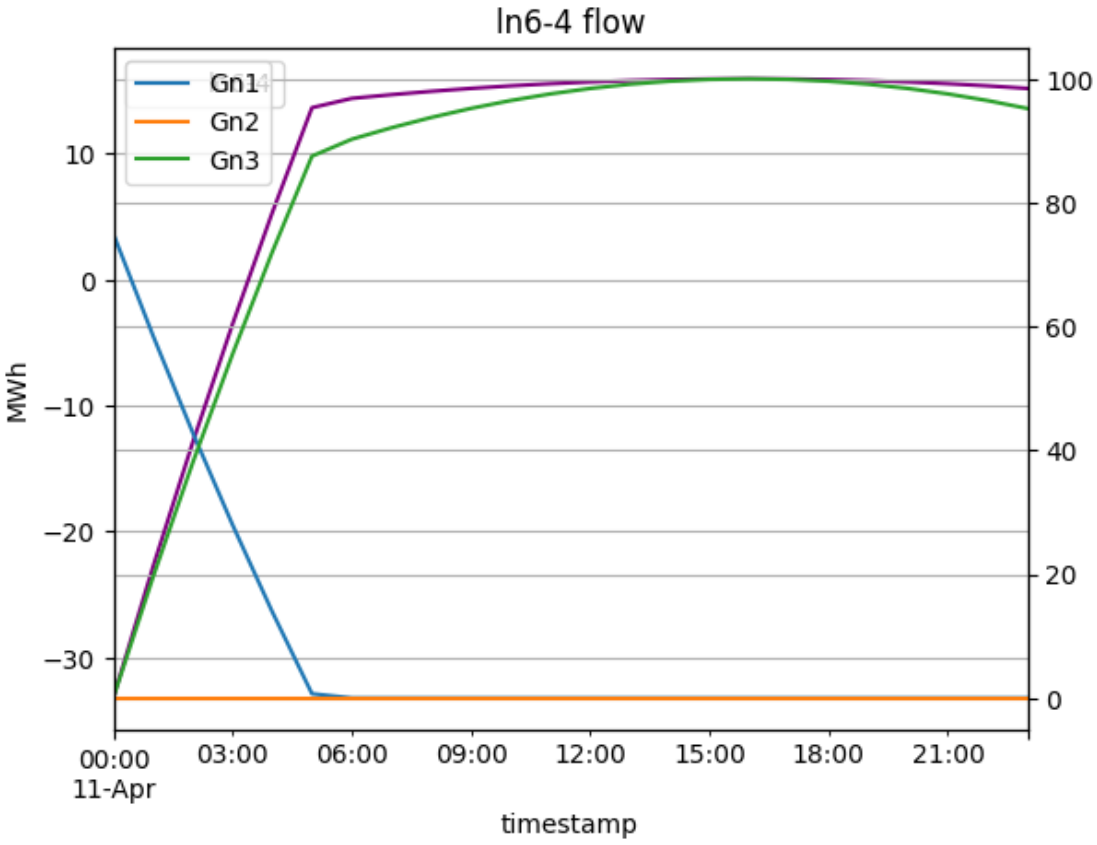
[ ]:
```
<Axes: title={'center': 'aggregate gen'}, xlabel='snapshot'>
```



[ ]:
```
# this all makes sense so far
# now let's take a look at how lines are flowing without n-1 security without␣
 ↪outages
# the particular lines we are interested in is ln6-4 which connects load bus 6␣
 ↪to bus 4
dfm_line = result_wo_sc.dfm_line[['snapshot','ln6-4']]
ax = dfm_line.set_index("snapshot").plot(grid=True, title="ln6-4 flow",␣
 ↪color='purple')
ax.set_xlabel("timestamp")
ax.set_ylabel("MWh")


# This is reasonable so far...let's overlay the gen output plot here
ax2 = ax.twinx()
dfm_gen.set_index("snapshot").plot(ax=ax2, grid=True)

# Explanation: In the early hours of morning, the wind profile is close to 0,
# so Gn1 that is more expensive is firing
# when the wind profile is bumped up throughout the day,
```

```
# most of the flow comes from the wind unit
# this is all good....next, however, we will see what happens to this case when
# ln7-8 goes out due to a storm
```

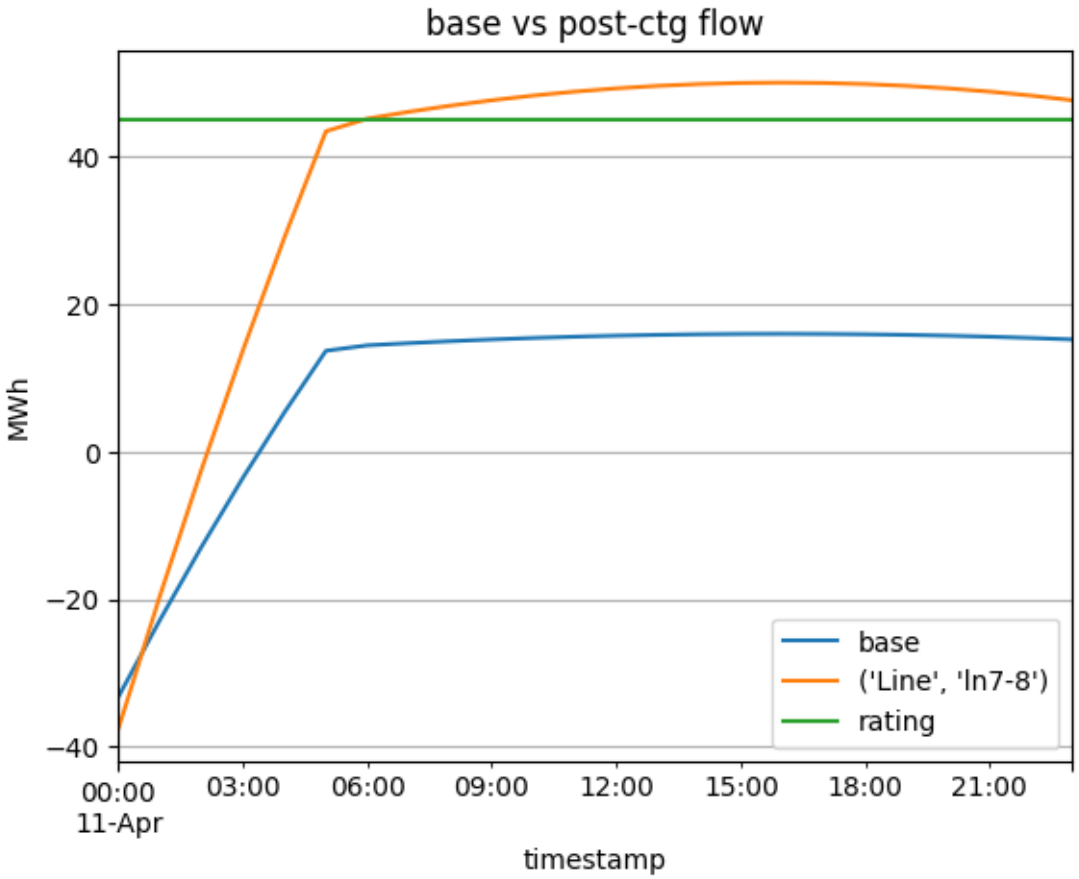
[ ]:
```
<Axes: xlabel='snapshot'>
```



[ ]:
```
# let's plot the post contingency flow for ln6-4 (the flow of ln6-4 after ln7-8␣
 ↪goes out)

dfm_post_ctg = result_wo_sc.dfm_post_ctg
dfm_post_ctg = dfm_post_ctg[dfm_post_ctg.label == 'ln6-4']
# let's include rating of ln6-4 (this can be obtained from static_csv)
dfm_post_ctg.loc[:, ["rating"]] = 45
ax = dfm_post_ctg.set_index("snapshot")[["base", "('Line', 'ln7-8')",␣
 ↪"rating"]].plot(grid=True, title="base vs post-ctg flow")
ax.set_xlabel("timestamp")
ax.set_ylabel("MWh")

# Do you notice the problem here? The solution offered violates the line rating␣
 ↪when
```

```
# the ln7-8 goes down.  During base case everything is ok, that is,
# the line flow is much lower than the rating. However, when the ln7-8 goes␣
 ↪down,
# the post contingency flow violates goes above the rating. This poses␣
 ↪additional
# damages to ln6-4 in the long run.  Next, we shall see that
# n-1 security optimization mitigates this...
```

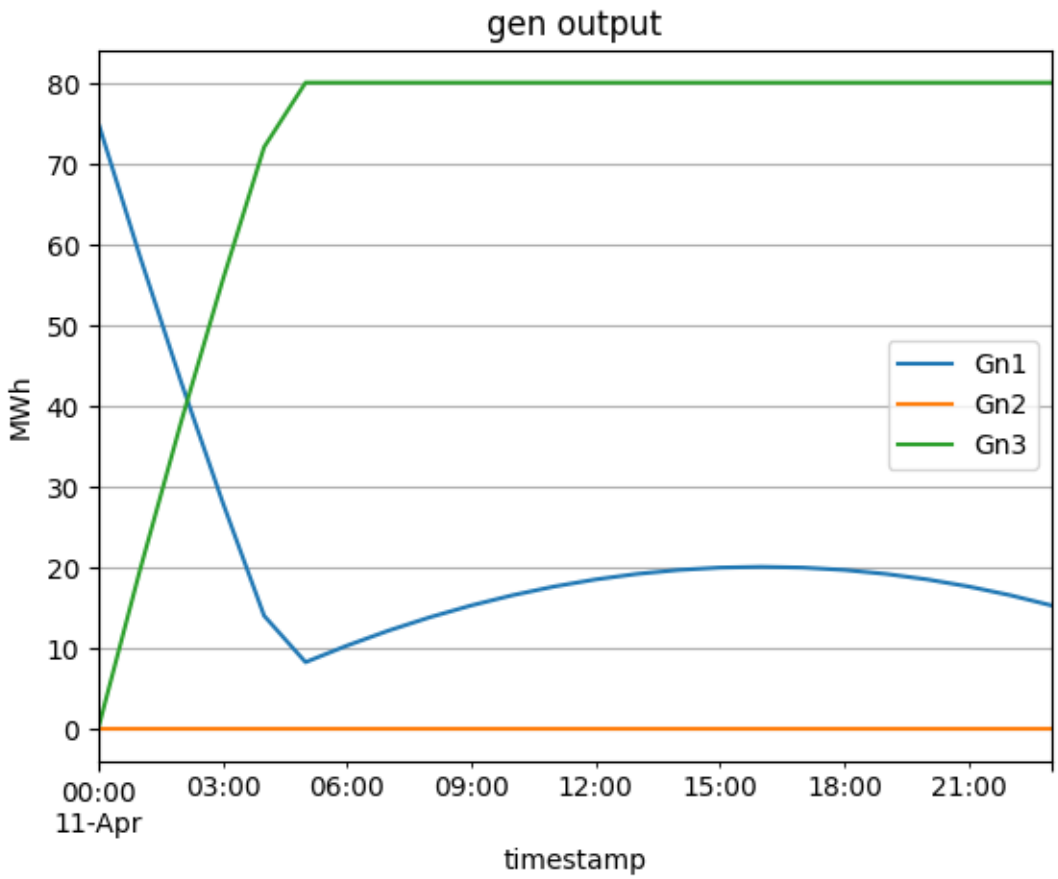
[ ]:
```
Text(0, 0.5, 'MWh')
```



[ ]:
```
# Let's start with how generators meet the load with n-1 security
# Recall that in result_wo_sc, wind (Gn3) meets the load during peak hours
# Here, Gn1 maintains some contribution although most of it is still met by␣
 ↪wind (Gn3)
# This is because in this case the network is ready for ln7-8 to fall out
dfm_gen = result_w_sc.dfm_gen
ax = dfm_gen.set_index("snapshot").plot(grid=True, title="gen output")
ax.set_xlabel("timestamp")
```

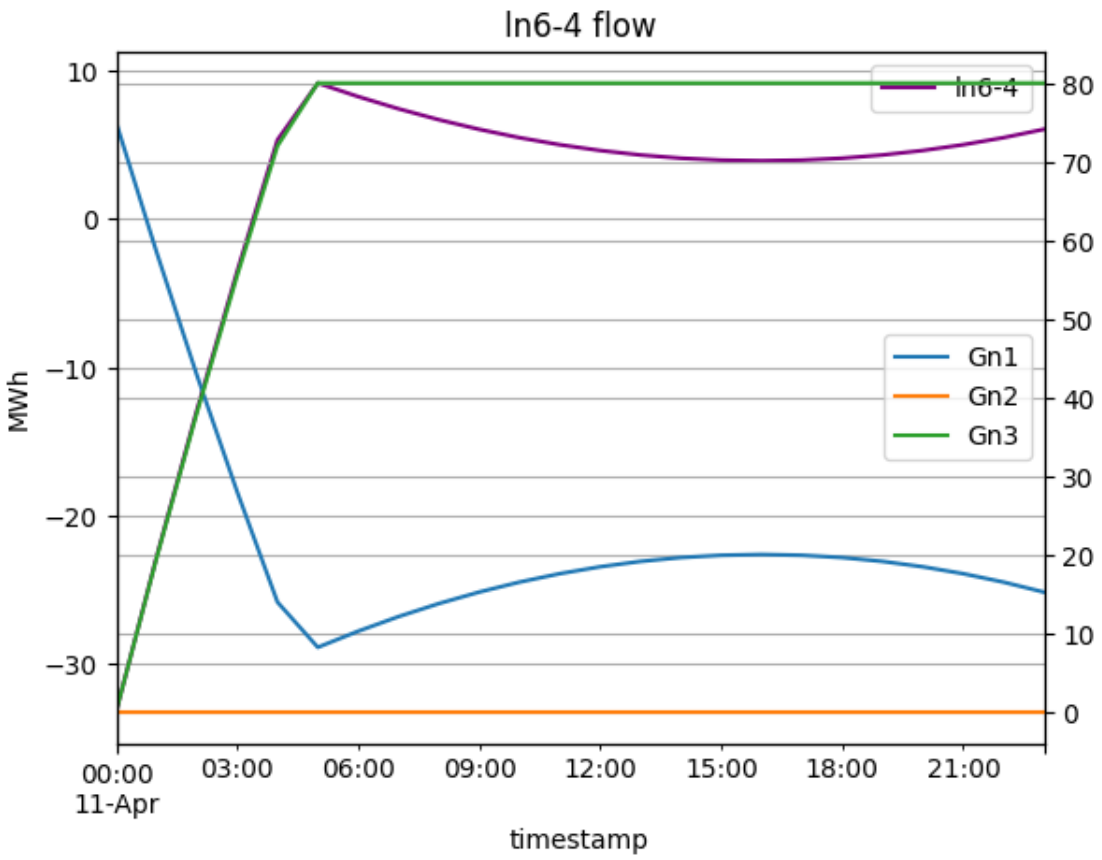
```
ax.set_ylabel("MWh")
```

[ ]: Text(0, 0.5, 'MWh')



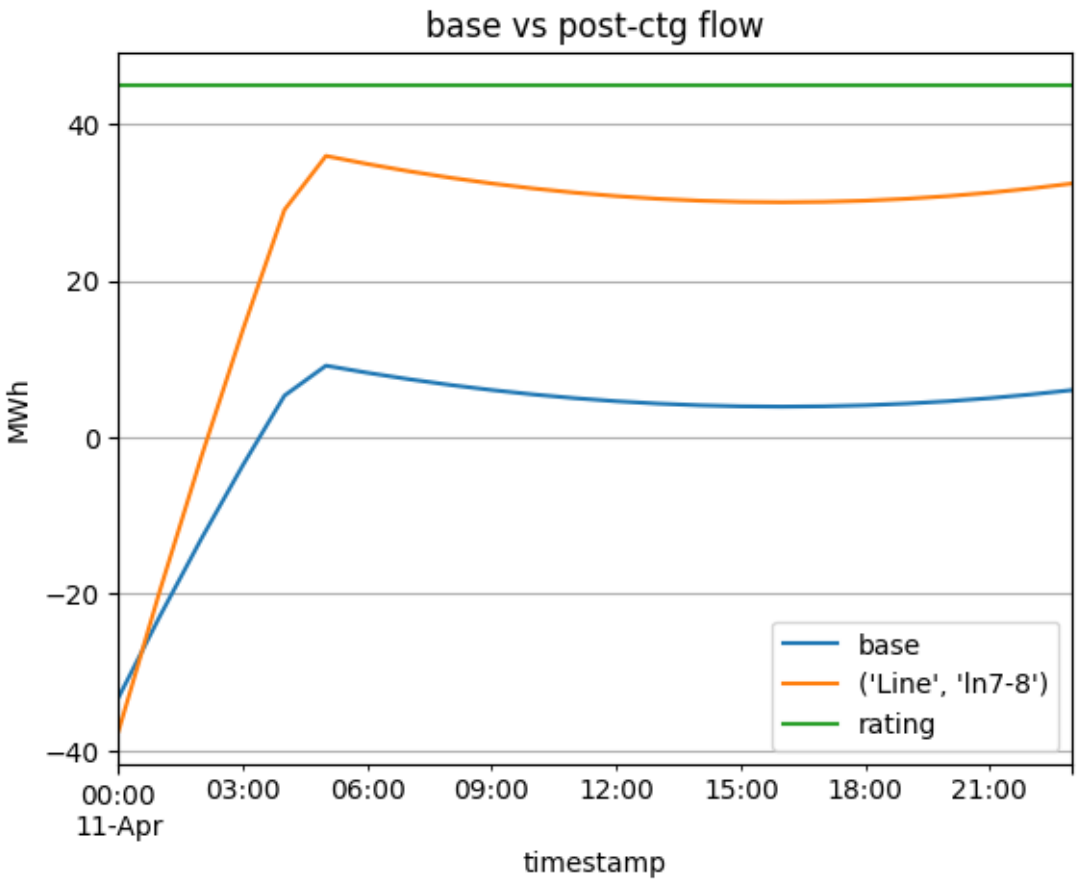
[ ]:
```
# you can examine the basecase flow
dfm_line = result_w_sc.dfm_line[['snapshot','ln6-4']]
ax = dfm_line.set_index("snapshot").plot(grid=True, title="ln6-4 flow",␣
 ↪color='purple')
ax.set_xlabel("timestamp")
ax.set_ylabel("MWh")


# Compare this with its analogue in wo n-1 security. Here, there is less␣
 ↪initial pressure on ln6-4
ax2 = ax.twinx()
dfm_gen.set_index("snapshot").plot(ax=ax2, grid=True)
```

[ ]: <Axes: xlabel='snapshot'>



[ ]:
```
# let's plot the post contingency flow for ln6-4 (the flow of ln6-4 after ln7-8␣
 ↪goes out)

dfm_post_ctg = result_w_sc.dfm_post_ctg
dfm_post_ctg = dfm_post_ctg[dfm_post_ctg.label == 'ln6-4']
# let's include rating of ln6-4 (this can be obtained from static_csv)
dfm_post_ctg.loc[:, ["rating"]] = 45
ax = dfm_post_ctg.set_index("snapshot")[["base", "('Line', 'ln7-8')",␣
 ↪"rating"]].plot(grid=True, title="base vs post-ctg flow")
ax.set_xlabel("timestamp")
ax.set_ylabel("MWh")

# Voila! The solution offered respects the line rating when
# the ln7-8 goes down.
```

[ ]:
```
Text(0, 0.5, 'MWh')
```


base vs post-ctg flow
40
20
0
−20
−40
MWh
00:00
11-Apr
03:00
06:00
09:00
12:00
15:00
18:00
21:00
timestamp
base
('Line', 'ln7-8')
rating