Final Project

Name: **Naman Makkar (nbm49)** and **Jan Carbonell (jc2897)**

Student ID: **nbm49** and **jc2897**

- Reasoning and work must be shown to gain partial/full credit
- Please include the cover-page on your homework PDF with your name and student ID. Failure of doing so is considered bad citizenship.

In this problem, you are going to run a unit commitment problem based on the software that is provided and your task is explain various aspects of the formulation.

Review of the Security-constrained Unit Commitment: The Security-constrained Unit Commitment (SUC) is the optimization minimizing the operational cost of generation that the ISO runs to make decisions about: 1) the generators units that will be online/offline (committed or not) in each of the next 24 hours; 2) the amount of reserves that are needed to balance the loss of a single generator (any generator), which is why is its name includes the adjective "security-constrained". The SCUC incorporates the dynamic safety constraints of generators (namely, how long can they stay on, and off once they are turned off, how fast they can ramp), which is the reason why one cannot solve it instant by instant.

Data for the problem: In this problem, you are given a single file (ieee118.xls) which contains multiple sheets. The metadata corresponding to each sheet is explained below,

Table 1: Generation metadata

Generation	
GenNumber	Generator number
Bus	Generator's bus
Pmax	Generator's maximum generation level, in MW
Pmin	Generator's minimum generation level, in MW
LinearCost	Generator's variable operating cost, in $/MWh
StartupCost	Startup cost of generator, in $
ShutdownCost	Shutdown cost of generator, in $
ReserveCost	Reseve cost of generator, in $
NoLoadCost	No-load cost of generator, in $
minDOWN	Minimum down time for generator, in hr
minUP	Minimum up time for generator, in hr
HourlyRamp	Generator hourly ramp rate, in MW/hr
Startup/ShutdownRamp	Startup/Shutdown level for generator, in MW/hr

Table 2: Bus metadata

Bus	
BusNumber	Bus number
Pload	Bus' real power load
Qload	Bus' reactive power load

Table 3: Branch metadata

Branch	
branch_number	Branch number
from_bus	From bus number of the branch
to_bus	To bus number of the branch
LineR	Resistance of line, in pu
LineX	Reactance of line, in pu
B(susceptance)	Susceptance of line
rateA	Continuous rating for branch, in MW
rateC	Emergency rating for branch, in MW

Table 4: Load metadata

Load	
Hour	Hour of day
Day	Percentage of load at a given hour and day X, referenced to base load

1. **SCUC formulation**: Let $\mathcal{G} := \{\mathcal{V}, \mathcal{E}\}$ be the graph associated with the grid where $\mathcal{V}$ is the set of buses and $\mathcal{E}$ is the set of lines/edges such that edge $e := \{i, j\} \in \mathcal{E}$ connects two buses $i, j \in \mathcal{V}$. Let $N := |\mathcal{V}|$ denotes the number of buses and $E := |\mathcal{E}|$ is the number of edges ($|\mathcal{S}|$ is the cardinality of a set $\mathcal{S}$). In the network, there is a set of generator $\mathcal{N}_g$ of size $N_g := |\mathcal{N}_g|$. All active power generators injections at time t are stacked in the column vector $\mathbf{p}_t^g \in \mathbb{R}^{N_g}$ where $t \in \mathcal{T}$ and $\mathcal{T}$ is the time horizon of 24h. Since the generators are only in some of the buses, the selection matrix $\mathbf{E}_g \in \{0,1\}^{N \times N_g}$ is such that entries of vector $\mathbf{E}_g \mathbf{p}_t^g$ is are either 0 (for buses without generation) or the generator injection (for the buses with generators). For each generator the SCUC decides three types of binary variables at time t, in the entries of the vectors $\mathbf{u}_t, \mathbf{v}_t, \mathbf{w}_t \in \{0,1\}^{N_g}$, where $\mathbf{u}_t$ are the commitment, $\mathbf{v}_t$ are the start-up and $\mathbf{w}_t$ are the shut-down status of the generators. The costs associated to the status being one are no-load, start-up and shut-down costs $\mathbf{c}_{nl}, \mathbf{c}_{su}$ and $\mathbf{c}_{sd}$ vectors. In addition to providing power $\mathbf{p}_t^g$ at cost $\mathbf{c}_l$, generators can provide reserves $\mathbf{p}_t^r \in \mathbb{R}^{N_g}$ at cost $\mathbf{c}_r$. A security-constrained unit commitment (SCUC) is formulated as a mixed-integer linear program of the form:

$$\min \quad \sum_{t\in\mathcal{T}} \left(\mathbf{c}_l^\intercal \mathbf{p}_t^g + \mathbf{c}_{nl}^\intercal \mathbf{u}_t + \mathbf{c}_{su}^\intercal \mathbf{v}_t + \mathbf{c}_{sd}^\intercal \mathbf{w}_t + \mathbf{c}_r^\intercal \mathbf{p}_t^r\right), \tag{1a}$$

$$\text{s.t.} \quad -\overline{\mathbf{P}}_\ell \leq \text{diag}(\mathbf{b})\mathbf{M}\boldsymbol{\theta}_t \leq \overline{\mathbf{P}}_\ell, \quad \forall t \in \mathcal{T}, \tag{1b}$$

$$\text{diag}(\underline{\mathbf{p}}^{\mathbf{g}})\mathbf{u}_t + \mathbf{p}_t^r \leq \mathbf{p}_t^g \leq \text{diag}(\overline{\mathbf{p}}^{\mathbf{g}})\mathbf{u}_t - \mathbf{p}_t^r, \quad \forall t \in \mathcal{T}, \tag{1c}$$

$$\mathbf{E}_g \mathbf{p}_t^g - \mathbf{p}_t^d = \mathbf{B}\boldsymbol{\theta}_t, \quad \forall t \in \mathcal{T}, \tag{1d}$$

$$\mathbf{p}_t^g - \mathbf{p}_{t-1}^g \leq \text{diag}(\overline{\mathbf{r}}_{hr})\mathbf{u}_{t-1} + \text{diag}(\overline{\mathbf{r}}_{su})\mathbf{v}_t, \quad \forall t \in \mathcal{T}, \tag{1e}$$

$$\mathbf{p}_{t-1}^g - \mathbf{p}_t^g \leq \text{diag}(\overline{\mathbf{r}}_{hr})\mathbf{u}_t + \text{diag}(\overline{\mathbf{r}}_{sd})\mathbf{w}_\mathbf{t}, \quad \forall t \in \mathcal{T}, \tag{1f}$$

$$\sum_{s=t-T_i^{on}+1}^{t} [\mathbf{v}_s]_i \leq [\mathbf{u}_t]_i, \quad \forall t \in \{T_i^{on}, \ldots, N\}, \forall i \in \{1, \ldots, N_g\}, \tag{1g}$$

$$\sum_{s=t-T_i^{\text{off}}+1}^{t} [\mathbf{w}_s]_i \leq 1 - [\mathbf{u}_t]_i, \quad \forall t \in \{T_i^{\text{off}}, \ldots, N\}, \forall i \in \{1, \ldots, N_g\}, \tag{1h}$$

$$\mathbf{v}_t - \mathbf{w}_t = \mathbf{u}_t - \mathbf{u}_{t-1}, \quad \forall t \in \mathcal{T}, \tag{1i}$$

$$\left(\mathbb{1}^{\text{T}} \mathbf{p}_t^r\right) \geq [\mathbf{p}_t^g + \mathbf{p}_t^r]_i, \quad \forall t \in \mathcal{T}, \quad \forall i \in \{1, \ldots, N_g\}, \tag{1j}$$

$$\mathbf{0} \leq \mathbf{p}_t^r \leq \text{diag}(\overline{\mathbf{p}}^{\mathbf{g}})\mathbf{u}_t, \quad \forall t \in \mathcal{T}. \tag{1k}$$

where $\overline{\mathbf{P}}_\ell \in \mathbb{R}^E$ is the vector of *Rate A* thermal capacity limits of the lines, $\mathbf{b} \in \mathbb{R}^E$ is the vector of line susceptances in p.u., $\mathbf{M} \in \{-1, 0, 1\}^{N\times E}$ is the directed incidence matrix of the graph $\mathcal{G}$ and $\boldsymbol{\theta}_t \in \mathbb{R}^N$ is the vector of voltage angles. The vectors $\underline{\mathbf{p}}^g$ and $\overline{\mathbf{p}}^g \in \mathbb{R}^{N_g}$ denote the minimum and maximum capacity limits of the generators. Furthermore, $\mathbf{p}_t^d \in \mathbb{R}^N$ denotes the vector of nodal demand and $\mathbf{B} := \mathbf{M}\text{diag}(\mathbf{b})\mathbf{M}^\intercal, \mathbf{B} \in \mathbb{R}^{N\times N}$ is the matrix of susceptances. Also, the vectors $\overline{\mathbf{r}}_{hr}, \overline{\mathbf{r}}_{su}, \overline{\mathbf{r}}_{sd} \in \mathbb{R}^{N_g}$ denote the maximum hourly, start-up and shut-down ramp rates the generators can tolerate. The scalars T_i^{on} and T_i^{off} denote the minimum up and down times of generator $i \in \mathcal{N}_g$ in hours. Finally, $[\mathbf{x}_t]_i$ denotes the i-th element of vector $\mathbf{x}$ at time t. Eq. (1a) is the total operational cost to be minimized and includes the cost of energy, reserves and commitment/start-up/shut-down costs. Eq. (1b) enforces the thermal capacity limits of the lines and Eq. (1c) includes the capacity limits of the generators. The power flow constraint is included in Eq. (1d). Eqs. (1e)-(1f) denote the ramping constraints and consider the start-up and shut-down ramping rates. The minimum up and down time constraints are given by Eqs. (1g) and (1h). Eq. (1i) ensures that the commitment variables are consistent with the start-up and shut-down variables. The N-1 reliability constraint is given by Eq. (1j) and Eq. (1k) ensure that the reserves provided by the generator are greater or equal than zero and less than the capacity of the generator.

2. (1–5 points) **Questions on the SCUC**:

   (a) (**1 points**) Consider the constraints that contain the binary variables, specifically: Eqs. (1c), (1e),(1f),(1g),(1h),(1i),(1k). Explain in words why each of the constraints ensures that the units are operated within the safety constraints. You can draw different examples to explain your words. (E.g. you can assume that a generator is up and turns off, or a generator is off and turns on, etc)

   Answer: The following constraints involving binary variables in the SCUC formulation cater to the generator's operational safety and reliability:

- Eq. (1c) regulates the generation capacity by ensuring that committed generators operate within their rated minimum and maximum limits, adjusted for reserves.
- Eqs. (1e) and (1f) manage the ramping constraints, ensuring the rate of change in power generation during start-ups and shut-downs adhere to specified ramp rates.
- Eqs. (1g) and (1h) ensure generators adhere to minimum up and down times to avoid frequent cycling and associated wear and tear.
- Eq. (1i) maintains consistency between generator commitment and its start-up or shut-down status.
- Eq. (1k) regulates reserves ensuring they are non-negative and within generator capacity, to handle unforeseen demand or supply variations.

These constraints together ensure the generators operate within their physical and safety margin limitations, guaranteeing grid reliability.

(b) (**0.25 points**) Here constraint (1j) ensures that each generator can fail and there are enough reserves to replace it. Explain why.

Answer: This equation corresponds to the N-1 reliability constraint. This constraint ensures that the total amount of reserves across all generators at any time is greater than or equal to the active power generation plus the reserves of any single generator.

This is useful because, in the event of the failure of one generator, the system should have enough reserves in place to compensate for that loss, thereby maintaining system reliability.

(c) (**0.25 points**) Install **gurobi** in your machine. Create an account in `gurobi.com` and download the software compatible with your OS from `https://www.gurobi.com/downloads/gurobi-software/`. You should request a *Named-User Academic* student license from `https://portal.gurobi.com/iam/licenses/list`. Once you get the license, activate it by running the *grbgetkey* command from terminal followed by your license key. You need to be on campus to complete this step (using the VPN form Cornell will not work). Then, install the python package **gurobipy**[1] on your anaconda environment. Run the code below and provide a screenshot of the output.

In [1]:
```
import cvxpy as cp
import numpy as np
# Generate a random problem
np.random.seed(0)
m, n= 40, 25

A = np.random.rand(m, n)
b = np.random.randn(m)
# Construct a CVXPY problem
x = cp.Variable(n, integer=True)
objective = cp.Minimize(cp.sum_squares(A @ x - b))
prob = cp.Problem(objective)
prob.solve(solver = cp.GUROBI,verbose = True)
```

(d) (**0.25 points**) Build the vectors of constants, i.e. the vectors $\mathbf{c}_l$, $\mathbf{c}_{nl}, \mathbf{c}_{su}$, $\mathbf{c}_{sd}$, $\mathbf{c}_r$, $\overline{\mathbf{P}}_\ell$, $\mathbf{b}$, $\mathbf{M}$, $\underline{\mathbf{p}}^g$, $\overline{\mathbf{p}}^g$, $\mathbf{E}_g$, $\mathbf{B}$, $\overline{\mathbf{r}}_{hr}$, $\overline{\mathbf{r}}_{su}$ and $\overline{\mathbf{r}}_{sd}$. Do not forget to express the susceptance, thermal

[1] `https://pypi.org/project/gurobipy/`

```
Set parameter Username
Academic license - for non-commercial use only - expires 2023-10-08
```

Out[1]: 13.660003257778829

Figure 1: Screenshot of the output of the code

limits, and capacity limits in p.u. The price constants do not need any transformations. You do not need to print the vectors or make any plots.

In [2]:

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
path_excel_file = '/Users/namanmakkar/Downloads/Final/ieee118.xls'
bus = pd.read_excel(open(path_excel_file, 'rb'), sheet_name='Bus')
branch = pd.read_excel(open(path_excel_file, 'rb'), sheet_name='Branch')
load = pd.read_excel(open(path_excel_file, 'rb'), sheet_name='Load')
generation = pd.read_excel(open(path_excel_file,'rb'), sheet_name='Generation')
pmax = np.array(generation['Pmax'])/100.0
pmin = np.array(generation['Pmin'])/100.0
hourly_ramp = np.array(generation['HourlyRamp'])
startup_shutdown_ramp = np.array(generation['Startup/ShutdownRamp'])
E = len(branch['from_bus'])
N = max(branch['to_bus'])
fbus = list(branch['from_bus'])
tbus = list(branch['to_bus'])
M = [[0 for _ in range(N)] for _ in range(E)]

for edge in range(E):
    node1 = fbus[edge]
    node2 = tbus[edge]
    M[edge][node1 - 1] = -1
    M[edge][node2 - 1] = 1
M = np.array(M)
M = M.T
susceptances = np.array(branch['B(susceptance)'])/100.0
b = np.diag(susceptances)
b = -b
B = M @ b @ M.T
N_g = max(generation['GenNumber'])
Egen = np.zeros((118,54))#[[0 for _ in range(N_g)] for _ in range(N)]
bus_gen = list(generation['Bus'])
gens = list(generation['GenNumber'])
for idx,bus_val in enumerate(bus_gen):
    Egen[bus_val-1, idx] = 1
noload_cost = np.array(generation['NoLoadCost'])
linear_cost = np.array(generation['LinearCost'])
startup_cost = np.array(generation['StartupCost'])
shutdown_cost = np.array(generation['ShutdownCost'])
reserve_cost = np.array(generation['ReserveCost'])
fmax = np.array(branch['rateA'])/100.0
fmin = -fmax
```

(e) (**0.25 points**) For each day, create a matrix of nodal demands $\mathbf{P}^d := [\mathbf{p}_1^d, \ldots, \mathbf{p}_{24}^d] \in \mathbb{R}^{N \times T}$ where T is the number of time intervals, in hours, i.e. $T := 24$. You should have one matrix per day. Then, for each day, sum over the columns of $\mathbf{P}^d$ and plot the resulting curves. You should have a plot with 5 curves, one per day, where the x-axis denotes hour of the day (1 through 24) and the y-axis denotes total load in MW. Which day has the highest load? Which has the lowest? What day would you expect the total operational

cost to be the highest? Why? Comment your results.
Solution: It can be observed by looking at the plot that **day 2** has the highest load and **day 4** has the lowest load.

In [3]:

```
power_demand_day1 = [[0 for _ in range(24)] for _ in range(N)]
power_demand_day2 = [[0 for _ in range(24)] for _ in range(N)]
power_demand_day3 = [[0 for _ in range(24)] for _ in range(N)]
power_demand_day4 = [[0 for _ in range(24)] for _ in range(N)]
power_demand_day5 = [[0 for _ in range(24)] for _ in range(N)]
demand_percent_day1 = np.array(load['Day1']/100.0)
demand_percent_day2 = np.array(load['Day2'])/100.0
demand_percent_day3 = np.array(load['Day3'])/100.0
demand_percent_day4 = np.array(load['Day4'])/100.0
demand_percent_day5 = np.array(load['Day5'])/100.0

demand_percentage_matrix = np.array([demand_percent_day1,
    demand_percent_day2, demand_percent_day3,
    demand_percent_day4, demand_percent_day5])

p_load = np.array(bus['Pload'])/100.0
bus_number = np.array(bus['BusNumber'])
for bus_idx in range(N):
    for hr in range(demand_percent_day1.shape[0]):
        power_demand_day1[bus_idx][hr] = p_load[bus_idx]*
    demand_percent_day1[hr]
        power_demand_day2[bus_idx][hr] = p_load[bus_idx]*
    demand_percent_day2[hr]
        power_demand_day3[bus_idx][hr] = p_load[bus_idx]*
    demand_percent_day3[hr]
        power_demand_day4[bus_idx][hr] = p_load[bus_idx]*
    demand_percent_day4[hr]
        power_demand_day5[bus_idx][hr] = p_load[bus_idx]*
    demand_percent_day5[hr]

power_demand_day1 = np.array(power_demand_day1)
power_demand_day2 = np.array(power_demand_day2)
power_demand_day3 = np.array(power_demand_day3)
power_demand_day4 = np.array(power_demand_day4)
power_demand_day5 = np.array(power_demand_day5)

total_demand_day1 = np.sum(power_demand_day1, axis=0)
total_demand_day2 = np.sum(power_demand_day2, axis=0)
total_demand_day3 = np.sum(power_demand_day3, axis=0)
total_demand_day4 = np.sum(power_demand_day4, axis=0)
total_demand_day5 = np.sum(power_demand_day5, axis=0)

min_up_time = np.array(generation['minUP'])
min_down_time = np.array(generation['minDOWN'])
fmin = fmin[:,np.newaxis]
fmax = fmax[:,np.newaxis]
pmin = pmin[:,np.newaxis]
pmax = pmax[:,np.newaxis]
hourly_ramp = hourly_ramp[:,np.newaxis]
startup_shutdown_ramp = startup_shutdown_ramp[:,np.newaxis]

hours = np.arange(1,25)
fig = plt.figure()
plt.plot(hours, total_demand_day1, 'r', label='Day 1')
plt.plot(hours, total_demand_day2, 'b', label='Day 2')
plt.plot(hours, total_demand_day3, 'g', label='Day 3')
plt.plot(hours, total_demand_day4, 'orange', label='Day 4')
plt.plot(hours, total_demand_day5, 'purple', label='Day 5')
plt.xlabel('Hours of the Day')
plt.ylabel('Demand in MW')
plt.legend()
fig.savefig('demand_by_hours.png')
plt.show()
```

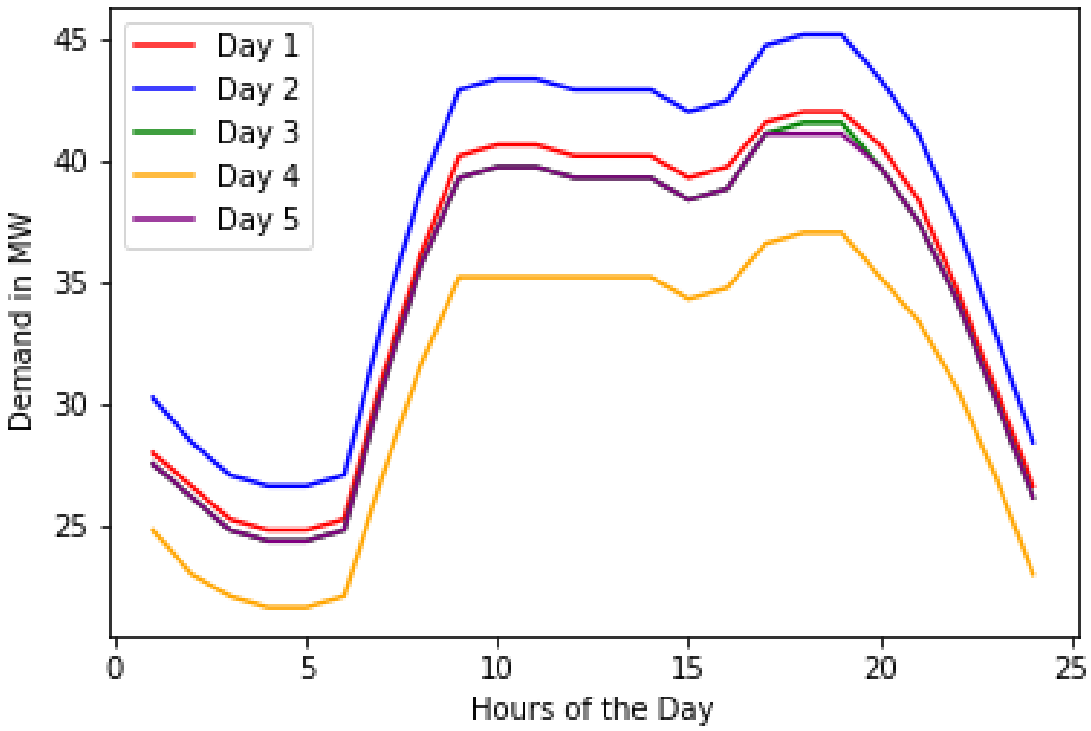

Figure 2: Load in MW vs hour of the day

(f) (**2 points**) Use the code provided in *uc.py*, parse the test case file *ieee118.xlsx* to solve Eq. (1) for the scenario provided in the first day.

1. (**1 points**) Solve Eq. (1) for the scenario provided in the first day. You need to feed the code with all the variables that you created before. You can use a *MIPgap = 0.01* to speed up the convergence (it should take around 5 minutes to run a single day). Provide all the code you have used to solve the problem (without the output) in a single cell.

In [4]:

```
##DAY 2 HAS THE HIGHEST LOAD AND DAY 4 HAS THE LOWEST LOAD
# First solving for day 1
d = power_demand_day1
T = 24

p = cp.Variable((generation.shape[0],T))
r = cp.Variable((generation.shape[0],T))
theta = cp.Variable((N,T))
u = cp.Variable((generation.shape[0],T),boolean = True) # Commitment variable
v = cp.Variable((generation.shape[0],T),boolean = True) # Startup variable
w = cp.Variable((generation.shape[0],T),boolean = True) # Shutdown variable

obj = cp.Minimize(cp.sum(linear_cost.T@p + startup_cost.T@v + shutdown_cost.T@w + noload_cost.T@u + reserve_cost.T@r))
# obj = cp.Minimize(cp.sum(linear_cost.T@p))

power_flow_constraints = [Egen@p - d == B@theta]
flow_limits = [fmin@np.ones((1,T)) <= b@M.T@theta, b@M.T@theta <= fmax@np.ones((1,T))]
generator_limits = [np.diag(pmin.T[0])@u + r <= p, p <= np.diag(pmax.T[0])@u - r]
ramp_limits_startup = [p[:,i] - p[:,i-1] <= np.diag(hourly_ramp.T[0])@u[:,i-1] + np.diag(startup_shutdown_ramp.T[0])@v[:,i] for i in range(1,T)]
ramp_limits_shutdown = [p[:,i-1] - p[:,i] <= np.diag(hourly_ramp.T[0])@u[:,i] + np.diag(startup_shutdown_ramp.T[0])@w[:,i] for i in range(1,T)]
commitment_constraints = [v[:,i] - w[:,i] == u[:,i] - u[:,i-1] for i in range(1,T)]
```

In [5]:

```
min_up_time_constraints = []
for idx, utime in enumerate(min_up_time):
    for t in range(utime-1,T):
        min_up_time_constraints += [cp.sum(v[idx,t-utime+1:t+1]) <= u[idx,t]]

min_down_time_constraints = []
for idx, dtime in enumerate(min_down_time):
    for t in range(dtime-1,T):
        min_down_time_constraints += [cp.sum(w[idx,t-dtime+1:t+1]) <= 1-u[idx,t]]

reserve_constraints = [cp.sum(r[:,t]) >= 0.07*cp.sum(d[:,t]) for t in range(T)]
reserve_constraints += [cp.sum(r[:,t]) >= p[idx,t] + r[idx,t] for t in range(T) for idx,_ in enumerate(pmax)]
reserve_constraints += [r <= np.diag(pmax.T[0])@u, r >= 0]

# constraints = power_flow_constraints + flow_limits + generator_limits + commitment_constraints + min_up_time_constraints + min_down_time_constraints + reserve_constraints + ramp_limits_startup + ramp_limits_shutdown
constraints = power_flow_constraints + flow_limits + generator_limits + commitment_constraints + reserve_constraints + ramp_limits_startup + ramp_limits_shutdown + min_up_time_constraints + min_down_time_constraints

prob = cp.Problem(obj, constraints)
prob.solve(solver = cp.GUROBI,verbose=True,MIPgap = 0.01)
```

2. (**0.25 points**) Report the objective cost and running time.

   Solution: The objective cost of **191855.99108233556** and a running time of **68.758 seconds** was achieved.

3. (**0.25 points**) Provide a table with 24 rows (one by hour) and 5 columns, with the aggregated generation, aggregated demand and aggregated reserves by hour. The fourth column should contain the ID of the generator that is providing the largest injection and reserves combined. The fifth column should contain the dispatch and reserve sum for the generator in column 4. What do you observe? Are your constraints being enforced?
   Solution: The table can be observed in **Figure 3**. It can be observed that for the first 21 hours the maximum generation is carried out by generator of id 6 while for the last 3 hours the maximum generation is carried out by generators 34 and 31.

4. (**0.25 points**) Use the matplotlib function *imshow*[2] to plot the matrices of binary variables $\mathbf{u}, \mathbf{v}$ and $\mathbf{w}$. What generators are being committed? What generators turn on/off the most? Is this consistent with the minimum up/down times? Comment your results.
   Solution: The plots for matrices $\mathbf{u}, \mathbf{v}$ and $\mathbf{w}$ can be observed in **Figures 4, 5 and 6**. The yellow lines in the image for matrix u show the generators that are being

[2] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imshow.html

	aggregate_generation	aggregate_demand	aggregate_reserve	max_gen_id	max_disp_res
0	0.518848	0.237439	0.074074	6	4.0
1	0.493743	0.225950	0.074074	6	4.0
2	0.468637	0.214461	0.074074	6	4.0
3	0.460269	0.210631	0.074074	6	4.0
4	0.460269	0.210631	0.074074	6	4.0
5	0.468637	0.214461	0.074074	6	4.0
6	0.577428	0.264247	0.074074	6	4.0
7	0.669481	0.306373	0.074074	6	4.0
8	0.744798	0.340840	0.074074	6	4.0
9	0.753167	0.344669	0.074074	6	4.0
10	0.753167	0.344669	0.074074	6	4.0
11	0.744798	0.340840	0.074074	6	4.0
12	0.744798	0.340840	0.074074	6	4.0
13	0.744798	0.340840	0.074074	6	4.0
14	0.728061	0.333181	0.074074	6	4.0
15	0.736430	0.337010	0.074074	6	4.0
16	0.769904	0.352329	0.074074	6	4.0
17	0.778272	0.356158	0.074074	6	4.0
18	0.778272	0.356158	0.074074	6	4.0
19	0.753167	0.344669	0.074074	6	4.0
20	0.711324	0.325521	0.074074	6	4.0
21	0.644376	0.294884	0.074074	34	4.0
22	0.569059	0.260417	0.074074	31	4.0
23	0.493743	0.225950	0.074074	31	4.0

Figure 3: Table with aggregated generation, aggregated demand, aggregated reserves, ids of the maximum generator and the dispatch and reserve sums of that generator

committed.

```
Out[88]: <matplotlib.image.AxesImage at 0x7fc899a40a00>
```

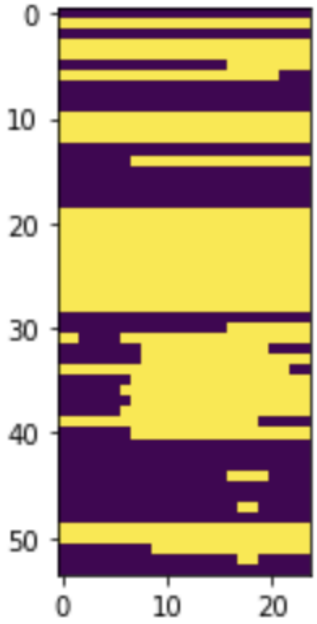

Figure 4: Plotted **u** matrix

5. (**0.25 points**) Calculate the reserve margin of the system by hour. The reserve margin is the amount of capacity left in the system to meet demand, i.e. $\mathbb{1}^\intercal \overline{\mathbf{p}^g} - \mathbb{1}^\intercal \mathbf{p}_t^g, \forall t$. Based on the reserve margin, would the system be able to meet the hourly demand if there is it increases hourly by 50%? Comment your results.
Solution:

In [6]:
```
ones_vector = np.ones((54, 1))
dot_product_p_max = ones_vector.T @ pmax
dot_product_p_g_t = ones_vector.T @ p_g_t
reserve_margin_by_hr = dot_product_p_max - dot_product_p_g_t
reserve_margin_by_hr
```

Out[6]:
```
array([[58.59  , 31.9279, 33.2836, 33.7355, 33.7355, 33.2836,
    27.4089,
        22.438 , 18.3709, 17.919 , 17.919 , 18.3709, 18.3709,
    18.3709,
        19.2747, 18.8228, 17.0152, 16.5633, 16.5633, 17.919 ,
    20.1785,
        23.7937, 27.8608, 31.9279]])
```

We increase generation by 50% and calculate the reserve margin. For all those hours where the reserve margin is negative the system will not be able to meet the hourly demand.

Out[91]: <matplotlib.image.AxesImage at 0x7fc90a551310>

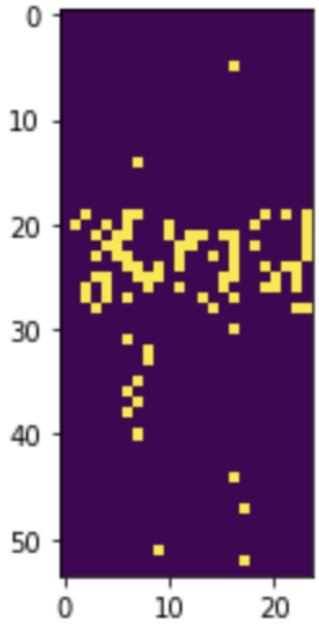

Figure 5: Plotted **v** matrix

Out[92]: <matplotlib.image.AxesImage at 0x7fc93bb04d00>

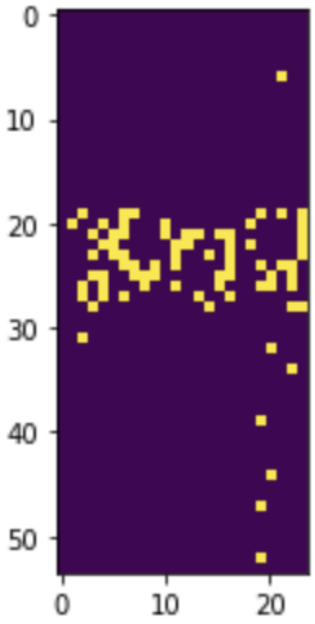

Figure 6: Plotted **w** matrix

In [7]:

```
ones_vector = np.ones((54, 1))
dot_product_p_max = ones_vector.T @ pmax
dot_product_p_g_t = 1.5*ones_vector.T @ p_g_t
reserve_margin_by_hr_incr = dot_product_p_max -
    dot_product_p_g_t
reserve_margin_by_hr_incr
```

Out[7]:

```
array([[16.5633 , 18.59685, 20.6304 , 21.30825, 21.30825,
    20.6304 ,
        11.81835,  4.362  , -1.73865, -2.4165 , -2.4165 ,
    -1.73865,
        -1.73865, -1.73865, -0.38295, -1.0608 , -3.7722 ,
    -4.45005,
        -4.45005, -2.4165 ,  0.97275,  6.39555, 12.4962 ,
    18.59685]])
```

(g) (**1 points**) Now, solve the problem for each day. You should solve each of day, one a time, using the load given in the excel file. For days $d = 2, \ldots, 5$, the inter-temporal constraints in Eqs. (1e)-(1h) should be consistent with the decisions you made the previous day, i.e. $d - 1$. For example, between Day 1 Hour 24, and Day 2, Hour 1, the generators should not violate the ramping constraints in Eq. (1e)-(1f). The same applies to the minimum up/down time constraints in Eqs. (1g)-(1h), e.g. if a generator turned off on Day 1 and hour 23 and the generator has a minimum down time of 4 hours, the generator should not be able to turn on until Day 2, hour 2. Also, on Day d, the decisions for Day d-1 are already taken, and you should treat those variables as given data (**do not re-optimize** variables from the previous day). For each day, report the running time and objective cost.

Solution: Running time is calculated as compile time + solver time. Objective Cost and running time for day 2 - **425236.436372819** and **58.94 seconds**. While the objective cost and runtime for day 3 were - **614147.081307615** and **281.32 seconds**. Objective cost and running time for day 4 were - **767461.7556687992** and **66.7 seconds**. Objective and running time for day 5 - **960194.936948663** and **95.72 seconds**. For days 2-5 an MIP gap of **0.05** was chosen.

In [8]:

```
power_demand_day1_and_2_combined = np.concatenate((
    power_demand_day1, power_demand_day2), axis=1)

#solving for day 2
d2 = power_demand_day1_and_2_combined
T = 48

p2 = cp.Variable((generation.shape[0],T))
r2 = cp.Variable((generation.shape[0],T))
theta2 = cp.Variable((N,T))
u2 = cp.Variable((generation.shape[0],T),boolean = True) #
    Commitment variable
v2 = cp.Variable((generation.shape[0],T),boolean = True) #
    Startup variable
w2 = cp.Variable((generation.shape[0],T),boolean = True) #
    Shutdown variable


obj = cp.Minimize(cp.sum(linear_cost.T@p2 + startup_cost.T@v2 +
     shutdown_cost.T@w2 + noload_cost.T@u2 + reserve_cost.T@r2))
# obj = cp.Minimize(cp.sum(linear_cost.T@p))

power_flow_constraints = [Egen@p2 - d2 == B@theta2]
flow_limits = [fmin@np.ones((1,T)) <= b@M.T@theta2, b@M.
    T@theta2 <= fmax@np.ones((1,T))]
generator_limits = [np.diag(pmin.T[0])@u2 + r2 <= p2, p2 <= np.
    diag(pmax.T[0])@u2 - r2]

#ramp_limits_startup = [p2[:,i] - p[:,i-1] <= np.diag(
    hourly_ramp.T[0])@u[:,i-1] + np.diag(startup_shutdown_ramp.T
    [0])@v[:,i]]
ramp_limits_startup = [p2[:,i] - p2[:,i-1] <= np.diag(
    hourly_ramp.T[0])@u2[:,i-1] + np.diag(startup_shutdown_ramp.
    T[0])@v2[:,i] for i in range(1,T)]
ramp_limits_shutdown = [p2[:,i-1] - p2[:,i] <= np.diag(
    hourly_ramp.T[0])@u2[:,i] + np.diag(startup_shutdown_ramp.T
    [0])@w2[:,i] for i in range(1,T)]
commitment_constraints = [v2[:,i] - w2[:,i] == u2[:,i] - u2[:,i
    -1] for i in range(1,T)]
prev_day_constraints_p = [p2[:,i] == p[:,i] for i in range
    (1,24)]
prev_day_constraints_u = [u2[:,i] == u[:,i] for i in range
    (1,24)]
prev_day_constraints_v = [v2[:,i] == v[:,i] for i in range
    (1,24)]
prev_day_constraints_w = [w2[:,i] == w[:,i] for i in range
    (1,24)]
prev_day_constraints_theta = [theta2[:,i] == theta[:,i] for i
    in range(1,24)]
prev_day_constraints_r = [r2[:,i] == r[:,i] for i in range
    (1,24)]
```

In [9]:

```
min_up_time_constraints = []
for idx, utime in enumerate(min_up_time):
    for t in range(utime-1,T):
        min_up_time_constraints += [cp.sum(v2[idx,t-utime+1:t+1]) <= u2[idx,t]]

min_down_time_constraints = []
for idx, dtime in enumerate(min_down_time):
    for t in range(dtime-1,T):
        min_down_time_constraints += [cp.sum(w2[idx,t-dtime+1:t+1]) <= 1-u2[idx,t]]

reserve_constraints = [cp.sum(r2[:,t]) >= 0.07*cp.sum(d2[:,t]) for t in range(T)]
reserve_constraints += [cp.sum(r2[:,t]) >= p2[idx,t] + r2[idx,t] for t in range(T) for idx,_ in enumerate(pmax)]
reserve_constraints += [r2 <= np.diag(pmax.T[0])@u2, r2 >= 0]

# constraints = power_flow_constraints + flow_limits + generator_limits + commitment_constraints + min_up_time_constraints + min_down_time_constraints + reserve_constraints + ramp_limits_startup + ramp_limits_shutdown
constraints = power_flow_constraints + prev_day_constraints_p + prev_day_constraints_u + prev_day_constraints_v + prev_day_constraints_w + prev_day_constraints_theta + prev_day_constraints_r + flow_limits + generator_limits + commitment_constraints + reserve_constraints + ramp_limits_startup + ramp_limits_shutdown + min_up_time_constraints + min_down_time_constraints

prob = cp.Problem(obj, constraints)
prob.solve(solver = cp.GUROBI,verbose=True,MIPgap = 0.05)
```

In [10]:

```
power_demand_combined_upto_day3 = np.concatenate((
    power_demand_day1_and_2_combined, power_demand_day3), axis
    =1)

#solving for day 3
d3 = power_demand_combined_upto_day3
T = 72

p3 = cp.Variable((generation.shape[0],T))
r3 = cp.Variable((generation.shape[0],T))
theta3 = cp.Variable((N,T))
u3 = cp.Variable((generation.shape[0],T),boolean = True) #
    Commitment variable
v3 = cp.Variable((generation.shape[0],T),boolean = True) #
    Startup variable
w3 = cp.Variable((generation.shape[0],T),boolean = True) #
    Shutdown variable


obj = cp.Minimize(cp.sum(linear_cost.T@p3 + startup_cost.T@v3 +
     shutdown_cost.T@w3 + noload_cost.T@u3 + reserve_cost.T@r3))


power_flow_constraints = [Egen@p3 - d3 == B@theta3]
flow_limits = [fmin@np.ones((1,T)) <= b@M.T@theta3, b@M.
    T@theta3 <= fmax@np.ones((1,T))]
generator_limits = [np.diag(pmin.T[0])@u3 + r3 <= p3, p3 <= np.
    diag(pmax.T[0])@u3 - r3]

#ramp_limits_startup = [p2[:,i] - p[:,i-1] <= np.diag(
    hourly_ramp.T[0])@u[:,i-1] + np.diag(startup_shutdown_ramp.T
    [0])@v[:,i]]
ramp_limits_startup = [p3[:,i] - p3[:,i-1] <= np.diag(
    hourly_ramp.T[0])@u3[:,i-1] + np.diag(startup_shutdown_ramp.
    T[0])@v3[:,i] for i in range(1,T)]
ramp_limits_shutdown = [p3[:,i-1] - p3[:,i] <= np.diag(
    hourly_ramp.T[0])@u3[:,i] + np.diag(startup_shutdown_ramp.T
    [0])@w3[:,i] for i in range(1,T)]
commitment_constraints = [v3[:,i] - w3[:,i] == u3[:,i] - u3[:,i
    -1] for i in range(1,T)]
prev_day_constraints_p = [p3[:,i] == p2[:,i] for i in range
    (1,48)]
prev_day_constraints_u = [u3[:,i] == u2[:,i] for i in range
    (1,48)]
prev_day_constraints_v = [v3[:,i] == v2[:,i] for i in range
    (1,48)]
prev_day_constraints_w = [w3[:,i] == w2[:,i] for i in range
    (1,48)]
prev_day_constraints_theta = [theta3[:,i] == theta2[:,i] for i
    in range(1,48)]
prev_day_constraints_r = [r3[:,i] == r2[:,i] for i in range
    (1,48)]
```

In [11]:

```
min_up_time_constraints = []
for idx, utime in enumerate(min_up_time):
    for t in range(utime-1,T):
        min_up_time_constraints += [cp.sum(v3[idx,t-utime+1:t
    +1]) <= u3[idx,t]]

min_down_time_constraints = []
for idx, dtime in enumerate(min_down_time):
    for t in range(dtime-1,T):
        min_down_time_constraints += [cp.sum(w3[idx,t-dtime+1:t
    +1]) <= 1-u3[idx,t]]

reserve_constraints = [cp.sum(r3[:,t]) >= 0.07*cp.sum(d3[:,t])
    for t in range(T)]
reserve_constraints += [cp.sum(r3[:,t]) >= p3[idx,t] + r3[idx,t
    ] for t in range(T) for idx,_ in enumerate(pmax)]
reserve_constraints += [r3 <= np.diag(pmax.T[0])@u3, r3 >= 0]

# constraints = power_flow_constraints + flow_limits +
    generator_limits + commitment_constraints +
    min_up_time_constraints + min_down_time_constraints +
    reserve_constraints + ramp_limits_startup +
    ramp_limits_shutdown
constraints = power_flow_constraints + prev_day_constraints_p +
     prev_day_constraints_u + prev_day_constraints_v +
    prev_day_constraints_w + prev_day_constraints_theta +
    prev_day_constraints_r + flow_limits + generator_limits +
    commitment_constraints + reserve_constraints +
    ramp_limits_startup + ramp_limits_shutdown +
    min_up_time_constraints + min_down_time_constraints

prob3 = cp.Problem(obj, constraints)
prob3.solve(solver = cp.GUROBI,verbose=True,MIPgap = 0.05)
```

In [12]:

```
power_demand_combined_upto_day4 = np.concatenate((
    power_demand_combined_upto_day3, power_demand_day4), axis=1)

#solving for day 4
d4 = power_demand_combined_upto_day4
T = 96

p4 = cp.Variable((generation.shape[0],T))
r4 = cp.Variable((generation.shape[0],T))
theta4 = cp.Variable((N,T))
u4 = cp.Variable((generation.shape[0],T),boolean = True) #
    Commitment variable
v4 = cp.Variable((generation.shape[0],T),boolean = True) #
    Startup variable
w4 = cp.Variable((generation.shape[0],T),boolean = True) #
    Shutdown variable

obj = cp.Minimize(cp.sum(linear_cost.T@p4 + startup_cost.T@v4 +
     shutdown_cost.T@w4 + noload_cost.T@u4 + reserve_cost.T@r4))
# obj = cp.Minimize(cp.sum(linear_cost.T@p))

power_flow_constraints = [Egen@p4 - d4 == B@theta4]
flow_limits = [fmin@np.ones((1,T)) <= b@M.T@theta4, b@M.
    T@theta4 <= fmax@np.ones((1,T))]
generator_limits = [np.diag(pmin.T[0])@u4 + r4 <= p4, p4 <= np.
    diag(pmax.T[0])@u4 - r4]

#ramp_limits_startup = [p2[:,i] - p[:,i-1] <= np.diag(
    hourly_ramp.T[0])@u[:,i-1] + np.diag(startup_shutdown_ramp.T
    [0])@v[:,i]]
ramp_limits_startup = [p4[:,i] - p4[:,i-1] <= np.diag(
    hourly_ramp.T[0])@u4[:,i-1] + np.diag(startup_shutdown_ramp.
    T[0])@v4[:,i] for i in range(1,T)]
ramp_limits_shutdown = [p4[:,i-1] - p4[:,i] <= np.diag(
    hourly_ramp.T[0])@u4[:,i] + np.diag(startup_shutdown_ramp.T
    [0])@w4[:,i] for i in range(1,T)]
commitment_constraints = [v4[:,i] - w4[:,i] == u4[:,i] - u4[:,i
    -1] for i in range(1,T)]
prev_day_constraints_p = [p4[:,i] == p3[:,i] for i in range
    (1,72)]
prev_day_constraints_u = [u4[:,i] == u3[:,i] for i in range
    (1,72)]
prev_day_constraints_v = [v4[:,i] == v3[:,i] for i in range
    (1,72)]
prev_day_constraints_w = [w4[:,i] == w3[:,i] for i in range
    (1,72)]
prev_day_constraints_theta = [theta4[:,i] == theta3[:,i] for i
    in range(1,72)]
prev_day_constraints_r = [r4[:,i] == r3[:,i] for i in range
    (1,48)]
```

In [13]:

```
min_up_time_constraints = []
for idx, utime in enumerate(min_up_time):
    for t in range(utime-1,T):
        min_up_time_constraints += [cp.sum(v4[idx,t-utime+1:t+1]) <= u4[idx,t]]

min_down_time_constraints = []
for idx, dtime in enumerate(min_down_time):
    for t in range(dtime-1,T):
        min_down_time_constraints += [cp.sum(w4[idx,t-dtime+1:t+1]) <= 1-u4[idx,t]]

reserve_constraints = [cp.sum(r4[:,t]) >= 0.07*cp.sum(d4[:,t]) for t in range(T)]
reserve_constraints += [cp.sum(r4[:,t]) >= p4[idx,t] + r4[idx,t] for t in range(T) for idx,_ in enumerate(pmax)]
reserve_constraints += [r4 <= np.diag(pmax.T[0])@u4, r4 >= 0]

# constraints = power_flow_constraints + flow_limits + generator_limits + commitment_constraints + min_up_time_constraints + min_down_time_constraints + reserve_constraints + ramp_limits_startup + ramp_limits_shutdown
constraints = power_flow_constraints + prev_day_constraints_p + prev_day_constraints_u + prev_day_constraints_v + prev_day_constraints_w + prev_day_constraints_theta + prev_day_constraints_r + flow_limits + generator_limits + commitment_constraints + reserve_constraints + ramp_limits_startup + ramp_limits_shutdown + min_up_time_constraints + min_down_time_constraints

prob4 = cp.Problem(obj, constraints)
prob4.solve(solver = cp.GUROBI,verbose=True,MIPgap = 0.05)
```

In [14]:

```
power_demand_combined_upto_day5 = np.concatenate((
    power_demand_combined_upto_day4, power_demand_day5), axis=1)

#solving for day 5
d5 = power_demand_combined_upto_day5
T = 120

p5 = cp.Variable((generation.shape[0],T))
r5 = cp.Variable((generation.shape[0],T))
theta5 = cp.Variable((N,T))
u5 = cp.Variable((generation.shape[0],T),boolean = True) #
    Commitment variable
v5 = cp.Variable((generation.shape[0],T),boolean = True) #
    Startup variable
w5 = cp.Variable((generation.shape[0],T),boolean = True) #
    Shutdown variable

obj = cp.Minimize(cp.sum(linear_cost.T@p5 + startup_cost.T@v5 +
     shutdown_cost.T@w5 + noload_cost.T@u5 + reserve_cost.T@r5))
# obj = cp.Minimize(cp.sum(linear_cost.T@p))

power_flow_constraints = [Egen@p5 - d5 == B@theta5]
flow_limits = [fmin@np.ones((1,T)) <= b@M.T@theta5, b@M.
    T@theta5 <= fmax@np.ones((1,T))]
generator_limits = [np.diag(pmin.T[0])@u5 + r5 <= p5, p5 <= np.
    diag(pmax.T[0])@u5 - r5]

ramp_limits_startup = [p5[:,i] - p5[:,i-1] <= np.diag(
    hourly_ramp.T[0])@u5[:,i-1] + np.diag(startup_shutdown_ramp.
    T[0])@v5[:,i] for i in range(1,T)]
ramp_limits_shutdown = [p5[:,i-1] - p5[:,i] <= np.diag(
    hourly_ramp.T[0])@u5[:,i] + np.diag(startup_shutdown_ramp.T
    [0])@w5[:,i] for i in range(1,T)]
commitment_constraints = [v5[:,i] - w5[:,i] == u5[:,i] - u5[:,i
    -1] for i in range(1,T)]
prev_day_constraints_p = [p5[:,i] == p4[:,i] for i in range
    (1,96)]
prev_day_constraints_u = [u5[:,i] == u4[:,i] for i in range
    (1,96)]
prev_day_constraints_v = [v5[:,i] == v4[:,i] for i in range
    (1,96)]
prev_day_constraints_w = [w5[:,i] == w4[:,i] for i in range
    (1,96)]
prev_day_constraints_theta = [theta5[:,i] == theta4[:,i] for i
    in range(1,96)]
prev_day_constraints_r = [r5[:,i] == r4[:,i] for i in range
    (1,96)]
```

In [15]:

```
min_up_time_constraints = []
for idx, utime in enumerate(min_up_time):
    for t in range(utime-1,T):
        min_up_time_constraints += [cp.sum(v5[idx,t-utime+1:t
    +1]) <= u5[idx,t]]

min_down_time_constraints = []
for idx, dtime in enumerate(min_down_time):
    for t in range(dtime-1,T):
        min_down_time_constraints += [cp.sum(w5[idx,t-dtime+1:t
    +1]) <= 1-u5[idx,t]]

reserve_constraints = [cp.sum(r5[:,t]) >= 0.07*cp.sum(d5[:,t])
    for t in range(T)]
reserve_constraints += [cp.sum(r5[:,t]) >= p5[idx,t] + r5[idx,t
    ] for t in range(T) for idx,_ in enumerate(pmax)]
reserve_constraints += [r5 <= np.diag(pmax.T[0])@u5, r5 >= 0]

# constraints = power_flow_constraints + flow_limits +
    generator_limits + commitment_constraints +
    min_up_time_constraints + min_down_time_constraints +
    reserve_constraints + ramp_limits_startup +
    ramp_limits_shutdown
constraints = power_flow_constraints + prev_day_constraints_p +
     prev_day_constraints_u + prev_day_constraints_v +
    prev_day_constraints_w + prev_day_constraints_theta +
    prev_day_constraints_r + flow_limits + generator_limits +
    commitment_constraints + reserve_constraints +
    ramp_limits_startup + ramp_limits_shutdown +
    min_up_time_constraints + min_down_time_constraints

prob5 = cp.Problem(obj, constraints)
prob5.solve(solver = cp.GUROBI,verbose=True,MIPgap = 0.05)
```

3. (1–3 points) **Electric vehicles (Lecture 23, Slides 54 onwards)**: In the next part, we can model the increase load that would incur due to EV charging in the unit commitment problem, assuming that at every bus b the electric load of charging vehicles is added to the demand considered in the previous problem. Recall that the model for the feasible set of aggregate load of EVs can be represented as follows, assuming that one can charge at an arbitrary rate (but not discharge):

$$\mathcal{L} = \Bigg\{ p^d(t) | p^d(t) = \rho \sum_{(\boldsymbol{x},\boldsymbol{x}')} (\boldsymbol{x}' - \boldsymbol{x})\, \partial d_{\boldsymbol{x},\boldsymbol{x}'}(t), \quad \partial d_{\boldsymbol{x},\boldsymbol{x}'}(t) \in \mathbb{Z}^+ \ \forall (\boldsymbol{x},\boldsymbol{x}') \tag{2}$$
$$n_{\boldsymbol{x}}(t) = a_{\boldsymbol{x}}(t) + \sum_{\boldsymbol{x}' \in \mathcal{U}_{\boldsymbol{x}}} \partial d_{\boldsymbol{x},\boldsymbol{x}'}(t) \ \forall \boldsymbol{x},$$
$$\mathcal{U}_{\boldsymbol{x}} = \Big\{ \boldsymbol{x}' \,|\, \|\boldsymbol{x}' - \boldsymbol{x}\|_1 = \min(\|\boldsymbol{x}\|_1, 1), \quad (\boldsymbol{x} - \boldsymbol{x}')_r \geq 0, (\boldsymbol{x} - \boldsymbol{x}')_s \geq 0 \Big\} \Bigg\}$$

and $\boldsymbol{x} = (r, s)$ refers to the residual time to fully charge and the slack time, $n_{\boldsymbol{x}}(t)$ is the number of cars plugged at time t that have residual time to fully charge and slack time $\boldsymbol{x} = (r, s)$, $n_{\boldsymbol{x}}(t)$, $a_{\boldsymbol{x}}(t)$ are the additional cars that plug in at time t and arrive in state $\boldsymbol{x}$, $\partial d_{\boldsymbol{x},\boldsymbol{x}'}(t) = d_{\boldsymbol{x},\boldsymbol{x}'}(t) - d_{\boldsymbol{x},\boldsymbol{x}'}(t-1)$ is the number of cars that change their state from $\boldsymbol{x}$ to $\boldsymbol{x}'$ as discussed in Lecture 23.

1. (**0.75 points**) Explain how you calculate the initial residual time to charge r from the battery capacity of the vehicle E, state of charge SoC_i and rate of charge ρ.

   Answer: In order to calculate the initial residual time to charge r with respect to:

   - The battery capacity of the vehicle E
   - state of charge SoC_i
   - rate of charge ρ

   We need to first determine how much more energy the battery needs to be fully charged. This is calculated in the following manner $(E - SoC_i)$, where SoC_i is the current state of charge of the battery, and E is the total energy capacity of the battery. The residual time to charge r is then obtained by dividing this energy requirement by the rate of charge ρ, which results in the following equation:

   $$r_i = \frac{E - SoC_i}{\rho} \tag{3}$$

2. (**0.75 points**) Explain why a state $\boldsymbol{x} = (r, s)$ of a car is such that: $||\boldsymbol{x}||_1 = r + s = \chi - t^a, \qquad r = 0, \ldots, T^r, \quad s = 0, \ldots, T^s.$
   The state of a car, denoted by $\boldsymbol{x} = (r, s)$, can be defined as the residual time to fully charge r and the slack time s. The residual r is the time it would take to charge the battery from its current state of charge to its full capacity at the given charging rate. The range of r goes from 0 when the battery is already fully charged to T^r -the maximum time it could take to fully charge the battery-.

   The slack s is the time remaining after the car is fully charged until the deadline χ. This value can range from 0 -if the deadline is exactly when the car will be fully charged- to

T^s, which corresponds to the maximum amount of time the car could be plugged in after it's fully charged.

$$S_i = \chi_i - t^a - r_i \tag{4}$$

Therefore, this is equal to the total time the car is available for charging:

$$||\boldsymbol{x}||_1 = r + s = \chi_i - t^a - r_i + r_i = \chi_i - t^a \tag{5}$$

Assume $r = 0, \ldots, T^r$, and the slack time $s = 0, \ldots, T^s$ and and that between t and $t+1$ the cars can only reduced r by one, because the rate of charge is fixed. Consider the following scenarios:

1. (**0.75 points**) The charging occurs as soon as the car plugs in. In this case in each time instant all cars $n_{\boldsymbol{x}}(t)$ charge and therefore:

$$\partial d_{(r,s),(r',s')}(t) = \begin{cases} n_{(r,s)}(t-1) & r' = r-1, s' = s,\ r > 0,\ r = 0, \ldots, T^r,\quad s = 0, \ldots, T^s \\ 0 & \text{else} \end{cases}.$$

   Explain why that is the case.

   Answer: The given equation models the number of cars that transition from state (r, s) to state (r', s') between times $t-1$ and t given the residual time to full charge r and the slack time s. Since the charging begins as soon as the car is plugged in, the cars can only reduce their time to charge by 1 between t and $t+1$ and their slack does not change.

   Therefore, the given equation is correct. It's effectively modeling the behavior of the charging system under the described assumptions, which results in an energy-first profile.

2. (**0.75 points**) The case where the load $p^d(t)$ profile is controlled and decided together with the schedule and commitment decisions considering what is the feasible aggregate load and selecting the best load profile. Explain why the action that can be taken $\partial d_{(r,s),(r',s')}(t)$ can only be within the set $\mathcal{U}_{\boldsymbol{x},\boldsymbol{x}'}$

   Answer: The action that can be taken $\partial d_{\boldsymbol{x},\boldsymbol{x}'}(t)$ represents the number of vehicles that transition from state $\boldsymbol{x} = (r, s)$ to state $\boldsymbol{x}' = (r', s')$ between times $t-1$ and t.

   The set $\mathcal{U}_{\boldsymbol{x},\boldsymbol{x}'}$ represents the feasible transitions from state $\boldsymbol{x}$ to another state $\boldsymbol{x}'$. Specifically, it's the set of states $\boldsymbol{x}'$ that are reachable from state $\boldsymbol{x}$ in one time step, considering the constraints of the system.

   From the previous answer, we have that these constraints are:

   - Fixed rate of charge $r' = r - 1$
   - the slack time s cannot decrease $s' = s$

   Thus, the action that can be taken $\partial d\boldsymbol{x}, \boldsymbol{x}'(t)$ can only be within the set $\mathcal{U}\boldsymbol{x}, \boldsymbol{x}'$ because this set represents the feasible transitions given the constraints of the system. Any action outside of this would violate the constraints and therefore it is not possible.