Optimal Decision Making - MGT-483

Professor: Daniel Kuhn

Energy Production

Economic Dispatch and Unit Commitment

Guillaume PARCHET (283294)
Kareen FALLAHA (343871)
Filip SLEZAK (286557)

May 20, 2022

Introduction

Electricity is key to the daily life of individuals at home, as well as to the production cycle of companies. However, future electricity demand is mostly uncertain, and the grid operators must supply enough electricity to meet its demand for smooth and continuous operations every day. Therefore, this report aims to find a safe dispatch schedule to generate electricity at optimized cost.

In the first section "Economic Dispatch", we will determine the amount of electricity that each generator should produce in each hour to meet the demand at minimal cost. The main idea behind this section is to utilize as much renewable energy sources as possible, for the sake of the environment and to meet sustainability standards. Any electricity demand above the potential of renewable sources can be covered by traditional sources, such as thermal or hydro-power energy. In the second section "Unit Commitment", we will go a step further to determine the schedule, i.e. start-up and shut-down time of the traditional generators. These actions introduce new costs to our problem, which should be minimized.

Note: The python code used to derive all the presented results can be found in the Appendix and will also be provided separately. The exhaustive display of the optimal dispatch plan values $\{g_i^t\}_{t\in\mathcal{T}}$ can also be found there.

1 Economic Dispatch Problem

1.1 Initial optimal dispatch plan

In this problem, there are 6 electricity generators, and the goal is to find the optimal dispatch plan for electricity generation, with the objective of minimizing the variable generation cost of these generators.

In order to do that, we first initialize the following variables with the given values per hour:

- r_t: power produced by the renewable energy sources (e.g. solar power plants and wind farms)
- d_t: total electricity demand

We also initialize the generation cost C_i^g, the capacity $\bar{g}_i$, ramp-up rate R_i^up and ramp-down rate R_i^dn.

In the next step, we run the following optimization algorithm using the simplex method (GLPK solver in Python):

$$
\begin{aligned}
\min_{\mathbf{g}} \quad & \sum_{t=1}^{T}\sum_{i=1}^{N} C_i^g g_i^t & \\
\text{s.t.} \quad & d^t = \sum_{i\in\mathcal{N}_g} g_i^t + r^t & \forall t \in \mathcal{T} \\
& 0 \le g_i^t \le \bar{g}_i & \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \\
& g_i^t - g_i^{t-1} \le R_i^{up} & \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T}\backslash\{1\} \\
& g_i^{t-1} - g_i^t \le R_i^{dn} & \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T}\backslash\{1\}
\end{aligned}
$$

The optimal dispatch plan shows that we can achieve a minimal cost of \$10'158.50 using the 6 generators for a total of 576.9MW, which completes the supply from renewable sources to match demand as seen in

Table 1:

	G1	G2	G3	G4	G5	G6	**Total**
Power produced (MW)	177.6	64.2	160.9	127.4	15.8	31	**576.9**
Generation costs ($/MW)	15	20	15	20	30	25	-
Total cost per generator ($)	2,664	1,284	2,413.5	2,548	474	775	**10,158.5**

Table 1: Optimal values for power generation in Economic dispatch problem

We can better visualize the generators used the most by plotting the optimal decisions $\{g_i^t\}_{t\in\mathcal{T}}$ in a line graph (Figure 1) or by aggregating total usage (Figure 2).

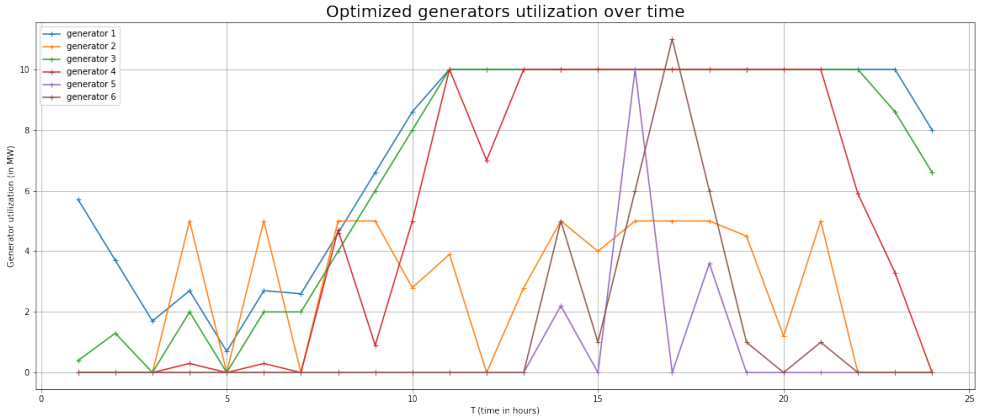


Figure 1: Generator utilization (in MW)

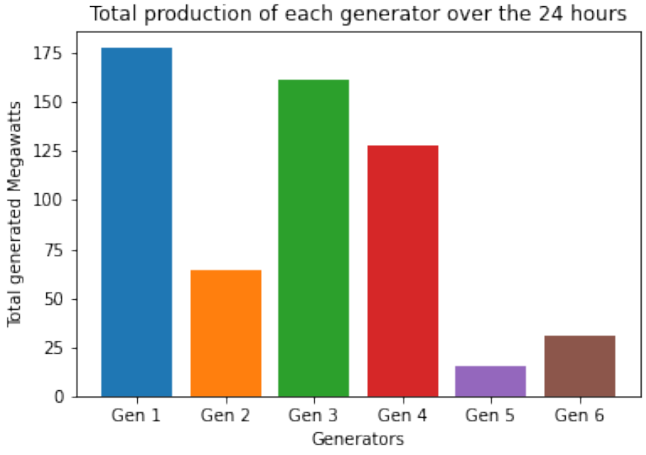


Figure 2: Total generator utilization (in MW)

1.2 Enhanced dispatch model: Lithium battery

In this section, we add a lithium battery as storage device to be charged when energy production is cheap and discharged when energy production is expensive. The characteristics of the battery are:

- Capacity $\bar{S} = 20$ MW
- Charging efficiency $\eta^c = 0.95$
- Discharging efficiency $\eta^d = 0.92$

The assumptions needed to initialize the problem are that the battery is empty at $t = 1$ and $t = T + 1$.

The new enhanced economic dispatch model can be formulated as follows:

$$
\begin{aligned}
\min_{\mathbf{g}} \quad & \sum_{t=1}^{T}\sum_{i=1}^{N} C_i^g g_i^t & \\
\text{s.t.} \quad & d^t = \sum_{i\in\mathcal{N}_g} g_i^t + r^t + \eta_d.b_d^t - \eta_c.b_c^t & \forall t \in \mathcal{T} \\
& 0 \le g_i^t \le \bar{g}_i & \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \\
& g_i^t - g_i^{t-1} \le R_i^{up} & \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T}\backslash\{1\} \\
& g_i^{t-1} - g_i^t \le R_i^{dn} & \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T}\backslash\{1\} \\
& S^{t+1} = S^t - b_d^t + \eta_c.b_c^t & \forall t \in \mathcal{T} \\
& b_d^t \le S^t + \eta_c.b_c^t & \forall t \in \mathcal{T} \\
& b_c^t \le (\bar{S} - S^t) + b_d^t & \forall t \in \mathcal{T} \\
& 0 \le S^t \le \bar{S} & \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T}\backslash\{1\} \\
& 0 \le b_d^t & \forall t \in \mathcal{T} \\
& 0 \le b_c^t & \forall t \in \mathcal{T} \\
& (S^1 = 0) & \\
& (S^{T+1} = 0) &
\end{aligned}
$$

1.3 Enhanced optimal dispatch plan: Lithium battery

By adding the storage device, the optimal cost lowers to 9'777.75\$ which represents savings of 3.75%.

We can visualize the usage of the generators in Figure 3 and Figure 4 as well as the battery usage in Figure 5. Again, all exact values can be found in Appendix.

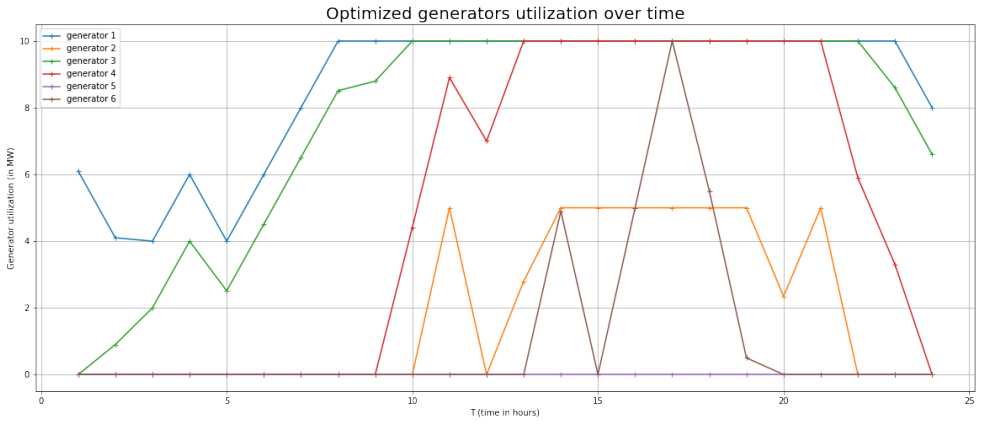


Figure 3: Generator utilization (in MW)

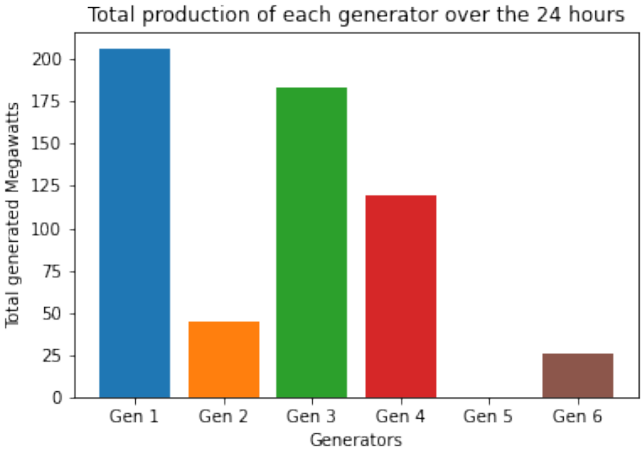


Figure 4: Total generator utilization (in MW)

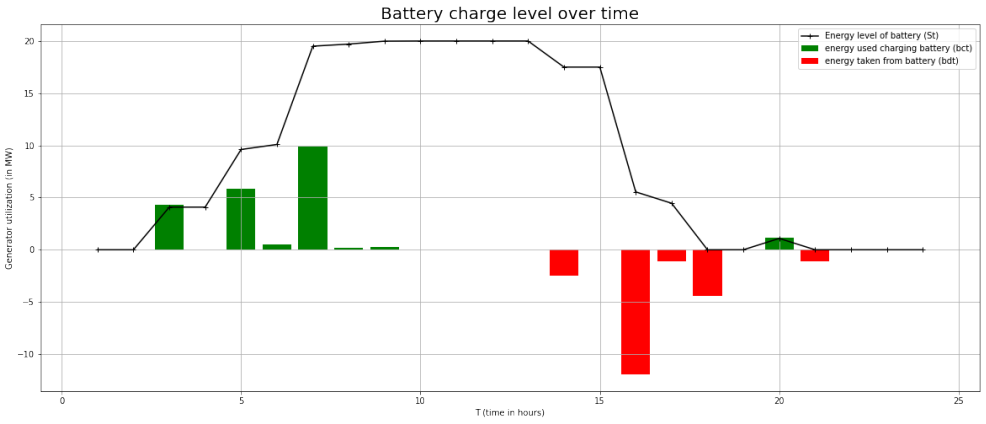


Figure 5: Storage device charging and discharging

With the storage device, the system now takes advantage of the hours with lots of renewable energy available and low demand to fill up the battery (for example at t=5 or t=7, one can find the full renewable energy table in Table2 of Handout). It can later on use this additional "free energy" when there are low renewable levels but high demand (for example at t=16), allowing to limit the activation of higher cost generators. Apart from this, we can see the system still prioritizes using generators 1 and 3 (as they are cheaper).

2 Unit Commitment

In this section, we will enhance the economic dispatch model by accounting for the start-up and shut-down costs, for the fixed costs of production, as well as for the minimum up-times and down-times of the generators.
The formulation of the model is now as follows:

$$\min_{\mathbf{g}} \quad \sum_{t=1}^{T}\sum_{i=1}^{N} C_i^u u_i^t + C_i^d v_i^t + C_i^n x_i^t + C_i^g g_i^t \tag{2a}$$

$$\text{s.t.} \quad d^t = \sum_{i\in\mathcal{N}_g} g_i^t + r^t \qquad \forall t \in \mathcal{T} \tag{2b}$$

$$0 \le g_i^t \le \bar{\bar{g}}_i x_i^t \qquad \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \tag{2c}$$

$$x_i^{t-1} - x_i^t + u_i^t \ge 0 \qquad \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T}\backslash\{1\} \tag{2d}$$

$$x_i^t - x_i^{t-1} + v_i^t \ge 0 \qquad \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T}\backslash\{1\} \tag{2e}$$

$$x_i^t - x_i^{t-1} \le x_i^\tau \qquad \forall i \in \mathcal{N}_g, \forall \tau \in \{t+1, \min\{t+T_i^{up}, T\}\}, \forall t \in \mathcal{T}\backslash\{1,T\} \tag{2f}$$

$$x_i^{t-1} - x_i^t \le 1 - x_i^\tau \qquad \forall i \in \mathcal{N}_g, \forall \tau \in \{t+1, \min\{t+T_i^{down}, T\}\}, \forall t \in \mathcal{T}\backslash\{1,T\} \tag{2g}$$

$$x_i^t,\ u_i^t,\ v_i^t \in 0,1 \qquad \forall t \in \mathcal{T}\backslash\{1,T\} \tag{2h}$$

2.1 Constraints

The constraints (2c)-(2g) have the following meaning:

- (2c) indicates that the energy produced by each generator i is always positive, and is upper bounded by the production capacity $\bar{g}_i$ when the generator is on ($x_i^t = 1$) and is zero when it is off ($x_i^t = 0$).

$$0 \le g_i^t \le \bar{g}_i x_i^t \qquad \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T}$$

- (2d) & (2e) both limit the state of the generator at time t compared to the previous state and based on the decision of turning the generator on (u_i^t) or off (v_i^t). In other words, these conditions prevent the "impossible states" such as turning off a generator that is off, and turning on one that is on:
  - if the generator was off at $t-1$ and still off at time t : $x_i^t = x_i^{t-1} = 0$
  - if the generator was off and turned on: $x_i^t = x_i^{t-1} + u_i^t = 1$
  - if the generator was on and no action performed: $x_i^t = x_i^{t-1} = 1$
  - if the generator was on then turned off $x_i^t = 0$ and $x_i^{t-1} = 1$, where both conditions are satisfied:
    (2d)

$$\begin{aligned} & x_i^{t-1} - x_i^t + u_i^t \ge 0 && \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T}\backslash\{1\} \\ \equiv \quad & x_i^t \le x_i^{t-1} + u_i^t && \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T}\backslash\{1\} \end{aligned}$$

and (2e)

$$\begin{aligned} & x_i^t - x_i^{t-1} + v_i^t \geq 0 && \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \backslash \{1\} \\ \equiv \quad & x_i^t \geq x_i^{t-1} - v_i^t && \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \backslash \{1\} \end{aligned} \tag{1}$$

- (2f) enforces that the generator is on for at least T_{up} between time $t+1$ and the minimum between $t+T_{up}$ and T. Similarly, (2g) enforces that the generator is off for at least T_{down} between time $t+1$ and the minimum between $t+T_{down}$ and T.

$$\begin{aligned} x_i^t - x_i^{t-1} &\leq x_i^\tau && \forall i \in \mathcal{N}_g, \forall \tau \in \{t+1, \min\{t+T_i^{up}, T\}\}, \forall t \in \mathcal{T} \backslash \{1, T\} \\ x_i^{t-1} - x_i^t &\leq 1 - x_i^\tau && \forall i \in \mathcal{N}_g, \forall \tau \in \{t+1, \min\{t+T_i^{down}, T\}\}, \forall t \in \mathcal{T} \backslash \{1, T\} \end{aligned}$$

  This means that if there was a change from off at $t-1$ to on at t, then the generator cannot be turned off between $t+1$ and $t+T_i^{up}$, in order to have a minimum up time of T_i^{up}.
  In the case where there was a change from on to off, then this condition also prohibits turning the generator back on between $t+1$ and $t+T_i^{down}$, to allow a minimum downtime of T_i^{down}.

2.2 Implementation and Optimal Decisions

When accounting Unit Commitment costs, the new minimum of the system is 11'024.50$.

Again, let's visualize the usage of the generators in Figure 6 and Figure 7. Additionally, we can see in Figure 8 that due to the newly introduced costs, it seems we now have an incentive to avoid turning on and turning off generators (occurs only 3 times and 2 times respectively).

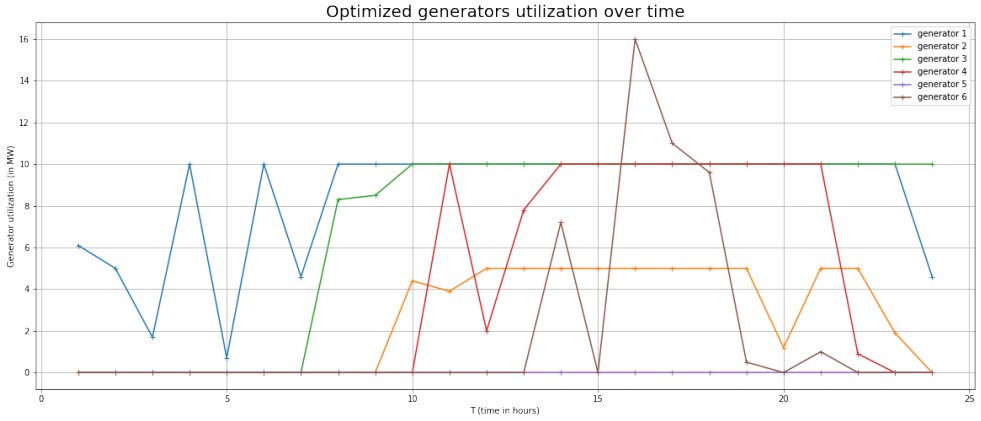


Figure 6: Generator production (in MW)

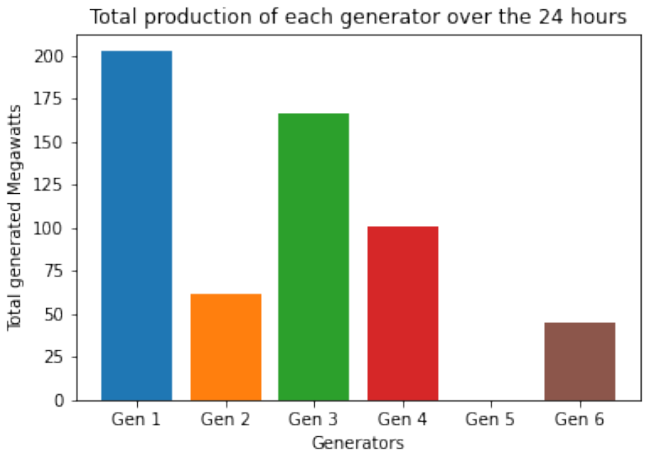


Figure 7: Total generator utilization (in MW)

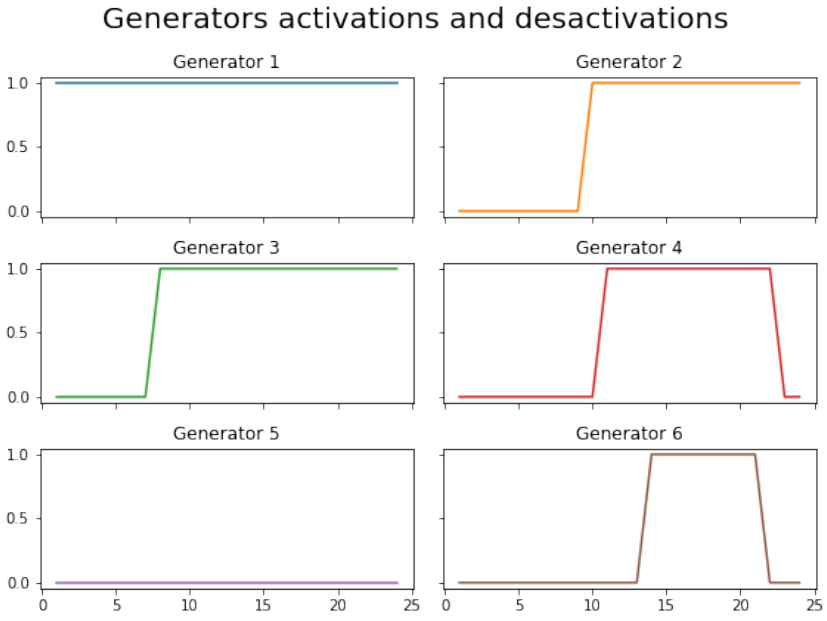


Figure 8: Storage device charging and discharging

2.3 Amendment to continuous variables

In this section, we change u_i^t and v_i^t from discrete to continuous variables between 0 and 1. All results and graphs are identical, reaching the same minimum cost of \$11'024.50.
Figure 6, Figure 7 and Figure 8 are still valid in this part.

The run-time in part 2.2 is 0.00934 seconds, while the run-time in part 2.3 is 0.00757 seconds. Enforcing the binary constraints is more complex than letting the variables have free value in interval [0,1]. Solving this Mixed Integer Linear Program requires to use a solver that is more complex than the ones we used for basic Linear Program to make sure our variables are integers. This in turns leads to a longer run-time.

As a recap of both sections 2.2 and 2.3, the power generation and cost for each generator is summarized in Table 2:

	G1	G2	G3	G4	G5	G6	**Total**
Total power generated (MW)	199.5	54.2	170.	107.9	0.	45.3	**576.9**
Total cost per generator ($)	3232.5	1259	2795	2388	0	1350	**11,024.5**

Table 2: Optimal values for power generation in deterministic unit commitment problem

2.4 Comparison and Theory

Theoretically, to check if this makes sense, we look into the two conditions related to u_i^t and v_i^t which are:

$$x_i^{t-1} - x_i^t + u_i^t \geq 0 \qquad \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \backslash \{1\} \quad \text{(2d)}$$
$$x_i^t - x_i^{t-1} + v_i^t \geq 0 \qquad \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \backslash \{1\} \quad \text{(2e)}$$

Respectively, the two conditions can be rewritten as:

$$u_i^t \geq x_i^t - x_i^{t-1} \qquad \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \backslash \{1\} \quad \text{(2d)}$$
$$v_i^t \geq x_i^{t-1} - x_i^t \qquad \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \backslash \{1\} \quad \text{(2e)}$$

Putting the continuous variable g_i^t aside, the previously integer problem (IP) having relaxed constraints is now the mixed integer linear problem (MILP). Therefore, the feasible set of the MILP is larger than the feasible set of the IP, which leads to an optimal value of the MILP better or equal to the optimal value of the IP. If the relaxed formulation is 'good' then the relaxed problem's feasible set is the convex hull containing the integers of interest at its extreme points. Since an LP solution lies on an extreme point of the feasible set, the variable in question will be an integer.

Since the constraints on u_i^t and v_i^t are now relaxed, then the feasible values of u_i^t and v_i^t are continuous and can fall within the range [0,1]:

$$0 \leq u_i^t \leq 1$$
$$0 \leq v_i^t \leq 1$$

Additionally, since the objective function is monotonically increasing in u_i^t and v_i^t, then the lowest value of the possible range will be the optimal solution. Therefore, the MILP will choose an optimal value to be on the boundaries of the feasible set, which will be either 0 or 1.

Looking at the possible values of x_i^t and x_i^{t-1} which are still binary, the difference between the two values will either be 0, 1 or -1. To view all feasible values of u_i^t and v_i^t, we perform a case by case analysis based on the constraints (2d) and (2e) as seen in Table 3:

x_i^t	x_i^{t-1}	u_i^t	v_i^t
0	0	≥ 0	≥ 0
0	1	≥ -1	≥ 1
1	0	≥ 1	≥ -1
1	1	≥ 0	≥ 0

Table 3: Feasible values of u_i^t and v_i^t with possible combinations of generator states

For scenarios where u_i^t and $v_i^t \geq 1$, the optimal value would be 1.
For the other scenarios where u_i^t, v_i^t are either ≥ 0 or ≥ -1, the optimal value would be 0.

3 Robust Unit Commitment

3.1

The presented problem has three factors that make its numerical resolution complex:

- not possible to enumerate all constraints implied by (3b) and (3c) as we have $g_i^t \ \forall r \in \mathcal{R}$
- potentially non-linear objective and constraints (3b) and (3c) as $g_i^t(r^1, ..., r^t)$ can be non-linear
- non-linear objective function (min-max over $\mathbf{g}(.)$ and $\mathbf{r}$)

3.2

With the approximate linear decision rules with limited memory, we can now rewrite the system with finitely many decision variables:

$$\min_{\mathbf{x},\ \mathbf{u},\ \mathbf{v}} \sum_{t=1}^{T}\sum_{i=1}^{N} C_i^u u_i^t + C_i^d v_i^t + C_i^n x_i^t + \min_{\mathbf{g(.)}} \max_{r\in\mathcal{R}} [\ \sum_{i=1}^{N} C_i^g.(a_i^1 r^1 + c_i^1) + \sum_{t=2}^{T}\sum_{i=1}^{N} C_i^g.(a_i^t r^t + b_i^t r^{t-1} + c_i^t)\ ]$$

$$\begin{aligned}
\text{s.t.} \quad & d^t = r^t + \sum_{i\in\mathcal{N}_g} (a_i^t r^t + b_i^t r^{t-1} + c_i^t) && \forall r \in \mathcal{R}, \forall t \in \mathcal{T}\backslash\{1\} \\
& d^1 = r^1 + \sum_{i\in\mathcal{N}_g} (a_i^1 r^1 + c_i^1) && \forall r \in \mathcal{R} \\
& 0 \le a_i^t r^t + b_i^t r^{t-1} + c_i^t \le \bar{g}_i x_i^t && \forall r \in \mathcal{R}, \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T}\backslash\{1\} \\
& 0 \le a_i^1 r^1 + c_i^1 \le \bar{g}_i x_i^1 && \forall r \in \mathcal{R}, \forall i \in \mathcal{N}_g \\
& \text{(2d)-(2h)}
\end{aligned}$$

Note we can equivalently re-write the objective function as ($r^0 = 0$ for simplicity of notation):

$$\min_{\mathbf{x},\ \mathbf{u},\ \mathbf{v}} \sum_{t=1}^{T}\sum_{i=1}^{N} C_i^u u_i^t + C_i^d v_i^t + C_i^n x_i^t + \min_{\mathbf{g(.)}} \max_{r\in\mathcal{R}} [\ \sum_{t=1}^{T}\sum_{i=1}^{N} C_i^g.(a_i^t r^t + b_i^t r^{t-1} + c_i^t)\ ]$$

$$\text{s.t.} \quad b_i^1 = 0 \qquad \forall i \in \mathcal{N}_g$$

We can also further simplify this formulation using the first Hint. Indeed, one can see constraint (3b) is equivalent to a set of three linear constraints:

$$\begin{aligned}
& d^t = r^t + \sum_{i\in\mathcal{N}_g} (a_i^t r^t + b_i^t r^{t-1} + c_i^t) && \forall r \in \mathcal{R}, \forall t \in \mathcal{T} \\
\Leftrightarrow \quad & d^t = r^t + r^t.\sum_{i\in\mathcal{N}_g} a_i^t + r^{t-1}.\sum_{i\in\mathcal{N}_g} b_i^t + \sum_{i\in\mathcal{N}_g} c_i^t && \forall r \in \mathcal{R}, \forall t \in \mathcal{T} \\
\Leftrightarrow \quad & \sum_{i\in\mathcal{N}_g} c_i^t = d^t \quad \text{and} \quad \sum_{i\in\mathcal{N}_g} b_i^t = 0 \quad \text{and} \quad \sum_{i\in\mathcal{N}_g} a_i^t = -1 && \forall t \in \mathcal{T}
\end{aligned}$$

Furthermore, let's replace the $\forall r \in \mathcal{R}$ in the problem's conditions by adding two more constraints to

ensure all $r \in \mathbb{R}^T : |r^t - \bar{r}^t| \leq \hat{r}^t \; \forall t \in \mathcal{T}$:

$$\begin{aligned} r^t &\leq \bar{r}^t + \hat{r}^t \qquad \forall t \in \mathcal{T} \\ r^t &\geq \bar{r}^t - \hat{r}^t \qquad \forall t \in \mathcal{T} \end{aligned}$$

3.3

To dualize the maximization problem over r, let's first check which parts of the initial problem are dependent on $\vec{r}$:

- Constraint (3b) depends on $\vec{r}$ but we have replaced it by the three linear constraints detailed in question 3.2, those are not dependant on $\vec{r}$ therefore we do not need to consider them.
- Constraint (3c) also depends on $\vec{r}$ but will be further dealt with in question 3.4, we can omit it for now as it does not impact on the solution of the objective dual.
- The two constraints $r^t \leq \bar{r}^t + \hat{r}^t$ and $r^t \geq \bar{r}^t - \hat{r}^t$ derived in question 3.2 depend on $\vec{r}$ and need to be included.
- As $\vec{r}$ represents the level of renewable energy at time t, we also add the condition $r^t \geq 0 \; \forall t \in \mathcal{T}$ as it would not make sense to observe negative energy generation.

Lastly let's rewrite the objective function to make the coefficients c_i^t of the dual problem more distinct:

$$\begin{aligned} &\max_{\mathbf{r}} \sum_{t=1}^{T} \sum_{i=1}^{N} C_i^g.(a_i^t r^t + b_i^t r^{t-1} + c_i^t) \\ &= \max_{\mathbf{r}} r^0 \sum_i C_i^g b_i^1 + r^1 \sum_i C_i^g.(a_i^1 + b_i^2) + ... + r^{T-1} \sum_i C_i^g.(a_i^{T-1} + b_i^T) + r^T \sum_i C_i^g a_i^T \end{aligned}$$

Where the first term $r^0 \sum_i C_i^g b_i^1$ is equal to 0 as we added the constraint $b_i^1 = 0 \; \forall i \in \mathcal{N}_g$. We also brought back the term $\sum_{t=1}^{T} \sum_{i=1}^{N} C_i^g c_i^t$ into the original problem as it does not depend on $\vec{\mathbf{r}}$.

Let's summarize all these elements into the following sub-problem primal:

$$\begin{aligned} \max_{\mathbf{r}} \; & r^1 \sum_i C_i^g.(a_i^1 + b_i^2) + ... + r^{T-1} \sum_i C_i^g.(a_i^{T-1} + b_i^T) + r^T \sum_i C_i^g.a_i^T \\ \text{s.t.} \quad & r^t \leq \bar{r}^t + \hat{r}^t \qquad \forall t \in \mathcal{T} \\ & -r^t \leq \hat{r}^t - \bar{r}^t \qquad \forall t \in \mathcal{T} \\ & \vec{\mathbf{r}} \geq \vec{\mathbf{0}} \end{aligned}$$

We can now compute the corresponding sub-problem dual:

$$\begin{aligned}
\min_{\mathbf{p}} \quad & \sum_{t=1}^{T}[(\bar{r}^t + \hat{r}^t)p_{1,t} + (\hat{r}^t - \bar{r}^t)p_{2,t}] \\
\text{s.t.} \quad & p_{1,t} - p_{2,t} \geq \sum_i C_i^g.(a_i^t + b_i^{t+1}) && \forall t \in \mathcal{T}\backslash\{T\} \\
& p_{1,T} - p_{2,T} \geq \sum_i C_i^g.a_i^T \\
& p_{1,t},\ p_{2,t} \geq 0 && \forall t \in \mathcal{T}
\end{aligned}$$

By merging back the dual of the maximization "adversarial" renewable energy problem into the original problem, we get the following global problem:

$$\begin{aligned}
\min_{\mathbf{x,\ u,\ v,\ p,\ g(.)}} \quad & \sum_{t=1}^{T}\sum_{i=1}^{N}[C_i^u u_i^t + C_i^d v_i^t + C_i^n x_i^t + C_i^g c_i^t] + \sum_{t=1}^{T}[(\bar{r}^t + \hat{r}^t)p_{1,t} + (\hat{r}^t - \bar{r}^t)p_{2,t}] \\
\text{s.t.} \quad & 0 \leq a_i^t r^t + b_i^t r^{t-1} + c_i^t \leq \bar{g}_i x_i^t && \forall r \in \mathcal{R}, \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \\
& \sum_{i \in \mathcal{N}_g} c_i^t = d^t && \forall t \in \mathcal{T} \\
& \sum_{i \in \mathcal{N}_g} b_i^t = 0 && \forall t \in \mathcal{T} \\
& \sum_{i \in \mathcal{N}_g} a_i^t = -1 && \forall t \in \mathcal{T} \\
& p_{1,t} - p_{2,t} \geq \sum_i C_i^g.(a_i^t + b_i^{t+1}) && \forall t \in \mathcal{T}\backslash\{T\} \\
& p_{1,T} - p_{2,T} \geq \sum_i C_i^g.a_i^T \\
& p_{1,t},\ p_{2,t} \geq 0 && \forall t \in \mathcal{T} \\
& b_i^1 = 0 && \forall i \in \mathcal{N}_g \\
& \text{(2d)-(2h)}
\end{aligned}$$

3.4

We have already reformulated constraint (3b) in question 3.2 (transformed it into three linear constraints that do not depend on $\vec{r}$).

As indicated in the Hint, we will now use duality to rewrite constraint (3c). First, let's separate the two constraints:

$$\begin{aligned}
& 0 \leq a_i^t r^t + b_i^t r^{t-1} + c_i^t && \forall r \in \mathcal{R}, \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \\
& a_i^t r^t + b_i^t r^{t-1} + c_i^t \leq \bar{g}_i x_i^t && \forall r \in \mathcal{R}, \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T}
\end{aligned}$$

Which is equivalent to the two systems:

$$\begin{aligned} -c_i^t &\leq \min_{\mathbf{x}\in\mathbb{R}^2: Cx\leq d} a_i^t r^t + b_i^t r^{t-1} && \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \\ c_i^t - \bar{g}_i x_i^t &\leq \min_{\mathbf{x}\in\mathbb{R}^2: Cx\leq d} -a_i^t r^t - b_i^t r^{t-1} && \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \end{aligned}$$

where $Cx \leq d$ are the conditions on r^t and r^{t-1}. Note this is indeed valid $\forall t \in \mathcal{T}$ as when $t = 0$ we have the condition $b_i^t = 0$ (so r^0 does not matter). Let's detail the primal of the first condition on fixed arbitrary i and t:

$$\begin{aligned} \min_{\mathbf{r}} \quad & a_i^t r^t + b_i^t r^{t-1} \\ \text{s.t.} \quad & r^t \leq \bar{r}^t + \hat{r}^t \\ & -r^t \leq \hat{r}^t - \bar{r}^t \\ & r^{t-1} \leq \bar{r}^{t-1} + \hat{r}^{t-1} \\ & -r^{t-1} \leq \hat{r}^{t-1} - \bar{r}^{t-1} \\ & r^t, r^{t-1} \geq 0 \end{aligned}$$

We can dualize this system similarly to what we did in question 3.3:

$$\begin{aligned} \max_{\mathbf{y}} \quad & (\bar{r}^t + \hat{r}^t) y_{1,i,t} + (\hat{r}^t - \bar{r}^t) y_{2,i,t} + (\bar{r}^{t-1} + \hat{r}^{t-1}) y_{3,i,t} + (\hat{r}^{t-1} - \bar{r}^{t-1}) y_{4,i,t} \\ \text{s.t.} \quad & y_{1,i,t} - y_{2,i,t} \leq a_i^t \\ & y_{3,i,t} - y_{4,i,t} \leq b_i^t \\ & y_{1,i,t}, y_{2,i,t}, y_{3,i,t}, y_{4,i,t} \leq 0 \end{aligned}$$

Finally, using the given Hint for 3.4, we can rewrite the equivalent conditions in the original problem:

$$\begin{aligned} -c_i^t \leq (\bar{r}^t + \hat{r}^t) y_{1,i,t} + (\hat{r}^t - \bar{r}^t) y_{2,i,t} + (\bar{r}^{t-1} + \hat{r}^{t-1}) y_{3,i,t} + (\hat{r}^{t-1} - \bar{r}^{t-1}) y_{4,i,t} && \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \\ y_{1,i,t} - y_{2,i,t} \leq a_i^t && \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \\ y_{3,i,t} - y_{4,i,t} \leq b_i^t && \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \\ y_{1,i,t}, y_{2,i,t}, y_{3,i,t}, y_{4,i,t} \leq 0 && \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \end{aligned}$$

By following the same procedure for the second condition, we get the additional conditions from the dual:

$$\begin{aligned} c_i^t - \bar{g}_i x_i^t \leq (\bar{r}^t + \hat{r}^t) y_{5,i,t} + (\hat{r}^t - \bar{r}^t) y_{6,i,t} + (\bar{r}^{t-1} + \hat{r}^{t-1}) y_{7,i,t} + (\hat{r}^{t-1} - \bar{r}^{t-1}) y_{8,i,t} && \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \\ y_{5,i,t} - y_{6,i,t} \leq -a_i^t && \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \\ y_{7,i,t} - y_{8,i,t} \leq -b_i^t && \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \\ y_{5,i,t}, y_{6,i,t}, y_{7,i,t}, y_{8,i,t} \leq 0 && \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \end{aligned}$$

Note that for simplicity of notation, we will assume $\bar{r}^0 = 0$ and $\hat{r}^0 = 0$. Otherwise, this is equivalent to add the conditions for special cases $t = 1$. (i.e. $c_i^1 - \bar{g}_i x_i^1 \leq (\bar{r}^1 + \hat{r}^1) y_{5,i,1} + (\hat{r}^1 - \bar{r}^1) y_{6,i,1} \quad \forall i \in \mathcal{N}_g$).

3.5

We are now able to summarize the full mixed-integer linear problem as follows:

$$\min_{\mathbf{x,\ u,\ v,\ p,\ y,\ g(.)}} \sum_{t=1}^{T}\sum_{i=1}^{N}[C_i^u u_i^t + C_i^d v_i^t + C_i^n x_i^t + C_i^g c_i^t] + \sum_{t=1}^{T}[(\bar{r}^t + \hat{r}^t)p_{1,t} + (\hat{r}^t - \bar{r}^t)p_{2,t}]$$

$$\begin{aligned}
\text{s.t.}\quad & -c_i^t \le (\bar{r}^t + \hat{r}^t)y_{1,i,t} + (\hat{r}^t - \bar{r}^t)y_{2,i,t} + (\bar{r}^{t-1} + \hat{r}^{t-1})y_{3,i,t} + (\hat{r}^{t-1} - \bar{r}^{t-1})y_{4,i,t} && \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \\
& c_i^t - \bar{g}_i x_i^t \le (\bar{r}^t + \hat{r}^t)y_{5,i,t} + (\hat{r}^t - \bar{r}^t)y_{6,i,t} + (\bar{r}^{t-1} + \hat{r}^{t-1})y_{7,i,t} + (\hat{r}^{t-1} - \bar{r}^{t-1})y_{8,i,t} && \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \\
& y_{1,i,t} - y_{2,i,t} \le a_i^t && \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \\
& y_{3,i,t} - y_{4,i,t} \le b_i^t && \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \\
& y_{5,i,t} - y_{6,i,t} \le -a_i^t && \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \\
& y_{7,i,t} - y_{8,i,t} \le -b_i^t && \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \\
& \sum_{i \in \mathcal{N}_g} c_i^t = d^t && \forall t \in \mathcal{T} \\
& \sum_{i \in \mathcal{N}_g} b_i^t = 0 && \forall t \in \mathcal{T} \\
& \sum_{i \in \mathcal{N}_g} a_i^t = -1 && \forall t \in \mathcal{T} \\
& p_{1,t} - p_{2,t} \ge \sum_i C_i^g.(a_i^t + b_i^{t+1}) && \forall t \in \mathcal{T}\backslash\{T\} \\
& p_{1,T} - p_{2,T} \ge \sum_i C_i^g.a_i^T \\
& y_{n,i,t} \le 0 \qquad \forall n \in \{1,2,...,8\}, \forall i \in \mathcal{N}_g, \forall t \in \mathcal{T} \\
& p_{1,t},\ p_{2,t} \ge 0 && \forall t \in \mathcal{T} \\
& b_i^1 = 0 && \forall i \in \mathcal{N}_g \\
& \text{(2d)-(2h)}
\end{aligned}$$

To solve the robust unit commitment problem, we implemented the full mixed-integer linear problem. The solver outputs an optimal cost of \$11'337 which is higher than the cost of \$11'024.5 we had in the deterministic unit commitment problem. This result was expected since the robust unit commitment problem translates into a worst case scenario problem (lowering risk when accounting renewable energy levels are uncertain). As a side note, setting the uncertainty on r^t to 0 instead of 0.6 brings us back to Problem 2 and yields the same result. This cross-check means the formulation is likely correct.

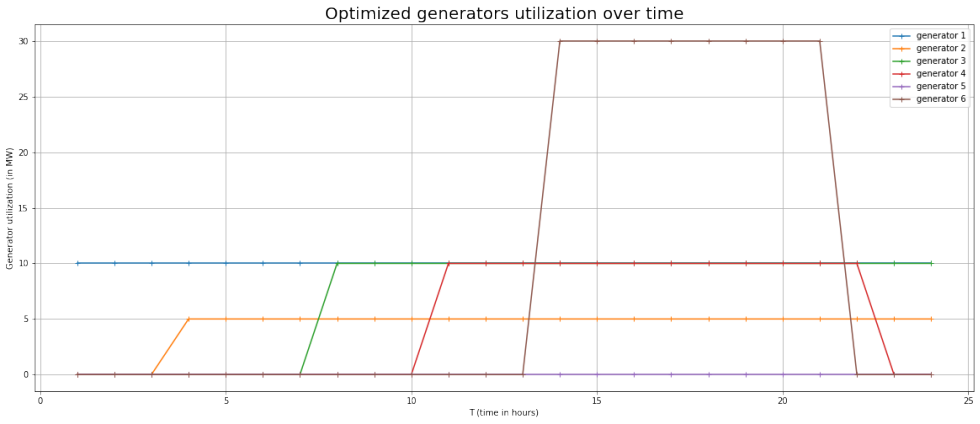


Figure 9: Generator utilization (in MW)

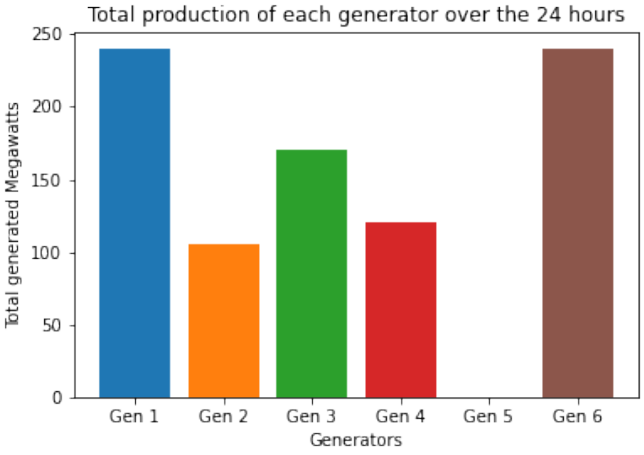


Figure 10: Total generator utilization (in MW)

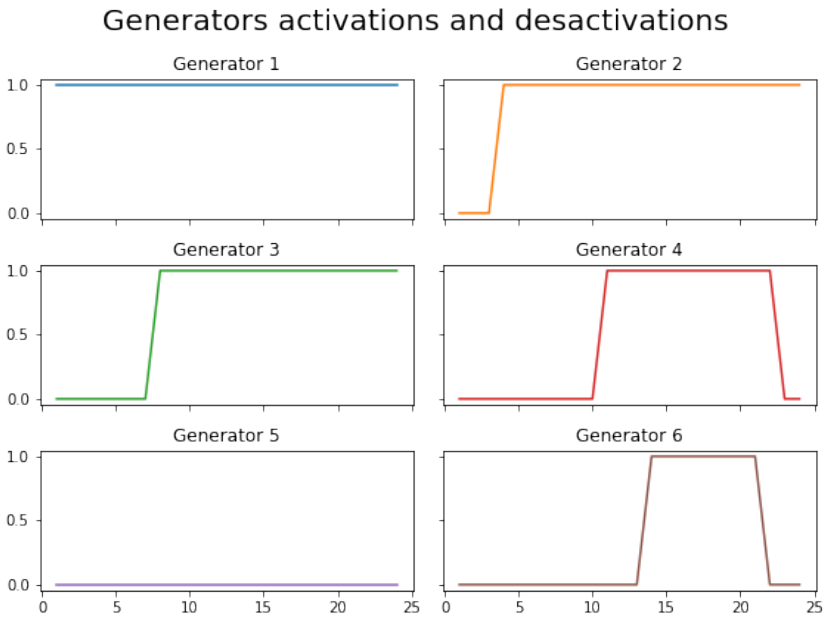


Figure 11: Storage device charging and discharging

Comparing generator states **x**, we note that the start-up and start-off timings of all generators are the same, except for Generator 2 which is turned on earlier than before.
This behavior can be compared between the deterministic unit commitment problem in Figure 8 where (t=8) and the robust unit commitment problem in Figure 11 where (t=4).
The main reason for which the algorithm first chose Generator 2 to provide additional energy is mostly due to its lower **generation cost * capacity** product ($100 for G2 vs. $150+ for the generators).

3.6

Currently we approximate the functional decisions by linear decision rules with quite limited memory. Indeed we decided the output of generator i at time t only based on the two renewable energy generations r^t and r^{t-1}. Adding more memory would result in better approximations of the functional decisions at the cost of increased complexity when solving the system (more decisions variables). Note we could estimate the best trade-off between accuracy and computation power required.

Let's take as an example the case we approximate the functional decisions with 4-memory:

$$\begin{aligned} g_i^1(r^1) &= a_i^1 r^1 + e_i^1 \\ g_i^2(r^1, r^2) &= a_i^2 r^2 + b_i^2 r^1 + e_i^2 \\ g_i^3(r^1, r^2, r^3) &= a_i^3 r^3 + b_i^3 r^2 + c_i^3 r^1 + e_i^3 \\ g_i^t(r^1, ..., r^t) &= a_i^t r^t + b_i^t r^{t-1} + c_i^t r^{t-2} + d_i^t r^{t-3} + e_i^t \qquad \forall t \in \mathcal{T} \backslash \{1, 2, 3\} \end{aligned}$$

Alternatively, we could try to forecast future energy production, base our linear approximation on r^{t+1} as well as r^{t-1}, the former could be based weather forecasts.

Problem1

May 20, 2022

1 APPENDIX

2 Problem 1 - Economic Dispatch

[24]:
```
import numpy as np
import cvxpy as cp

import matplotlib.pyplot as plt
```

3 Definition of data

[6]:
```
generation_costs = np.array([15, 20, 15, 20, 30, 25.])
capacity = np.array([10, 5, 10, 10, 20, 30.])
ramp_up_rate = np.array([2, 5, 2, 5, 10, 5.])
ramp_down_rate = np.array([2, 5, 2, 5, 10, 5.])
```

[7]:
```
rt = np.array([15.2, 16.4, 16.1, 10.9, 14.8, 7.6, 15.6, 5.5, 9.2, 5.7, 1.5, 12.
 ↪4, 10.4, 4.8, 14.3, 0.5, 6.6, 5.7, 11.5, 11.9, 2.8, 7.3, 6.7, 9.7])
dt = np.array([21.3, 21.4, 17.8, 20.9, 15.5, 17.6, 20.2, 23.8, 27.7, 30.1, 35.
 ↪4, 39.4, 43.2, 47.0, 49.3, 51.5, 52.6, 50.3, 47.0, 43.1, 38.8, 33.2, 28.6,␣
 ↪24.3])
```

4 Question 1.1

[8]:
```
# YOUR CODE HERE
# We recommend GLPK solver
Ng = 6  # number of traditional generators
T = 24  # planning horizon
# Define variables
gt = cp.Variable((Ng, T))  # production of each traditional generator
                           # over the planning horizon
# Construct the problem
```

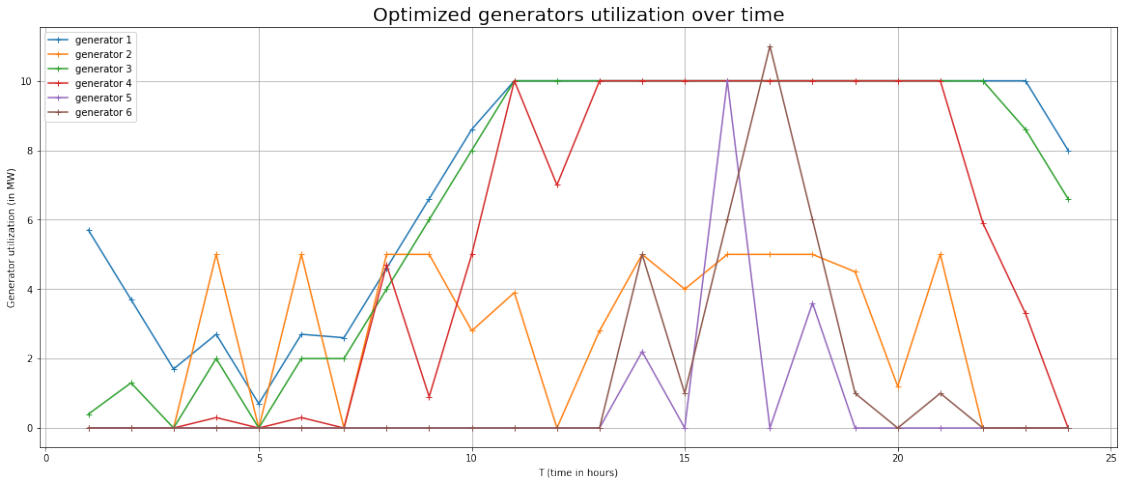
```
objective = cp.Minimize(cp.sum(generation_costs @ gt))
constraints = [dt == cp.sum(gt, axis=0) + rt,
               0 <= gt, gt <= np.tile(capacity, (T, 1)).T,
               gt[:, 1:] - gt[:, :-1] <= np.tile(ramp_up_rate, (T-1, 1)).T,
               gt[:, :-1] - gt[:, 1:] <= np.tile(ramp_down_rate, (T-1, 1)).T]
problem = cp.Problem(objective, constraints)

# Solve with GLPK.
problem.solve(solver=cp.GLPK)
print(f"optimal value with GLPK: {problem.value}", np.sum(gt.value, axis=1),␣
 ↪sep='\n')
print("per generator", np.sum(gt.value))
```

```
GLPK Simplex Optimizer, v4.65
588 rows, 144 columns, 984 non-zeros
optimal value with GLPK: 10158.5
[177.6  64.2 160.9 127.4  15.8  31. ]
per generator 576.9
      0: obj =   0.000000000e+00 inf =   5.769e+02 (24)
    138: obj =   1.041700000e+04 inf =   0.000e+00 (0)
*   195: obj =   1.015850000e+04 inf =   0.000e+00 (0) 1
OPTIMAL LP SOLUTION FOUND
```

[9]:
```
opti_git_values = np.round_(gt.value, 1)

for i in range(0, len(opti_git_values)):
    print('Optimal g{}^t values:\n'.format(i+1), opti_git_values[i])
```

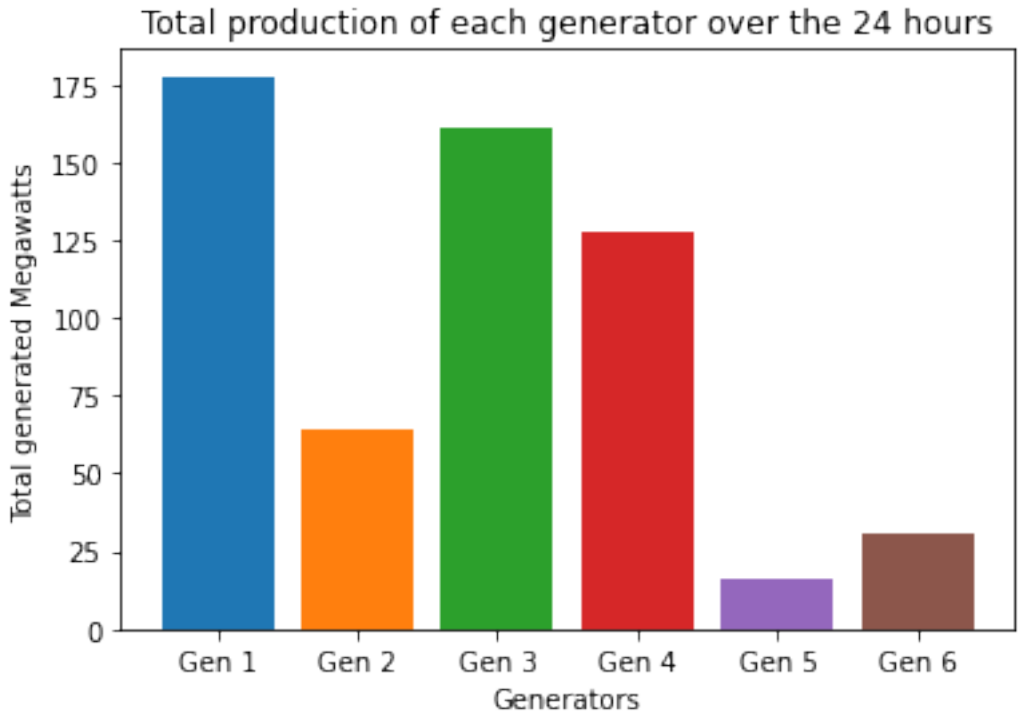
```
Optimal g1^t values:
 [ 5.7  3.7  1.7  2.7  0.7  2.7  2.6  4.6  6.6  8.6 10.  10.  10.  10.
 10.  10.  10.  10.  10.  10.  10.  10.  10.   8. ]
Optimal g2^t values:
 [0.  0.  0.  5.  0.  5.  0.  5.  5.  2.8 3.9 0.  2.8 5.  4.  5.  5.  5.
 4.5 1.2 5.  0.  0.  0. ]
Optimal g3^t values:
 [ 0.4  1.3  0.   2.   0.   2.   2.   4.   6.   8.  10.  10.  10.  10.
 10.  10.  10.  10.  10.  10.  10.  10.   8.6  6.6]
Optimal g4^t values:
 [ 0.   0.   0.   0.3  0.   0.3  0.   4.7  0.9  5.  10.   7.  10.  10.
 10.  10.  10.  10.  10.  10.  10.   5.9  3.3  0. ]
Optimal g5^t values:
 [ 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   2.2
  0.  10.   0.   3.6  0.   0.   0.   0.   0.   0. ]
Optimal g6^t values:
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  5.  1.  6. 11.  6.
  1.  0.  1.  0.  0.  0.]
```

[11]:
```
plt.figure(figsize=(20,8))

x = range(1, 25)

for i in range(0, len(gt.value)):
    plt.plot(x, gt.value[i], label='generator '+str(i+1), marker='+')

plt.ylabel('Generator utilization (in MW)')
plt.xlabel('T (time in hours)')
plt.title('Optimized generators utilization over time', fontsize=20)
plt.legend()
plt.grid()

plt.savefig('ex1.1.a.png')
plt.show()
```



[12]:
```
data = np.sum(gt.value, axis=1)

labels = ['Gen 1', 'Gen 2', 'Gen 3', 'Gen 4', 'Gen 5', 'Gen 6']
default_numpy_colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd',␣
 ↪'#8c564b', '#e377c2', '#7f7f7f', '#bcbd22', '#17becf']

plt.figure(figsize=(6,4))

plt.xticks(range(len(data)), labels)
plt.xlabel('Generators')
plt.ylabel('Total generated Megawatts')
plt.title('Total production of each generator over the 24 hours')
plt.bar(range(len(data)), data, color = default_numpy_colors[0:6])
```

```
plt.savefig('ex1.1.b.png')
plt.show()
```



5 Question 1.2

[13]:
```
nu_d = 0.92
nu_c = 0.95
S_bar = 20
```

[14]:
```
# YOUR CODE HERE
# We recommend GLPK solver
Ng = 6  # number of traditional generators
T = 24  # planning horizon
# Define variables
gt = cp.Variable((Ng, T))  # production of each traditional generator
S = cp.Variable(T+1)  # battery state of charge
bdt = cp.Variable(T)  # battery discharge over planning horizon
bct = cp.Variable(T)  # battery charge over the planning horizon
# Construct the problem
objective = cp.Minimize(cp.sum(generation_costs @ gt))
constraints = [dt == cp.sum(gt, axis=0) + rt + nu_d*bdt - bct,
```

```
                0 <= gt, gt <= np.tile(capacity, (T, 1)).T,
                gt[:, 1:] - gt[:, :-1] <= np.tile(ramp_up_rate, (T-1, 1)).T,
                gt[:, :-1] - gt[:, 1:] <= np.tile(ramp_down_rate, (T-1, 1)).T,
                # 0 <= bdt, 0 <= bct,
                # 0 <= bdt, bdt <= S[:-1], 0 <= bct, bct <= S_bar - S[1:],
                # 0 <= bdt, bdt <= S[:-1] + nu_c*bct, 0 <= bct, bct <= (S_bar -␣
 ↪S[1:]) + bdt,
                0 <= S, S <= S_bar,
                S[0] == 0, S[1:] == S[:-1] - bdt + nu_c*bct, S[-1] == 0,
                0 <= bdt, bdt <= S[:-1] + nu_c*bct,
                0 <= bct, bct <= (S_bar - S[:-1]) + bdt, ]
problem = cp.Problem(objective, constraints)

# Solve with GLPK.
problem.solve(solver=cp.GLPK, verbose=0)
print(f"optimal value with GLPK: {problem.value}", np.sum(gt.value, axis=1),␣
 ↪' charge', nu_c*bct.value - bdt.value, 'charge', S.value, sep='\n')
```

```
optimal value with GLPK: 9777.750582125813GLPK Simplex Optimizer, v4.65
760 rows, 217 columns, 1372 non-zeros

[2.06200000e+02 4.51441666e+01 1.82937069e+02 1.19515561e+02
 1.22124533e-15 2.59000001e+01]
 charge
[ 0.00000000e+00  0.00000000e+00  4.08500000e+00  8.88178420e-16
  5.51982759e+00  4.84827586e-01  9.41482759e+00  1.99827586e-01
  2.80905172e-01  1.40452586e-02  7.02262931e-04  3.51131465e-05
  1.75565733e-06 -2.49999991e+00  0.00000000e+00 -1.19565217e+01
 -1.08695652e+00 -4.45652174e+00  0.00000000e+00  1.08695652e+00
 -1.08695652e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]      0: obj =
0.000000000e+00 inf =   5.769e+02 (24)
    156: obj =   1.036290108e+04 inf =   0.000e+00 (0)
*   301: obj =   9.777750582e+03 inf =   1.776e-15 (0) 1
OPTIMAL LP SOLUTION FOUND

charge
[ 0.          0.          0.          4.085       4.085       9.60482759
 10.08965517 19.50448276 19.70431034 19.98521552 19.99926078 19.99996304
 19.99999815 19.99999991 17.5        17.5         5.54347826  4.45652174
  0.          0.          1.08695652  0.          0.          0.
  0.        ]
```

[15]:
```
opti_git_values = np.round_(gt.value, 1)

for i in range(0, len(opti_git_values)):
    print('Optimal g{}^t values:\n'.format(i+1), opti_git_values[i])
```

```
Optimal g1^t values:
 [ 6.1  4.1  4.   6.   4.   6.   8.  10.  10.  10.  10.  10.  10.  10.
 10.  10.  10.  10.  10.  10.  10.  10.  10.  10.  10.   8. ]
Optimal g2^t values:
 [ 0.   0.   0.   0.   0.  -0.   0.   0.  -0.  -0.   5.   0.   2.8  5.
  5.   5.   5.   5.   5.   2.3  5.   0.   0.   0. ]
Optimal g3^t values:
 [ 0.   0.9  2.   4.   2.5  4.5  6.5  8.5  8.8 10.  10.  10.  10.  10.
 10.  10.  10.  10.  10.  10.  10.  10.   8.6  6.6]
Optimal g4^t values:
 [ 0.   0.   0.   0.   0.   0.   0.   0.   0.   4.4  8.9  7.  10.  10.
 10.  10.  10.  10.  10.  10.  10.  10.   5.9  3.3  0. ]
Optimal g5^t values:
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -0.  0.  0.  0.  0.  0.  0. -0.
  0.  0.  0.  0.  0.  0.]
Optimal g6^t values:
 [ 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   4.9
  0.   5.  10.   5.5  0.5  0.   0.   0.   0.   0. ]
```

[16]:
```
print('Energy in battery (S_t):\n', np.round_(S.value, 1))
```

```
Energy in battery (S_t):
 [ 0.   0.   0.   4.1  4.1  9.6 10.1 19.5 19.7 20.  20.  20.  20.  20.
 17.5 17.5  5.5  4.5  0.   0.   1.1  0.   0.   0.   0. ]
```

[17]:
```
print('Energy used charging battery (bct):\n', np.round_(bct.value, 1))
```

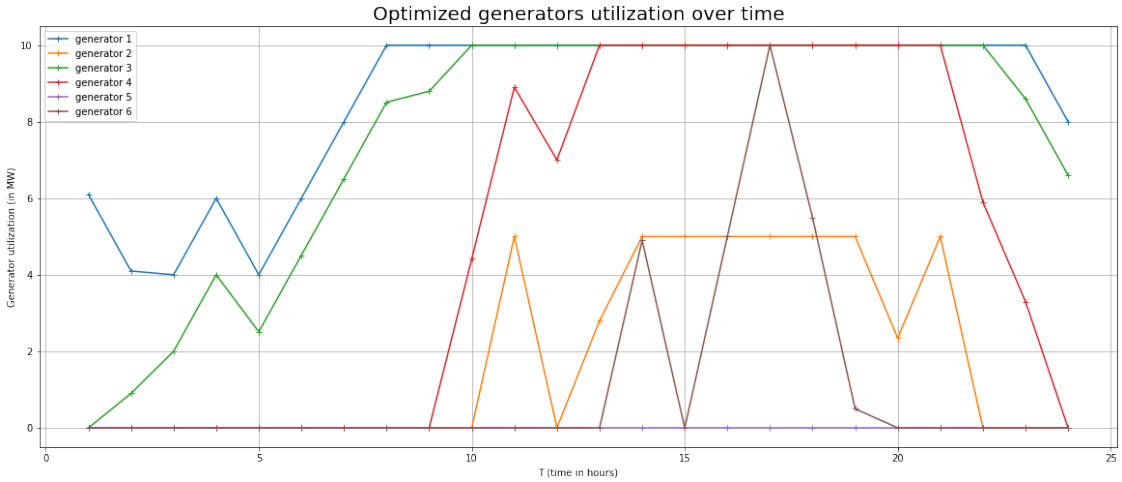
```
Energy used charging battery (bct):
 [0.  0.  4.3 0.  5.8 0.5 9.9 0.2 0.3 0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  1.1 0.  0.  0.  0. ]
```

[18]:
```
print('Energy taken from battery (bdt):\n', np.round_(bdt.value, 1))
```

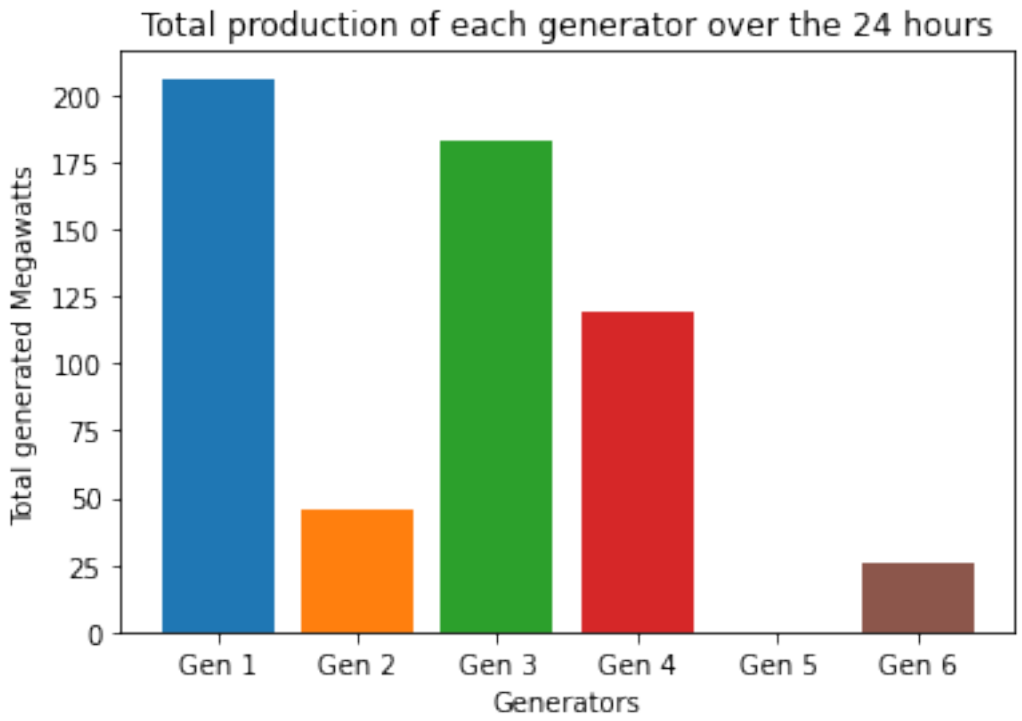
```
Energy taken from battery (bdt):
 [ 0.   0.   0.  -0.   0.   0.   0.  -0.   0.   0.   0.   0.   0.   2.5
  0.  12.   1.1  4.5  0.   0.   1.1  0.   0.   0. ]
```

[21]:
```
plt.figure(figsize=(20,8))

x = range(1, 25)

for i in range(0, len(gt.value)):
    plt.plot(x, gt.value[i], label='generator '+str(i+1), marker='+')

plt.ylabel('Generator utilization (in MW)')
plt.xlabel('T (time in hours)')
plt.title('Optimized generators utilization over time', fontsize=20)
plt.legend()
```

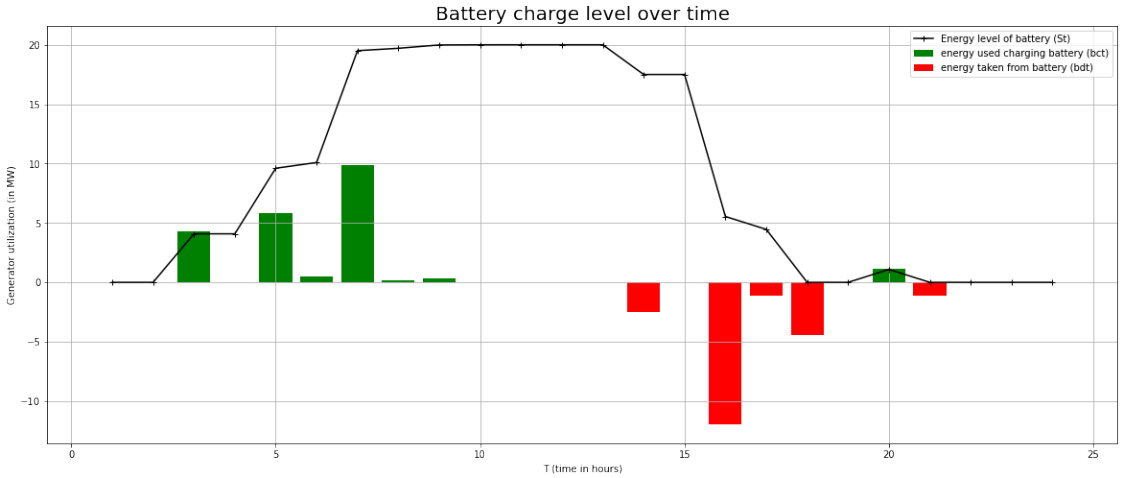
```
plt.grid()

plt.savefig('ex1.3.a.png')
plt.show()
```



[20]:
```
data = np.sum(gt.value, axis=1)

labels = ['Gen 1', 'Gen 2', 'Gen 3', 'Gen 4', 'Gen 5', 'Gen 6']
default_numpy_colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd',␣
 ↪'#8c564b', '#e377c2', '#7f7f7f', '#bcbd22', '#17becf']

plt.figure(figsize=(6,4))

plt.xticks(range(len(data)), labels)
plt.xlabel('Generators')
plt.ylabel('Total generated Megawatts')
plt.title('Total production of each generator over the 24 hours')
plt.bar(range(len(data)), data, color = default_numpy_colors[0:6])

plt.savefig('ex1.3.b.png')
plt.show()
```



[22]:
```
plt.figure(figsize=(20,8))

x = range(1, 25)

plt.bar(x, bct.value, color='green', label='energy used charging battery (bct)')
plt.bar(x, -1*bdt.value, color='red', label='energy taken from battery (bdt)')
plt.plot(x, S.value[1:], label='Energy level of battery (St)', marker='+',␣
 ↪color='black')

plt.ylabel('Generator utilization (in MW)')
plt.xlabel('T (time in hours)')
plt.title('Battery charge level over time', fontsize=20)
plt.legend()
plt.grid()

plt.savefig('ex1.3.c.png')
plt.show()
```


Battery charge level over time
Energy level of battery (St)
energy used charging battery (bct)
energy taken from battery (bdt)
Generator utilization (in MW)
T (time in hours)
20
15
10
5
0
−5
−10
0
5
10
15
20
25

[ ]:

Problem2

May 20, 2022

1 Problem 2 - Unit Commitment

[1]:
```
import numpy as np
import cvxpy as cp
import timeit

import matplotlib.pyplot as plt
```

2 Definition of data

[2]:
```
generation_costs = np.array([15, 20, 15, 20, 30, 25.])
startup_costs = np.array([75, 100, 75, 100, 100, 125.])
shutdown_costs = np.array([7.5, 10.0, 7.5, 10.0, 10.0, 12.5])
running_costs = np.array([10, 5, 10, 10, 10, 10.])

capacity = np.array([10, 5, 10, 10, 20, 30.])
ramp_up_rate = np.array([2, 5, 2, 5, 10, 5.])
ramp_down_rate = np.array([2, 5, 2, 5, 10, 5.])

initial_state = np.array([1, 0, 0, 0, 0, 0])
```

[3]:
```
rt = np.array([15.2, 16.4, 16.1, 10.9, 14.8, 7.6, 15.6, 5.5, 9.2, 5.7, 1.5, 12.
↪4, 10.4, 4.8, 14.3, 0.5, 6.6, 5.7, 11.5, 11.9, 2.8, 7.3, 6.7, 9.7])
dt = np.array([21.3, 21.4, 17.8, 20.9, 15.5, 17.6, 20.2, 23.8, 27.7, 30.1, 35.
↪4, 39.4, 43.2, 47.0, 49.3, 51.5, 52.6, 50.3, 47.0, 43.1, 38.8, 33.2, 28.6,␣
↪24.3])
```

3 Question 2.2

[4]:
```
Tup = 3
Tdown = 2
```

[9]:
```
# YOUR CODE HERE
# Hint: Define decision variables in CVXPY using
# shape = (2,2) # example: 2x2 binary matrix
# binary_example_variable = cp.Variable(shape, boolean=True)
# We recommend GLPK_MI solver
Ng = 6  # number of traditional generators
T = 24  # planning horizon
# Define variables
u = cp.Variable((Ng, T), boolean=True)  # generator is starting (flag)
v = cp.Variable((Ng, T), boolean=True)  # generator is shutting down (flag)
x = cp.Variable((Ng, T), boolean=True)  # generator is running (flag)
gt = cp.Variable((Ng, T))  # production of each traditional generator
# Construct the problem
objective = cp.Minimize(cp.sum(startup_costs @ u + shutdown_costs @ v
                               + running_costs @ x + generation_costs @ gt))
constraints = [dt == cp.sum(gt, axis=0) + rt,
               0 <= gt, gt <= cp.multiply(np.tile(capacity, (T, 1)).T, x),
               x[:, :-1] - x[:, 1:] + u[:, 1:] >= 0,
               x[:, 1:] - x[:, :-1] + v[:, 1:] >= 0,
               *[x[:, t] - x[:, t-1] <= x[:, tau]
                    for t in range(1, T) for tau in range(t+1, min(t + Tup,␣
 ↪T))],
               *[x[:, t-1] - x[:, t] <= 1 - x[:, tau]
                    for t in range(1, T) for tau in range(t+1, min(t + Tdown,␣
 ↪T))],
               u[:, 0] == 0, x[:, 0] == initial_state, v[:, 0] == 0]
problem = cp.Problem(objective, constraints)
```

[10]:
```
# Solve with GLPK.
problem.solve(solver=cp.GLPK_MI, verbose=0)
print(f"optimal value with GLPK_MI: {problem.value}", np.sum(gt.value, axis=1),␣
 ↪'running', x.value, sep='\n')
timeit.timeit()
```

```
Long-step dual simplex will be used
optimal value with GLPK_MI: 11024.5
[202.7  61.4 166.8 100.7   0.   45.3]
running
[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0.]]
```

[10]: 0.008301086723804474

[11]:
```
cost_per_gen=[]
for i in range(Ng):
    for t in range(T):
        cost_per_gen.append(startup_costs[i] * u.value[i][t] +␣
↪shutdown_costs[i] * v.value[i][t] + running_costs[i] * x.value[i][t] +␣
↪generation_costs[i] * gt.value[i][t])
    print(sum(cost_per_gen))
    cost_per_gen=[]
```

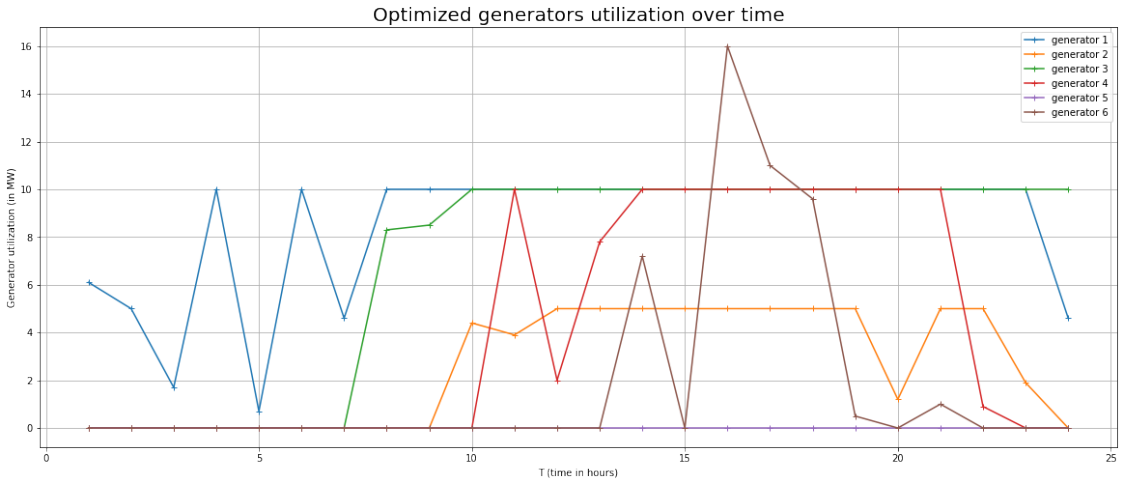
```
3280.5
1403.0
2747.0
2244.0
0.0
1349.9999999999998
```

[12]:
```
opti_git_values = np.round_(gt.value, 1)

for i in range(0, len(opti_git_values)):
    print('Optimal g{}^t values:\n'.format(i+1), opti_git_values[i])
```

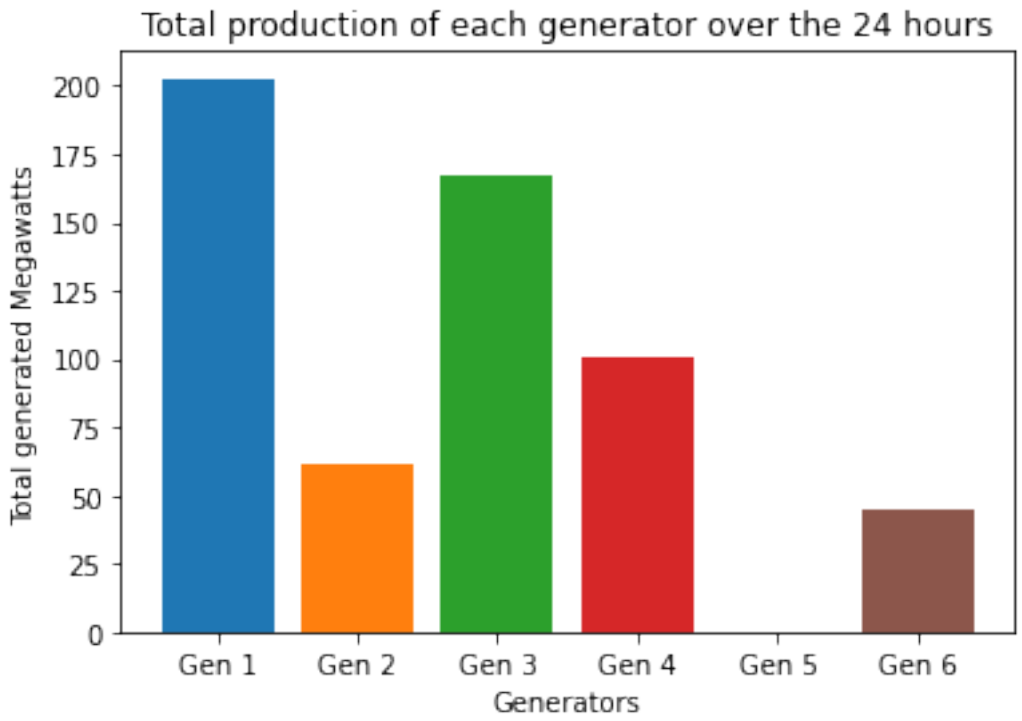
```
Optimal g1^t values:
 [ 6.1  5.   1.7 10.   0.7 10.   4.6 10.  10.  10.  10.  10.  10.  10.
 10.  10.  10.  10.  10.  10.  10.  10.  10.  10.  10.   4.6]
Optimal g2^t values:
 [-0.  -0.   0.  -0.  -0.  -0.  -0.  -0.  -0.   4.4  3.9  5.   5.   5.
  5.   5.   5.   5.   5.   1.2  5.   5.   1.9 -0. ]
Optimal g3^t values:
 [-0.  -0.   0.  -0.   0.   0.  -0.   8.3  8.5 10.  10.  10.  10.  10.
 10.  10.  10.  10.  10.  10.  10.  10.  10.  10. ]
Optimal g4^t values:
 [-0.  -0.   0.  -0.  -0.  -0.  -0.  -0.  -0.  -0.  10.   2.   7.8 10.
 10.  10.  10.  10.  10.  10.  10.   0.9 -0.  -0. ]
Optimal g5^t values:
 [-0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0.
 -0. -0. -0. -0. -0. -0.]
Optimal g6^t values:
 [-0.  -0.  -0.  -0.  -0.  -0.  -0.  -0.  -0.  -0.  -0.  -0.  -0.   7.2
 -0.  16.  11.   9.6  0.5 -0.   1.  -0.  -0.  -0. ]
```

[13]:
```
plt.figure(figsize=(20,8))

x_ticks = range(1, 25)

for i in range(0, len(gt.value)):
    plt.plot(x_ticks, gt.value[i], label='generator '+str(i+1), marker='+')
```

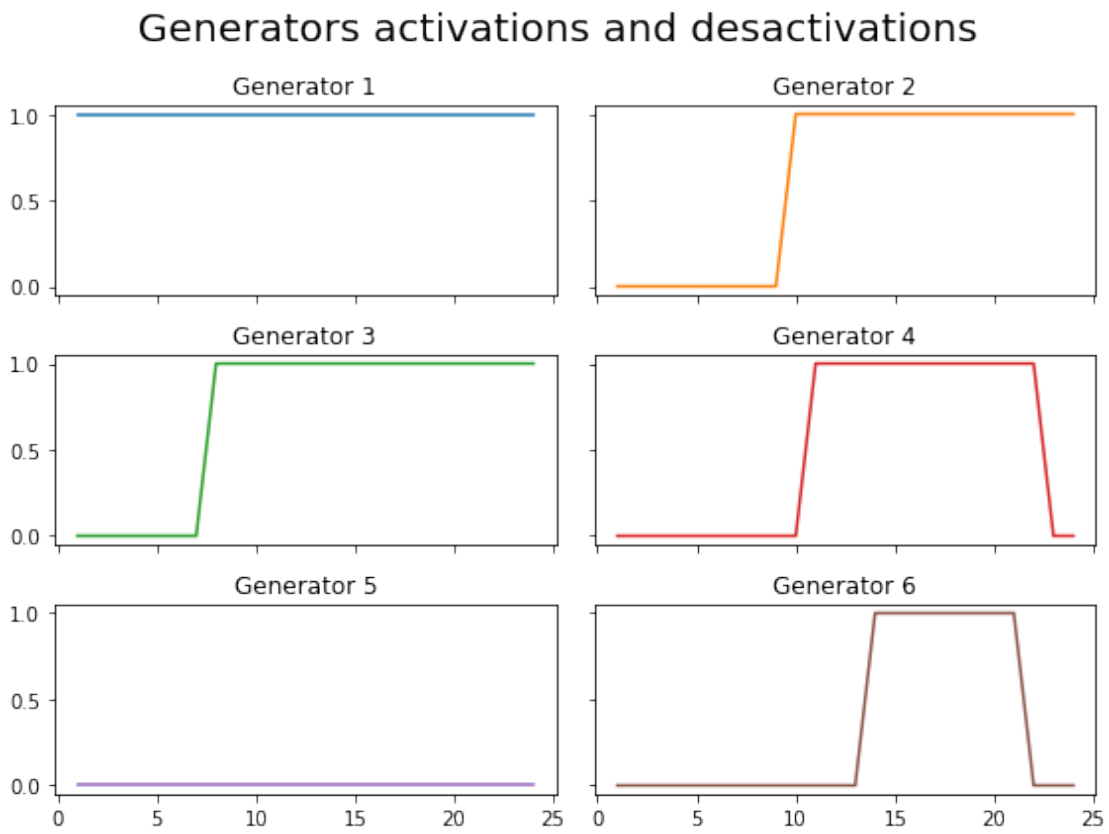
```
plt.ylabel('Generator utilization (in MW)')
plt.xlabel('T (time in hours)')
plt.title('Optimized generators utilization over time', fontsize=20)
plt.legend()
plt.grid()

plt.savefig('ex2.2.a.png')
plt.show()
```



[10]:
```
data = np.sum(gt.value, axis=1)

labels = ['Gen 1', 'Gen 2', 'Gen 3', 'Gen 4', 'Gen 5', 'Gen 6']
default_numpy_colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd',␣
 ↪'#8c564b', '#e377c2', '#7f7f7f', '#bcbd22', '#17becf']

plt.figure(figsize=(6,4))

plt.xticks(range(len(data)), labels)
plt.xlabel('Generators')
plt.ylabel('Total generated Megawatts')
plt.title('Total production of each generator over the 24 hours')
plt.bar(range(len(data)), data, color = default_numpy_colors[0:6])

plt.savefig('ex2.2.b.png')
plt.show()
```



[14]:
```
x_ticks = range(1, 25)

fig, ((ax11, ax12), (ax21, ax22), (ax31, ax32)) = plt.subplots(3, 2, sharex =␣
 ↪True, sharey = True,figsize=(8.0, 6.0))

ax11.plot(x_ticks, x.value[0], color='#1f77b4')
ax11.set_title('Generator 1')
ax12.plot(x_ticks, x.value[1], color='#ff7f0e')
ax12.set_title('Generator 2')
ax21.plot(x_ticks, x.value[2], color='#2ca02c')
ax21.set_title('Generator 3')
ax22.plot(x_ticks, x.value[3], color='#d62728')
ax22.set_title('Generator 4')
ax31.plot(x_ticks, x.value[4], color='#9467bd')
ax31.set_title('Generator 5')
ax32.plot(x_ticks, x.value[5], color='#8c564b')
ax32.set_title('Generator 6')

fig.suptitle('Generators activations and desactivations', fontsize=20)
plt.tight_layout()
plt.savefig('ex2.2.c.png')
plt.show()
```



4 Question 2.3 Relaxed

[15]:
```
Tup = 3
Tdown = 2
```

[16]:
```
# YOUR CODE HERE
# We recommend GLPK_MI solver
Ng = 6  # number of traditional generators
T = 24  # planning horizon
# Define variables
u = cp.Variable((Ng, T))  # generator is starting (flag)
v = cp.Variable((Ng, T))  # generator is shutting down (flag)
x = cp.Variable((Ng, T),boolean=True)  # generator is running (flag)
gt = cp.Variable((Ng, T))  # production of each traditional generator
# Construct the problem
objective = cp.Minimize(cp.sum(startup_costs @ u + shutdown_costs @ v +␣
 ↪running_costs @ x
                               + generation_costs @ gt))
```

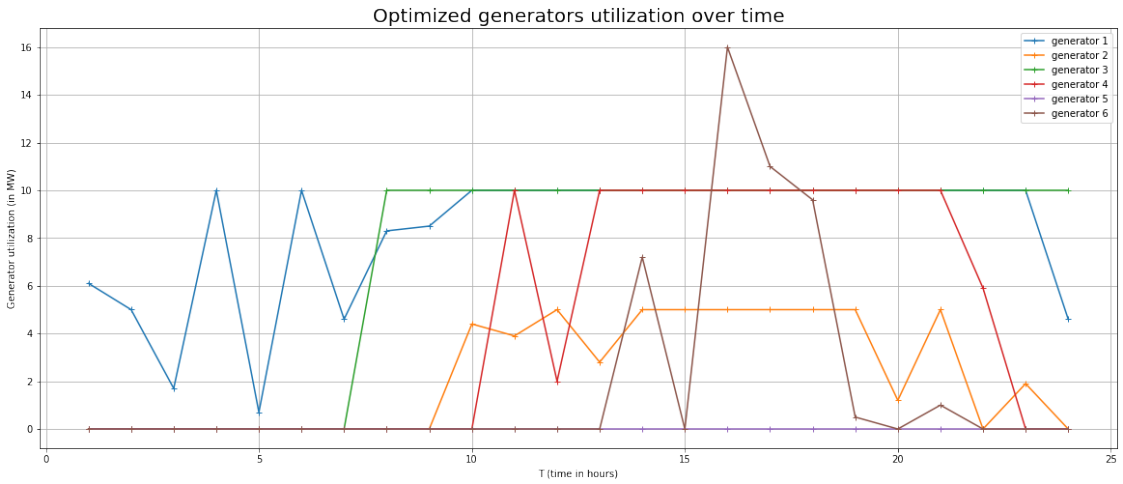
```
constraints = [dt == cp.sum(gt, axis=0) + rt,
               0 <= gt, gt <= cp.multiply(np.tile(capacity, (T, 1)).T, x),
               x[:, :-1] - x[:, 1:] + u[:, 1:] >= 0,
               x[:, 1:] - x[:, :-1] + v[:, 1:] >= 0,
               *[x[:, t] - x[:, t-1] <= x[:, tau]
                 for t in range(1, T) for tau in range(t+1, min(t + Tup, T))],
               *[x[:, t-1] - x[:, t] <= 1 - x[:, tau]
                 for t in range(1, T) for tau in range(t+1, min(t + Tdown, T))],
               u[:, 0] == 0, x[:, 0] == initial_state, v[:, 0] == 0,
               0 <= u, u <= 1, 0 <= v, v <= 1]
problem = cp.Problem(objective, constraints)
```

[17]:
```
# Solve with GLPK.
problem.solve(solver=cp.GLPK_MI, verbose=0)
print(f"optimal value with GLPK_MI: {problem.value}", np.sum(
    gt.value, axis=1), 'running', x.value, sep='\n')
timeit.timeit()
```

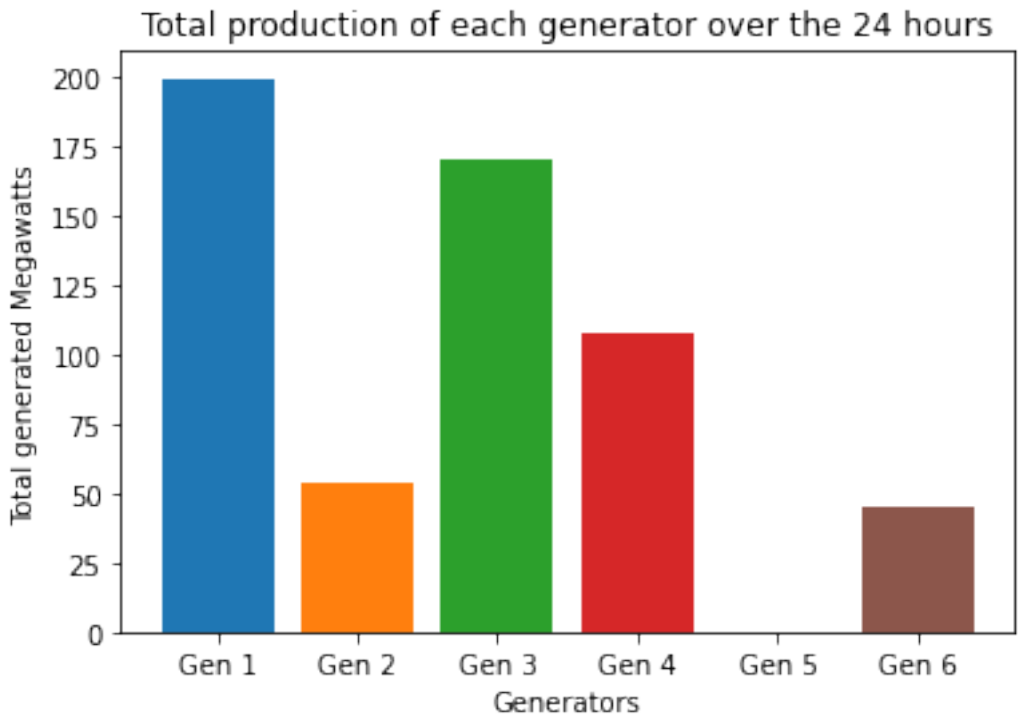
```
Long-step dual simplex will be used
optimal value with GLPK_MI: 11024.5
[199.5  54.2 170.  107.9   0.   45.3]
running
[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0.]]
```

[17]:
```
0.008580219000577927
```

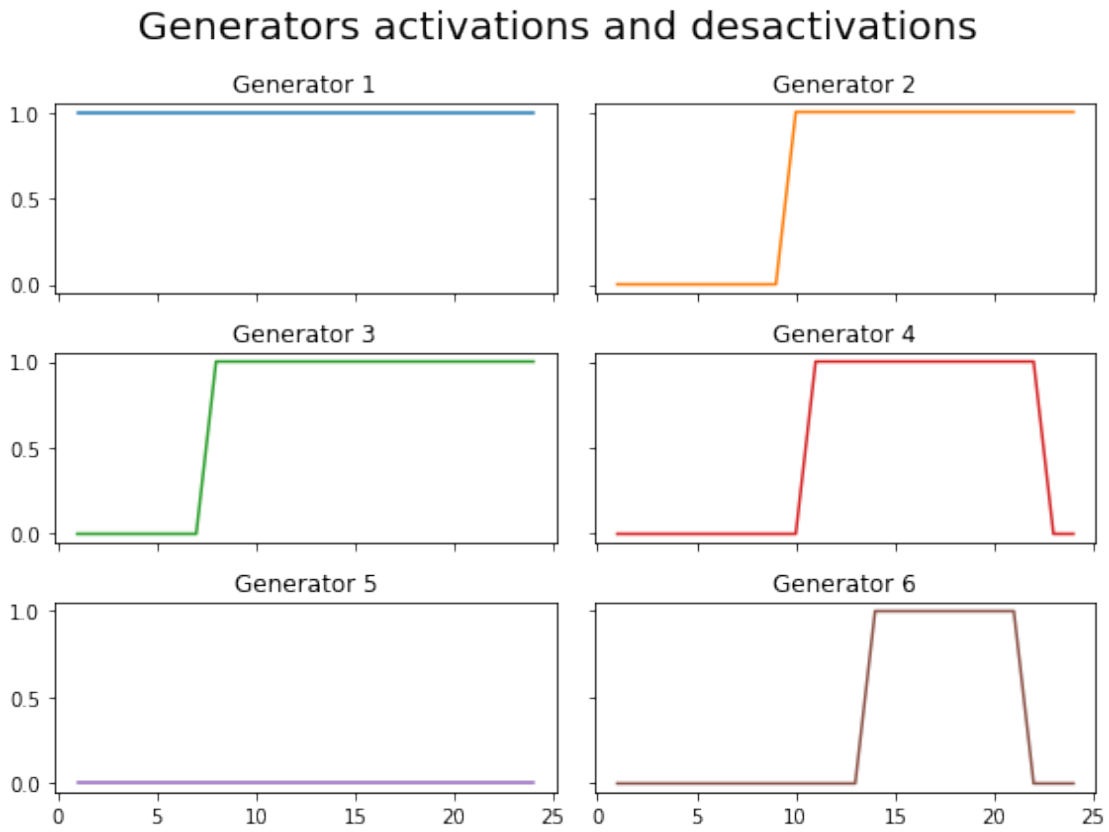
[20]:
```
plt.figure(figsize=(20,8))

x_ticks = range(1, 25)

for i in range(0, len(gt.value)):
    plt.plot(x_ticks, gt.value[i], label='generator '+str(i+1), marker='+')

plt.ylabel('Generator utilization (in MW)')
plt.xlabel('T (time in hours)')
plt.title('Optimized generators utilization over time', fontsize=20)
plt.legend()
plt.grid()

plt.savefig('ex2.3.a.png')
plt.show()
```



[21]:
```
data = np.sum(gt.value, axis=1)

labels = ['Gen 1', 'Gen 2', 'Gen 3', 'Gen 4', 'Gen 5', 'Gen 6']
default_numpy_colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd',␣
 ↪'#8c564b', '#e377c2', '#7f7f7f', '#bcbd22', '#17becf']

plt.figure(figsize=(6,4))

plt.xticks(range(len(data)), labels)
plt.xlabel('Generators')
plt.ylabel('Total generated Megawatts')
plt.title('Total production of each generator over the 24 hours')
plt.bar(range(len(data)), data, color = default_numpy_colors[0:6])

plt.savefig('ex2.3.b.png')
plt.show()
```



[22]:
```
x_ticks = range(1, 25)

fig, ((ax11, ax12), (ax21, ax22), (ax31, ax32)) = plt.subplots(3, 2, sharex =␣
 ↪True, sharey = True,figsize=(8.0, 6.0))

ax11.plot(x_ticks, x.value[0], color='#1f77b4')
ax11.set_title('Generator 1')
ax12.plot(x_ticks, x.value[1], color='#ff7f0e')
ax12.set_title('Generator 2')
ax21.plot(x_ticks, x.value[2], color='#2ca02c')
ax21.set_title('Generator 3')
ax22.plot(x_ticks, x.value[3], color='#d62728')
ax22.set_title('Generator 4')
ax31.plot(x_ticks, x.value[4], color='#9467bd')
ax31.set_title('Generator 5')
ax32.plot(x_ticks, x.value[5], color='#8c564b')
ax32.set_title('Generator 6')

fig.suptitle('Generators activations and desactivations', fontsize=20)
plt.tight_layout()
plt.savefig('ex2.3.c.png')
plt.show()
```


Generators activations and desactivations
Generator 1
Generator 2
Generator 3
Generator 4
Generator 5
Generator 6
1.0
0.5
0.0
0
5
10
15
20
25

Problem3

May 20, 2022

1 Problem 3 - Robust Unit Commitment

[15]:
```
import numpy as np
import cvxpy as cp

import matplotlib.pyplot as plt
```

2 Definition of data

[2]:
```
s = slice(0, 6, None)
startup_costs = np.array([75, 100, 75, 100, 100, 125.])[s]
shutdown_costs = np.array([7.5, 10.0, 7.5, 10.0, 10.0, 12.5])[s]
running_costs = np.array([10, 5, 10, 10, 10, 10.])[s]
generation_costs = np.array([15, 20, 15, 20, 30, 25.])[s]

capacity = np.array([10, 5, 10, 10, 20, 30.])[s]
ramp_up_rate = np.array([2, 5, 2, 5, 10, 5.])[s]
ramp_down_rate = np.array([2, 5, 2, 5, 10, 5.])[s]

initial_state = np.array([1, 0, 0, 0, 0, 0])[s]
```

[3]:
```
rt_bar = np.array([15.2, 16.4, 16.1, 10.9, 14.8, 7.6, 15.6, 5.5, 9.2, 5.7, 1.5,␣
↪12.4, 10.4, 4.8, 14.3, 0.5, 6.6, 5.7, 11.5, 11.9, 2.8, 7.3, 6.7, 9.7])
rt_hat = 0.6*np.ones_like(rt_bar)
dt = np.array([21.3, 21.4, 17.8, 20.9, 15.5, 17.6, 20.2, 23.8, 27.7, 30.1, 35.
↪4, 39.4, 43.2, 47.0, 49.3, 51.5, 52.6, 50.3, 47.0, 43.1, 38.8, 33.2, 28.6,␣
↪24.3])
```

3 Question 3.5

[4]:
```
Tup = 3
Tdown = 2
```

[5]:
```
# %%timeit
# YOUR CODE HERE
# We recommend GLPK_MI solver
Ng = len(initial_state)  # number of traditional generators
T = len(dt)  # planning horizon
# Define variables
a = cp.Variable((Ng, T))  # rt param
b = cp.Variable((Ng, T))  # rt param
c = cp.Variable((Ng, T))  # rt param
u = cp.Variable((Ng, T), boolean=True)  # generator is starting (flag)
v = cp.Variable((Ng, T), boolean=True)  # generator is shutting down (flag)
x = cp.Variable((Ng, T), boolean=True)  # generator is running (flag)

# problem dual variables
p1 = cp.Variable(T)
p2 = cp.Variable(T)
# constraints dual variables
y1 = cp.Variable((Ng, T))
y2 = cp.Variable((Ng, T))
y3 = cp.Variable((Ng, T))
y4 = cp.Variable((Ng, T))
y5 = cp.Variable((Ng, T))
y6 = cp.Variable((Ng, T))
y7 = cp.Variable((Ng, T))
y8 = cp.Variable((Ng, T))

gx = cp.multiply(np.tile(capacity, (T, 1)).T, x)
rp, rm = rt_hat + rt_bar, rt_hat - rt_bar

'''
objective = cp.Minimize(
    cp.sum(startup_costs @ u + shutdown_costs @ v + running_costs @ x)
    + np.repeat(generation_costs, T) @ c.flatten()
    + p1 @ (rt_hat + rt_bar) + p2 @ (rt_hat - rt_bar)
)

constraints3 = [
    # 3b & linearization
    cp.sum(a, axis=0) == -1,
    cp.sum(b, axis=0) == 0,
    cp.sum(c, axis=0) == dt,
```

```
    # 3c & uncertainty box
    b[:,0] == 0
]

'''

###

objective = cp.Minimize(
    cp.sum( [rp[t] * p1[t] + rm[t] * p2[t] for t in range(T)] )
    + cp.sum( [ (startup_costs[i] * u[i][t] + shutdown_costs[i] * v[i][t] +␣
 ↪running_costs[i] * x[i][t] + generation_costs[i] * c[i][t])
               for i in range(Ng) for t in range(T)] )
)

constraints2 = [
    # 2d
    x[:, :-1] - x[:, 1:] + u[:, 1:] >= 0,
    # 2e
    x[:, 1:] - x[:, :-1] + v[:, 1:] >= 0,
    # 2f
    *[x[:, t] - x[:, t-1] <= x[:, tau]
      for t in range(1, T) for tau in range(t+1, min(t + Tup, T))],
    # 2g
    *[x[:, t-1] - x[:, t] <= 1 - x[:, tau]
      for t in range(1, T) for tau in range(t+1, min(t + Tdown, T))],
    # 2h
    u[:, 0] == 0, x[:, 0] == initial_state, v[:, 0] == 0,
]

constraints3 = []

for i in range(Ng):
    constraints3.append(b[i][0] == 0)

for t in range(T):
    constraints3.append(sum([c[i][t] for i in range(Ng)]) == dt[t])
    constraints3.append(sum([b[i][t] for i in range(Ng)]) == 0)
    constraints3.append(sum([a[i][t] for i in range(Ng)]) == -1)


for t in range(T-1):
    constraints3.append( p1[t] - p2[t] >= sum([generation_costs[i] *␣
 ↪(a[i][t]+b[i][t+1]) for i in range(Ng)]) )

constraints3.append( p1[T-1] - p2[T-1] >= sum([generation_costs[i] * a[i][T-1]␣
 ↪for i in range(Ng)]) )
```

```
for i in range(Ng):
    for t in range(T):
        if t == 0:
            constraints3 += [-c[i][t] <= rp[t]*y1[i][t] + rm[t]*y2[i][t]]
            constraints3 += [c[i][t] - gx[i][t] <= rp[t]*y5[i][t] +␣
 ↪rm[t]*y6[i][t]]
        else:
            constraints3 += [-c[i][t] <= rp[t]*y1[i][t] + rm[t]*y2[i][t] +
                                                  rp[t-1]*y3[i][t] +␣
 ↪rm[t-1]*y4[i][t]]
            constraints3 += [c[i][t] - gx[i][t] <= rp[t]*y5[i][t] +␣
 ↪rm[t]*y6[i][t] +
                                                  rp[t-1]*y7[i][t] +␣
 ↪rm[t-1]*y8[i][t]]

        constraints3.append(y1[i][t] - y2[i][t] <= a[i][t])
        constraints3.append(y3[i][t] - y4[i][t] <= b[i][t])
        constraints3.append(y5[i][t] - y6[i][t] <= -a[i][t])
        constraints3.append(y7[i][t] - y8[i][t] <= -b[i][t])



constraints3 += [y1 <= 0]
constraints3 += [y2 <= 0]
constraints3 += [y3 <= 0]
constraints3 += [y4 <= 0]
constraints3 += [y5 <= 0]
constraints3 += [y6 <= 0]
constraints3 += [y7 <= 0]
constraints3 += [y8 <= 0]
constraints3 += [p1 >= 0]
constraints3 += [p2 >= 0]

problem = cp.Problem(objective, constraints2 + constraints3)
problem.solve(solver=cp.GLPK_MI, verbose=1)
print("Total cost", problem.value)
```

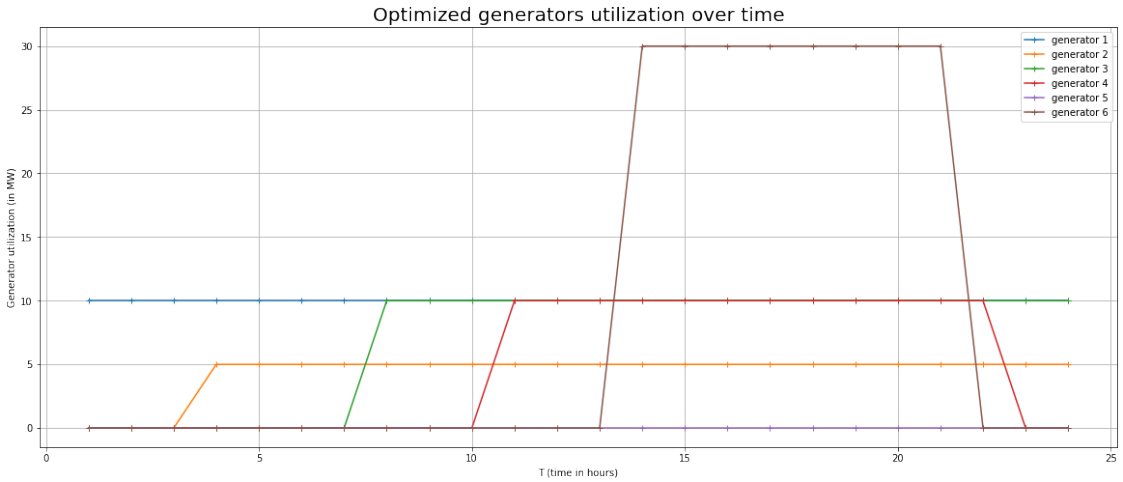
```
===============================================================================
                                     CVXPY
                                    v1.1.18
===============================================================================
(CVXPY) May 20 08:49:49 PM: Your problem has 2064 variables, 1046 constraints,
and 0 parameters.
(CVXPY) May 20 08:49:49 PM: It is compliant with the following grammars: DCP,
DQCP
(CVXPY) May 20 08:49:49 PM: (If you need to solve this problem multiple times,
```

```
but with different data, consider using parameters.)
(CVXPY) May 20 08:49:49 PM: CVXPY will first compile your problem; then, it will
invoke a numerical solver to obtain a solution.
-------------------------------------------------------------------------------
                                  Compilation
-------------------------------------------------------------------------------
(CVXPY) May 20 08:49:50 PM: Compiling problem (target solver=GLPK_MI).
(CVXPY) May 20 08:49:50 PM: Reduction chain: Dcp2Cone -> CvxAttr2Constr ->
ConeMatrixStuffing -> GLPK_MI
(CVXPY) May 20 08:49:50 PM: Applying reduction Dcp2Cone
(CVXPY) May 20 08:49:50 PM: Applying reduction CvxAttr2Constr
(CVXPY) May 20 08:49:50 PM: Applying reduction ConeMatrixStuffing
(CVXPY) May 20 08:49:51 PM: Applying reduction GLPK_MI
(CVXPY) May 20 08:49:51 PM: Finished problem compilation (took 1.887e+00
seconds).
-------------------------------------------------------------------------------
                                Numerical solver
-------------------------------------------------------------------------------
(CVXPY) May 20 08:49:51 PM: Invoking solver GLPK_MI  to obtain a solution.
      0: obj =   2.001000000e+04 inf =   5.116e+02 (49)
    368: obj =   2.073461667e+04 inf =   4.441e-15 (0)
*   974: obj =   1.116789167e+04 inf =   0.000e+00 (0) 4
Long-step dual simplex will be used
+   974: mip =     not found yet >=              -inf        (1; 0)
+  1178: >>>>>   1.159400000e+04 >=   1.117415833e+04   3.6% (34; 0)
+  1868: >>>>>   1.159200000e+04 >=   1.129726667e+04   2.5% (125; 11)
+  2714: >>>>>   1.157000000e+04 >=   1.130815000e+04   2.3% (228; 25)
+  3146: >>>>>   1.145900000e+04 >=   1.131176667e+04   1.3% (288; 32)
+  5531: >>>>>   1.143700000e+04 >=   1.132463333e+04   1.0% (385; 267)
+  6124: >>>>>   1.139400000e+04 >=   1.132660000e+04   0.6% (364; 423)
+  6942: >>>>>   1.136900000e+04 >=   1.132951667e+04   0.3% (344; 577)
+  7660: >>>>>   1.134700000e+04 >=   1.133165000e+04   0.1% (336; 701)
+  7839: >>>>>   1.133700000e+04 >=   1.133260000e+04 < 0.1% (191; 969)
+  8282: mip =   1.133700000e+04 >=     tree is empty   0.0% (0; 1569)
-------------------------------------------------------------------------------
                                     Summary
-------------------------------------------------------------------------------
(CVXPY) May 20 08:49:58 PM: Problem status: optimal
(CVXPY) May 20 08:49:58 PM: Optimal value: 1.134e+04
(CVXPY) May 20 08:49:58 PM: Compilation took 1.887e+00 seconds
(CVXPY) May 20 08:49:58 PM: Solver (including time spent in interface) took
6.713e+00 seconds
Total cost 11337.0
```

[6]:
```
len(constraints3)
```

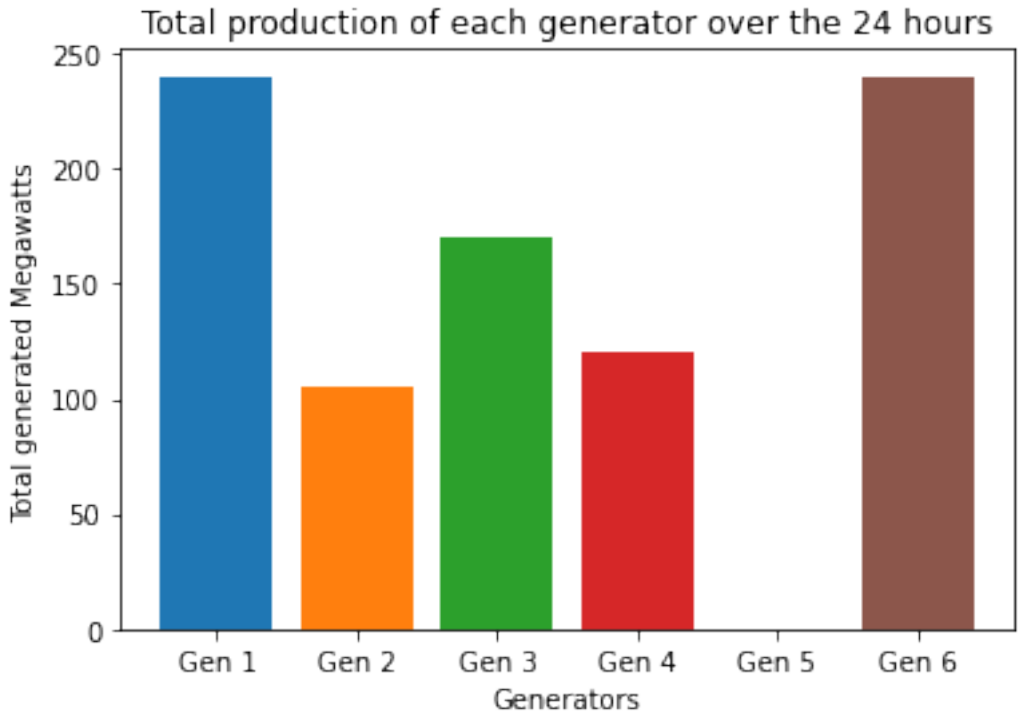
[6]: 976

[10]:
```
plt.figure(figsize=(20,8))

x_ticks = range(1, 25)

for i in range(0, len(gx.value)):
    plt.plot(x_ticks, gx.value[i], label='generator '+str(i+1), marker='+')

plt.ylabel('Generator utilization (in MW)')
plt.xlabel('T (time in hours)')
plt.title('Optimized generators utilization over time', fontsize=20)
plt.legend()
plt.grid()

plt.savefig('ex3.5.a.png')
plt.show()
```



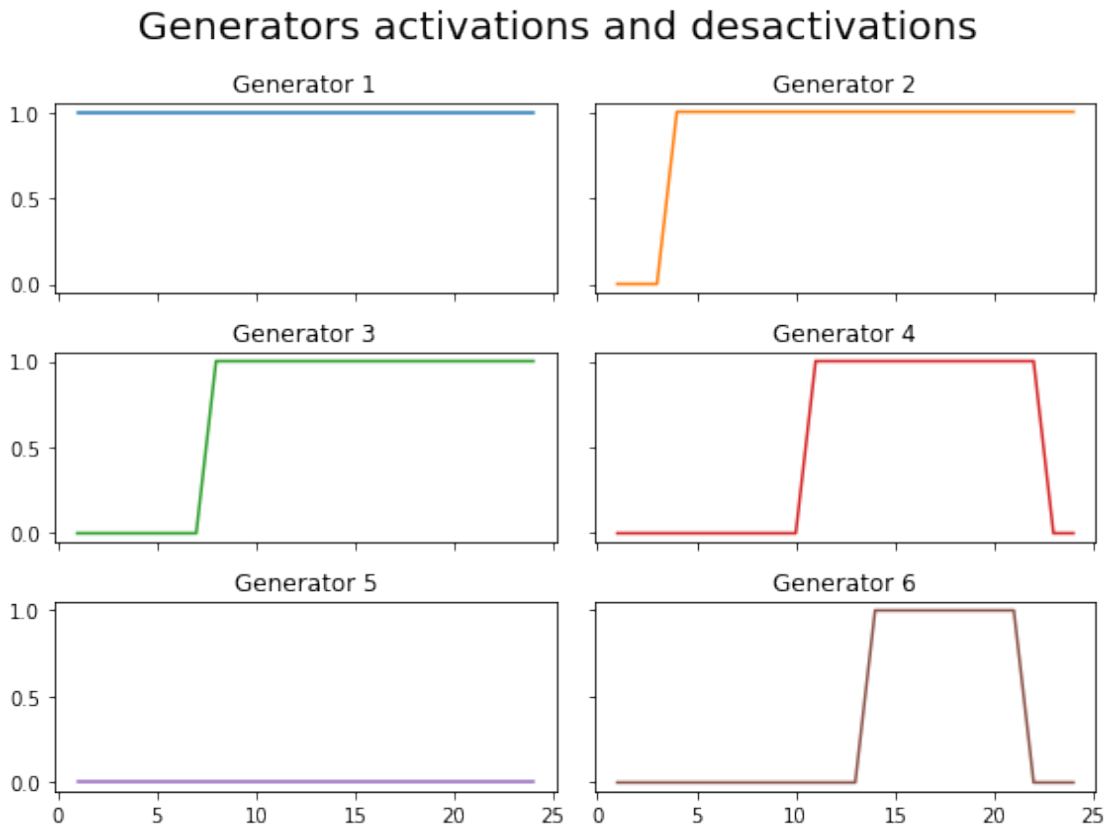
[11]:
```
data = np.sum(gx.value, axis=1)

labels = ['Gen 1', 'Gen 2', 'Gen 3', 'Gen 4', 'Gen 5', 'Gen 6']
default_numpy_colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd',␣
↪'#8c564b', '#e377c2', '#7f7f7f', '#bcbd22', '#17becf']

plt.figure(figsize=(6,4))

plt.xticks(range(len(data)), labels)
plt.xlabel('Generators')
plt.ylabel('Total generated Megawatts')
plt.title('Total production of each generator over the 24 hours')
plt.bar(range(len(data)), data, color = default_numpy_colors[0:6])
```

```
plt.savefig('ex3.5.b.png')
plt.show()
```



[13]:
```
x_ticks = range(1, 25)

fig, ((ax11, ax12), (ax21, ax22), (ax31, ax32)) = plt.subplots(3, 2, sharex =␣
 ↪True, sharey = True,figsize=(8.0, 6.0))

ax11.plot(x_ticks, x.value[0], color='#1f77b4')
ax11.set_title('Generator 1')
ax12.plot(x_ticks, x.value[1], color='#ff7f0e')
ax12.set_title('Generator 2')
ax21.plot(x_ticks, x.value[2], color='#2ca02c')
ax21.set_title('Generator 3')
ax22.plot(x_ticks, x.value[3], color='#d62728')
ax22.set_title('Generator 4')
ax31.plot(x_ticks, x.value[4], color='#9467bd')
ax31.set_title('Generator 5')
ax32.plot(x_ticks, x.value[5], color='#8c564b')
ax32.set_title('Generator 6')

fig.suptitle('Generators activations and desactivations', fontsize=20)
plt.tight_layout()
```

```
plt.savefig('ex3.5.c.png')
plt.show()
```



[ ]: