School of electronic and Computer Engineering

Prosumer of Power Grid as Convex Optimization Problem

Aggelos Aggelidakis
Charilaos Akasiadis

1. Introduction

The depletion of fossil fuel resources, as well as the global warming due to the "greenhouse effect", have led electricity suppliers to turn to renewable energy generation. The use of such sources can help reduce harmful gases emissions and promise a more sustainable generation process [7]. Due to the fact that renewable energy plants are not centralized and may cover broad geographical regions, researchers focus has turned to distributed generation, where electric power generation sources are connected directly to the distribution network or on the customer side of the meter [14]. The installation of additional sensors and controllers, on every level of the power network, that are able to communicate with each other, makes it possible to effectively manage the flow of energy, as if we had a large and centralized power plant, that sells energy to neighboring plants when it generates surplus, or buys from them when there is shortage. Such infrastructures are known as Virtual Power Plants or Microgrids [1].

The aggregation of the aforementioned energy systems is called the *Smart Grid*. More specifically, the Smart Grid aspires to use secure, two-way communication technologies to exploit valuable information across every stage, from generators and distribution networks to final consumption. This way we can achieve clean production, safety, security, reliability, resilience, efficiency and sustainability in the electricity grid [4]. In fact, the Smart Grid operation requires complex control and decision making, so it is common to represent each entity by *autonomous agents*. As Smart Grid entities we consider consumers, producers and *prosumers* of electricity.

A prosumer is an entity that both consumes and produces energy [11]. Residential prosumers in particular, are expected to be of much importance for the smartgrid era, as they possess increased awareness and the ability to adjust their behavior according to dynamic indicators [8]. There also exist commercial prosumers, industrial wind farms or electric car fleets for example, whose contribution in the electricity market also has a large impact. Due to the economic nature of a prosumer, it is desired to implement selfish and economically thinking autonomous agents that make effective decisions, regarding when to use which source, so that to meet demand at a minimum cost, or even make a profit by selling the electricity surplus to other entities that need it [15]. In this work, we present an algorithm that could be used by prosumers to efficiently manage the operation of their generating and consuming devices, so as to maximize their own profits while contributing to the stability of the overall electricity grid.

The proposed algorithm takes into account the cost of electricity generation by each available device or the charging of a utility company, as well as the predicted consumption and production levels of the facility, and generates an operation plan that results to the maximum profitability of the prosumer. As we will be explaining, the electricity generation cost function can be considered to be convex and so are the constraints that need to be met. Based on this feature, we develop a Primal-Dual interior point algorithm and find the optimal point, that is the *exact amount of energy that each controllable generating device must produce* in order to achieve the largest monetary profit.

The proposed algorithm is evaluated on three prosumer profiles that have different generation and storage characteristics. In our simulations we have also considered a variable pricing scheme for the electricity sold or bought by the utility company, as well as a scenario where the prosumers are not allowed to sell to the utility company. Results show that our method can be used by Smart Grid prosumers to manage their generation process effectively.

This report is structured as follows. Section 2 discusses some of the recent related work. In Section 3 a description of the problem is presented. In Section 4 we provide the prosumer profiles which we chose to evaluate. Section 5 presents the results from the experimental evaluation. Finally, in Section 6 we conclude.

2. Related Work

The implementation and operation of the Smart Grid is a large and new venture. From the modernization of the infrastructures, to the redesign of operational strategies, there are a lot to be done in order to end up with stable, secure and sustainable systems. The work of [13] discusses the need of intelligent online algorithms for the management of grids operation. They propose two methods for tackling uncertainties regarding the future, followed by an analysis based on regret. In our approach, we assume that there exist mechanisms that are able to precisely predict unknown or uncertain indicators e.g., the weather, the output of wind turbines and PV panels, as well as the price rates that a utility company charges electricity during a day.

As argued by authors of [8], “key benefit for the residential prosumer would be the possibility of optimizing his energy usage by adjusting his behaviour to dynamic market prices as well as have some financial benefits for the electricity he produces (and does not use)". Our method takes into consideration the aforementioned indicators and generates hourly prosumer operation plans that are cost-efficient.

In [12] authors aim to minimize grid's operational and generation costs, by solving a mixed-integer programming problem. We are also concerned about the same costs, but use convex optimization to obtain solutions.

The use of convex optimization methods in Smart Grid applications is not new. In [10], authors use convex optimization to select the best among 64 electricity storage portfolios, by minimizing a combined cost function. The portfolios, i.e. set-ups of different storage devices, are characterized by their capital cost and their average operating cost. Then, Pareto optimal portfolios are found, with respect to these two cost types. The difference with our approach is that we consider a single portfolio that can both store and generate electricity, and we minimize a single cost function, that indicates the capital cost of the system's operation, while meeting electricity demand. Our problem formulation is generic and can be configured to describe any portfolio, though.

The work of [6] describes a distributed convex optimization framework to manage energy flows between islanded microgrids. A scenario of two microgrids that exchange energy is presented. Electricity is transmitted and generated in an optimal manner, by minimizing the respective cost functions. Although authors include a distributed approach, we choose to use a multi-agent formulation, because it is more natural to use when modeling trades and markets. In our work we are concerned about the electricity generation costs of an economically minded prosumer only and seek a method that can be adopted by every prosumer in the Grid.

The most related work to ours is that of [16], where authors present VERA, that is a software system for energy management of contemporary microgrids. They present a cost and constraint formulation that is very close to that adopted in our work. Then, the minimization problem is solved using sequential quadratic programming. Similarly, we implement a Primal-Dual interior point method to solve the minimization problem which we prove that is convex.

We now proceed to a detailed description of our case.

3. Problem Formulation

We begin by defining the operating costs of every type of electricity generating or storage devices that are probable to find in a prosumer facility. The sum of these costs is the objective function which we want to minimize. In our model, we consider plans on a daily basis, that is for 24 time intervals, representing one hour of the day each.

3.1. Electricity Costs

A prosumer can employ any type of renewable energy generation system, such as wind turbines and solar panels. These generators generate power at each time interval t, and their output values are noted as P_t^w and $P_t^{pv} \in \mathbf{R}^+$, representing the KWh produced.

Electricity generated by such devices is considered to have zero cost. The amount of energy can be accurately estimated, though it can not be controlled. There are techniques to obtain estimates about renewable production size and we use them in this work. In more detail, the units, for both production and consumption, are modelled based on the work of [9]. Authors provide the methodology for optimal sizing of stand-alone photovoltaic/wind-generator systems using genetic algorithms, and also model the 24-hour consumption curve well enough 3.1. Note that there exist two peaks during the day, the first one at midday and the second one early at night. Furthermore, the work [5] presents a model of photovoltaic arrays based on the circuits that are contained in their cells.

The power output of a PV module depends on the number of cells in the module, the type of cells, and the total surface area of the cells. All modules are rated by manufacturers in terms of their peak power (Wp) under standard test conditions: ie. 1000 W/m^2 of sunlight ('peak sun'), T = 25 C, and air mass of 1.5. The type of the cell used in this study is Crystalline Silicon. Open circuit voltage is $Vg = 1.12eV$ and shot circuit current is $Isc = 5.75A$.

The second renewable energy unit that our system supports, wind-turbine, are based on the model that is created by [3]. The most important feature of the model is the swept area and the rotor diameter that characterize each wind-turbine.

The prosumer unit combines the above units in order to create a more realistic environment and simulates the behaviour of different units of the prosumer during the optimal management of the energy system. The modelling of the consumption based on real facts aims to bring the simulation closer to reality.

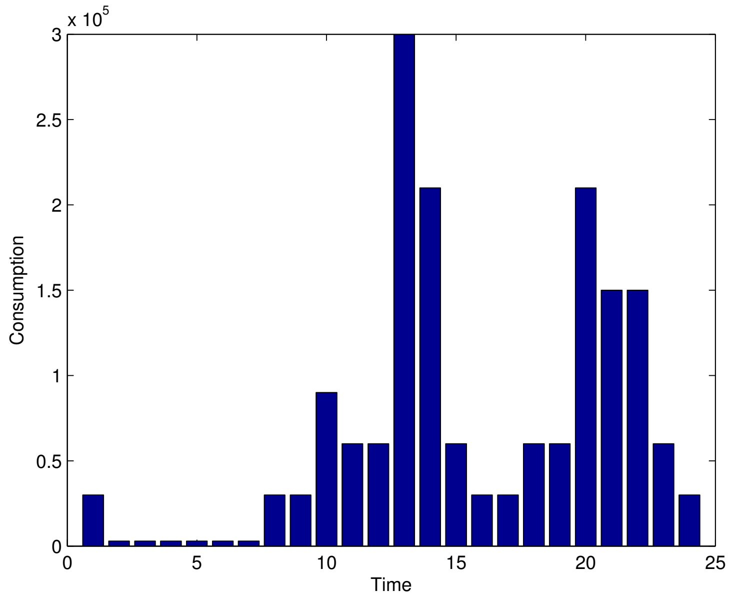


Figure 3.1: 24-hour Consumption Curve

The energy that the prosumer requests to consume at each t is noted as P_t^{cons}.

3.1.1. Utility Company

Prosumers can buy from or sell to a utility company that may exist externally. The pricing of the utility company might be either *flat* or *variable among time intervals* and is noted as c_t^{uc}. The amount of energy exchanged between the prosumer and the utility company at each time interval t is noted as $P_t^{uc} \in \mathbf{R}$ and it is measured in KWh. The product $c_t^{uc} \cdot P_t^{uc}$ is the total cost of the utility company services at a time interval t, and when its value is negative it denotes the profit of the prosumer for selling excessive energy.

3.1.2. Micro Turbines

There may also be micro turbines (MTs) available, that run on a variety of fuels, like biogas, diesel, or natural gas. Each MT has a maximum generation capability noted as $P_{\max}^{mt}$. For each time interval t, the amount of the energy that we need the microturbine to produce is P_t^{mt}, where $0 \leq P_t^{mt} \leq P_{\max}^{mt}$. The cost for producing energy using the MT is noted as c_t^{mt}, and is given by the sum of the fuel cost of the generator and the start-up cost of the generator.

The fuel cost of a typical diesel generator at the time interval t is usually expressed as [17]:

$$\hat{c}_t^{mt}(P_t^{mt}) = a \cdot (P_t^{mt})^2 + b \cdot P_t^{mt} + c$$

where, P_t^{mt} is the output power of the generator during t; a, b and c are constants determined by the type of generator, here $a = 0.0547$, $b = 1.7362$, $c = 3.2456$. A binary control signal $X_t \in \{0, 1\}$ is also defined, enabling the prosumer to decide when to turn the MT on or off. We must note that the generator has a startup cost SC_t^{mt} that is added to the overall generation cost c_t^{mt} at times when the generator needs to be turned on. Thus, each time interval t the MT generates P_t^{mt} KWh, the associated cost is $c_t^{mt}(P_t^{mt}) = X_t \cdot (\hat{c}_t^{mt}(P_t^{MT}) + SC_t^{mt})$.

3.1.3. Electricity Storage

Apart from generators, a prosumer can possess electricity storage systems, such as batteries. In the stationary battery operation, the amount of charged power corresponds to the surplus renewable energy sources output over the threshold for charge. The amount of discharged power corresponds to the surplus load power over the threshold for discharge. Hence, the amount of discharged power is restricted so that no reverse power flow is generated by the stationary batteries.

Each battery has a certain state of charge at every t that must be within certain levels:

$$SOC_t^{min} \leq SOC_t \leq SOC_t^{max}$$

The output power of the battery at t, P_t^b, is also subject to constraints:

$$P_{ch\_max}^b \leq P_t^b \leq P_{disch\_max}^b$$

where $P^b_{ch\_max}$ is the maximum charging capability of the battery and $P^b_{disch\_max}$ its maximum discharge capability. $Cap_{\max}$ denotes the battery capacity. The state of charge for the next time interval depends on P^b_t:

$$SOC_t = SOC_{t-1} - \frac{P^b_t}{Cap_{\max}}$$

3.1.4. Cost Function

The operating cost at a time interval for producing $\mathbf{x} = [P^{uc}, P^{mt}, P^b]$ amount of energy, is given by:

$$Cost = c^{uc} \cdot P^{uc} + \sum_{j=1}^{n} c^{mt}_j (P^{mt}_j)$$

where n is the number of micro turbines.

3.1.5. Convex Minimization Problem

Our main concern in this work is to keep the generation costs of $\mathbf{x} = [P^{uc}, P^{mt}, P^b]$ at a minimum, while satisfying specific constraints.

First of all, demand must be met at all times, i.e.:

$$P^{uc} + P^{mt} + P^b = P^{cons} - P^w - P^{pv}$$

Next, the battery needs to operate at certain levels. The constraints are the following:

$$-P^b - P^b_{ch\_max} \leq 0$$

$$-P^b - (1 - SOC_{t-1}) \cdot Cap_{max} \leq 0$$
$$P^b - (SOC_{t-1} - 0.2) \cdot Cap_{max} \leq 0$$
$$P^b - SOC_{t-1} \cdot Cap_{max} \leq 0$$
$$P^b - P^b_{disch\_max} \leq 0$$

that is, we can charge no more than the amount of load stored in the battery, charged during a time interval has a maximum value that depends on the battery, state of charge must not drop below 0.2, we can discharge no more than the amount of load stored in the battery, and the electricity discharged during a time interval has a maximum value that depends on the battery. The above inequalities are all affine.

Consider that there are constraints regarding the MT operation too. More specifically,

$$P^{mt} \geq 0$$

$$P^{mt} - X \cdot P^{mt}_{\max} \leq 0$$

that is, we can only produce energy and not store, and there is a limit in the amount of production governed by the specifications of each MT. Once again the inequalities are affine.

The amount of energy exchanged with the grid is unlimited in the general case, but in the experimental section we test a scenario where P^{uc} is strictly positive, and another one that is strictly negative.

Finally, we examine the convexity of the cost function. To begin, the domain of the cost function is $\mathbf{R}$, that is convex. The first term of the sum, is an affine combination of P^{uc} and can be considered convex. The second term is quadratic functions of P^{mt}. The second derivative of the MT term is 2α, $\alpha > 0$, that is positive, thus the function is convex. The sum of convex functions is also convex, concluding with the convexity of the cost function.

Summarizing, to find the minimum cost, we must obtain the appropriate values for

$$\mathbf{x} = [P^{uc}, P^{mt}]$$

namely:

$$\begin{aligned}
\min_{\mathbf{x}} \quad & f(\mathbf{x}) = c^{uc} \cdot P^{uc} + \sum_{j=1}^{n} c_j^{mt}\left(P_j^{mt}\right) \\
\text{subject to} \quad & P^{uc} + \sum_{j=1}^{n}(P_j^{mt}) + \sum_{j=1}^{m}(P_j^{b}) + P^{w} + P^{pv} - P^{cons} = 0 \\
& -P_j^b - P^b_{ch_max,j} \leq 0 \ , \forall j \in M \\
& -P_j^b - (1 - SOC_{t-1,j}) \cdot Cap_{max,j} \leq 0 \ , \forall j \in M \\
& P_j^b - (SOC_{t-1,j} - 0.2) \cdot Cap_{max,j} \leq 0 \ , \forall j \in M \\
& P_j^b - SOC_{t-1,j} \cdot Cap_{max,j} \leq 0 \ , \forall j \in M \\
& P_j^b - P^b_{disch_max,j} \leq 0 \ , \forall j \in M \\
& -P_j^{mt} \leq 0 \ , \forall j \in N \\
& P_j^{mt} - X \cdot P^{mt}_{\max,j} \leq 0 \ , \forall j \in N
\end{aligned} \tag{3.1}$$

For ease of presentation, from now on we assume that there exist one MT and one battery array.

3.2. Primal-Dual Method

For the minimization procedure we use the Primal-Dual interior point method [2]. The linear system that we solve to obtain the search direction $\boldsymbol{\Delta} x_{pd}$, in order to minimize the problem (3.3) is given by:

$$\begin{bmatrix} \nabla^2 f_0(x) + \sum \lambda_i \nabla^2 f_i(x) & Df(x)^T & A^T \\ -diag(\lambda) Df(x) & -diag(-f(x)) & 0 \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\Delta} x \\ \boldsymbol{\Delta} \lambda \\ \boldsymbol{\Delta} v \end{bmatrix} = - \begin{bmatrix} \nabla f_0(x) + Df(x)^T \lambda + A^T v \\ -diag(\lambda) f(x) - \frac{1}{t}\mathbf{1} \\ Ax - b \end{bmatrix} \tag{3.2}$$

Prosumer is permitted to buy & sell to the Utility

We have two different cases, one for MT control signal $X = 0$ and another for $X = 1$. When the MT is turned on ($X = 1$):

$$\nabla^2 f_0(x) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2\alpha & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$\nabla^2 f_i(x) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \forall i$$

$$f(x) = \begin{bmatrix} P^{mt} - P_{\max} \\ -P^{mt} \\ -P^b - P^b_{ch\_max} \\ -P^b - (1 - SOC_{t-1}) \cdot Cap_{max} \\ P^b - (SOC_{t-1} - 0.2) \cdot Cap_{max} \\ P^b - SOC_{t-1} \cdot Cap_{max} \\ P^b - P^b_{disch\_max} \end{bmatrix}$$

$$Df(x) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\lambda = -\frac{1}{f(x)}$$

$$A = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

$$b = P^{cons} - P^w - P^{pv}$$

In the case it is turned off ($X = 0$):

$$\nabla^2 f_0(x) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix},$$

$$\nabla^2 f_i(x) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$f(x) = \begin{bmatrix} -P^b - P^b_{ch\_max} \\ -P^b - (1 - SOC_{t-1}) \cdot Cap_{max} \\ P^b - (SOC_{t-1} - 0.2) \cdot Cap_{max} \\ P^b - SOC_{t-1} \cdot Cap_{max} \\ P^b - P^b_{disch\_max} \end{bmatrix}$$

$$Df(x) = \begin{bmatrix} 0 & -1 \\ 0 & -1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

$$\lambda = -\frac{1}{f(x)}$$

$$A = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

$$b = P^{cons} - P^{w} - P^{pv}$$

Prosumer is not permitted to sell to the Utility

This problem is slightly different from the previous one. When the renewable energy sources produce more power than the power that consumers need, surplus is created. However the prosumer is not permitted to sell that amount of power to the utility. Hence, "Dump Load" is created. As a result of that, we have to create a new decision variable maps to the amount of power of the "Dump Load". Consider that amount of power as P^{dump}. Let formulate the problem again:

$$\begin{aligned}
\min_{\mathbf{x}} \quad & f(\mathbf{x}) = c^{uc} \cdot P^{uc} + \sum_{j=1}^{n} c_j^{mt}\left(P_j^{mt}\right) \\
\text{subject to} \quad & P^{uc} + \sum_{j=1}^{n}(P_j^{mt}) + \sum_{j=1}^{m}(P_j^{b}) + P^{w} + P^{pv} - P^{cons} - P^{dump} = 0 \\
& -P_j^{b} - P_{ch_max,j}^{b} \leq 0 \ , \forall j \in M \\
& -P_j^{b} - (1 - SOC_{t-1,j}) \cdot Cap_{max,j} \leq 0 \ , \forall j \in M \\
& P_j^{b} - (SOC_{t-1,j} - 0.2) \cdot Cap_{max,j} \leq 0 \ , \forall j \in M \\
& P_j^{b} - SOC_{t-1,j} \cdot Cap_{max,j} \leq 0 \ , \forall j \in M \\
& P_j^{b} - P_{disch_max,j}^{b} \leq 0 \ , \forall j \in M \\
& -P_j^{mt} \leq 0 \ , \forall j \in N \\
& P_j^{mt} - X \cdot P_{\max,j}^{mt} \leq 0 \ , \forall j \in N \\
& -P_j^{dump} \leq 0 \ , \forall j \in N
\end{aligned} \tag{3.3}$$

We have two different cases, one for MT control signal $X = 0$ and another for $X = 1$. When the MT is turned on ($X = 1$):

$$\nabla^2 f_0(x) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 2\alpha & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$\nabla^2 f_i(x) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \forall i$$

$$f(x) = \begin{bmatrix} P^{mt} - P_{\max} \\ -P^{mt} \\ -P^b - P^b_{ch\_max} \\ -P^b - (1 - SOC_{t-1}) \cdot Cap_{max} \\ P^b - (SOC_{t-1} - 0.2) \cdot Cap_{max} \\ P^b - SOC_{t-1} \cdot Cap_{max} \\ P^b - P^b_{disch\_max} \\ -P^{dump} \end{bmatrix}$$

$$Df(x) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

$$\lambda = -\frac{1}{f(x)}$$

$$A = \begin{bmatrix} 1 & 1 & 1 & -1 \end{bmatrix}$$

$$b = P^{cons} - P^w - P^{pv}$$

In the case it is turned off ($X = 0$):

$$\nabla^2 f_0(x) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$\nabla^2 f_i(x) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$f(x) = \begin{bmatrix} -P^b - P^b_{ch\_max} \\ -P^b - (1 - SOC_{t-1}) \cdot Cap_{max} \\ P^b - (SOC_{t-1} - 0.2) \cdot Cap_{max} \\ P^b - SOC_{t-1} \cdot Cap_{max} \\ P^b - P^b_{disch\_max} \\ -P^{dump} \end{bmatrix}$$

$$Df(x) = \begin{bmatrix} 0 & -1 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

$$\lambda = -\frac{1}{f(x)}$$

$$A = \begin{bmatrix} 1 & 1 & -1 \end{bmatrix}$$

$$b = P^{cons} - P^{w} - P^{pv}$$

The Primal dual algorithm is given as:

Algorithm 1 Primal Dual

Input: $\mathbf{x}$ satisfying $f_1(\mathbf{x}) < 0$, ..., $f_n(\mathbf{x}) < 0, \lambda > 0$, $\mu > 1$, $\epsilon > 0$
while TRUE **do**
  Determine t, $t := n\mu/\hat{\eta}$
  Compute primal-dual search direction Δy_{pd}
  Line search and update
  Quit if $||r_{pri}||_2 \leq \epsilon, ||r_{dual}||_2 \leq \epsilon, \hat{\eta} \leq \epsilon$
end while

4. Prosumer Profiles

The unit of the prosumer includes photovoltaic arrays, wind turbines, storage unit (batteries) and micro-turbines (in our scenario micro-turbines are diesel generators) to produce power. In our setting we consider three different prosumers, each possessing electricity generation units of different capabilities.

More specifically, the capabilities of each prosumer are summarized in Table 4.1.

Prosumer Id	Battery Capacity	MT Capability
a	80000	200000
b	100000	100000
c	250000	200000

Table 4.1: Prosumer equipment.

The price of the utility company for each time interval is a uniformly distributed random variable normalized in the range (0.0785 – 0.09) Euros/KWh, modeling the recent trend on variable pricing.

We now proceed with the experimental evaluation of our setting.

5. Experimental Results

In this section we test the performance of the Primal-Dual algorithm we presented, when adopted by the prosumers we discussed above. Each experiment simulates the energy flow management for 24-hours. We consider two scenarios:

1. Prosumers can both buy and sell electricity to the utility company.

2. Prosumers only buys (and not sells) electricity from the utility company.

5.1. Scenario 1

The typical costs of the prosumer profiles for a single day, according to the first scenario where prosumers fully interact with the utility company, are presented in Table 5.1.

Prosumer Id	Daily profit
a	39.20 Euros
b	57.63 Euros
c	43.74 Euros

Table 5.1: Generation costs when the prosumers fully interact with the utility company (buy and sell).

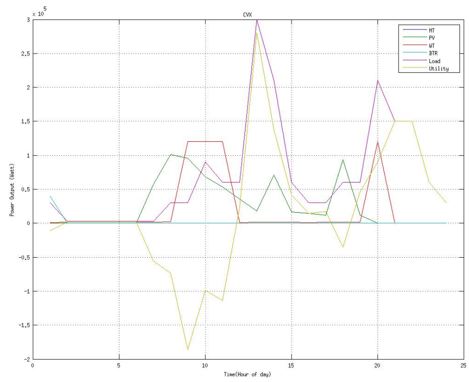
The algorithm "weighs" the generation costs of each available device and chooses a discharging profile for each of them that generates the requested amount of energy at a minimum cost. The minimum cost operation profiles are shown in Figure 5.1.

5.2. Scenario 2

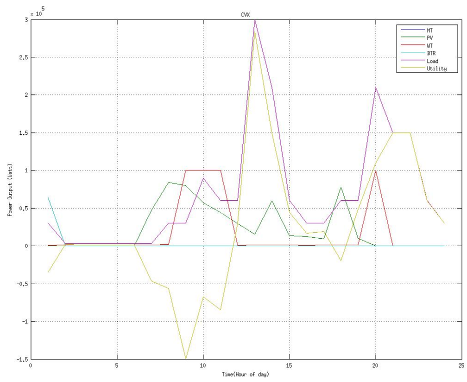
In this case, prosumers are not able to sell their energy surplus to the utility company, and thus any excess is considered dump load. Here, the algorithm actually tries to minimize the amount of the dump load, as the prosumer looses its produced energy. It is natural that the prosumers in this scenario have increased costs, because the utility company does not pay them. The prosumer costs for a typical day are shown in 5.2. The minimum cost operation profiles are shown in Figure 5.2.

Prosumer Id	Daily profit
a	82.96 Euros
b	68.21 Euros
c	86.08 Euros

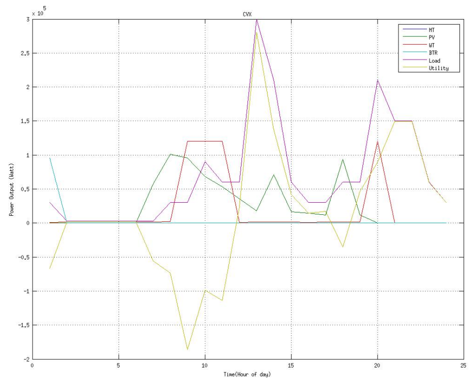
Table 5.2: Generation costs when the prosumers only buy electricity from the utility company.



(a) Prosumer a

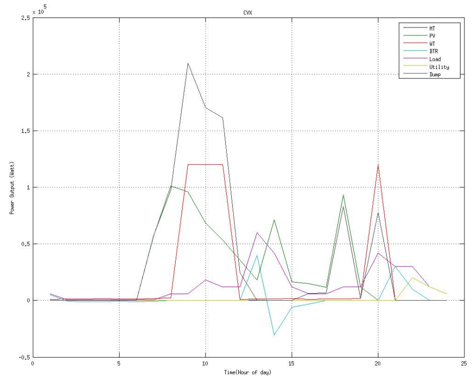


(b) Prosumer b

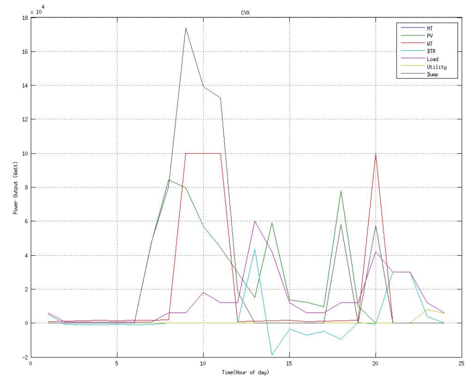


(c) Prosumer c

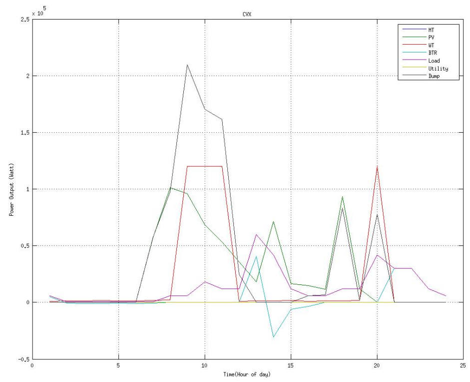
Figure 5.1: Operation levels of each source during the day (scenario 1).



(a) Prosumer a



(b) Prosumer b



(c) Prosumer c

Figure 5.2: Operation levels of each source during the day (scenario 2).

6. Conclusions and Future Work

In this work, we presented a method for minimizing the operating costs of prosumer entities in the Smart Electricity Grid. We studied the application of convex optimization techniques in Smart Grid environments that have been applied in past work. A problem formulation is provided, which we prove that is convex. For the solution of the minimization problem we constructed a Primal-Dual algorithm and effectively applied it in our simulations of realistic prosumer profiles.

There are many additions that can be researched in the near future. Firstly, it will be interesting to explore the behavior of a larger number of prosumers, who are trying to exchange energy with a retailer instead of the utility directly, while the retailer might have contradicting interests. Secondly, the use of different type of micro-turbines, as well as, different type of batteries, will make the simulation even more realistic. Furthermore, the possibilities of non-convex scenarios is quite interesting, specifically the ones where we can convert them to covnex ones, and thus efficiently solve them.

References

[1] Peter Asmus. Microgrids, virtual power plants and our distributed energy future. *The Electricity Journal*, 23(10):72–82, 2010.

[2] Stephen Poythress Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[3] Zhe Chen, J.M. Guerrero, and F. Blaabjerg. A review of the state of the art of power electronics for wind turbines. *Power Electronics, IEEE Transactions on*, 24(8):1859–1875, 2009.

[4] Xi Fang, Satyajayant Misra, Guoliang Xue, and Dejun Yang. Smart gridâĂŤthe new and improved power grid: a survey. 2011.

[5] Francisco M González-Longatt. Model of photovoltaic module in matlab. *II CIBELEC*, 2005:1–5, 2005.

[6] David Gregoratti and Javier Matamoros. Distributed convex optimization of energy flows: The two-microgrid case. In *Communications and Networking (BlackSeaCom), 2013 First International Black Sea Conference on*, pages 201–205. IEEE, 2013.

[7] Mark Z Jacobson and Mark A Delucchi. A path to sustainable energy by 2030. *Scientific American*, 301(5):58–65, 2009.

[8] S. Karnouskos. Demand side management via prosumer interactions in a smart city energy marketplace. In *Innovative Smart Grid Technologies (ISGT Europe), 2011 2nd IEEE PES International Conference and Exhibition on*, pages 1–7, 2011.

[9] Eftichios Koutroulis, Dionissia Kolokotsa, Antonis Potirakis, and Kostas Kalaitzakis. Methodology for optimal sizing of stand-alone photovoltaic/wind-generator systems using genetic algorithms. *Solar Energy*, 80(9):1072 – 1088, 2006.

[10] Matt Kraning, Yang Wang, Ekine Akuiyibo, and Stephen Boyd. Operation and configuration of a storage portfolio via convex optimization. In *Proceedings of the 18th IFAC World Congress*, pages 10487–10492, 2011.

[11] I. Lampropoulos, G. M A Vanalme, and W.L. Kling. A methodology for modeling the behavior of electricity prosumers within the smart grid. In *Innovative Smart Grid Technologies Conference Europe (ISGT Europe), 2010 IEEE PES*, pages 1–8, 2010.

[12] Lian Lu, Jinlong Tu, Chi-Kin Chau, Minghua Chen, Zhao Xu, and Xiaojun Lin. Towards real-time energy generation scheduling in microgrids with performance guarantee. In *Power and Energy Society General Meeting (PES), 2013 IEEE*, pages 1–5. IEEE, 2013.

[13] Balakrishnan Narayanaswamy, Vikas K Garg, and TS Jayram. Online optimization for the smart (micro) grid. In *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*, page 19. ACM, 2012.

[14] G. Pepermans, J. Driesen, D. Haeseldonckx, R. Belmans, and W. DâĂŹhaeseleer. Distributed generation: definition, benefits and issues. *Energy Policy*, 33(6):787 – 798, 2005.

[15] Sarvapali D Ramchurn, Perukrishnen Vytelingum, Alex Rogers, and Nicholas R Jennings. Putting the'smarts' into the smart grid: a grand challenge for artificial intelligence. *Communications of the ACM*, 55(4):86–97, 2012.

[16] Petr Stluka, Datta Godbole, and Tariq Samad. Energy management for buildings and microgrids. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, pages 5150–5157. IEEE, 2011.

[17] Mian Xing, Ling Ji, and Baiting Xu. Multiple objective optimizations for energy management system under uncertainties. *TELKOMNIKA Indonesian Journal of Electrical Engineering*, 11(12):7044–7051, 2013.

A. Matlab Scripts

A.1. First Prosumer's Profile

B. Energy Management System – CVX – Primal Dual Interior Point Method

```
clear all; close all; clc;

%% READ CSV FILE
row=1;

numm=[  120000   -100000  120000   120000   40000
        80000    -75000   80000    80000    750000
        50000    -45000   120000   120000   80000   ];

for row=1:3
%% READ DATA
fprintf('Simulation Started\n\n') ;

num = csvread('Data.csv');

BATTpmax = num(row,1);
BATTpmin = num(row,2);
PVpmax = num(row,3);
WTpmax = num(row,4);
MTpmax = num(row,5);

%% TIME STEPS, AC BUS
time_step = 24;
totalUnits = 5;
p_bus = zeros(totalUnits,time_step);
index_GenMT = 1; index_GenPV = 2; index_GenWT = 3; index_GenBTR
    = 4; index_Load = 5; index_Util = 6;

% MICROURBINE
mut = 6000;% minimum up time (sec)
mdt = 200;% minimum down time (sec)
ton=0;% counting the times turned on
toff=0;% counting the times turned off
statusMT=ones(1,time_step);

%% WIND TURBINE
```

```
statusWT=ones(1,time_step);

%% BATTERY
Vb=12;% initial voltage
pbold = 0;
[soc_cvx, qq1, qq2,num_of_batt] = battery( 0, pbold, BATTpmax);

%% UTILITY
vwind = [10.12 11.654 12.43 12.7654 12.123 12.5643 12.626
    13.845 15.113 16.322 17.11 10.12 11.654 12.43 12.7654 10.12
    11.654 12.43 12.7654 24.6363 25.346 26.7562 27.23 28.88];
irrad = [0.0075 0.3948 0.674 0.789 0.519 0.555  0.493  0.935
    0.891 0.583 0.514 0.35 0.169 0.6472  0.1508 0.1324 0.10385
    0.8206 0.1069 0.3603 0.7578 0.8679 0.0767 0.5666];
cutil = 10^(-3)*[0.0787 0.0785 0.0785 0.0789 0.088 0.088 0.089
    0.09 0.0885 0.0805 0.0815 0.0825 0.079 0.078 0.089 0.0866
    0.0799 0.082 0.08 0.085 0.09 0.0788 0.0788 0.085];
tempr = [33.57  66.38  54.61  65.75 42.07 70.09 30.81  54.57
    57.26  28.02 65.66  72.92 62.93  49.59  54.64 43.63 32.40
    39.59  56.84  47.46 68.34 59.68 55.18 44.52];
stp=3600;
%% EVALUATION
for i=1:time_step
     %% CONSTRAINTS MT
    if( ton >= mut/stp )
         statusMT(i) = 0;
     end

     if( toff >= mdt/stp )
         statusMT(i) = 1;
     end

     if( statusMT(i) == 0 )
         ton=0;
         toff=toff+1;
         if(i<length(statusMT))
             statusMT(i+1)=0;
         end
     else
         toff = 0;
         ton = ton+1;
         if(i<length(statusMT))
             statusMT(i+1)=1;
         end
```

```
    end

    %% AC BUS
    p_bus(index_GenWT,i) = windturbine(vwind(i),statusWT(i),WTpmax);
    p_bus(index_Load,i) = power_bus_per_dayhour(time_step,i);
    if(mod(i,time_step)<=19 && mod(i,time_step)>=7)
        p_bus(index_GenPV,i) = pv_array(irrad(i),tempr(i),PVpmax);
    end

    if soc_cvx<=0.201
       soc_cvx=0.201;
    end
    %% CVX
%       n=3;
%       cvx_begin
%           variable x(n,1)
%           minimize objFun(x(1), x(2), statusMT, toff, cutil(i), i);
%           subject to
%           x(1)+x(2)+x(3)*Vb==p_bus(index_Load,i)-(p_bus(index_GenWT,i)+p_bus(index_GenPV,i));
%           %-x(3)<=0;
%           %#############################
%           -x(3)*Vb-abs(BATTpmin)<=0;
%           -x(3)*Vb-(1-soc_cvx)*BATTpmax<=0;
%           %#############################
%           x(3)*Vb-(soc_cvx-0.2)*BATTpmax<=0;
%           x(3)*Vb-soc_cvx*BATTpmax<=0;
%           x(3)*Vb-abs(BATTpmin)<=0;
%           x(2)-MTpmax*statusMT(i)<=0;
%           -x(2)<=0;
%       cvx_end
%       x_cvx_x = cvx_optpnt.x;
%       fprintf('\nCVX finished. ');

    %% PRIMAL DUAL INTERIOR POINT METHOD
    clear r_t_d r_t_c r_t_p r_t
    clear array_mtr
    clear Dy Dx_pd Dlamda_pd Dv_pd
    clear x lamda  v
    clear eta tau f Df
```

```
    p=1;
    mu=10;
    alpha = 1e-2;
    beta = 5e-1;
    epsilon_feas = 1e-6;
    epsilon = 1e-6;

    b = p_bus(index_Load,i)-p_bus(index_GenWT,i)-p_bus(
        index_GenPV,i);

    v = rand(p,1);

    if statusMT(i)==1
        m=7;
        n=3;
        A = [1 1 Vb];
        x = rand(n,1);

        f = [ x(2)-MTpmax;
              -x(2);
              -x(3)*Vb-abs(BATTpmin);
              -x(3)*Vb-(1-soc_cvx)*BATTpmax;
              x(3)*Vb-(soc_cvx-0.2)*BATTpmax;
              x(3)*Vb-soc_cvx*BATTpmax;
              x(3)*Vb-abs(BATTpmin)];

        Df = [0 1 0;
              0 -1 0;
              0 0 -Vb;
              0 0 -Vb;
              0 0 Vb;
              0 0 Vb;
              0 0 Vb];
    else
        m=5;
        n=2;
        A = [1 Vb];
        x = rand(n,1);

        f = [ -x(2)*Vb-abs(BATTpmin);
              -x(2)*Vb-(1-soc_cvx)*BATTpmax;
              x(2)*Vb-(soc_cvx-0.2)*BATTpmax;
              x(2)*Vb-soc_cvx*BATTpmax;
              x(2)*Vb-abs(BATTpmin)];
```

```
        Df = [0 -Vb;
              0 -Vb;
              0 Vb;
              0 Vb;
              0 Vb];
    end

    Hessian = zeros(m,n);
    Hessian_f0 = D2_f_obj(statusMT(i));

    lamda = -1./f;

    k = 1;

    %% Repeat
    while (1)

        %% DETERMINE tau
        if statusMT(i)==1
              f = [ x(2,k)-MTpmax;
              -x(2,k);
              -x(3,k)*Vb-abs(BATTpmin);
              -x(3,k)*Vb-(1-soc_cvx)*BATTpmax;
              x(3,k)*Vb-(soc_cvx-0.2)*BATTpmax;
              x(3,k)*Vb-soc_cvx*BATTpmax;
              x(3,k)*Vb-abs(BATTpmin)];

        else
                f = [ -x(2,k)*Vb-abs(BATTpmin);
              -x(2,k)*Vb-(1-soc_cvx)*BATTpmax;
              x(2,k)*Vb-(soc_cvx-0.2)*BATTpmax;
              x(2,k)*Vb-soc_cvx*BATTpmax;
              x(2,k)*Vb-abs(BATTpmin)];
        end

        eta(k) = -(f)' * lamda(:,k);
        tau = (mu * n)/eta(k);
        t_max = 1;
        t = 0.99*t_max;
        %% UPDATE residuals
        if statusMT(i)
              r_t_d(:,k) = D_f_obj(x(2,k), statusMT(i), cutil(i))
                    + Df'*lamda(:,k) + A'*v(:,k);
```

```
            r_t_c(:,k) = -diag(lamda(:,k))*f - (1/tau)*ones(m,1);
            r_t_p(:,k) = A*x(:,k)-b;
            r_t(:,k) = [r_t_d(:,k); r_t_c(:,k); r_t_p(:,k)];
        else
            r_t_d(:,k) = D_f_obj(0, statusMT(i), cutil(i)) + Df'*lamda(:,k) + A'*v(:,k);
            r_t_c(:,k) = -diag(lamda(:,k))*f - (1/tau)*ones(m,1);
            r_t_p(:,k) = A*x(:,k)-b;
            r_t(:,k) = [r_t_d(:,k); r_t_c(:,k); r_t_p(:,k)];
        end

        %% UPDATE the invertible matrix for ÎŤy
        array_mtr = [Hessian_f0 Df' A';
            -diag(lamda(:,k))*Df -diag(f) zeros(m,p);
            A zeros(p,m) zeros(p,p)];

        %% UPDATE ÎŤy
        Dy = -inv(array_mtr)*r_t(:,k);
        Dx_pd = Dy(1:n);
        Dlamda_pd = Dy(n+1:n+m);
        Dv_pd = Dy(n+m+1:size(Dy,1));

        %% Îż+DÎż>0
        while ~( sum( (lamda(:,k)+t*Dlamda_pd) > 0 ) == m )
            t=beta*t;
        end

        %% f(x+)<0
        if statusMT(i)==1
            f = [ (x(2,k)+t*Dx_pd(2))-MTpmax;
            -(x(2,k)+t*Dx_pd(2));
            -(x(3,k)+t*Dx_pd(3))*Vb-abs(BATTpmin);
            -(x(3,k)+t*Dx_pd(3))*Vb-(1-soc_cvx)*BATTpmax;
            (x(3,k)+t*Dx_pd(3))*Vb-(soc_cvx-0.2)*BATTpmax;
            (x(3,k)+t*Dx_pd(3))*Vb-soc_cvx*BATTpmax;
            (x(3,k)+t*Dx_pd(3))*Vb-abs(BATTpmin)];
        else
           f = [  -(x(2,k)+t*Dx_pd(2))*Vb-abs(BATTpmin);
                -(x(2,k)+t*Dx_pd(2))*Vb-(1-soc_cvx)*BATTpmax;
               (x(2,k)+t*Dx_pd(2))*Vb-(soc_cvx-0.2)*BATTpmax;
            (x(2,k)+t*Dx_pd(2))*Vb-soc_cvx*BATTpmax;
```

```
        (x(2,k)+t*Dx_pd(2))*Vb-abs(BATTpmin)];
    end

    while ~( sum ( f < 0 ) == m )
        t=beta*t;

        if statusMT(i)==1
            f = [ (x(2,k)+t*Dx_pd(2))-MTpmax;
            -(x(2,k)+t*Dx_pd(2));
            -(x(3,k)+t*Dx_pd(3))*Vb-abs(BATTpmin);
            -(x(3,k)+t*Dx_pd(3))*Vb-(1-soc_cvx)*BATTpmax;
            (x(3,k)+t*Dx_pd(3))*Vb-(soc_cvx-0.2)*BATTpmax;
            (x(3,k)+t*Dx_pd(3))*Vb-soc_cvx*BATTpmax;
            (x(3,k)+t*Dx_pd(3))*Vb-abs(BATTpmin)];
        else
           f = [ -(x(2,k)+t*Dx_pd(2))*Vb-abs(BATTpmin);
            -(x(2,k)+t*Dx_pd(2))*Vb-(1-soc_cvx)*BATTpmax;
                (x(2,k)+t*Dx_pd(2))*Vb-(soc_cvx-0.2)*BATTpmax;
                (x(2,k)+t*Dx_pd(2))*Vb-soc_cvx*BATTpmax;
                (x(2,k)+t*Dx_pd(2))*Vb-abs(BATTpmin)];
        end

    end

    %% BACKTRACKING
    while norm( func_res( x(:,k) + t*Dx_pd, lamda(:,k) + t*Dlamda_pd, v(:,k) + t*Dv_pd, A, b, m, tau, statusMT(i), Vb, MTpmax, cutil(i), BATTpmax,BATTpmin,soc_cvx ) ) >...
            (1-alpha*t) * norm(r_t(:,k))
        t = beta*t;
    end

    %% UPDATE x,Îż,v
    x(:,k+1) = x(:,k) + t*Dx_pd ;
    lamda(:,k+1) = lamda(:,k) + t*Dlamda_pd;
    v(:,k+1) = v(:,k) + t*Dv_pd;

    %% TERMINATION CONDITION
    if norm(r_t_p(:,k))<=epsilon_feas && norm(r_t_d(:,k))<=epsilon_feas && eta(k)<=epsilon
        break;
```

```
            end
            k = k+1;
        end

        if statusMT(i)==1
            [optimum_primal_dual_interior_point , Futility_prim, Fgen_prim, suc_prim] = objFun(x(1,k), x(2,k), statusMT, toff,cutil(i),i);
        else
            [optimum_primal_dual_interior_point , Futility_prim, Fgen_prim, suc_prim] = objFun(x(1,k), 0, statusMT, toff,cutil(i),i);
        end

        fprintf('\n\nPrimal Dual interior point method finished. ');

        fprintf('\n\nOptimum Primal Dual Interior Point: %d', optimum_primal_dual_interior_point);

%       fprintf('\n\nOptimal Points from CVX: [ %d %d %d ]', x_cvx_x(1),x_cvx_x(2),x_cvx_x(3));

        if statusMT(i)==1
            fprintf('\n\nOptimal Points from Primal Dual Interior Point: [ %d %d %d ]',x(1,k),x(2,k),x(3,k));
        else
            fprintf('\n\nOptimal Points from Primal Dual Interior Point: [ %d 0 %d ]',x(1,k),x(2,k));
        end
        %% WRITE TO BUS
%           p_bus(index_Util,i) = x_cvx_x(1);
%           p_bus(index_GenMT,i) = x_cvx_x(2);
%           p_bus(index_GenBTR,i) = x_cvx_x(3)*Vb;
%           val_obj(i) = cvx_optval;

            p_bus(index_Util,i) = x(1,k);
            if statusMT(i)==1
                p_bus(index_GenMT,i) = x(2,k);
                p_bus(index_GenBTR,i) = x(3,k)*Vb;
            else
                p_bus(index_GenMT,i) = 0;
                p_bus(index_GenBTR,i) = x(2,k)*Vb;
            end
```

```
        val_obj(i) = optimum_primal_dual_interior_point;
    %% UPDATE BATTERY
    pbold = [pbold  p_bus(index_GenBTR,i)];
    [soc_cvx, pout_btr, qq3, num_of_batt] = battery( p_bus(index_GenBTR,i), pbold(1:end-1), BATTpmax);
end

loss_24_hour = sum(val_obj)
num(row,5+i+1) = loss_24_hour;
num(row,6:5+i) = val_obj;
csvwrite('Data.csv',num)
FileName =  sprintf('%d_%d_%d_%d_DATA.csv', BATTpmax, PVpmax, WTpmax, MTpmax);
csvwrite(FileName,p_bus)
plot_from_excel(FileName);
fprintf('Simulation Ended\n\n') ;
end
```

C. Objective Function

```
function [ObjFun,Futility,Fgen,suc] = objFun(putil, pmt , status_mt, toff,cutil,index)

    if(index==1)
        statusMT = status_mt(1);
        statusMT_prev = 0;
    else
        statusMT = status_mt(index);
        statusMT_prev = status_mt(index-1);
    end

    %% UTILITY
    Futility = cutil' .* putil;

    %% GENERATOR MAINTAINANCE AND START-UP
    stp = 3600; %
    hot_startup = 30; %sec
    cold_startup = 200; %sec

    cooling_time = 520; %sec

    if statusMT_prev==0 && statusMT==1
        suc = ( ( hot_startup / stp ) + ( cold_startup / stp ) * (1 - exp ( - toff / (cooling_time/stp) )) ); %
```

```
            start-up cost
    else
        suc=0;
    end

    alpha=0.0074; beta=0.2333; cu=0.4333;
    Fgen = ( alpha * (pmt*10^(-3)).^2 + beta * pmt*10^(-3) + cu
        + suc) .* statusMT;

    %% OBJECTIVE FUNCTION
     ObjFun = sum( Futility + Fgen); % $/Wh
end
```

D. Gradient of Objective

```
function [gradient] = D_f_obj(pmt, statusMT,cutil)
    alpha=0.0074*10^(-3); beta=0.2333*10^(-3);

    if statusMT==1
        gradient = [cutil ( 2*alpha*pmt + beta ) 0]';
    else
        gradient = [cutil 0]';
    end
end
```

E. Hessian of Objective

```
function [Hess] = D2_f_obj(statusMT)
    alpha=0.0074*10^(-3);
    if statusMT==1
        Hess = [0 0 0; 0 2*alpha 0; 0 0 0];
    else
        Hess = [0 0;0 0];
    end
end
```

F. Calculation of Residuals

```
function r_t = func_res(x,lamda,v,A,b,m,tau,statusMT,Vb,MTpmax,
    cutil,BATTpmax,BATTpmin,soc)

    if statusMT==1
        f = [ x(2)-MTpmax;
              -x(2);
```

```
                -x(3)*Vb-abs(BATTpmin);
                -x(3)*Vb-(1-soc)*BATTpmax;
                x(3)*Vb-(soc-0.2)*BATTpmax;
                x(3)*Vb-soc*BATTpmax;
                x(3)*Vb-abs(BATTpmin)];

            Df = [0  1  0;
                0  -1  0;
                0  0  -Vb;
                0  0  -Vb;
                0  0  Vb;
                0  0  Vb;
                0  0  Vb];

            r_t_d = D_f_obj(x(2), statusMT, cutil) + Df'*lamda + A'*v;

        else
            f = [ -x(2)*Vb-abs(BATTpmin);
                -x(2)*Vb-(1-soc)*BATTpmax;
                x(2)*Vb-(soc-0.2)*BATTpmax;
                x(2)*Vb-soc*BATTpmax;
                x(2)*Vb-abs(BATTpmin)];

            Df = [0  -Vb;
                0  -Vb;
                0  Vb;
                0  Vb;
                0  Vb];

            r_t_d = D_f_obj(0, statusMT, cutil) + Df'*lamda + A'*v;

        end

        r_t_c = -diag(lamda)*f - (1/tau)*ones(m,1);
        r_t_p = A*x-b;
        r_t = [r_t_d; r_t_c; r_t_p];
end
```

G. DC AC Inverter

```
function Pout = dc_ac_inverter(P)
        Pout=0.97*P;
end
```

H. Battery Unit

```
function [soc, pout, Ah_cur_con, K] = battery( pbnew, pbold,
    pbmax)

%      nb=0.9;      %efficiency
     nb=1;       %efficiency
         dT=1;           %1 hour
         Vb=12;          %inital Voltage
         Ahinit=25;  %inital Ampere-hours

     K = pbmax/(Vb*Ahinit*dT/nb);% total batteries

     Ahinit= Ahinit*K;

     Ah_cur_con = (pbnew/pbmax)*Ahinit;

     Ah_con = (sum(pbold)/pbmax)*Ahinit;

         Ah_remain = (Ahinit)-Ah_con;

     soc = (Ah_remain-Ah_cur_con)/(Ahinit);

         pout(soc>=0) = (Vb*(Ah_cur_con));

     pout(soc>1) = -(Vb*(Ah_con));

     pout(soc<0) = 0;

     soc(soc<0) = (Ah_remain)/(Ahinit);

     soc(soc>1) = 1;
end
```

I. Micro-Turbine

```
function [pout, emission] = microturbine(p,status,prated)
     emissionSO2 = 0.206*10^(-6); % kg/Wh
     emissionCO2 = 649*10^(-6);
     emissionNOx = 9.89*10^(-6);

     if(status==1)
         pout(p>prated) = prated;
         pout(p<=prated && p>=0)=p;
```

```
    else
        pout = 0;
    end

    emission = [emissionSO2*pout emissionCO2*pout emissionNOx*
        pout];
end
```

J. Plot figures from CSV file

```
function plot_from_excel(name)
    Dat=csvread(name,0,0);
    h=figure;
    set(gcf,'Visible','off');
    t=1:24;
    plot(t,Dat(1,1:24),t,Dat(2,1:24),t,Dat(3,1:24),t,Dat
        (4,1:24),t,Dat(5,1:24),t,Dat(6,1:24));
    xlabel('Time(Hour of day)')
    ylabel('Power Output (Watt)')
    grid on;
    title('CVX')
    legend('MT','PV','WT','BTR','Load','Utility');
    baseFileName = sprintf('%s.jpg', name);
    saveas(h, baseFileName );
end
```

K. Power Bus per Dayhour

```
function [p_bus] = power_bus_per_dayhour(pertime,stp_time)

    if(stp_time<=pertime)
        if (stp_time==1 || stp_time==8 || stp_time==9 ||
            stp_time==16 || stp_time==17 || stp_time==24)
            p_bus = 30000;
        elseif (stp_time==11 || stp_time==12 || stp_time==15 ||
            stp_time==18 || stp_time==19 || stp_time==23)
            p_bus = 60000;
        elseif (stp_time==10)
            p_bus = 90000;
        elseif (stp_time==21 || stp_time==22)
            p_bus = 150000;
        elseif (stp_time==14 || stp_time==20)
            p_bus = 210000;
        elseif (stp_time==13)
```

```
                p_bus = 300000;
            else
                p_bus = 3000;
            end
        else
            if (mod(stp_time,pertime)==1 || mod(stp_time,pertime)==8 || mod(stp_time,pertime)==9 || mod(stp_time,pertime)==16 || mod(stp_time,pertime)==17 || mod(stp_time,pertime)==24)
                p_bus = 30000;
            elseif (mod(stp_time,pertime)==11 || mod(stp_time,pertime)==12 || mod(stp_time,pertime)==15 || mod(stp_time,pertime)==18 || mod(stp_time,pertime)==19 || mod(stp_time,pertime)==23)
                p_bus = 60000;
            elseif (mod(stp_time,pertime)==10)
                p_bus = 90000;
            elseif (mod(stp_time,pertime)==21 || mod(stp_time,pertime)==22)
                p_bus = 150000;
            elseif (mod(stp_time,pertime)==14 || mod(stp_time,pertime)==20)
                p_bus = 210000;
            elseif (mod(stp_time,pertime)==13)
                p_bus = 300000;
            else
                p_bus = 3000;
            end
        end

end
```

L. Photovoltaic Array

```
function p = pv_array(Irrad,Tempr,prated)
    %% Initialize

    p=0;
    totalPanels = (40*prated)/20000;
    totalCells = 172;
    Vcell  = 0:0.01:0.665; % voltage vector??
    Ppv = zeros(totalPanels,length(Vcell));
    Tempr(1:totalPanels) = Tempr;
    Irrad(1:totalPanels) = Irrad;
```

```
    %% Panels Computation
    for i=1:totalPanels
        Pcell(i,:) = Vcell .* solarcell( Vcell , Irrad(i) ,
          Tempr(i));

        Ppv(i,:) = totalCells * Pcell(i,:);

        pmax(i) = max(Ppv(i,:));

        p= p + dc_ac_inverter( pmax(i) );
    end
end
```

M. Solar Cell of Photovoltaic array

```
function Ia = solarcell(Va,Suns,TaC)
    k = 1.38e-23; % Boltzmann constant
    q = 1.60e-19; % Electron
    n = 1.2; % Quality factor for the diode. n=1.2 for
      crystaline
    Vg = 1.12; %  [eV]
    T1 = 273 + 25; % Kelvin

    Voc_T1 = 0.665; % Open-current voltage at T1 [V]
    Isc_T1 = 5.75; % Short-circuit current at T1 [A]

    K0 = 3.5/1000; % Current/Temperature coefficient [A/K]

    dVdI_Voc = -0.00985; % dV/dI coefficient at Voc [A/V]

    TaK = 273 + TaC; % Convert cell's temperature from Celsius
      to Kelvin [K]

    IL_T1 = Isc_T1 * Suns; % Compute IL depending the suns at
      T1
    IL = IL_T1 + K0 * (TaK - T1); % Apply the temperature
      effect

    I0_T1 =  Isc_T1 / ( exp( q * Voc_T1 / ( n * k * T1 ) ) - 1
      );
    I0 = I0_T1 * ( TaK / T1 ) .^ (3/n) .* exp( -q * Vg / ( n *
      k ) .* ( (1./TaK) - (1/T1) ) );

```

```
    Xv = I0_T1 * q / ( n * k * T1 ) * exp( q * Voc_T1 / ( n * k * T1 ) );
    Rs = - dVdI_Voc - 1/Xv; %Compute Rs Resistance

    Vt_Ta = n * k * TaK / q;
    Ia = zeros(size(Va)); %Initialize Ia vector

    % Compute Ia with Newton method
    for j=1:5;
        Ia = Ia - (IL - Ia - I0.*( exp( (Va + Ia .* Rs) ./ Vt_Ta ) - 1 ) ) ./ (-1 - ( I0.*( exp( (Va+Ia.*Rs) ./ Vt_Ta ) -1 ) ) .* Rs ./ Vt_Ta );
    end
end
```

N. Windturbine

```
function p = windturbine(v,status,prated)
    % wind.m
    % v =[10,11,12,12,12,12,12,13,15,16,17,17,18,19,19,20,21,22,23,24,25,26,27

    von = 3;          % cut-on speed (m/s)
    vc = 14;          % corner speed (m/s)
    vout = 25;        % cut-out speed (m/s)
    diameter = 10;
    rho = 0.1;
    capacity_factor=0.2;
    %p = zeros( size(v) );

    if(status==1)
        swept_area = pi * (diameter/2)^2;
        % Below cut-on
        p(v < von) = 0;
        % Ramp up (use model)
        I = (v >= von & v < vc);
        p(I) = 0.5 * swept_area * (v(I).^3) * rho * capacity_factor;    % P = 1/2 * v^3 * A * p *Cp
        % At rated power
        p(v >= vc & v <= vout) = prated;
        % Above cut-out
        p(v > vout) = 0;

```

```
%           figure(1);%Plot the I-V characteristic curve
%           axis([0,30,0,800])
%           plot(v,p);
%           grid;
        end
end
```

N.1. Second Prosumer's Profile (not permitted to sell to utility)

O. Energy Management System – CVX – Primal Dual Interior Point Method

```
clear all; close all; clc;

%% READ CSV FILE
row=1;

numm=[  120000   -100000 120000   120000   40000
        80000    -75000  100000   100000   750000
        50000    -45000  120000   120000   80000    ];

for row=1:3
%% READ DATA
fprintf('Simulation Started\n\n') ;

num = csvread('Data.csv');

BATTpmax = num(row,1);
BATTpmin = num(row,2);
PVpmax = num(row,3);
WTpmax = num(row,4);
MTpmax = num(row,5);

    %% INIALIZATION

    %TIME STEPS, AC BUS
    time_step = 24;
    totalUnits = 5;
    p_bus = zeros(totalUnits,time_step);
    index_GenMT = 1; index_GenPV = 2; index_GenWT = 3;
       index_GenBTR = 4; index_Load = 5; index_Util = 6;
       index_Dump = 7;

    % MICROURBINE
```

```
mut = 6000;% minimum up time (sec)
mdt = 200;% minimum down time (sec)
ton=0;% counting the times turned on
toff=0;% counting the times turned off
statusMT=ones(1,time_step);

% WIND TURBINE
statusWT=ones(1,time_step);

% BATTERY
Vb=12;% initial voltage
pbold = 0;
[soc_cvx, qq1, qq2, num_of_batt] = battery( 0, pbold,
    BATTpmax);

% UTILITY
vwind = [10.12 11.654 12.43 12.7654 12.123 12.5643 12.626
    13.845 15.113 16.322 17.11 10.12 11.654 12.43 12.7654
    10.12 11.654 12.43 12.7654 24.6363 25.346 26.7562 27.23
    28.88];
irrad = [0.0075 0.3948 0.674 0.789 0.519 0.555  0.493
    0.935  0.891 0.583 0.514 0.35 0.169 0.6472  0.1508
    0.1324 0.10385 0.8206 0.1069 0.3603 0.7578 0.8679 0.0767
     0.5666];
cutil = 10^(-3)*[0.0787 0.0785 0.0785 0.0789 0.088 0.088
    0.089 0.09 0.0885 0.0805 0.0815 0.0825 0.079 0.078 0.089
     0.0866 0.0799 0.082 0.08 0.085 0.09 0.0788 0.0788
    0.085];
tempr = [33.57  66.38  54.61  65.75 42.07 70.09 30.81
    54.57 57.26  28.02 65.66  72.92 62.93  49.59  54.64
    43.63 32.40  39.59  56.84  47.46 68.34 59.68 55.18
    44.52];

stp=3600;
%% EVALUATION
for i=1:time_step
    %% CONSTRAINTS MT
    if( ton >= mut/stp )
        statusMT(i) = 0;
    end

    if( toff >= mdt/stp )
        statusMT(i) = 1;
    end
```

```
        if( statusMT(i) == 0 )
            ton=0;
            toff=toff+1;
            if(i<length(statusMT))
                statusMT(i+1)=0;
            end
        else
            toff = 0;
            ton = ton+1;
            if(i<length(statusMT))
                statusMT(i+1)=1;
            end
        end

        %% AC BUS
        p_bus(index_GenWT,i) = windturbine(vwind(i),statusWT(i),WTpmax);
        p_bus(index_Load,i) = power_bus_per_dayhour(time_step,i);
        if(mod(i,time_step)<=19 && mod(i,time_step)>=7)
            p_bus(index_GenPV,i) = pv_array(irrad(i),tempr(i),PVpmax);
        end
       %% CVX
%          n=4;
%          cvx_begin
%              variable x(n,1)
%              minimize objFun(x(1), x(2), statusMT, toff, cutil(i), i, x(4));
%              subject to
%               x(1)+x(2)+x(3)*Vb-x(4)==p_bus(index_Load,i)-(p_bus(index_GenWT,i)+p_bus(index_GenPV,i)); %%% x(4) ---> DUMP LOAD
%               %-x(3)<=0;
%               %###############################
%               -x(3)*Vb-abs(BATTpmin)<=0;
%               -x(3)*Vb-(1-soc_cvx)*BATTpmax<=0;
%               %###############################
%               x(3)*Vb-(soc_cvx-0.2)*BATTpmax<=0;
%               x(3)*Vb-soc_cvx*BATTpmax<=0;
%               x(3)*Vb-abs(BATTpmin)<=0;
%               x(2)-MTpmax*statusMT(i)<=0;
%               -x(2)<=0;
```

```
%                   -x(1)<=0;
%                   -x(4)<=0;
%            cvx_end
%
%            x_cvx_x = cvx_optpnt.x;
%
%            fprintf('\nCVX finished. ');

          if soc_cvx<0.201;
               soc_cvx=0.201;
          end
          %% PRIMAL DUAL
          clear r_t_d r_t_c r_t_p r_t
          clear array_mtr
          clear Dy Dx_pd Dlamda_pd Dv_pd
          clear x lamda  v
          clear eta tau f Df

          p=1;
          mu=10;
          alpha = 1e-2;
          beta = 5e-1;
          epsilon_feas = 1e-6;
          epsilon = 1e-6;

          b = p_bus(index_Load,i)-p_bus(index_GenWT,i)-p_bus(
              index_GenPV,i);

          v = rand(p,1);

          if statusMT(i)==1
               m=9;
               n=4;
               A = [1 1 Vb -1];
               x = rand(n,1);

               f = [ x(2)-MTpmax;
                     -x(2);
                     -x(3)*Vb-abs(BATTpmin);
                     -x(3)*Vb-(1-soc_cvx)*BATTpmax;
                     x(3)*Vb-(soc_cvx-0.2)*BATTpmax;
                     x(3)*Vb-soc_cvx*BATTpmax;
                     x(3)*Vb-abs(BATTpmin);
                     -x(1);
```

```
                    -x(4)];

                Df = [0  1  0  0;
                      0  -1  0  0;
                      0  0  -Vb  0;
                      0  0  -Vb  0;
                      0  0  Vb  0;
                      0  0  Vb  0;
                      0  0  Vb  0;
                      -1  0  0  0;
                      0  0  0  -1];
            else
                m=7;
                n=3;
                A = [1  Vb  -1];
                x = rand(n,1);

                f = [ -x(2)*Vb-abs(BATTpmin);
                -x(2)*Vb-(1-soc_cvx)*BATTpmax;
                      x(2)*Vb-(soc_cvx-0.2)*BATTpmax;
                      x(2)*Vb-soc_cvx*BATTpmax;
                      x(2)*Vb-abs(BATTpmin);
                      -x(1);
                      -x(3)];

                Df = [0  -Vb  0;
                      0  -Vb  0;
                      0  Vb  0;
                    0  Vb  0;
                    0  Vb  0;
                    -1  0  0;
                    0  0  -1];
            end

            Hessian = zeros(m,n);
            Hessian_f0 = D2_f_obj(statusMT(i));

            lamda = -1./f;

            k = 1;

            %% Repeat
            while (1)
                %% DETERMINE tau
```

```
        if statusMT(i)==1
            f = [ x(2,k)-MTpmax;
                  -x(2,k);
                  -x(3,k)*Vb-abs(BATTpmin);
-x(3,k)*Vb-(1-soc_cvx)*BATTpmax;
                  x(3,k)*Vb-(soc_cvx-0.2)*BATTpmax;
                  x(3,k)*Vb-soc_cvx*BATTpmax;
                  x(3,k)*Vb-abs(BATTpmin);
                  -x(1,k);
                  -x(4,k)];
        else
            f = [ -x(2,k)*Vb-abs(BATTpmin);
-x(2,k)*Vb-(1-soc_cvx)*BATTpmax;
                  x(2,k)*Vb-(soc_cvx-0.2)*BATTpmax;
                  x(2,k)*Vb-soc_cvx*BATTpmax;
                  x(2,k)*Vb-abs(BATTpmin);
                  -x(1,k);
                  -x(3,k)];
        end

        eta(k) = -(f)' * lamda(:,k);
        tau = (mu * n)/eta(k);
        t_max = 1;
        t = 0.99*t_max;
        %% UPDATE residuals
        if statusMT(i)
            r_t_d(:,k) = D_f_obj(x(2,k), statusMT(i), cutil(i)) + Df'*lamda(:,k) + A'*v(:,k);
            r_t_c(:,k) = -diag(lamda(:,k))*f - (1/tau)*ones(m,1);
            r_t_p(:,k) = A*x(:,k)-b;
            r_t(:,k) = [r_t_d(:,k); r_t_c(:,k); r_t_p(:,k)];
        else
            r_t_d(:,k) = D_f_obj(0, statusMT(i), cutil(i)) + Df'*lamda(:,k) + A'*v(:,k);
            r_t_c(:,k) = -diag(lamda(:,k))*f - (1/tau)*ones(m,1);
            r_t_p(:,k) = A*x(:,k)-b;
            r_t(:,k) = [r_t_d(:,k); r_t_c(:,k); r_t_p(:,k)];
        end

```

```
        %% UPDATE the invertible matrix for ÎŤy
        array_mtr = [Hessian_f0 Df' A'; ...
        -diag(lamda(:,k))*Df -diag(f) zeros(m,p);...
        A zeros(p,m) zeros(p,p)];

        %% UPDATE ÎŤy
        Dy = -inv(array_mtr)*r_t(:,k);
        Dx_pd = Dy(1:n);
        Dlamda_pd = Dy(n+1:n+m);
        Dv_pd = Dy(n+m+1:size(Dy,1));

        %% Îż+DÎż>0
        while ~( sum( (lamda(:,k)+t*Dlamda_pd) > 0 ) == m )
            t=beta*t;
        end

        %% f(x+)<0
        if statusMT(i)==1
            f = [ (x(2,k)+t*Dx_pd(2))-MTpmax;
            -(x(2,k)+t*Dx_pd(2));
            -(x(3,k)+t*Dx_pd(3))*Vb-abs(BATTpmin);
        -(x(3,k)+t*Dx_pd(3))*Vb-(1-soc_cvx)*BATTpmax;
            (x(3,k)+t*Dx_pd(3))*Vb-(soc_cvx-0.2)*BATTpmax;
            (x(3,k)+t*Dx_pd(3))*Vb-soc_cvx*BATTpmax;
            (x(3,k)+t*Dx_pd(3))*Vb-abs(BATTpmin);
            -(x(1,k)+t*Dx_pd(1));
             -(x(4,k)+t*Dx_pd(4))];
        else
           f = [ -(x(2,k)+t*Dx_pd(2))*Vb-abs(BATTpmin);
            -(x(2,k)+t*Dx_pd(2))*Vb-(1-soc_cvx)*BATTpmax;
            (x(2,k)+t*Dx_pd(2))*Vb-(soc_cvx-0.2)*BATTpmax;
            (x(2,k)+t*Dx_pd(2))*Vb-soc_cvx*BATTpmax;
            (x(2,k)+t*Dx_pd(2))*Vb-abs(BATTpmin);
            -(x(1,k)+t*Dx_pd(1));
             -(x(3,k)+t*Dx_pd(3))];
        end

        while ~( sum ( f < 0 ) == m )
            t=beta*t;

            if statusMT(i)==1
                f = [ (x(2,k)+t*Dx_pd(2))-MTpmax;
                -(x(2,k)+t*Dx_pd(2));
```

```
                    -(x(3,k)+t*Dx_pd(3))*Vb-abs(BATTpmin);
            -(x(3,k)+t*Dx_pd(3))*Vb-(1-soc_cvx)*BATTpmax;
                    (x(3,k)+t*Dx_pd(3))*Vb-(soc_cvx-0.2)*
                        BATTpmax;
                    (x(3,k)+t*Dx_pd(3))*Vb-soc_cvx*BATTpmax;
                    (x(3,k)+t*Dx_pd(3))*Vb-abs(BATTpmin);
                    -(x(1,k)+t*Dx_pd(1));
                     -(x(4,k)+t*Dx_pd(4))];
                else
                    f = [ -(x(2,k)+t*Dx_pd(2))*Vb-abs(BATTpmin);
                -(x(2,k)+t*Dx_pd(2))*Vb-(1-soc_cvx)*BATTpmax;
                    (x(2,k)+t*Dx_pd(2))*Vb-(soc_cvx-0.2)*
                        BATTpmax;
                    (x(2,k)+t*Dx_pd(2))*Vb-soc_cvx*BATTpmax;
                    (x(2,k)+t*Dx_pd(2))*Vb-abs(BATTpmin);
                    -(x(1,k)+t*Dx_pd(1));
                     -(x(3,k)+t*Dx_pd(3))];
                end

            end

            %% BACKTRACKING
            while norm( func_res( x(:,k) + t*Dx_pd, lamda(:,k)
                + t*Dlamda_pd, v(:,k) + t*Dv_pd, A, b, m, tau,
                statusMT(i), Vb, MTpmax, soc_cvx, cutil(i),
                BATTpmax,BATTpmin)) >...
                    (1-alpha*t) * norm(r_t(:,k))
                t = beta*t;
            end

            %% UPDATE x,Îż,v
            x(:,k+1) = x(:,k) + t*Dx_pd ;
            lamda(:,k+1) = lamda(:,k) + t*Dlamda_pd;
            v(:,k+1) = v(:,k) + t*Dv_pd;

            %% TERMINATION CONDITION
            if norm(r_t_p(:,k))<=epsilon_feas && norm(r_t_d(:,k
                ))<=epsilon_feas && eta(k)<=epsilon
                break;
            end
            k = k+1;
        end

```

```
            if statusMT(i)==1
                [optimum_primal_dual_interior_point , Futility_prim
                    , Fgen_prim, suc_prim] = objFun(x(1,k), x(2,k),
                    statusMT, toff,cutil(i),i,x(4,k));
            else
                [optimum_primal_dual_interior_point , Futility_prim
                    , Fgen_prim, suc_prim] = objFun(x(1,k), 0,
                    statusMT, toff,cutil(i),i,x(3,k));
            end

            optimum_primal_dual_interior_point

            fprintf('\nPrimal Dual interior point method finished.
                ');

            %% WRITE TO BUS
%                   p_bus(index_Util,i) = x_cvx_x(1);
%                   p_bus(index_GenMT,i) = x_cvx_x(2);
%                   p_bus(index_GenBTR,i) = x_cvx_x(3)*Vb;
%                   p_bus(index_Dump,i) = x_cvx_x(4);
%                   val_obj(i) = cvx_optval;

                p_bus(index_Util,i) = x(1,k);
                if statusMT(i)==1
                    p_bus(index_GenMT,i) = x(2,k);
                    p_bus(index_GenBTR,i) = x(3,k)*Vb;
                     p_bus(index_Dump,i) = x(4,k);
                else
                    p_bus(index_GenBTR,i) = x(2,k)*Vb;
                     p_bus(index_Dump,i) = x(3,k);
                end
                val_obj(i) = optimum_primal_dual_interior_point;
            %% UPDATE BATTERY
            pbold = [pbold  p_bus(index_GenBTR,i)];
            [soc_cvx, pout_btr, qq1,num_of_batt] = battery( p_bus(
                index_GenBTR,i), pbold(1:end-1), BATTpmax);
        end

        loss_24_hour = sum(val_obj)
        num(row,6+i+1) = loss_24_hour;
        num(row,7:6+i) = val_obj;
        csvwrite('Data.csv',num)
        FileName =  sprintf('%d_%d_%d_%d_DATA.csv', BATTpmax,
            PVpmax, WTpmax, MTpmax);
```

```
    csvwrite(FileName,p_bus)
    plot_from_excel(FileName);
    fprintf('Simulation Ended\n\n') ;
end
```

P. Objective Function

```
function [ObjFun,Futility,Fgen,suc] = objFun(putil, pmt, status_mt, toff,cutil,index, pdump)

    if(index==1)
        statusMT = status_mt(1);
        statusMT_prev = 0;
    else
        statusMT = status_mt(index);
        statusMT_prev = status_mt(index-1);
    end

    %% UTILITY
    Futility = cutil * putil;

    %% GENERATOR MAINTAINANCE AND START-UP
    stp = 3600; %
    hot_startup = 30; %sec
    cold_startup = 200; %sec

    cooling_time = 520; %sec

    if statusMT_prev==0 && statusMT==1
        suc = ( ( hot_startup / stp ) + ( cold_startup / stp ) * (1 - exp ( - toff / (cooling_time/stp) )) ) * (1 - statusMT_prev); % start-up cost
    else
        suc=0;
    end

    alpha=0.0074; beta=0.2333; cu=0.4333;
    Fgen = ( alpha * (pmt*10^(-3)).^2 + beta * pmt*10^(-3) + cu + suc) .* statusMT;

    %% DUMP LOAD
    Fdump = cutil * pdump;
    %% OBJECTIVE FUNCTION
    ObjFun = sum( Futility + Fgen + Fdump); % $/Wh
```

```
end
```

Q. Gradient of Objective

```
function [gradient] = D_f_obj(pmt, statusMT,cutil)
    alpha=0.0074*10^(-3); beta=0.2333*10^(-3);
    if statusMT==1
        gradient = [cutil ( 2*alpha*pmt + beta ) 0 cutil]';
    else
        gradient = [cutil 0 cutil]';
    end
end
```

R. Hessian of Objective

```
function [Hess] = D2_f_obj(statusMT)
    alpha=0.0074*10^(-3);
    if statusMT==1
        Hess = [0 0 0 0; 0 2*alpha 0 0; 0 0 0 0; 0 0 0 0];
    else
        Hess = [0 0 0;0 0 0;0 0 0];
    end
end
```

S. Calculation of Residuals

```
function r_t = func_res(x,lamda,v,A,b,m,tau,statusMT,Vb,MTpmax,
   soc,cutil,BATTpmax,BATTpmin)

    if statusMT==1
        f = [ x(2)-MTpmax;
                        -x(2);
                        -x(3)*Vb-abs(BATTpmin);
                -x(3)*Vb-(1-soc)*BATTpmax;
                        x(3)*Vb-(soc-0.2)*BATTpmax;
                        x(3)*Vb-soc*BATTpmax;
                        x(3)*Vb-abs(BATTpmin);
                        -x(1);
                        -x(4)];

        Df = [0 1 0 0;
              0 -1 0 0;
              0 0 -Vb 0;
              0 0 -Vb 0;
```

```
                  0 0 Vb 0;
                  0 0 Vb 0;
                  0 0 Vb 0;
                  -1 0 0 0;
                  0 0 0 -1];

        r_t_d = D_f_obj(x(2), statusMT, cutil) + Df'*lamda + A'*v;

    else
        f = [ -x(2)*Vb-abs(BATTpmin);
              -x(2)*Vb-(1-soc)*BATTpmax;
              x(2)*Vb-(soc-0.2)*BATTpmax;
              x(2)*Vb-soc*BATTpmax;
              x(2)*Vb-abs(BATTpmin);
              -x(1);
              -x(3)];

        Df = [0 -Vb 0;
              0 -Vb 0;
              0 Vb 0;
              0 Vb 0;
              0 Vb 0;
              -1 0 0;
              0 0 -1];

        r_t_d = D_f_obj(0, statusMT, cutil) + Df'*lamda + A'*v;
    end

    r_t_c = -diag(lamda)*f - (1/tau)*ones(m,1);
    r_t_p = A*x-b;
    r_t = [r_t_d; r_t_c; r_t_p];
end
```