# OPTIMUM ECONOMIC DISPATCH AND PRICING STRATEGY FOR LOCALISED ELECTRICITY MARKET WITH PV-BATTERY INTEGRATION

by

Smitali Patnaik

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Applied Energy and Electromechanical Systems

Charlotte

2020

## ABSTRACT

SMITALI PATNAIK. Optimum Economic Dispatch and Pricing Strategy for Localised Electricity Market with PV-Battery Integration. (Under the direction of DR.MACIEJ NORAS)

As the ingress of Renewable Systems and Energy Storage is gaining pace, the concept of local markets is emerging as an attractive alternative to utility grid services. Although, local markets are naive and at emerging stage, their advantages are being realized at both technical and financial aspects. The local market for trading electricity includes prosumers who own Distributed Energy Resources (like PV, Battery Storage) and sell their surplus generation of energy to their existing peers in the community. The local market is based on co-operative sharing economy where all users can participate to meet their demands at a chance of lower prices than that offered by the utility companies. The Thesis looks forward to develop an effective model containing PV and Battery combination in residential community based on two sets of historical demand data and PV generation, and compare the results through performance indices to see how PV and energy storage contribute to savings and help reduce overall grid dependency. The local market has been set up for a community of houses in New South Wales, Australia and local prices have been considered according to the grid prices and feed-in tariff prices prevailing in the market. The overall aim of the research is to optimize the electricity dispatch for these particular demand data sets with an appropriate pricing strategy to achieve cost minimization in terms of energy purchase, increase Self sufficiency, Self consumption and Social Welfare. The simulation of the electricity trading has been carried out using Python programming and Gurobi 9.0 (academic license) and SciPy library as the solver.

## DEDICATION

Dedicated to my Parents, who taught me that its never too late to chase your dreams.

## ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr.Maciej Noras and Dr.Umit Cali to give me this opportunity of work on thesis and providing me with continued guidance and support throughout the program. I am also thankful to Dr.Wesley Williams and Dr.Weimin Wang for being in the committee and inspiring me with new ideas.

I am thankful to my classmates for their help and standing by me whenever required. I also extend my gratitude to all faculty members and staff members who helped me out directly or indirectly in completion of this thesis.

.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

## LIST OF ABBREVIATIONS

| | |
|---|---|
| $buy_{i,t}$ | Buy Decision, takes 0 or 1 |
| $c_{i,t}$ | Battery energy sold locally (kWh) |
| $ch_{i,t}$ | Charge Decision, takes 0 or 1 |
| $disch_{i,t}$ | Discharge Decision, takes 0 or 1 |
| $e_{bt,i,t+1}$ | Battery Status at t+1 hour (kWh) |
| $e_{bt,i,t}$ | Battery status at t hour (kWh) |
| $e_{btsell,grid,i,t}$ | Battery energy sold to grid (kWh) |
| $e_{btsell,loc,i,t}$ | Maximum transaction(charge/discharge) limit for battery at hour t (kWh) |
| $e_{btuse,i,t}$ | Battery energy used to meet demand (kWh) |
| $e_{buy,grid,i,t}$ | Energy purchased from grid(kWh) for meeting demand |
| $e_{buy,loc,i,t}$ | Energy purchased from local market(kWh) for meeting demand |
| $e_{buych,grid,i,t}$ | Buy charge from grid (kWh) |
| $e_{buych,loc,i,t}$ | Buy charge locally (kWh) |
| $e_{d,i,t,max}$ | Maximum Demand (kWh) within the Pool |
| $e_{d,i,t,min}$ | Minimum Demand (kWh) within the Pool |
| $e_{d,i,t}$ | Demand (kWh) of ith household at t hour (t = 0, 1, etc.) |
| $e_{dnet,i,t}$ | Surplus net Demand (kWh) of ith household for purchase after self DER usage at t hour |
| $e_{dnew,i,t}$ | Adjusted Demand (kWh) of ith household adjusted at t hour |

| | |
|---|---|
| $e_{pvcharge,i,t}$ | PV energy used for battery charging (kWh) |
| $e_{pvsell,grid,i,t}$ | PV energy sold to Grid (kWh) |
| $e_{pvsell,loc,i,t}$ | PV energy sold to local market (kWh) |
| $e_{pvuse,i,t}$ | PV energy used to meet demand (kWh) |
| $F(X)$ | Fairness Index |
| $n$ | Number of households |
| $o_i$ | Optimal throughput |
| $p_{ft}$ | Feed in Tarrif |
| $p_g$ | Grid Price |
| $p_{loc}$ | Local price for each hour |
| $sell_{i,t}$ | Sell Decision, takes 0 or 1 |
| $t_i$ | Actual throughput |
| $x_i$ | Normalized throughput (in Kbps) of the ith TCP flow |
| ADMM | Alternating Direction Multiplier |
| CREST | Centre for Renewable Energy Systems Technology |
| cv | Coefficient of variation |
| DERs | Distributed Energy Resources |
| DSM | Demand Side Management |
| ESC | Energy Sharing Coordinator |
| GST | Goods and Services Tax |

| | |
|---|---|
| ICT | Information and Communications Technology |
| kW | Kilo Watt |
| kWh | Kilo Watt Hour |
| LP | Linear Programming |
| MILP | Mixed Integer Linear Programming |
| MINLP | Mixed Integer Non-Linear Programming |
| MMR | Mid Market Rate |
| NEC | National Electrcity Market |
| NLP | Non Linear Programming |
| P2P | Peer to Peer |
| PV | Photovoltaic |
| SC | Self Consumption |
| SDR | Supply and Demand Ratio |
| SS | Self-Sufficiency |
| STEP | Smart Electricity Exchange Program |
| VCG | Vickrey Clarke Groves |

# CHAPTER 1: INTRODUCTION

## 1.1 Concept

A vertically integrated market is still flourishing in many regions and consumers have no choice or say on the prices they have to pay for consuming the electricity. Increasing use of Distributed Energy Resources (DERs) and Battery storage systems with smart trading platforms can help residential consumers to generate energy and sell it to peers or inject to the grid. This newly emerging local market system using DERs has opened a great opportunity for consumers to take control of the electricity consumption and earn revenue as a new category of producers called *Prosumers* [1]. Energy trading in local market is being discussed for more than five years and many countries are coming forward to utilize the idea for increasing sustainability, bring down emissions and carve out a budget friendly model of buying electricity by households using small scale renewable generators [2]. The local energy trading seems to be promising venture as it will not only solve these aforementioned problems but also help create a free market model where consumer participation is prevalent and allows them to choose from whom they want to buy electricity. Prosumers get advantage of investment in residential DERs like Solar PV as solar energy is intermittent in nature and during low demand, the surplus energy produced can be sold to energy deficient consumers or charge the Battery Storage. Thus, a small scale sharing economy can be expected to be achieved within small communities, where neighbors can maximize their utility function (which is to extract maximum savings for consumers and generate revenue for prosumers) by meeting their demand through co-operative actions within their comfort zone.

## 1.2 Objective

The thesis aims to study and analyze various local market configurations currently being studied in the market through literature review and look forward to develop suitable scenario based model for optimizing the dispatch of electricity with suitable pricing strategy, using PV and Battery storage in a residential community set up and see which model was suitable for the given community in terms of performance. Battery Storage has been integrated into selected households based on their load or PV generation profiles. The research will help understand how the penetration of solar and energy storage capabilities will impact the economics of community as a whole and savings achieved with respect to the conventional trading with utility grid. The pricing mechanism and dispatch models simulated in the thesis will help understand the local market dynamics and required parametric for increasing efficiency of local trading.

## 1.3 Scope

A small set of housing community in New South Wales (Australia) with real time demand and PV generation data is considered. Two sets Community mix containing number of Prosumer and Consumer with PV/Battery ownership is created in which dispatch and pricing models are tested for trading outputs. The prosumer households have PV or both PV and Storage. A 48 hours trading instance is generated through optimization and auction based algorithms. The model performance is measured through Community and Individual Savings, Self-Sufficiency, Self-Consumption and Fairness Index.

# CHAPTER 2: LITERATURE REVIEW

Many studies and models have been implemented in the local market based energy trading and lot of commercial platforms have been created through corporate investments which have successfully produced promising results in this field in terms of reducing grid dependency, maximizing savings, managing surplus generation, earning revenue etc. Some interesting model proposals and commercial solutions available have been covered in this literature review in order to study about characteristics of market, understand the governing policies, if any and observe the impact of the proposed and commercial solutions on these markets. Local markets may differ in load usage patterns, income, total demand, utility prices, renewable policy implementation etc. Thus, this uniqueness in market features necessitates flexible solution strategies that suits all the users in a community considering all aspects like user interest, generation capacity of the households, grid integration and budget. For example, if the residential batteries are cheaper in a given region, users can maintain individual large sets of battery storage and use them in the local market for revenue. In some places, bulk investment in a centralized battery storage can be more cost effective option as it may be providing more risk sharing options to users with better lending rates and returns.

## 2.1 Local Energy Trading Platforms

Various small and medium level trading models have been introduced in many countries as pilot projects or commercialized solutions. For example, Vandebron in Netherlands [3] was introduced in 2013, which facilitated trading electricity generated from solar, wind and biomass and the prosumers were free to set their prices and consumers were free to choose their suppliers. Another platform called Piclo [4], was introduced by a start up company called Open Utility in 2015 in partnership with Good Energy and due to its popularity, it got approval from the Energy Regulator in UK. Piclo offered transaction services similar to Vendebron for a time span of every half an hour and allowed users to select electricity supplier during the trading process. Peer Energy Cloud [5] and SmartWatts [6] were Information and Communications Technology (ICT) platforms which provided cloud based virtual trading platform and used excess generation from DERs for setting local market conditions. Sonnen Batteries in Germany introduced Sonnen Community in which the battery owners could charge their batteries with PV and sell the excess power in the market through virtual market platform eradicating most of the dependency on utility grid [7]. Mosaic was a test project in USA which focused on community sharing through investments in PV and allowed consumers without PV to participate in local trading and get opportunity for savings [8]. Yelaho was a similar project, but got discontinued within a year of operation due to its unpopularity among users, because of lack of credit and funding, prevailing apartment based communities, regulatory constraints like restriction to integrate more solar systems with grid after specified solar cap is met [8][9]. Lichblick Swarm Energy by Lichtblick Swarm Conductor used cloud-based platform solution for integrating DERs, known as Swarm Dirigent[10]. The platform could integrate more than 1000 DERs, that included photovoltaics, energy storage, wind power and electric vehicles thereby balancing the generation from DERs and managing peak shaving, solar load shifting and grid operation. A solution provided by investment from LO3 and next47 called Transactive Grid had capacity to integrate more than 40 homes in a local market network using Ethereum Blockchain concept and power was tokenized for convenience and transactions were carried out using smart contracts. The Brooklyn microgrid implemented for NewYork was one of the most promising pilot projects [11]. Another popular model was Elecbay, which used game theory for local market transactions. The format used non-cooperative game theory between users for finding nash equilibrium, and individual utility was based on price of electricity supply and flexible demand of the user [12][13]. Blockchain based platforms are gaining popularity due to the transparency in operations, provide valid transactions and do not allow tampering of records. Some models related to Blockchain can be referred in a paper by Goravonic [14]. Powerledger is another blockchain based platform and provides security and privacy in transactions [15]. Many Blockchain models are proposing the use of crypto-currency for fast transactions like Bankymoon, which is a pilot project that focuses on developing African schools, and has prepaid meters installations that work on blockchain. The users who want to support can directly put crypto-currencies like Bitcoins to the meter to finance electricity to the school, thereby providing means to fund for the community development [16].

One of the key factors for the success of trading platforms is the security and privacy, that is why models like Sonnen and Powerledger have been popular. Power ledger has a Ethereum-based platform, where trading of energy tokens are democratized and there is full privacy to the users with transparency in transactions and pricing. Another reason for Powerledger's success and expansion in other countries is the government policies in these locations where use of DERs are being supported extensively. For example, to make such platforms popular Australia adopted strategy where through local market consumers can get reward for purchasing and selling energy in real time [17]. Another factor is the adaptability to the local policies and existing power infrastructure through experimental setup. Thus, Power Ledger is currently setting up projects in the US, Thailand, Japan, Austria etc., where focus is on testing and customizing their platform with existing renewable energy infrastructure [18].

Market solutions discussed above are an effective platform similar to ebay, AirBnB

,etc., providing more alternatives to consumers for purchasing electricity. However, considering electricity consumption to be a never ending process, it is difficult to set one hour or half an hour transactions, and to be always dependent on manual decisions for initiating transactions. Hence, effective market model not only requires a good communication platform for trading but also needs consideration of efficient pricing and dispatch in order to be economically attractive to the users and capable of automatic implementation, that satisfies all aspects user comfort and utility. Many novel market mechanisms have been proposed, and a lot of research studies are under development to shape a generalized energy trading market for local communities considering balance between social welfare, revenue generation, market trends and regulations. Some of these are discussed below.

## 2.2 Market Models

A model using different scenarios: 1) a Peer to Peer (P2P) energy trading, 2) Order Book Market, 3) Zero Intelligence Agent and 4) Intelligent Agent was simulated by Mengelkamp [19]. In P2P trading scenario, agents were randomly matched and a transaction was carried out, whereas in Order Book Market, the trading was based on function of buying price and selling price. The Zero Intelligence Agent used single random pricing between grid price and Feed in Tariff, while the Intelligent Agent model considered agents' behavior based on their savings and revenue to plan their decisions. The P2P, Order Booklet and Intelligent Agent model showed self consumption of about 38% while, that provided by Zero Intelligence model was at 35%. The lowest local pricing was achieved with P2P market model combined with Intelligent pricing scenario and the lower self-consumption (ratio of local use to total DER generation) was visible due to timed gap between PV generation and night consumption. A model using Supply and Demand Ratio (SDR) was proposed for determining local pricing by Liu et al [20]. This model used load shifting and had provision to select time horizon (day ahead or hour ahead) for transaction. The SDR $>$1 denoted excess supply in the pool, whereas SDR $<1$ signified greater demand in the trading period. Thus, pricing mechanism followed simple economic demand-supply principle with fairness index of 0.165 indicating unfair allocation of cost benefit achieved from the solution, with fairness index calculated as variance in ratio of benefit to cost of all users. Another model using the Bill sharing, MMR (Mid Market Rate) and SDR mechanism used game theory with shapely value for optimization of allocation [21]. The results showed that individual savings was improved but community savings did not improve significantly. A Linear Programming Optimization based feasibility test model was proposed by Long [22], which focused on checking the possibility of trading by maximizing the balance between demand and supply and establishing P2P index number. It used k-means clustering for classifying the demand and usage profiles in to different categories of low voltage distribution networks which was further feeded into LP for optimization. The P2P index number of 1 was the desired value to establish the feasibility and case results were used to establish the required DERs penetration in the network. The model used excel based demand profile calculator by Centre for Renewable Energy Systems Technology (CREST) to establish the load and DERs profiles, however, with real time data whether the model can be effective or not, cannot be determined as generation undergoes lot of impact due to local climate changes. Another trading model by Long [23] included three types: 1) Bill sharing scenario in which users share the single community bill at the meter and pay their share as per their import and export. 2) The Mid Market Rate pricing model where local price is the midpoint of buy price and sell price based on supply and demand totals. 3) Auction based model using Reclusive Least Square method for finding clearing price of the market. The savings from these three models achieved were close to 30%. Demand Side Management was simulated in the P2P trading model by Alam [24] that proposed to reduce unfair cost distribution among the users through Pareto Optimality by restricting the maximum cost payable by a household. It considered loads, disutility (discomfort from delaying or reducing appliance use), energy storage, renewables, and focused on using all the energy in trading model in order to minimize overall community costs through Mixed Integer Non Linear Programming(MILNP). Demand Side Management (DSM) can be a critical factor in implementing energy trading in local market as the consumption profile has direct impact on the costs of households. A model using combination of Battery and PV was proposed by Long [25] in which two stage control was applied for a location UK using CREST demand modeling tool (by Centre for Renewable Energy Systems Technology). At the first stage, the billing and payment was set up after 24 hours of transaction which was optimized using constrained non Linear optimization using data points from forecasting of load and PV and using previous 24 hour battery discharging and charging schedule. The local pricing was decided by modified SDR model that provided compensation. The second stage used rule based control for sending equal control signals to prosumers for charging and discharging batteries based on surplus energy and demand by Energy Sharing Coordinator (ESC). The savings achieved through this model was around 30% with consumers saving around 12.5% and prosumers making extra 57 Euros per household. The model performance was measured with different battery sizes and seasons for 100 households. The self sufficiency ranged from 24.2% to 63.3% for battery sizes ranging from 0 to 16 and self consumption from 62% to 100% for similar battery range. Another model for UK was tested by [26] in which first scenario used decentralized battery penetration at individual premises and another scenario proposing centralized battery storage common for the community. The decentralized model achieved maximum savings and least transactions with the grid compared to the centralized battery storage model. A MILP based model using McCormick relaxation by Jing [27] was applied to three cases comprising of residential and commercial prosumers interacting with grid and in second, in which a residential community is connected to commercial prosumers. The cost savings achieved was about 4.9% cost savings when second model was implemented with allocation fairness. A concept called Smart Electricity Exchange Platform (STEP) was proposed by Zepter [28] using residential buildings in UK that used the excerpts from intra-day and day-ahead trading markets to model a stochastic programming based sequenced decision-making energy trading system with battery banks, wind and PV and concluded savings of about 60% when the battery storage was used with PV. The forecasting of demand is a good concept to model an efficient demand- response based system, however, switching the system continuously to balance demand and supply and interact with the grid can be an expensive set off. A multi-objective optimization environment was set up for implementing a User Dominated Demand Response schema with P2P energy trading by Zhou et al [29] in which demand response bids were set up with schedules and optimization algorithm optimized the demand in the pool along with energy trading process to divert surplus generation in the pool. The results showed savings up to 13.6% for higher PV generation levels. A hierarchical model using two step process was proposed by Park et al [30]. It used self scheduling by prosumers to optimize their utility function (extract maximum revenue), consider depreciation cost of Battery etc. The results were further fed to derive pricing to increase social welfare and a MILP based algorithms was implemented for a 24 hour trading and decrease in operational costs was the criteria for consideration. Nash equilibrium and Lyapunov-based methods were implemented by De Paola [31], to devise a new iterative control algorithm to always converge at minimizing energy costs of the consumers by changing their scheduled power flat demand profile. The Nash equilibrium strategy provided 24% savings in the pool.

It is difficult to compare the models proposed by different works discussed above in the literature review and conclude which model is better and in what perspective, as various metrics have been emphasized for each work and the models are applied on different geographical locations, commercial and households setups with varied incomes, consumption levels, and different user willingness criteria to participate. The previous work as been enlisted to throw light into developments, and get idea of the technical terms used in methodologies of pricing, dispatch, and various trading platforms persisting in local market system and use excepts from these works to develop suitable solution for the selected market in the thesis.

## 2.3 Thesis Statement

The research work associated with local market trading is still in its naive state as different locations have distinct climate conditions, usage patterns, generation from DERs like solar and wind, regulations etc., and to study the diversity of all these factors is time consuming exercise. The study of these variations is necessary as this can help construct customized solutions for the different community features, which requires detailed analysis of historical data for these locations in terms of demand patterns, generation or weather, electricity bills etc. It was observed from literature review, many proposed mechanisms for price optimizations and dispatch have utilized simulation tools to generate demand and generation patterns and have not utilized historical datasets for trading simulations except a very few. Also, both community and individual savings were not considered together in many previous works. Hence, much focus has not been put on the overall efficiency of models at both individual and community levels. Thus, performance measurement becomes a key requirement to know whether the model is feasible solution for the given community or not and whether social welfare can be achieved for all the participants of the trading. Social welfare is one of key requirement for a market model, as one user not gaining welfare from the trading will lose the willingness to participate. The individual social welfare in local trading can be in terms of savings and receiving appropriate share from the local market.

The main contributions of this work is to introduce a new pricing strategy based on demand and perform a dispatch using known Mixed Integer Linear Programming

(MILP) optimization techniques. It also looks forward to utilize excerpts from Vickrey Clarke Groves(VCG) mechanism in a novel way to setup another trading dispatch through transparent bids based on the new pricing strategy developed. And also deduce which model was suitable for the given community. It is expected from this work that, by consolidating known performance metrics like Self-Sufficiency and Self-Consumption in dispatch models used here, the importance of performance metrics in deciding suitable dispatch and pricing model for Local market trading can be realized. It is important to look at every aspect of performance while working with trading mechanisms, as the dispatch scheme needs to be modeled considering requirements at both community and individual level in order to make model lucrative for users to adopt. One of the performance metrics, the work focuses on is the Fairness Index adopted from Jain's Fairness Index [32], which can be used as another metric for measuring equal allocation of electricity by taking demand as the benchmark for the measurement.

# CHAPTER 3: MARKET SELECTION

## 3.1 About Australian Electricity Market

The National Electricity Market (NEM) in Australia was created during the process of privatization of electricity generation and retail sectors between the year 1995 and 2010. There are 22 electricity networks in Australia with both public and private ownership. The Australian Market has been witnessing hike in adoption of local energy trading schemes after many households are opting for renewables and storage to shift focus from conventional utility purchases [33]. The various factors leading to this change are increasing usage of technologies by new generations, social inequality, decreasing income, realization of concepts of demand management, increasing rates from utility companies, need for control over purchases, climate change awareness, growth in small scale business environment, increasing cost of operations for grid maintenance and decreasing government subsidies or Feed In Tariff rates [34].

## 3.2 Location for Local Market Assumption

Considering the market boosting initiatives from both Consumer and Regulatory end, the market based in New South Wales has been found suitable for study in this thesis and models were simulated using this base for understanding techno-commercial aspects of local trading market in this region as it suffices maximum favorable factors for trading market that can be assumed to be available and hence, market model can be expected to be simple and robust, for example, consumer willingness for participation can be considered constant and positive as all users encourage community based sharing model. Grid Prices and Feed In Tariff can be adopted from the market to devise pricing strategies which can help in creating upper bound and lower bound in local price range. The market in New South Wales has one fully private electricity network, two privately ownership networks with 50.4% shares and one fully public owned network. The selected community for model simulation has distribution managed by Ausgrid [35] which was turned into in public-private owned network having 50.4% shares to private investors in the year 2016 and captures one of the major portion of the market with Sydney, Central Coast and Hunter regions of New South Wales under it.

## 3.3 Demand and PV Generation Data Selection

Historical Demand data and PV generation data was obtained from Ausgrid Database for the month of July 2011 and 2012 [36]. The solar home electricity data contains the 365 days log of consumption (based on domestic tariff) and PV generated from the gross energy meters installed in the premises of the 300 households, in a time gap of 30 minutes with total PV generation.

## 3.4 Grid Price and Feed-In-Tariff

The Grid Price is the price at which users buy electricity from the utility company. And Feed-in Tariff is the price at which utility company buys electricity from prosumers. The grid price considered is total 46.04 cents (inclusive GST(Goods and Services Tax 10%, payment deduction fee of 0.45%) and Feed in Tariff as 10.4 cents [37][38]. The currency considered is Australian Dollars. These two prices have been considered for purchase and sales transactions with the grid and in setting trading price for the local market.

# CHAPTER 4: SCENARIO CASES

The Load scenario cases have been used to simulate the pricing strategy and dispatch mechanisms for the market under consideration. The real time data from the Australian market(Ausgrid) has been adopted and historical demand and solar PV generation data has been taken for few set of houses. These set of houses are categorized into three groups.

Group A : having No DERs

Group B : with only PV

Group C : with both PV and Battery Storage.

The number of houses considered for trading comprise of 8 sets of houses in first scenario case and 10 set of houses in second scenario case. The zip codes have been selected randomly from the dataset of 300 households. The scenario test is being done with 8 to 10 houses considering hardware limitations and processor speed. Also, number of houses were found have very small PV sizes within the dataset, thus, only those houses were selected which have higher PV sizes and were expected to provide better chances of producing surplus energy for day time trading and charging of batteries. Demand and pv pattern has been analyzed for each scenario to have better understanding of how much variations is persisting between the pv generation and demand. This has been done by taking the cumulative historical demand and pv generation and plotting them together for the total 48 hour duration. Battery sizes have been calculated using certain assumptions on demand of the group C households. Battery sizes need to be designed appropriately to make sure that sufficient surplus is created every hour for sales.

## 4.1 Case-I: Consumer-Prosumer Mix

Eight houses were considered for this scenario and categorized into 3 groups with Group A having 2 houses not owning any DERs, Group B with 2 houses having PV, and Group C having both PV and Battery Storage. The respective historical PV generation log were considered for the Group B and Group C households. The PV generation of some houses were scaled up to increase PV capacity as some households (C4,C5) had very small PV size unfit for local trading. Maximum PV size in the entire pool is limited to 10 kW considering that its a residential set up and installation area and budget constraints prevail for such investment (Table 4.1, Household Classification).

Table 4.1: Household Classification.

| Group | House ID # | DERs | PV (kW) |
|---|---|---|---|
| A | C7 | NA | NA |
| A | C8 | NA | NA |
| B | C1 | PV | 4.8 |
| B | C2 | PV | 6.2 |
| C | C3 | PV+Battery | 9.99 |
| C | C4 | PV+Battery | 10.2 |
| C | C5 | PV+Battery | 10.5 |
| C | C6 | PV+Battery | 4.55 |

### 4.1.1 Demand Data and PV Generation Analysis

The historical demand data from the Ausgrid for all the eight houses has been added for each hour and has been plotted showing how much total demand is prevailing in the 48 hours starting from 0th Hour corresponding to 12 AM midnight, and consists of cumulative demand and PV generation of all users for each hour. This period has been considered in the model testing because it adequately covers day and night transactions and explains the role of PV and battery in the trading system sufficiently. From both the curves (Fig.4.1, Cumulative Demand and Generation Profile), considerable variation was noted between the demand and PV generation profiles and thus, battery storage requirement for meeting the user demand and local trading was found to be apparent as it can help improve the performance of trading in terms of savings with more DERs penetration [25][26][39].

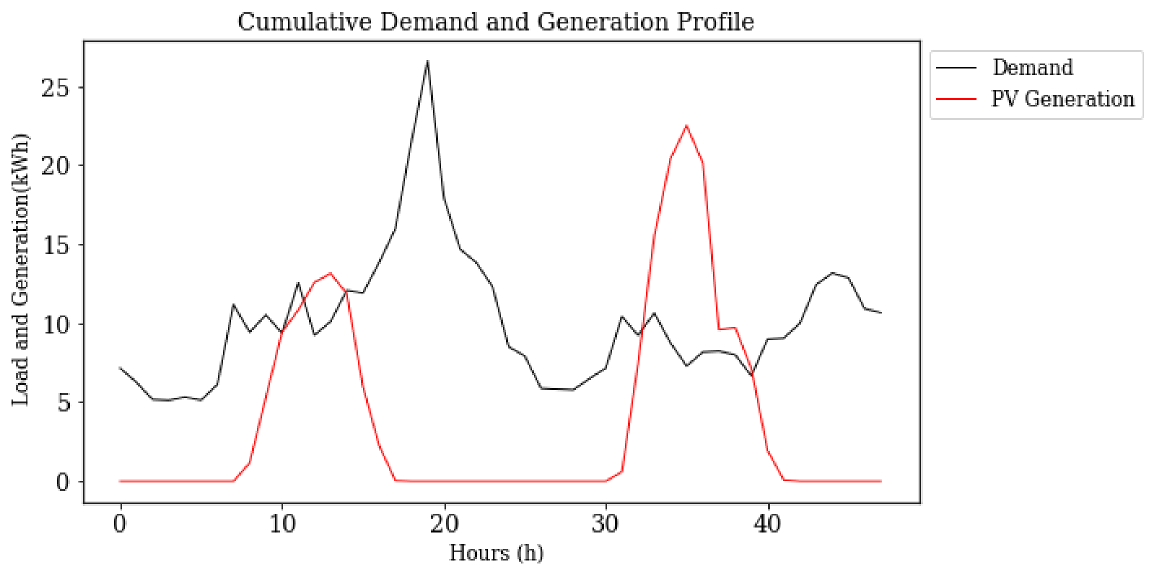

Figure 4.1: Cumulative Demand and PV Generation profile.

The highly intermittent nature of demand and narrow peak of PV generation means that a storage system needs to charge within the narrow PV window and discharge during evening and night hours. PV generation is weak compared to the demand on first day, however, it showed moderate generation for the second day. Thus, the peak demand can be seen in the night time and making need of storage evident to meet community total demand. The total community load demand and PV generation used for this case is stated below (Table 4.2, Community Totals).

Table 4.2: Community Totals

| Hours | Total demand (kWh) | Total PV (kWh) |
|---|---|---|
| 48.00 | 486.29 | 187.63 |

### 4.1.2 Battery Selection

Lithium-ion batteries are a promising candidates for residential energy storage due to continuously declining costs [40] and thus, have been considered in the trading models (Table 4.3, Battery sizing). But most of the battery characteristics are not required to be used in the simulation as energy from the battery is the only parameter which is of use in the trading. A general Battery Sizing has been assumed based on average load demand in kWh. Hence, the Battery functionality has been made linear through basic charge and discharge limits for the battery dispatch formulation [41]. As the trading mechanism takes place in kWh, energy from the battery has been taken to consideration in this unit. The discharge (or charge) limit was calculated based on number of discharge hours required and considers required back up in emergencies for prosumer use by keeping it limited to a constant value. It is to be noted that the discharge hours are assumed for calculations only, actual discharge and charge time taken by battery may vary based on the minimum in the objective function being achieved, constraints set for the charging and discharging of the batteries and solver configuration. The minimum battery limit for simulation is assumed to be 0 kWh and maximum battery limit is the battery size, thus, battery can be dispatched in the local market only when it is within this range.

,

Table 4.3: Battery Sizing

| | | | | |
|---|---|---|---|---|
| Average Hourly Energy Demand per Day kWh | 26.23 | 23.84 | 17.93 | 27.2 |
| @ 75% of demand kWh | 19.6 | 17.88 | 13.44 | 20.44 |
| Battery Units in kWh | 25 | 25 | 17.6 | 25 |
| Battery @ 90% efficiency kWh | 22.5 | 22.5 | 15.8 | 22.5 |
| Discharge/Charge limit kW per hour | $1.8kW \sim 2kW$ | $1.6kW \sim 2kW$ | $1.3kW \sim 1kW$ | $1.6kW \sim 2kW$ |
| Approx. Discharge hours | 12 | 12 | 12 | 12 |

25 KWh and 17.6 kWh Battery Banks have been considered with efficiency of 90% (assumed as the worst case). The 25 kWh and 17.6 kWh is taken from the standard battery product range available in the market just to standardize or round-off the capacity derived from the calculation above in the table. The maximum charge and discharge limit every one hour has been set as 2 kW for 25kWh units and 1 kW for 17.6 kWh unit (Table 4.3, Battery Sizing). This means battery owners are allowed to charge/discharge their batteries in this specified limit in a single transaction hour. The charge and discharge hours has calculated considering minimum 12 hours for the battery to discharge completely. That is, at the rate of 2 kW each hour, the battery gets discharged in 12 hours. However, this calculation has been inserted to derive suitable discharge and charge limits for the battery bank and based on the assumption that typical consumption of a household ranges between 1 kWh to 2 kWh in a given hour. And there will be fewer instances when it goes above this capacity (for example, cooking or washer/dryer) and can be met from the grid supply or local market purchases. Load demand below this can help create surplus battery energy during a hour and help in revenue generation.

## 4.2 Case-II: Consumer-Prosumer Mix

The Community in this Scenario Case consists of 10 households in a different zip code of New South Wales with 2 set of Houses in Group A (without DERs), 4 set of Houses in Group B (with PV) and 4 set of houses in group C (with both PV and Batteries). Both demand data and PV generation log has been adopted from Ausgrid database [36]. This scenario case was created in order to verify the functionality of the dispatch and pricing strategy devised for the trading (Table 4.1, Household Classification).

Table 4.4: Household Classification

| Group | House ID # | DERs | PV (kW) |
|---|---|---|---|
| A | H9 | NA | NA |
| A | H10 | NA | NA |
| B | H1 | PV | 5.4 |
| B | H2 | PV | 4.0 |
| B | H3 | PV | 4.2 |
| B | H4 | PV | 4.0 |
| C | H5 | PV+Battery | 5.9 |
| C | H6 | PV+Battery | 8.0 |
| C | H7 | PV+Battery | 6.2 |
| C | H8 | PV+Battery | 5.6 |

### 4.2.1 Demand Data and PV Generation Analysis

The demand data of all the household is added and mapped for 1 hour intervals and total time period under consideration is 48 hours. The plot has been created to check the total demand of all the households and PV generation to observe the extent of gap between the solar availability and usage patterns.

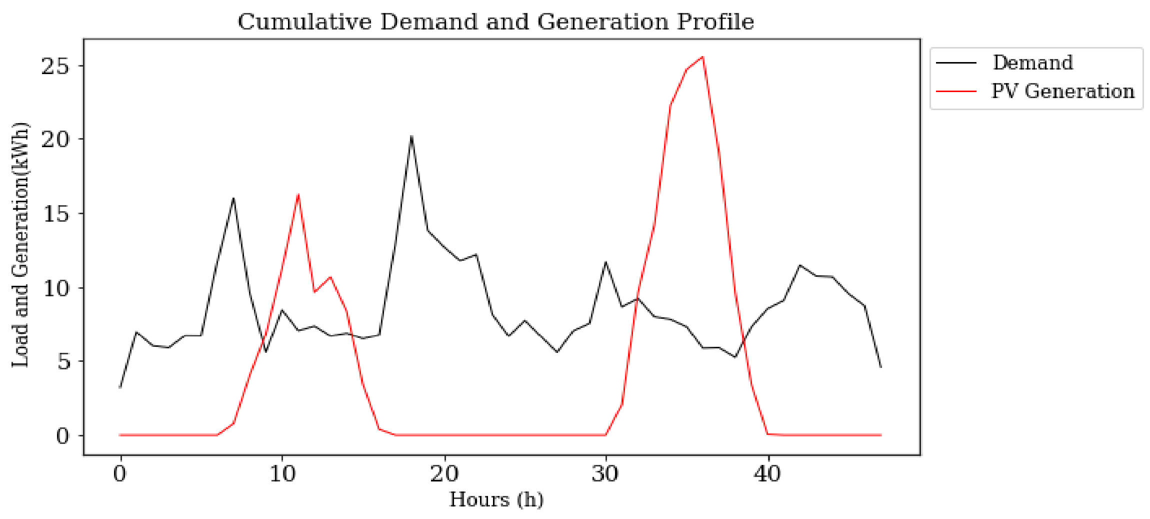

Figure 4.2: Cumulative Demand and PV Generation profile.

The demand was higher during the evening and night hours, and PV generation was surplus in day hours as compared to demand patterns for both the days (Fig.4.2, Cumulative Demand and PV Generation Profile). Thus, the battery size becomes necessary for setting up local market trading during the night time. The demand and generation peaks were not synchronized, which was similar to scenario-I data set. The total demand was 411.48 kWh for the 48 hour duration and PV generation was recorded as 202 kWh (Table 4.5, Community Totals).

Table 4.5: Community Totals

| Hours | Total demand (kWh) | Total PV (kWh) |
|---|---|---|
| 48 | 411.48 | 202.00 |

### 4.2.2 Battery Selection

Battery in this case has been sized based on the peak maximum demand for the respective households in the pool for 6 hours. The battery size is considered higher for some households, thus, a worst case scenario of peak demand running continuously for 6 hours of non-sunshine period is assumed. The intention is to use a higher battery capacity to meet the peak demand in the non-sunshine hours and create sufficient surplus in the pool for trading (Table 4.6, Battery Sizing). The charge/discharge limit has been derived from hours of backup for all battery capacities as in case-I. The battery backup hours are used merely for calculation to set charge and discharge limit for the batteries and actual discharge or charge time may vary depending on prosumers' decisions on battery dispatch. The battery cannot be over sized considering the budget and overall PV in the pool for battery charging. It is to be noted that reasonable size of battery has been taken for the households to make sure energy trading takes place. Schedule of equipments is not available for households to perform a detailed calculation considering back up hours, system voltage and autonomy. Also, this calculation is not needed right now in energy trading set up. The battery size has been planned suitability based on the fact that it is a residential set up and cost constraints will prevail, thus, very large battery sizes are not advisable. The battery size has been rounded off using multiples of a standard 13.5 kWh battery storage just to ease calculation. Hence, Battery sizes obtained were 24.3 kWh and 36.45 kWh with discharge/charge limit at each hour to be 3.3 kW (Table 4.6, Battery). Hourly demand was noted to be slightly higher for households in this case, hence, a higher discharge/charge limit was considered to create sufficient surplus for trading and meeting prosumer demand. The minimum battery limit for simulation is assumed to be 0 kWh and maximum battery limit is the battery size. Thus, trading with battery will be operational, when battery is between this range.

Table 4.6: Battery Sizing

| Parameter/User | H5 | H6 | H7 | H8 |
|---|---|---|---|---|
| Max Demand in a hour kWh | 5.44 | 7.89 | 6.02 | 7.55 |
| Max Demand in 6 hours kWh | 32.63 | 47.34 | 36.11 | 45.28 |
| @75% of 6 hour demand kWh | 24.47 | 35.51 | 27.09 | 33.96 |
| Battery size rounded-off kWh | 27.00 | 40.50 | 27.00 | 40.5 |
| Battery @ 90% efficiency kWh | 24.30 | 36.45 | 24.30 | 36.45 |
| Charge/Discharge limit kW per hour | 3.30 | 3.30 | 3.30 | 3.30 |
| Back up in hrs | 7 | 11 | 7 | 7 |

# CHAPTER 5: SIMULATION PACKAGE

## 5.1 Python

Python is the object oriented scripting language released in 1991 and developed by Guido Van Rossum of National Research Institute for Mathematics and Computer Science in Amsterdam [42]. It has become a widely used programming language and is recognized for educational and scientific computing.

Python programming has been selected for as platform for simulation of the energy trading models as it incorporates rich library packages facilitating speedy computation, and reduce coding complexity of the algorithms. Many Literature works have worked on GAMS, Matlab etc., for running optimization algorithms. Python is continuously developing its libraries for optimization algorithms and hence, it can be considered as another choice due to the simplicity in its coding, availability as a free of cost and an open source platform.

## 5.2 Python Libraries

Various python libraries utilized for the simulation include [42]:

NumPy (Numerical Python): As Python does not have a built-in array data structure. NumPy provides N-dimensional array object, linear algebra, Fourier transform, and random number capabilities.

SciPy (Scientific Python): SciPy facilitates scientific calculations like integrals, differential equations, additional matrix processing and optimization algorithms.

Pandas: is used for data manipulations and uses NumPy's ndarray. DataFrames, a two dimensional data structure feature is used here to export the demand data file into the python environment for simulation.

Matplotlib and Seaborn: have been utilized for data visualization of results as bar graph or line graphs etc.

## 5.3 Gurobi Solver

The Gurobi Solver is a commercial optimization solver for linear programming (LP), quadratic programming (QP), mixed integer linear programming (MILP), mixed-integer quadratic programming (MIQP) etc. Gurobi solver can run on number of platforms like GAMS, C++, Java, .NET, MATLAB, R and Python [43][44]. Gurobi academic license has been utilized for the simulation of the models and this available with full features for one year.

Gurobi solver has been selected for implementing MILP models for the trading because the hardware constraints require computer to have a solver that can do fast computation and adapt to the slow processor. As number of iterations will be many for the trading dispatch models, and Gurobi solver can run multiple iterations within seconds. It is equipped with inbuilt packages that can ease the long and complex loops, and can be processed using simple one line codes.

## 5.4 Data Preparation

The demand and generation dataset for New South Wales was available in the form of csv file. The values contained data in 30 minute intervals for all the households. The selected household data was taken and a separate csv file was created. Using Numpy, and Pandas library from Python the 30 minutes data was totaled and converted to one hour interval for trading with total time horizon of 48 hours for simulation. The houses were renamed with appropriate House IDs like (C1,C2,H1,H2 etc.) to make them easy to recognize and prepared for simulation. The system used is Lenovo MT 2325 with hardware of 8 GB RAM and Intel core i5-3320M CPU @2.60 Ghz.

# CHAPTER 6: PRICING STRATEGIES AND DISPATCH MECHANISMS

## 6.1 Introduction to Dispatch Mechanisms

A number dispatch techniques have been used before to implement local trading mechanisms. These techniques included Constrained Optimization techniques like Linear Programming (LP), Mixed Integer Linear Programming (MILP), Alternating Direction Method of Multipliers (ADMM) etc., and other different techniques like Nonlinear Programming, Models(NLP), Gametheory based Models, Auction based Models etc [45].

Linear programming (LP) is used to achieve the best maximum or minimum output in a mathematical model whose required variables are set by linear relationships. Linear programming methods are powerful and robust algorithms able to solve large-scale optimization problems. In Manufacturing or Supply Chain, Linear programming calculates the optimal planning or use of a resource to maximize or minimize a cost and can be solved graphically, algebraically, through Simplex Algorithm, barrier method and primal-dual IP method etc [45].

Alternating Direction Method of Multipliers (ADMM) solves the problems by segregating them into pieces making it easier to find the solution [46].

Non-Linear Programming Models are used for solving those problems that are non-linear in nature or the constraints are non-linear in nature [47].

Game Theory based models are of two types: cooperative and non-cooperative game theory. The outcome of a game theory model is based on strategic decisions of the players and the decision of one player affects the decision of other players [48].

The MILP and Auction mechanism are discussed separately in the subsections.

The dispatch mechanism used in thesis includes four cases of Mixed Integer Linear Programming (MILP)and a case of Vickrey Clarke Groves Auction(VCG Mechanism).
1) MILP using Fixed Demand-Variable Pricing
2) MILP using Fixed Demand-Variable Pricing (with only PV Charging)
3) MILP using Adjusted Demand-Minimum Local Price
4) MILP using Adjusted Demand-Minimum Local Price (with only PV Charging)
5) Vickrey Clarke Groves Auction Model

All dispatch mechanisms above utilize the local pricing obtained from a pricing strategy. Mechanisms in point 1 to 4 utilize same pricing equations, with points 3 and 4 using an additional optimization algorithm on this pricing formula to obtain a lower local price by adjusting the demand of each household. Auction mechanism uses a different price structure based on bids.

### 6.1.1 Mixed Integer Linear Programming(MILP)

The MILP model is preferred over other models for setting up dispatch in the thesis because, MILP algorithm allows to use binary variables (0,1) for setting up Consumer and Prosumer decisions easily and controlling them effectively during trading set up by a simple linear formulation of the minimization problem. If variables are integers, it is called a (pure) integer linear program (ILP, IP) and if all variables are allocated as 0 or 1 (Binary, Boolean), it becomes a 0-1 Linear program. For example, the decision to set up plant in yes or no can be converted into 0-1 Linear Program and used to minimize costs[49]. The MILP model used in the trading dispatch consists of two categories of decision variables for setting up trading dispatch and allocation.
a) Continuous Decision Variables
b) Binary Decision Variables.

The continuous decision variables comprise of real numbers (or an interval) and can take any value between a lower and upper bound. By Default, the numbers are positive and lower bound is set as 0 and the upper bound is set to infinite unless specified explicitly [50].

The binary variables can take only one option 0 or 1, indicating selection or rejection, a yes or no in a choice. For example, suppose a drug manufacturer wants to decide whether or not to use a fermentation tank. This decision is defined by a variable x. The choice can be modeled easily by setting this variable x to 0 or 1. In energy trading the binary variables have been used to set up selling and buying decisions for households and charging and discharging decisions for battery owners.

Constraints help in structuring the market model correctly to ensure that allocations are done properly and within the resources available and households do not transact energy outside their given specified limits. For example, Battery should be discharged only when its available energy is within minimum and maximum range or a household should not be buying energy beyond its given demand as this proves computation error. A simple MILP formulation is given below where a cost function is to be minimized by taking suitable value of x. The x takes value only when binary variable D takes value 1. And the D is set to 1 at only that value of x where cost function is found to be minimum and the constraint relation with A and B is satisfied [49][51].

$$\begin{aligned} Minimize\ C_{i,j}x \\ Subject\ to\ constraints: \\ Ax \leq B * D, D\epsilon(0,1) \\ x \geq 0 \end{aligned} \tag{6.1.1}$$

where,

| | |
|---|---|
| $C_{i,j}$ | = Cost Function |
| $x$ | = Continuous decision variable to be allocated to minimize cost |
| $D$ | = Binary variable that can take 0 or 1 |
| $A, B$ | = Real numbers |

Most constrained optimization models have been used before with simulated demand data sets like CREST, Homer etc. Also many simulated models used older meteo information to estimate solar PV generation. It was noted that very few models used actual real time data for setting up trading environments. The advantage of testing MILP based model helps in establishing the user and supplier decision environment virtually and also optimize the cost simultaneously.

The Mixed Integer Programming Model has been used before in many works before like a MILP model was proposed by Nguyen [41], for Australian Market which considered Battery Storage and PV and their respective investments for optimizing the savings. This model categorized the households into four groups with one group having no DERs, second group with only PV, third group with only Batteries and fourth group with both PV and batteries. However, Household with only batteries does not suit the trading market as owners with only Battery will be dependent on Grid and Local Market purchases only for charging their batteries. And the only way to use batteries profitably is to charge them when prices are low and sell the energy when the prices are high in the market, which may not be possible due to random changes in demand and generation. The MILP model tested in the thesis here does not considerv the group of only battery owners or any investments by the user as allocations may get biased towards prosumers while trying to achieve lower Levelized Cost of Energy (LCOE) and Levelized Cost of Storage (LCOS) in objective function. The focus of the MILP program is to attain best savings from the local pricing devised.

One of the approximate ways to solve 0-1 MILP is to use $2^n$ possible assignments of all variables (where n is number of variables used), and use the solution with minimum value out of those solution sets that are able satisfy constraints. But in many solvers, MILP models are solved by series of continuous (linear programming) relaxations one of them being branch-and-bound algorithm [52]. Gurobi Solver provides the Method parameter, which allows us choose the algorithm used to solve continuous models and offers various settings like cutting planes, heuristics, and search techniques for MIP models specifically. A major challenge in initiating local market trading using binary decision variables is that network constraints are not violated during the energy transactions [53]. Thus, MILP solver needs to deal with floating point inaccuracies encountered along the way in case on binary variables. For example, buy and sell are two binary decision variables for sales and purchases respectively, and we do not want them to happen simultaneously. We want that the sell should be set to 1, when buy is set to zero or vice versa. Thus, if floating point error is significant, the constraint may get ignored and both sell and buy could get allocated to 1, which is undesirable. Hence, Heuristics settings was not considered for the simulations as the results indeed provided a aggressive minimum solution but nature of heuristics tend to make it unreliable as a feasible solution, if constraints are relaxed heavily or precision is sacrificed, and this is not intended when working with binary decision variables [54]. The model implemented in the thesis utilizes dual method for solving MILP, which is default setting in the solver. It uses the LP relaxation technique and these underlying LP relaxations are solved by the dual method. Cuts have also been set to default, that is solver can automatically decide to use cutting planes method and apply cuts based on the problem [50]. The solver was also tested with Parameters set to Branch and Cut method as well but, it produced similar results for the load scenarios considered (as with default settings), hence, default setting was considered appropriate enough to set up trading dispatch.

### 6.1.2 Auction Models

An auction is a sales transaction in which the formation of prices commodities is through bidding process. Most common type of auction is the English Auction in which commodity is priced to zero first, then, bids are solicited from bidders with highest bid price set as item price. The Dutch Auction starts at a high price, which exceeds the item price, and subsequently decreases till a price is accepted by a bidder. The sealed bid option is a First price auction and is commonly used where each bidder submits a single bid in a sealed envelope and all envelopes are opened together to announce the highest bidder, and the item is sold at the highest bid price [55]. In the second-price sealed-bid auctions, bidders submit sealed envelopes in one round of bid submission and highest bid wins the item, but item is sold at second highest price bid as highest bid price often over-estimates the actual price value of item and thus, second price offers more truthfulness in price estimation. The sealed auctions are categorized into one sided auctions as the buyers participate in bidding process [55].

In a double-sided auction, all the buyers will submit their bid and the sellers also set specific prices for the commodities. Therefore, additional variables are formulated in auction model. Many energy trading models have proposed double auction theory [56] [57] and in auction models, it is often assumed that the participants are truthful in their actions. However, it is necessary the model should have an effective structure that can make user actions and bids transparent in nature [58][59].

This thesis looks forward to use some excerpt from Vickrey Clarke Groves Auction theory (a second price auction in which highest bidder buys commodity on second highest bid price) and use it in a simple first auction method to model a dispatch in which buyer bids are transparent and do not over estimate the local electricity using a suitable pricing strategy to devise buyer's bid price [60][61][62]. A VCG Auction has not been practiced much in the energy trading models before except a very few which have used technical aspects of network in their works [63]. The ideas from VCG model have been used to shape a new local trading dispatch model to see if it can help achieve fair allocation to all the users from the trading or not. The VCG auction has been simulated with standard python libraries as model does not have complex calculations as in MILP.

## 6.2 Pricing Strategy for MILP Models

This section explains the local pricing formula that will be implemented in dispatch mechanism stated in points 1 to 4 above (MILP schema). The auction model (VCG) in point 5 discussed later, uses pricing schema which is slightly modified version of this pricing strategy and has been covered in its respective section. The local pricing is set for MILP models is based on changing demand profiles of the consumer for the given hour, thus, a new rate at every hour for transaction is calculated that is used as the local price. The local prices has been proposed simply as a Average Function of Relative Normalized Demand Profile of the all users in a particular hour. Making Prices function of the demand can enhance response from the consumers to balance out usage and the prices consecutively, thereby, generating possibility of a cooperative or competitive decision making by the consumers. This is will increase community welfare and individual utility function of all users. The Local Pricing at each hour is calculated as [64][65]:

$$p_{loc} = \frac{1}{n} \sum_{i=1}^{n} \left\{ \frac{e_{d,i,t} - e_{d,i,t,min}}{e_{d,i,t,max} - e_{d,i,t,min}} * (p_g - p_{ft}) + (p_{ft}) \right\} \tag{6.2.1}$$

where,

| | |
|---|---|
| $p_{loc}$ | = Local price for t hour |
| $n$ | = Number of households |
| $e_{d,i,t}$ | = Demand (kWh) of ith household at t hour |
| $e_{d,i,t,min}$ | = Minimum Demand (kWh) within the Pool |
| $e_{d,i,t,max}$ | = Maximum Demand (kWh) within the Pool |
| $p_g$ | = Grid Price |
| $p_{ft}$ | = Feed in Tarrif |

The grid price $p_g$ is the price at which consumer can buy energy from the utility company for the surplus demand not met by DERs and $p_{ft}$ is the tariff price at which prosumers can sell their surplus energy to the utility company. The local price $p_{loc}$ always remains within the grid price $p_g$ and feed-in tariff price $p_{ft}$ so that households will feel motivated to adopt local trading. The prices below grid price is attractive choice for households selling their surplus energy get higher price than feed-in tariff price, if they sell energy at local price. The working of the pricing strategy can be assumed to help user control their demand share in next hour in order to impact the prices in the local market. For example, prosumers can increase or decrease prices in the market by analyzing the maximum profit they can derive from the market by sales by lowering their demand and increasing surplus energy in pool. Similarly, the consumer can control its demand and decide buy or not buy based on the price prevailing in market. Similarly, the households can also work together to optimize their demands to reduce the local pricing to achieve maximum benefit in terms of consumption and savings. It is assumed that, electricity need is inelastic, and users will consume their minimum requirements every day. The prices do not bend drastically based on demand, and it is ensured that the local prices always stay between grid price and tariff price so that prosumers are able to generate revenue and consumers are able to earn savings from their purchases. If a household does not have any demand in the pool, it need not to participate in the local market and the pricing is derived from the number of users participating in the market. The impact of pricing strategy on user demand can be visualized through following example:

Table 6.1 (Response Configuration-Initial State) below consists of demand at 0th hour for households C1 to C8, where households from C1 to C4 are prosumers and C7, C8 are consumers without any DERs. Suppose, Prosumer C4 is having total energy as 2 kWh for usage and trading. The surplus C4 can offer in the local market is 2 kWh-1.762 kWh = 0.238 kWh, making revenue from the sales to be 0.238 x 25.1 cents = 5.9 cents. However, C4 can change its revenue, if it changes its demand numbers and affect local price in the pool.

Table 6.1: Response Configuration-Initial state

| Hour | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | Price ($) |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.281 | 1.693 | 0.829 | 1.762 | 0.308 | 1.019 | 0.293 | 0.963 | 25.1 cents |

In Table 6.2 (Response Configuration-Prosumer C4), Suppose C4 changes its demand to 0.700 kWh. The local price at 0th hour reduces to 22.5 cents and C4 can generate revenue of (2 kWh-.700 kWh) x 22.5 cents = 29.25 cents, if it sells this surplus energy to local market. It is expected that consumer will take advantage of the reduction is prices and prefer buying at a lower cost.

Table 6.2: Response configuration -Prosumer C4

| Hour | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | Price ($) |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.281 | 1.693 | 0.829 | 0.7 | 0.308 | 1.019 | 0.293 | 0.963 | 22.5 cents |

Table 6.3 (Response configuration-Consumer C8) denotes consumer response to local price. C8 (a consumer without any storage or PV) reduces consumption in the 0th hour. This reduces the local price in the pool at 0th hour to 21 cents. Prosumers C1 to C6 can still sell energy at a considerable profit, if they have any surplus generation.

Table 6.3: Response Configuration-Consumer C8

| Hour | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | Price ($) |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.281 | 1.693 | 0.829 | 0.7 | 0.308 | 1.019 | 0.293 | 0.5 | 21.0 cents |

The impact of the pricing schema can be effective when combined with suitable demand response configurations in a game environment and it can help both consumer and prosumer some control over market prices. The game theory practice for price optimization is not in the scope of the thesis right now and an optimization algorithm is implemented to model a dispatch scheme that mimics households' purchase and sales decision. The results are used to measure the amount of savings, self-sufficiency and self-consumption of the community based on this variable pricing schema.

## 6.3 Fixed Demand-Variable Pricing

The fixed Demand and Variable pricing scheme is implemented for each hour in following steps for the load scenarios:

1. Local Price Calculation
2. Problem Formulation and Objective Function
3. Setting up Constraints
4. Trading Allocation

The price calculation has been done using Equation 6.3.1. The local pricing is calculated for each hour and used for community trading between set of households.

The Problem formulation involves setting the goal of the energy trading model which is to minimize the consumption from the grid, reduce the household bill and increase community savings. Hence, the problem is derived as a Minimization Problem that aims to minimize purchase costs from the grid. The minimization problem is solved at each hour and total 48 iterations take place.

Thus, the Objective Function is to minimize grid consumption subject to certain constraints. This is stated by:

$$minimize : \sum_{i=1}^{n} e_{buy,grid,i,t} * p_g + \sum_{k=1}^{m} e_{ch,k,t} * p_g \tag{6.3.1}$$

where,

$e_{buy,grid,i,t}$ = Energy bought from grid (kWh) by all user Groups (A,B,C) at t hour

$e_{ch,grid,k,t}$ = Charging bought from grid (kWh) by Group C users at t hour

Subject to Constraints [41][66] :

1) Group A (without PV/Battery Storage):

$$e_{d,i,t} = e_{buy,loc,i,t} + e_{buy,grid,i,t} \tag{6.3.2}$$

$$buy_{i,t} = 1 \tag{6.3.3}$$

where,

| | |
|---|---|
| $e_{d,i,t}$ | = Demand (kWh) of ith household at t hour |
| $e_{buy,grid,i,t}$ | = Energy purchased from grid (kWh) for meeting demand |
| $e_{buy,loc,i,t}$ | = Energy purchased from local market (kWh) for meeting demand |
| $buy_{i,t}$ | = Buy Decision |

The Continuous Decision variables set for the users are $e_{buy,grid,i,t}$ and $e_{buy,loc,i,t}$ and $buy_{i,t}$ is the Binary Decision Variable. It is set to 1, if the consumer decides to buy energy from local market or grid else stays 0. Equation 6.3.2 satisfies the condition that demand for this Group of users is met by purchases from grid and local market. As the Users in this group do not have any PV or Battery Storage, so the buying decision will always be set to 1 (given by Equation 6.3.3).

2) Group B (with PV Only) [41][66] :

$$e_{d,i,t} = e_{pvuse,i,t} + e_{buy,loc,i,t} + e_{buy,grid,i,t} \tag{6.3.4}$$

$$e_{pvuse,i,t} + e_{sell,loc,i,t} + e_{sell,grid,i,t} = e_{pv,i,t} \tag{6.3.5}$$

$$sell_{i,t} + buy_{i,t} \leq 1 \tag{6.3.6}$$

$$e_{pvsell,loc,i,t} + e_{pvsell,grid,i,t} \leq e_{pv,i,t} * sell_{i,t} \tag{6.3.7}$$

$$e_{buy,loc,i,t} + e_{buy,grid,i,t} \leq e_{d,i,t} * buy_{i,t} \tag{6.3.8}$$

where,

| | |
|---|---|
| $e_{d,i,t}$ | = Demand (kWh) of ith household at t hour |
| $e_{buy,grid,i,t}$ | = Energy purchased from grid (kWh) for meeting demand |
| $e_{buy,loc,i,t}$ | = Energy purchased from local market (kWh) for meeting demand |
| $e_{pvuse,i,t}$ | = PV energy used to meet demand (kWh) |

| | |
|---|---|
| $e_{pvsell,loc,i,t}$ | = PV energy sold to local market (kWh) |
| $e_{pvsell,grid,i,t}$ | = PV energy sold to Grid (kWh) |
| $e_{pv,i,t}$ | = Total PV Generation (kWh) |
| $buy_{i,t}$ | = Buy Decision |
| $sell_{i,t}$ | = Sell Decision |

Continuous Decision Variables for this user group includes $e_{buy,grid,i,t}$, $e_{buy,loc,i,t}$ , $e_{pvuse,i,t}$, $e_{sell,loc,i,t}$, and $e_{sell,grid,i,t}$ allocated by solver. Group B users are Prosumer in the day time and Consumers in the night time as they have only PV generator installed. Equation 6.3.4 ensures that household demand is fulfilled from PV usage, local and grid purchases. The sum of PV used and that sold to local market or Grid will equal the total PV generated by prosumer in given hour (Equation 6.3.5). Binary Variables $buy_{i,t}$ and are $sell_{i,t}$ given by Equation 6.3.6 which ensures that user can either buy or sell or not trade at all in a particular hour. Equations 6.3.7 and 6.3.8 use the Binary Variables for sales and buying transactions by allotting 1, if transaction is taking place, else sets them to 0.

2) Group C (with PV and Battery) [41][66]: The constraints for this user group is specified as:

$$e_{d,i,t} = e_{pvuse,i,t} + e_{btuse,i,t} + e_{buy,loc,i,t} + e_{buy,grid,i,t} \tag{6.3.9}$$

$$e_{pvuse,i,t} + e_{pvcharge,i,t} + e_{sell,loc,i,t} + e_{sell,grid,i,t} = e_{pv,i,t} \tag{6.3.10}$$

$$sell_{i,t} + buy_{i,t} \leq 1 \tag{6.3.11}$$

$$e_{pvsell,loc,i,t} + e_{pvsell,grid,i,t} + e_{btsell,loc,i,t} + e_{btsell,grid,i,t} \leq (e_{pv,i,t} + c_{i.t}) * sell_{i,t} \tag{6.3.12}$$

$$e_{buy,loc,i,t} + e_{buy,grid,i,t} \leq e_{d,i,t} * buy_{i,t} \tag{6.3.13}$$

$$e_{buych,loc,i,t} + e_{buych,grid,i,t} \leq c_{i,t} * buy_{i,t} \tag{6.3.14}$$

where,

| | |
|---|---|
| $e_{d,i,t}$ | = Demand (kWh) of ith household at t hour |
| $e_{buy,grid,i,t}$ | = Energy purchased from grid (kWh) for meeting demand |
| $e_{buy,loc,i,t}$ | = Energy purchased from local market (kWh) for meeting demand |

| | |
|---|---|
| $e_{pvuse,i,t}$ | = PV energy used to meet demand (kWh) |
| $e_{pvsell,loc,i,t}$ | = PV energy sold to local market (kWh) |
| $e_{pvsell,grid,i,t}$ | = PV energy sold to Grid (kWh) |
| $e_{pv,i,t}$ | = Total PV Generation (kWh) |
| $e_{btuse,i,t}$ | = Battery energy used (kWh) |
| $e_{pvcharge,i,t}$ | = PV energy used for battery charging (kWh) |
| $e_{btsell,grid,i,t}$ | = Battery energy sold to grid (kWh) |
| $e_{btsell,loc,i,t}$ | = Battery energy sold locally (kWh) |
| $c_{i,t}$ | = Maximum transaction(charge/discharge) limit for battery at hour t (kWh) |
| $buy_{i,t}$ | = Buy Decision |
| $sell_{i,t}$ | = Sell Decision |

Group C households have additional continuous variables to set up transactions with battery storage. The Demand of the households in this group is met by PV generation, battery storage, purchases from grid and local market (Equation 6.3.9). The sum of energy used from PV for meeting demand, for charging batteries, PV sold locally and to grid at each hour is equal to total PV generated at that time (Equation 6.3.10). The buying and selling cannot be done together by group C households in a particular hour, same as group B households. Also, group C households can sell battery energy at night, if it is available (Equation 6.3.11). Buying decisions are set to continuous buying variables and selling decisions are set to all continuous sales variables (Equation 6.3.12 and 6.3.13). This means that, prosumer household can sell energy by setting $sell_{i,t}$ to 1 and $buy_{i,t}$ to 0. And buy from local market or grid by setting $sell_{i,t}$ to 0 and $buy_{i,t}$ to 1 to get the allocation. Additional buying transactions involve buying battery charge from local market and grid (Equation 6.3.14). The maximum discharge or charge limit for battery storage is specified by limit $c_{i,t}$. Transaction with battery (charge or discharge) is done within a specified limit as battery is required to serve purpose of back up and also for energy trading. For example, a household with a battery size of 25 kWh cannot discharge the battery more than $c_{i,t}$ of 2 kWh at each hour, so that sufficient back up is maintained for emergencies.

The Transactions for the battery are controlled using binary decision variables for charging ($ch_{i,t}$) and discharging ($disch_{i,t}$). Charging decisions are used for buying charge from the local market ($e_{buych,loc,i,t}$) or grid ($e_{buych,grid,i,t}$) and charging the battery bank through PV ($e_{pvcharge,i,t}$) (Equation 6.3.15). Similarly, discharge decisions include using the battery for meeting demand ($e_{btuse,i,t}$), selling battery energy to grid ($e_{btsell,grid,i,t}$) or to local market ($e_{btsell,loc,i,t}$) (Equation 6.3.16). These battery transaction variables are allocated by solver based on the constraint and minimization objective. Additional constraints for the battery transactions are formulated below.

$$e_{pvcharge,i,t} + e_{buych,loc,i,t} + e_{buych,grid,i,t} \leq c_{i,t} * ch_{i,t} \tag{6.3.15}$$

$$e_{btuse,i,t} + e_{btsell,loc,i,t} + e_{btsell,grid,i,t} \leq c_{i,t} * disch_{i,t} \tag{6.3.16}$$

where,

| | |
|---|---|
| $e_{btuse,i,t}$ | = Battery used |
| $e_{pvcharge,i,t}$ | = PV energy used for battery charging (kWh) |
| $e_{buych,grid,i,t}$ | = Buy charge from grid (kWh) |
| $e_{buych,loc,i,t}$ | = Buy charge locally (kWh) |
| $e_{btsell,grid,i,t}$ | = Battery energy sold to grid (kWh) |
| $e_{btsell,loc,i,t}$ | = Battery energy sold locally (kWh) |
| $c_{i,t}$ | = Maximum transaction(charge/discharge) limit for battery at hour t (kWh) |
| $buy_{i,t}$ | = Buy Decision |
| $sell_{i,t}$ | = Sell Decision |
| $ch_{i,t}$ | = Charge Decision |
| $disch_{i,t}$ | = Discharge Decision |

The Battery status is supposed to always stay within its designated battery limits and should not go below the minimum battery energy or exceed its maximum energy rating (Equation 6.3.17 and Equation 6.3.18). The maximum energy is the battery rating in kWh and minimum energy is set to zero units or some percentage of total battery size in each load case scenarios (for example, a 22.5 kWh Battery can operate between 0 kWh and 22.5 kWh, that is it will go to charging mode, when it reaches close to 0 kWh and will participate in trading, only when it goes above 0 kWh and is able to charge at-least up to its $c_{i,t}$ value). After every charge or discharge transaction in a given hour, the battery status is updated i.e., the battery status calculated at t hour is fed as input to the t+1 hour for next transaction and the decision to charge or discharge is taken based on this battery status (Equation 6.3.19 and Equation 6.3.20) [41].

$$e_{bt,i,t} \geq e_{Minbt,i,t} \tag{6.3.17}$$

$$e_{bt,i,t} \leq e_{Maxbt,i,t} \tag{6.3.18}$$

$$e_{bt,i,t+1} = e_{bt,i,t} + e_{buych,loc,i,t} + e_{buych,grid,i,t} + e_{pvcharge,i,t} \tag{6.3.19}$$

$$e_{bt,i,t+1} = e_{bt,i,t} - (e_{sellch,loc,i,t} + e_{sellch,grid,i,t} + e_{btuse,i,t}) \tag{6.3.20}$$

where,

| | |
|---|---|
| $e_{bt,i,t+1}$ | = Battery Status at t+1 hour (kWh) |
| $e_{bt,i,t}$ | = Battery status at t hour (kWh) |
| $e_{btuse,i,t}$ | = Battery energy used (kWh) |
| $e_{pvcharge,i,t}$ | = PV energy used for battery charging (kWh) |
| $e_{buych,grid,i,t}$ | = Buy charge from grid (kWh) |
| $e_{buych,loc,i,t}$ | = Buy charge locally (kWh) |
| $e_{btsell,grid,i,t}$ | = Battery energy sold to grid (kWh) |
| $e_{btsell,loc,i,t}$ | = Battery energy sold locally (kWh) |
| $c_{i,t}$ | = Maximum transaction(charge/discharge) limit for battery at hour t (kWh) |

$buy_{i,t}$ = Buy Decision

$sell_{i,t}$ = Sell Decision

$ch_{i,t}$ = Charge Decision

$disch_{i,t}$ = Discharge Decision

## 6.4 Fixed Demand-Variable Pricing (Only PV charging)

Simulation of above section 6.3 was further extended with another situation, in which battery charging was restricted by usage of PV surplus only, and charging purchases from grid and local market for the battery prosumers was removed for Group C households. It was expected that with local pricing strategy, this change can reduce additional purchases from grid or local market and reduce the expenses, however, the dependency on PV charging may affect overall supply of battery in the pool for some given time periods and battery storage may not be able to discharge or participate in the local market with given limitation. But it was necessary to test this criteria to see the impact on the individual and community savings and how measurement indices perform with this.

The objective function and constraints for Group A and B Households remain same as in section 6.3 (Equation 6.3.2 to Equation 6.3.8) and pricing strategy also works on same calculation as in section 6.2 (Equation 6.2.1).

Group C users own both PV and Battery System, but they are now restricted to use PV surplus to charge the battery. The usage from PV is gets prioritized in following manner.

a) The PV is used to meet self demand first.

b) The battery status is continuously monitored and it is checked whether battery goes below specified limit or needs charging. If self demand is met from the PV and surplus PV is available, it is utilized for charging the battery.

c) After both self demand and Battery needs is fulfilled, the remaining surplus PV is sold to local market or grid based on overall demand in the pool and minimum obtained for the objective function (which is to minimize grid purchases).

d) If demand is met and Battery does not need charging. Prosumers can decide to sell surplus PV directly to local market or grid.

e) Thus, variables $e_{buych,loc,i,t}$ and $e_{buych,grid,i,t}$ are not used in this transaction anymore. The changes in charging constraints can be seen through Equation 6.4.6. The Group C Constraints are formulated as :

$$e_{d,i,t} = e_{pvuse,i,t} + e_{btuse,i,t} + e_{buy,loc,i,t} + e_{buy,grid,i,t} \tag{6.4.1}$$

$$e_{pvuse,i,t} + e_{pvcharge,i,t} + e_{sell,loc,i,t} + e_{sell,grid,i,t} = e_{pv,i,t} \tag{6.4.2}$$

$$sell_{i,t} + buy_{i,t} \leq 1 \tag{6.4.3}$$

$$e_{pvsell,loc,i,t} + e_{pvsell,grid,i,t} + e_{btsell,loc,i,t} + e_{btsell,grid,i,t} \leq (e_{pv,i,t} + c_{i.t}) * sell_{i,t} \tag{6.4.4}$$

$$e_{buy,loc,i,t} + e_{buy,grid,i,t} \leq e_{d,i,t} * buy_{i,t} \tag{6.4.5}$$

$$e_{pvcharge,i,t} \leq c_{i,t} * ch_{i,t} \tag{6.4.6}$$

$$e_{btuse,i,t} + e_{btsell,loc,i,t} + e_{btsell,grid,i,t} \leq c_{i,t} * disch_{i,t} \tag{6.4.7}$$

$$e_{bt,i,t} \geq e_{Minbt,i,t} \tag{6.4.8}$$

$$e_{bt,i,t} \leq e_{Maxbt,i,t} \tag{6.4.9}$$

$$e_{bt,i,t+1} = e_{bt,i,t} + e_{pvcharge,i,t} \tag{6.4.10}$$

$$e_{bt,i,t+1} = e_{bt,i,t} - e_{sellch,loc,i,t} + e_{sellch,grid,i,t} + e_{btuse,i,t} \tag{6.4.11}$$

where,

| | |
|---|---|
| $e_{d,i,t}$ | = Demand (kWh) of ith household at t hour |
| $e_{pvuse,i,t}$ | = PV energy used to meet demand (kWh) |
| $e_{btuse,i,t}$ | = Battery energy used (kWh) |
| $e_{buy,grid,i,t}$ | = Energy purchased from grid (kWh) for meeting demand |
| $e_{buy,loc,i,t}$ | = Energy purchased from local market (kWh) for meeting demand |
| $e_{pvsell,loc,i,t}$ | = PV energy sold to local market (kWh) |
| $e_{pvsell,grid,i,t}$ | = PV energy sold to Grid (kWh) |
| $e_{pv,i,t}$ | = Total PV Generation |

| | |
|---|---|
| $e_{pvcharge,i,t}$ | = PV energy used for battery charging (kWh) |
| $e_{btsell,grid,i,t}$ | = Battery energy sold to grid (kWh) |
| $e_{btsell,loc,i,t}$ | = Battery energy sold locally (kWh) |
| $c_{i,t}$ | = Maximum transaction(charge/discharge) limit for battery at hour t (kWh) |
| $buy_{i,t}$ | = Buy Decision |
| $sell_{i,t}$ | = Sell Decision |
| $ch_{i,t}$ | = Charge Decision |
| $disch_{i,t}$ | = Discharge Decision |
| $e_{bt,i,t+1}$ | = Battery Status at t+1 hour (kWh) |
| $e_{bt,i,t}$ | = Battery status at t hour(kWh) |

## 6.5 Adjusted Demand-Minimum Local Price

A Simple adjustment in demand without assuming any case of user willingness or profit considerations of the prosumers is fed to the MILP computation to see the impact of trading, the model excludes time of use and complex equipment adjustments as in [67] and looks forward to analyze economic aspects of demand adjustment and its effect on savings and local pricing strategy that was devised in section 6.2 (Equation 6.2.1). It was assumed that consumers and prosumers are cooperating with each other to reduce local price, by adjusting their respective demand. It has been considered that households have a minimum consumption and this has been incorporated in the load scenario cases to ensure that the mandatory equipments consume some power and household demand never goes to zero. The prosumers are not risk averse and hence, are not greedy for revenue. Most of the optimization models and game theory have tested greedy algorithms or non-cooperative game theory for resource allocation [68][69][70]. This model was tested to check whether community based savings can be improved by reducing the local price to a good value or not.

### 6.5.1 Adjusted Demand and Local Price Calculation

The idea of co-operative game in which the agents work together to bring down the local market pricing by adjusting their demand [71][72] is formulated below in steps: a) First, the prices are displayed for the local market based on the current demand. b) The demand is then optimized for each agent targeting to get a new minimum local price in the pool based on the PV and Battery Supply available at a given hour. c) Dispatch is then optimized to obtain Minimum grid utilization and effective local trading of resources with MILP program.

The equation for demand adjustment is simply though a Linear Programming equation using scipy optimization toolbox in Python.

$$Minimize: \; p_{loc} \tag{6.5.1}$$

subject to constraints:

$$\sum_{i=1}^{n} e_{dnew,i,t} \leq \sum_{i=1}^{n} e_{pv,i,t} + \sum_{i=1}^{n} c_{i,t} \tag{6.5.2}$$

$$\frac{1}{n}\sum_{i=1}^{n} e_{dnew,i,t} \leq \frac{1}{n}\sum_{i=1}^{n} e_{d,i,t} \tag{6.5.3}$$

$$e_{dnew,i,t} > 0 \tag{6.5.4}$$

where,

| | |
|---|---|
| $n$ | = Number of households |
| $e_{dnew,i,t}$ | = Adjusted Demand (kWh) of ith household adjusted at t hour |
| $e_{d,i,t}$ | = Demand (kWh) of ith household seen initially at t hour |
| $e_{pv,i,t}$ | = Total PV Generation (kWh) |
| $c_{i,t}$ | = Maximum transaction(charge/discharge) limit for battery at hour t (kWh) |

The above equation uses the local pricing formula in Equation 6.2.1 for calculating

$p_{loc}$ and adjusts the demand of each household till a minimum local price is reached in the pool. The local price stays between feed in tariff ($p_{ft}$) and grid price($p_g$).

The objective function and constraints are all set up for this model based on adjusted demand ($e_{dnew,i,t}$) instead of the actual demand data ($e_{d,i,t}$). The trading allocation is generated through MILP based algorithm and follows same rules as in section 6.3 with demand $e_{d,i,t}$ replaced with adjusted demand $e_{dnew,i,t}$ in objective function and uses all constraints same as section 6.3.

## 6.6 Adjusted Demand-Minimum Local Price (Only PV Charging)

The Adjusted demand scenario was also tested with restriction of charging with PV only to see if model performance can be improved further. The Group C constraints were changed to remove charging purchases from grid and local market, and batteries were made to charge with surplus PV only. The Equations were set similarly as in section 6.4 to see whether the adjusted demand can work out with the given PV and battery supply as it was expected that adjusted demand may work within the given supply pool and may boost local market usage, increase savings and improve measurement indices. New adjusted demand was computed ($e_{dnew,i,t}$) for a minimum local price $p_{loc}$ (through Equations 6.5.1 to 6.5.4 and Equation 6.2.1 respectively), and this updated demand and local price was fed to the MILP program for creating trading instance with similar constraints as in section 6.4 for Group C (Equation 6.4.1 to 6.4.11).

## 6.7 Vickrey Clarke Groves Auction Model (VCG Auction)

Vickrey Clarke Groves Auction Model or VCG Auction is a sealed price bid auction technique and uses bidding system for transaction of commodities. The commodity to be sold is broadcasted to the buyer in real time and bids are collected based on the utility function of the buyer[61]. VCG Auction model is used in both first price and second price bidding, however, second price auction has been more popular as first price auction often over estimates the actual price of the commodity [73][74][75]. Hence, it motivates truthfulness of the bidder and maximizes social welfare. In local market trading case, the excerpts from the VCG auction model have been applied to remove problem associated with first price auction by calculating bids through a systematic pricing strategy and aligning the buying bids from highest to lowest. The transaction begins from highest to lowest bids till energy from DERs is fully consumed or all demand is met in the pool [61]. For prosumers, the net demand is the surplus demand after all the DERs is utilized and they become buyer to procure energy from the local market, if all their generated energy is self-utilized. For consumers without any DERs the net surplus demand and demand are same. The household becomes a prosumer only when he has surplus energy in the pool and cannot buy or sell at same instant.

### 6.7.1 Pricing Strategy

The bid price has been derived as function of net demand similar the previous MILP case, but the average one-time local price has been discarded and the individual prices have been considered and stated as bid price. Thus, price is directly proportional to the demand that is, higher demand makes the price bid higher in the market pool and customer with lower demand gets a lower buying bid. This makes the bid fair for each buyer as the bids cannot over-estimate or under-estimate the value of the electricity. The price bids always range between Feed in Tariff and Grid Price, and it is assumed that demand of every consumer will always be different. Thus, bid price of each consumer in pool is given by:

$$p_{bid,i,t} = p_g - \left\{ \frac{\sum e_{dnet,i,t} - e_{dnet,i,t}}{\sum e_{dnet,i,t}} * (p_g - p_{ft}) \right\} \tag{6.7.1}$$

where,

$p_{bid,i,t}$ = Bid price for ith consumer for t hour

$e_{dnet,i,t}$ = Surplus net Demand (kWh) of ith household for purchase after self DER usage at t hour

$p_g$ = Grid Price

$p_{ft}$ = Feed in Tarrif

### 6.7.2 Trading Mechanism

A cycle in hour 0 has be used to exemplify the procedure. The auction is modeled in following steps: Suppose Households C1 to C6 are prosumers with DERs and, C7 and C8 are consumers without any DERs. Transaction is initiated based on energy available with the prosumers.

1) At a given hour the total demand and supply is assessed (PV+Battery) and the usable demand of the prosumer is first fulfilled by their own generator. Any surplus in its generation after self-usage qualifies the prosumer to become a supplier. If prosumer has more demand than the generation, then it consumes all its generation and meets the surplus demand by becoming a consumer in the pool. If all demand is met by prosumer and there is no surplus demand and supply, the prosumer does not participate in trading (Table 6.4, VCG: Step 1).

Table 6.4: VCG: Step 1

| Demand kWh | | | | | | | | Supply kWh | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C1 | C2 | C3 | C4 | C5 | C6 |
| 0.281 | 1.693 | 0.829 | 1.762 | 0.308 | 1.019 | 0.293 | 0.963 | 0 | 0 | 2 | 2 | 1 | 2 |

2) The surplus supply in the market is obtained once demand of the prosumers is met and it is ordered from lowest to highest (Table 6.5, VCG: Step 2).

Table 6.5: VCG: Step 2

| Net Demand/Consumer kWh | | | | Surplus Supply kWh | | | |
|---|---|---|---|---|---|---|---|
| C1 | C2 | C7 | C8 | C4 | C5 | C6 | C3 |
| 0.281 | 1.693 | 0.293 | 0.963 | 0.238 | 0.692 | 0.981 | 1.171 |

3) Net-Demand in the pool is obtained and bids are calculated and posted in the trading screen along with the surplus supply in local market. Bids are lined up the pool in descending order, and supply in ascending order. The transactions starts with C4 as the supplier and C2 as the priority buyer. The supplier pool has been sorted in ascending order to let all the prosumers earn revenues in unbiased manner (Table 6.6, VCG: Step 3). At the end of the local trading, either all the supply gets finished or all demand is met. If surplus supply is left in the pool after all the demand met by the supply, the surplus energy is sold to grid. Similarly, with supply consumed by all households and surplus demand remaining, grid is used for meeting any extra demand. Thus, grid interaction is minimized and used only after local trading is completed.

Table 6.6: VCG: Step 3

| Net Demand & Bids/Consumers | | | | | Surplus Supply /Sellers | | | |
|---|---|---|---|---|---|---|---|---|
| Consumer | C2 | C8 | C7 | C1 | C4 | C5 | C6 | C3 |
| Demand kWh | 1.693 | 0.963 | 0.293 | 0.238 | 0.238 | 0.692 | 0.981 | 1.171 |
| Consumer Bids cents | 29.08 | 21.02 | 13.63 | 13.5 | | | | |

The Table 6.7 (Trading sequence for 0th Hour) below summarizes the trading sequence for the 0th hour between suppliers and buyers. C2 is the highest bidder in the pool and C4 is the supplier having lowest share of DERs. To get fair revenue between suppliers, C4 gets priority to sell first. C4 sells energy to C2 at highest bid price of 29 cents. After C4's energy is used up, C5 and C6 are the next suppliers in the queue and sell their energy at 29 cents (bid price of C2) to C2. After C2's demand is met, C8 being the next highest bidder buys remaining energy from C6 at 21 cents (C8's bidding price). Thus, C6 gets to sell its energy to two buyers at their respective bid prices. The sequence continues till all local energy is used up in the pool. C1 being the lowest bidder in the pool buys some portion of energy from local market and remaining from the grid. It is expected that this pricing scheme and dispatch will be fair to both prosumers and consumers in terms of savings and revenue respectively.

Table 6.7: Trading sequence for 0th Hour

| 0th Hour/Iteration | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Supplier | C4 | C5 | C6 | C6 | C3 | C4 | C5 | Grid |
| Supply | 0.24 | 0.69 | 0.98 | 0.22 | 1.17 | 0.43 | 0.13 | 0.00 |
| Buyer kWh | C2 | C2 | C2 | C8 | C8 | C7 | C1 | C1 |
| Demand kWh | 1.69 | 1.46 | 0.76 | 0.96 | 0.75 | 0.29 | 0.28 | 0.15 |
| Residual Supply kWh | 0.00 | 0.00 | 0.22 | 0 | 0.43 | 0.13 | 0.00 | 0.00 |
| Residual Demand kWh | 1.46 | 0.76 | 0.00 | 0.75 | 0.00 | 0.00 | 0.15 | 0.00 |
| Energy sold kWh | 0.24 | 0.69 | 0.76 | 0.22 | 0.75 | 0.29 | 0.13 | 0.15 |
| Bid Price ($) | 0.29 | 0.29 | 0.29 | 0.21 | 0.21 | 0.14 | 0.14 | 0.46 |
| Revenue/Cost ($) | 0.07 | 0.20 | 0.22 | 0.05 | 0.16 | 0.04 | 0.02 | 0.07 |

# CHAPTER 7: MEASUREMENT INDICES

The trading results have been compared and analyzed based on individual allocations to households and community totals for 48 hours. Model performance is measured in terms of savings, Self Sufficiency (SS), Self Consumption (SC), and Fairness Index ($F(X)$). For each model, the user allocations for the 48 hours are summarized with their net savings compared to the conventional bill. Similarly, community trading results for 48 hours are also calculated to observe savings at the macro level.

## 7.1 Savings

The savings are calculated using conventional bill with grid price and is calculated as difference between Conventional bill and Net Purchase cost, which is the difference between local trading expenditure and Revenue. The Percentage Savings shall be with respect to the Conventional bill and will be 100% if the prosumers earn profit from the local trading or expenditure in electricity purchase is nullified by sales revenue. The individual and community savings are calculated based on following formula.

$$Savings = \sum e_{d,i,t} * p_g - \sum e_{buy,grid,i,t} * p_g + \left(\sum e_{buych,loc,i,t} + \sum e_{buy,loc,i,t}\right) * p_{loc} - \sum e_{pvsell,lgrid,i,t} * p_{ft} + \left(\sum e_{pvsell,grid,i,t} + \sum e_{btsell,loc,i,t} + \sum e_{pvsell,loc,i,t}\right) * p_{loc} \tag{7.1.1}$$

where,

| | |
|---|---|
| $e_{d,i,t}$ | = Demand |
| $p_g$ | = Grid Price |
| $e_{buych,grid,i,t}$ | = Buy charging from grid |
| $e_{buych,loc,i,t}$ | = Buy charge locally |
| $e_{btsell,grid,i,t}$ | = Battery sold to grid |

| | |
|---|---|
| $e_{btsell,loc,i,t}$ | = Battery sold locally |
| $e_{pvsell,loc,i,t}$ | = PV sold locally |
| $e_{pvsell,grid,i,t}$ | = PV Sold to grid |
| $p_{ft}$ | = Local Price |

The individual savings are based on transaction totals of 48 hours for each household, whereas community totals add up all households as well for the 48 hours.

## 7.2 Self-Sufficiency (SS)

Self-Sufficiency (SS) is defined as amount of demand that can be met by local market or self generation. It indicates reliability that can be extracted from the local generation measures, when grid supply is not available [76]. It is formulated as:

$$SS\% = \frac{\sum_{t=1}^{48} e_{pvuse,i,t} + \sum_{t=1}^{48} e_{btuse,i,t} + \sum_{t=1}^{48} e_{buy,loc,i,t}}{\sum_{t=1}^{48} e_{d,i,t}} \tag{7.2.1}$$

where,

| | |
|---|---|
| $e_{d,i,t}$ | = Demand |
| $e_{btuse,i,t}$ | = Battery used |
| $e_{pvuse,i,t}$ | = PV used to meet demand |
| $e_{buy,loc,i,t}$ | = Energy purchased from local market(kWh) for meeting demand |

## 7.3 Self-Consumption (SC)

Self Consumption (SC) is defined by Long [25] as the ratio between PV energy used to the Total generation. This used energy is not exported to grid, but used locally [77]. For the given dispatch models, the self consumption can be calculated as ratio of sum of total PV used (for load and PV charging), Battery used, PV/battery purchased locally to the Total Supply (sum of Total PV generated and Total Battery limit available for that time). The transaction of the battery depends on the battery status which in turn affects the total supply in a given hour. Battery dispatch depends on whether it is within its maximum and minimum range or not, and the battery discharge limit will be added to the total supply only when it is within this range and is used in the dispatch mechanism. For example, if the battery of 25.2 kWh has discharge limit of 2kWh and is available for 3 hours, the net availability is taken as 3*2 kWh = 6 kWh. If the battery is discharged to its minimum value (say 0 kWh) after using or selling 6 kWh and does not get charged for the remaining 48-3 = 45 hours, the total battery which was available for trading becomes 6 kWh after the total trading period (48 hours). Thus, the notation $\sum_{t=1}^{48} c_{net,t}$ can vary for the prosumers in the time period based on their discharge decisions and hence, total supply (PV+Battery) for use and in local market/grid also varies for the trading hour.

$$SC\% = \frac{\sum_{t=1}^{48} e_{pvuse,i,t} + \sum_{i=1}^{48} e_{pvcharge,i,t} + \sum_{i=1}^{48} e_{buych,loc,i,t} + \sum_{t=1}^{48} e_{btuse,i,t} + \sum_{t=1}^{48} e_{buy,loc,i,t}}{\sum_{t=1}^{48} e_{pv,i,t} + \sum_{t=1}^{48} c_{net,t}} \tag{7.3.1}$$

where,

| | |
|---|---|
| $e_{btuse,i,t}$ | = Battery used |
| $e_{pvuse,i,t}$ | = PV used to meet demand |
| $e_{pvcharge,i,t}$ | = PV used for battery charging |
| $e_{buych,loc,i,t}$ | = Buy charge locally |
| $c_{net,t}$ | = transaction (discharge) limit available for battery for hour t for local use |
| $e_{buy,loc,i,t}$ | = Energy purchased from local market(kWh) for meeting demand |

## 7.4 Fairness Index F(X)

Social Welfare can be measured in terms of Fairness Index as we need to assess individual amount benefit received from trading to each community member. The welfare is achieved when every member gets a fair share of the allocation from the pool through self usage and local trading, and receives optimum allocation with respect to their demand. Fairness index was proposed by Jain [32] and was used to measure TCP fairness in network engineering and in congestion control mechanisms for determining whether users were receiving a fair share of system resources. Fairness index is formulated as follows [78].

$$F(X) = \frac{(\sum_{i=1}^{n} x_i)^2}{(n * \sum_{i=1}^{n} x_i^2)} \tag{7.4.1}$$

where,

$x_i$ = normalized throughput (in Kbps) of the ith TCP flow

$n$ = Number of connections

xi is the ratio between Actual throughput and Optimal throughput and is calculated as .

$$x_i = \frac{t_i}{o_i} \tag{7.4.2}$$

where,

$x_i$ = Normalized throughput (in Kbps) of the ith TCP flow

$t_i$ = Actual throughput

$o_i$ = Optimal throughput

we can present Fairness index equivalent to [79] :

$$F(X) = \frac{1}{1 + cv^2} \tag{7.4.3}$$

where,

$\bar{x}^2$ = Square of the mean

$\overline{x^2}$ = Variance

$cv$ = Coefficient of variation

Coefficient of variation (CV) is defined as the ratio of standard deviation to the mean and measures variability with respect to the mean of the population [80]. The range of fairness index varies between 0 and 1 that is $0 \leq F(X) \leq 1$. Jain's Fairness index is one of the widely studied fairness measures and can be used generally for fairness study in various fields. The ideal value of Fairness Index $F(X)$ is 1, if resources are fairly allocated among all the users.

The Fairness Index uses the assumption that each user deserves its share with respect to its demand criteria. For example, a sports person requires 2500 calories a day and a normal person requires calories of 1500. Suppose, one day meal having 2000 calories is to be distributed between these two people based on their body requirement. The fairness index of 1 will be achieved from Equation 7.4.1 (and Equation 7.4.3), if 1250 and 750 is the allocated calories to each person respectively from 2000 calories, and their respective normalized throughput comes out be $(1250/2500 = 0.5)$ and $(750/1500 = 0.5)$, i.e., the distribution is fair based on benchmark criteria of their required calories. Fairness index has many properties: Fairness index is scale independent i.e., it does not matter which unit of measurement is used, it is continuous in nature, it has direct relationship (higher the index value, fairer is the distribution) [81].

**How used in Local Market trading:**

The social welfare has been understood here as the overall user utility (measure of satisfaction like revenue earned by prosumers or savings achieved by all households etc.) received from consuming the service provided by the system after deducting expenditures [82]. As the requirement from models is to extract maximum DERs usage and local exchange for meeting the user demand, the Fairness Index can be measured here in terms of usage and local allocation for each user. The actual throughput/allocation for Consumer will be the amount of energy purchased from local market or used from DERs. The Optimal Throughput or allocation will be the fulfillment of entire demand of the users by local trading or DERs usage. This means that a household having demand of 33 kWh will have his optimum throughput as 33 kWh ($o_i$), but if its allocated only 11 kWh from the local market or DERs usage, the Actual Throughput will be 11 kWh ($t_i$) and normalized throughput ($x_i$) will be $11/33 = 0.33$. Additional advantage expected from Fairness index is that, it can be used in checking which household is misusing the trading schema and it can be penalized for having a higher demand in trading pool. Further, with community households cooperating with each other, demand can be managed and balanced by each households to bring the fairness index to 1, in such a way that every household is able to obtain local electricity without sacrificing their minimum needs. The Fairness Index can be understood more clearly from the results stated for this metric in Calculations and Results in Chapter 8.

# CHAPTER 8: CALCULATIONS AND RESULTS

## 8.1 Scenario Case-I

### 8.1.1 Fixed Demand-Variable Pricing

#### 8.1.1.1 Local Pricing Calculation

The pricing formula stated in section 6.2 (Equation 6.2.1) was utilized to find the local pricing. The local price $p_{loc}$ obtained for 48 hours scenario ranged between 19 cents to 30.5 cents. It can be observed from Figure 8.1 (Pricing and Normalized Demand for 48 hours), that pricing is the function of normalized demand and never exceeds the grid price and never goes below the feed in tariff. The pricing strategy can be well suited for households to satisfy their utility functions, which is to increase their respective savings through revenues from local sales and local purchases [83].

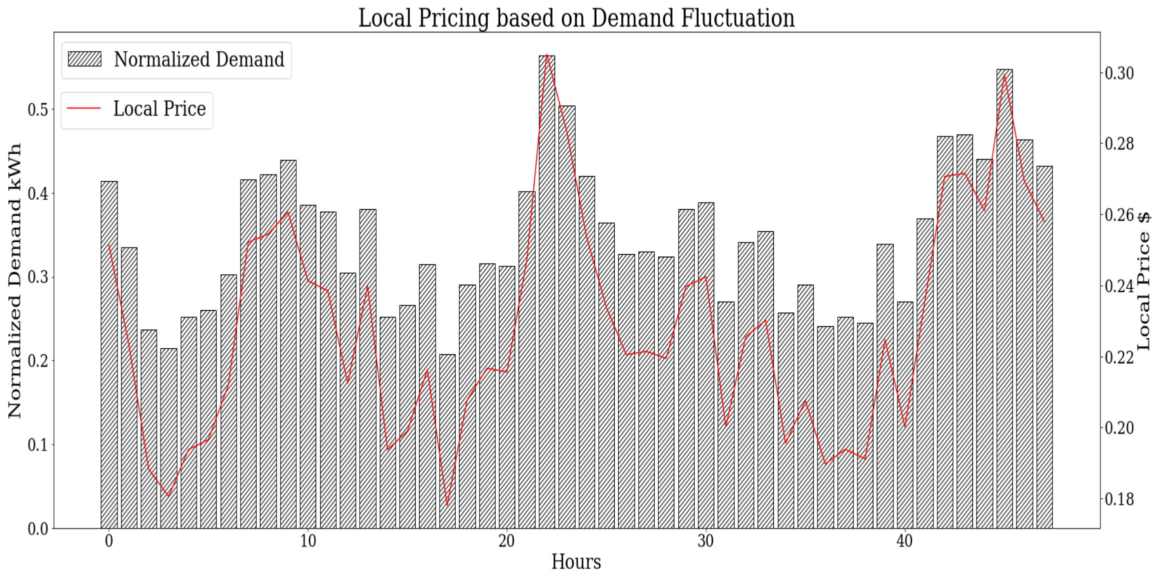

Figure 8.1: Pricing and Normalized Demand for 48 hours

The community totals and savings are calculated from Table 8.1 (Community Trading Totals) and Table 8.2 (Community Closing Accounts):

Table 8.1: Community Trading Totals

| Grid buy total (kWh) | Local Buy total (kWh) | Grid sell total (kWh) | Local sell total (kWh) | Use PV Total (kWh) | Use Battery Total (kWh) |
|---|---|---|---|---|---|
| 368.70 | 23.75 | 122.04 | 23.75 | 93.02 | 62.82 |

Table 8.2: Community Closing Accounts

| Total Purchase costs ($) | Total Sales revenue ($) | Net Local($) | Conventional Bill ($) | Savings ($) | %Savings |
|---|---|---|---|---|---|
| 175.35 | 18.29 | 157.06 | 223.89 | 66.83 | 30% |

The individual allocation stated in Table 8.3 (Individual Allocation) below is used further to calculate the fairness in distribution of DERs to the households and their respective savings. It is important to assess how well the dispatch is allocating resources and what amount of savings is achieved at user level. The decision variables include the amount of electricity and battery charge bought locally and from the grid, battery and PV sales to the grid and local market, and share of PV and battery used by respective prosumers. The expenses and revenue generated from the trading and savings obtained is calculated with respect to the conventional bill using grid price.

Table 8.3: Individual Allocation

| **Allocation Variables/User** | **C1** | **C2** | **C3** | **C4** | **C5** | **C6** | **C7** | **C8** |
|---|---|---|---|---|---|---|---|---|
| **Total Demand (kWh) 48 hrs** | 33.45 | 70.63 | 89.40 | 115.57 | 34.17 | 36.22 | 29.50 | 77.36 |
| **Total PV Generation (kWh)** | 22.76 | 19.31 | 45.62 | 31.52 | 42.65 | 25.76 | 0.00 | 0.00 |
| **Buy from grid (kWh)** | 20.91 | 51.40 | 49.06 | 58.11 | 22.43 | 21.97 | 26.05 | 68.37 |
| **Buy locally (kWh)** | 0.69 | 2.19 | 1.67 | 2.78 | 0.00 | 1.02 | 3.45 | 8.99 |
| **Buy Charging Locally(kWh)** | 0.00 | 0.00 | 0.57 | 2.40 | 0.00 | 0.00 | 0.00 | 0.00 |
| **Buy charging from Grid (kWh)** | 0.00 | 0.00 | 15.86 | 13.99 | 5.74 | 14.81 | 0.00 | 0.00 |
| **PV sold Locally(kWh)** | 0.00 | 0.00 | 1.22 | 0.19 | 2.07 | 0.82 | 0.00 | 0.00 |
| **PV sold to Grid (kWh)** | 10.92 | 2.26 | 20.95 | 5.92 | 34.08 | 16.18 | 0.00 | 0.00 |
| **Use PV (kWh)** | 11.84 | 17.05 | 21.88 | 21.80 | 6.26 | 5.56 | 0.00 | 0.00 |
| **Use Battery (kWh)** | 0.00 | 0.00 | 16.79 | 32.88 | 5.49 | 7.66 | 0.00 | 0.00 |
| **Use PV Charging(kWh)** | 0.00 | 0.00 | 1.57 | 3.61 | 0.26 | 3.20 | 0.00 | 0.00 |
| **Sell Battery Locally (kWh)** | 0.00 | 0.00 | 1.80 | 0.39 | 3.53 | 13.73 | 0.00 | 0.00 |
| **Sell Battery to Grid (kWh)** | 0.00 | 0.00 | 9.42 | 0.73 | 8.98 | 12.61 | 0.00 | 0.00 |
| **Grid Buy cost $** | 9.63 | 23.66 | 29.89 | 33.20 | 12.97 | 16.93 | 11.99 | 31.48 |
| **Local buy cost $** | 0.15 | 0.52 | 0.47 | 1.07 | 0.00 | 0.26 | 0.85 | 2.28 |
| **Grid sales revenue $** | 1.14 | 0.24 | 3.16 | 0.69 | 4.48 | 2.99 | 0.00 | 0.00 |
| **Local salesl revenue $** | 0.00 | 0.00 | 3.16 | 0.69 | 1.22 | 3.55 | 0.00 | 0.00 |
| **Net Purchase cost $** | 8.64 | 23.95 | 24.04 | 32.88 | 7.27 | 10.65 | 12.85 | 33.76 |
| **Conventional Bill $** | 15.40 | 32.52 | 41.16 | 53.21 | 15.73 | 16.67 | 13.58 | 35.62 |
| **Net savings $** | 6.76 | 8.57 | 17.12 | 20.33 | 8.46 | 6.03 | 0.73 | 1.85 |
| **%Savings** | 44% | 26% | 42% | 38% | 54% | 36% | 5% | 5% |

#### 8.1.1.2 Measurement Indices

The model in this case provides Self Sufficiency (SS) = 34.55%., which means, local transactions and DERs usage meets this percentage of the total demand. This indicates that, a major portion of the dependency still prevails on the grid. For calculation of self sufficiency index, PV charging has not been taken into consideration in this case as it has no role in meeting the demand of the user.

The self consumption (SC) = 59.54% for the community which means only 59.54% of the total DERs was used for the local trading.

Normalized throughput was calculated in Table 8.4 (Fairness Index Throughput) by Equation 7.4.2 of section 7.4 :

Table 8.4: Fairness Index Throughput

| **User** | **Self Use/Local Buy or $t_i$** | **Demand or $o_i$** | $x_i$ | $x_i^2$ |
|---|---|---|---|---|
| C1 | 12.53 | 33.45 | 0.37 | 0.14 |
| C2 | 19.23 | 70.63 | 0.27 | 0.07 |
| C3 | 40.34 | 89.40 | 0.45 | 0.20 |
| C4 | 57.46 | 115.57 | 0.50 | 0.25 |
| C5 | 11.75 | 34.17 | 0.34 | 0.12 |
| C6 | 14.24 | 36.22 | 0.39 | 0.15 |
| C7 | 3.45 | 29.50 | 0.12 | 0.01 |
| C8 | 8.99 | 77.36 | 0.12 | 0.01 |
| Sum | | | 2.57 | 0.97 |

Using the last two columns and Equation 7.4.1 we get , $F(X) = 0.852$. This can be verified with mean and standard deviation for the $x_i$ and substituting it in Equation 7.4.3 to get same results.

### 8.1.2 Fixed Demand-Variable Pricing (Only PV charging)

Local Price range remains same as in section 8.1.1.1 that is, between 19 cents to 30.5 cents. Battery charging from grid and local purchases is removed. Battery is charge from surplus PV only. This led to reduction in purchases from grid and local market, and improved purchase costs of the users, but the overall supply of DERs reduced considerably due to absence of alternate battery charging means. Also, the measurement index was reduced due to overall reduction in DERs availability as battery was not able to get charged and participate in local market. Community Results is summarized through Table 8.5 (Community Trading Results) and Table 8.6 (Community Closing Accounts). Individual Allocation is summarized as consumption, expenditures, revenue and savings of each household by Table 8.7 (Individual Allocation).

Table 8.5: Community Trading Results

| Grid Buy total (kWh) | Local Buy total (kWh) | Grid sales total(kWh) | Local sales total (kWh) | Use PV Total (kWh) | Use Battery Total (kWh) |
|---|---|---|---|---|---|
| 351.23 | 12.63 | 112.10 | 12.63 | 96.40 | 32.50 |

Table 8.6: Community Closing Accounts

| Total Purchase costs ($) | Total sales revenue ($) | Net Local($) | Conventional Bill ($) | Savings($) | %Savings |
|---|---|---|---|---|---|
| 164.45 | 14.40 | 150.05 | 223.89 | 73.84 | 33% |

Table 8.7: Individual Allocation

| **Allocation Variables/User** | **C1** | **C2** | **C3** | **C4** | **C5** | **C6** | **C7** | **C8** |
|---|---|---|---|---|---|---|---|---|
| **Demand** | 33.45 | 70.63 | 89.40 | 115.57 | 34.17 | 36.22 | 29.50 | 77.36 |
| **PV** | 22.76 | 19.31 | 45.62 | 31.52 | 42.65 | 25.76 | 0.00 | 0.00 |
| **Buy from Grid (kWh)** | 21.61 | 53.54 | 59.51 | 71.71 | 23.84 | 21.67 | 28.18 | 71.19 |
| **Buy Locally (kWh)** | 0.00 | 0.05 | 1.92 | 2.84 | 0.00 | 0.33 | 1.32 | 6.17 |
| **PV Sold Locally (kWh)** | 1.16 | 0.60 | 2.38 | 0.00 | 3.34 | 3.34 | 0.00 | 0.00 |
| **PV sold to Grid (kWh)** | 9.77 | 1.66 | 20.28 | 7.72 | 32.32 | 8.68 | 0.00 | 0.00 |
| **Use PV (kWh)** | 11.84 | 17.05 | 21.96 | 23.31 | 6.88 | 8.89 | 0.00 | 0.00 |
| **Use Battery (kWh)** | 0.00 | 0.00 | 6.01 | 17.71 | 3.45 | 5.33 | 0.00 | 0.00 |
| **Use PV Charging (kWh)** | 0.00 | 0.00 | 1.00 | 0.50 | 0.12 | 4.85 | 0.00 | 0.00 |
| **Sell Battery Locally (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 0.69 | 1.12 | 0.00 | 0.00 |
| **Sell Battery to Grid (kWh)** | 0.00 | 0.00 | 9.99 | 0.29 | 7.86 | 13.55 | 0.00 | 0.00 |
| **Grid Buy cost($)** | 9.95 | 24.65 | 27.40 | 33.02 | 10.98 | 9.97 | 12.97 | 32.78 |
| **Local Buy Cost ($)** | 0.00 | 0.01 | 0.43 | 0.63 | 0.00 | 0.08 | 0.31 | 1.28 |
| **Grid Sales Revenue ($)** | 1.02 | 0.17 | 3.15 | 0.83 | 4.18 | 2.31 | 0.00 | 0.00 |
| **Local Sales Revenue($)** | 0.27 | 0.14 | 0.53 | 0.00 | 0.86 | 0.94 | 0.00 | 0.00 |
| **Net Purchase cost ($)** | 8.66 | 24.34 | 24.15 | 32.81 | 5.94 | 6.80 | 13.29 | 34.06 |
| **Conventional Bill ($)** | 15.40 | 32.52 | 41.16 | 53.21 | 15.73 | 16.67 | 13.58 | 35.62 |
| **Net savings ($)** | 6.74 | 8.18 | 17.01 | 20.40 | 9.79 | 9.87 | 0.29 | 1.56 |
| **%Savings** | 44% | 25% | 41% | 38% | 62% | 59% | 2% | 4% |

#### 8.1.2.1 Measurement Indices

The Self Sufficiency (SS) was noted to be 27.77% and Self Consumption Index (SC) was 55.80%. The Fairness index ($F(X)$) was calculated to be 0.815 which was slightly lower than the previous transactions with grid and local trading purchases (Table 8.8, Fairness Index Throughput). As observed the cumulative DERs penetration was reduced due to new trading rules.

Table 8.8: Fairness Index Throughput

| **User** | **Self Use/Local Buy or** $t_i$ | **Demand or** $o_i$ | $x_i$ | $x_i^2$ |
|---|---|---|---|---|
| C1 | 11.84 | 33.45 | 0.35 | 0.13 |
| C2 | 17.10 | 70.63 | 0.24 | 0.06 |
| C3 | 29.89 | 89.40 | 0.33 | 0.11 |
| C4 | 43.86 | 115.57 | 0.38 | 0.14 |
| C5 | 10.33 | 34.17 | 0.30 | 0.09 |
| C6 | 14.55 | 36.22 | 0.40 | 0.16 |
| C7 | 1.32 | 29.50 | 0.04 | 0.00 |
| C8 | 6.17 | 77.36 | 0.08 | 0.01 |
| Sum | | | 2.14 | 0.70 |

### 8.1.3 Adjusted Demand-Minimum Local Price

#### 8.1.3.1 Adjusted Demand and Local Price Calculation

The adjusted demand is obtained from Equation 6.5.1 in section 6.5.1 and total adjusted demand $e_{dnew,i,t}$ was calculated as 488.01 kWh (Table 8.9, Community Totals) with local price $p_{loc}$ ranging from 14.8 cents to 21 cents for the 48 hour span (Fig.8.2, Pricing and Normalized Demand for 48 hours). The total adjusted demand increased as the optimization of local pricing and hence, demand as per available supply pool increased the demand of some users for few time periods. High spikes in demand still persisted for some time periods. However, minimum local price achieved was lower than the price results obtained in section 8.1.1.1.

Table 8.9: Community Totals

| Hours | Total demand (kWh) | Total PV (kWh) |
|---|---|---|
| 48 | 488.01 | 187.63 |

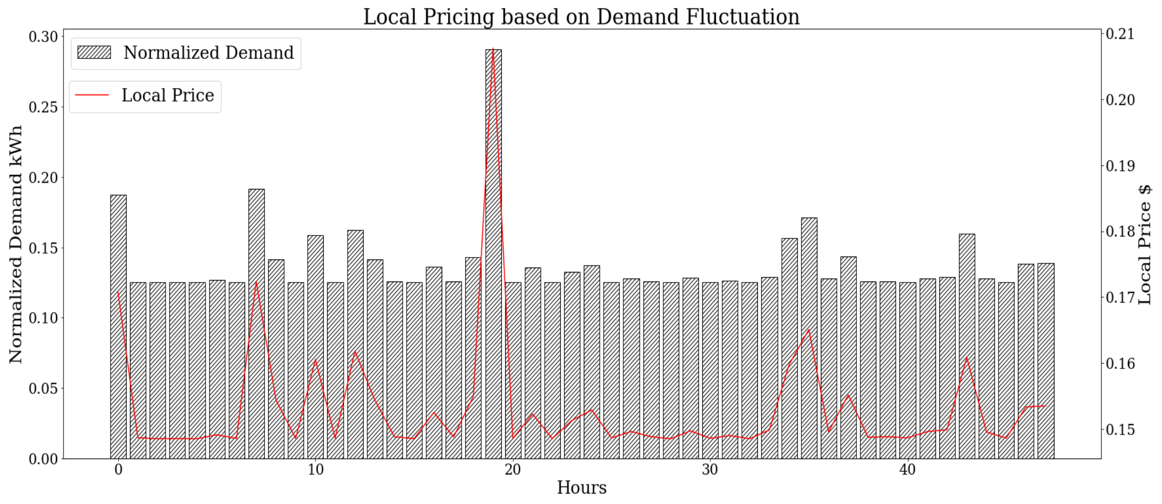

Figure 8.2: Pricing and Normalized Demand for 48 hours

The individual Household allocations and percentage savings are deduced in Table 8.10 (Individual Allocation) and Community results are summarized in Table 8.11 (Community Trading Totals) and Table 8.12 (Community Closing Accounts) .

Table 8.10: Individual Allocation

| **Allocation Variables/User** | **C1** | **C2** | **C3** | **C4** | **C5** | **C6** | **C7** | **C8** |
|---|---|---|---|---|---|---|---|---|
| **Demand** | 38.14 | 44.55 | 82.09 | 117.75 | 39.62 | 45.27 | 37.86 | 82.75 |
| **PV** | 22.76 | 19.31 | 45.62 | 31.52 | 42.65 | 25.76 | 0.00 | 0.00 |
| **Buy from Grid (kWh)** | 26.99 | 31.94 | 56.60 | 71.98 | 23.81 | 27.80 | 33.52 | 76.36 |
| **Buy Locally (kWh)** | 0.00 | 0.22 | 1.06 | 5.50 | 0.00 | 0.18 | 4.34 | 6.39 |
| **Buy Charging Locally (kWh)** | 0.00 | 0.00 | 0.00 | 1.54 | 0.00 | 0.00 | 0.00 | 0.00 |
| **Buy Chargingfrom Grid (kWh)** | 0.00 | 0.00 | 15.54 | 19.15 | 3.00 | 3.45 | 0.00 | 0.00 |
| **PV Sold Locally (kWh)** | 0.00 | 0.00 | 2.70 | 0.00 | 5.62 | 0.18 | 0.00 | 0.00 |
| **PV sold to Grid (kWh)** | 11.61 | 6.92 | 27.06 | 6.80 | 29.49 | 12.88 | 0.00 | 0.00 |
| **Use PV (kWh)** | 11.15 | 12.39 | 13.40 | 23.41 | 7.54 | 10.15 | 0.00 | 0.00 |
| **Use Battery (kWh)** | 0.00 | 0.00 | 11.03 | 16.86 | 8.28 | 7.14 | 0.00 | 0.00 |
| **Use PV Charging (kWh)** | 0.00 | 0.00 | 2.46 | 1.31 | 0.00 | 2.55 | 0.00 | 0.00 |
| **Sell Battery Locally (kWh)** | 0.00 | 0.00 | 5.68 | 2.53 | 0.76 | 1.74 | 0.00 | 0.00 |
| **Sell Battery to Grid (kWh)** | 0.00 | 0.00 | 15.29 | 10.61 | 5.96 | 13.12 | 0.00 | 0.00 |
| **Grid Buy cost($)** | 12.43 | 14.70 | 33.21 | 41.95 | 12.34 | 14.39 | 15.43 | 35.15 |
| **Local Buy Cost ($)** | 0.00 | 0.03 | 0.16 | 1.05 | 0.00 | 0.03 | 0.67 | 0.98 |
| **Grid Sales Revenue ($)** | 1.21 | 0.72 | 4.40 | 1.81 | 3.69 | 2.70 | 0.00 | 0.00 |
| **Local Sales Revenue($)** | 0.00 | 0.00 | 1.26 | 0.38 | 0.99 | 0.29 | 0.00 | 0.00 |
| **Net Purchase cost ($)** | 11.22 | 14.02 | 27.70 | 40.82 | 7.66 | 11.43 | 16.10 | 36.13 |
| **Conventional Bill ($)** | 17.56 | 20.51 | 37.79 | 54.21 | 18.24 | 20.84 | 17.43 | 38.10 |
| **Net savings ($)** | 6.34 | 6.49 | 10.09 | 13.39 | 10.58 | 9.42 | 1.33 | 1.97 |
| **%Savings** | 36% | 32% | 27% | 25% | 58% | 45% | 8% | 5% |

Table 8.11: Community Trading Totals

| Grid buy total (kWh) | Local Buy total (kWh) | Grid sell total (kWh) | Local sell total (kWh) | Use PV Total (kWh) | Use Battery Total (kWh) |
|---|---|---|---|---|---|
| 390.12 | 19.22 | 139.74 | 19.22 | 84.36 | 43.31 |

Table 8.12: Community Closing Accounts

| Total Purchase costs ($) | Total sales revenue ($) | Net Local ($) | Conventional Bill ($) | Savings ($) | %Savings |
|---|---|---|---|---|---|
| 182.52 | 17.45 | 165.08 | 224.68 | 59.60 | 27% |

#### 8.1.3.2 Measurement Indices

The measurement indices are stated to be: Self Sufficiency(SS) as 28.5%, Self Consumption (SC) as 51.2% and Fairness Index ($F(X)$) to be 0.856 (Table 8.13, Fairness Index Throughput)

Table 8.13: Fairness Index Throughput

| **User** | **Self Use/Local Buy or** $t_i$ | **Demand or** $o_i$ | $x_i$ | $x_i^2$ |
|---|---|---|---|---|
| C1 | 11.15 | 38.14 | 0.29 | 0.09 |
| C2 | 12.61 | 44.55 | 0.28 | 0.08 |
| C3 | 25.49 | 82.09 | 0.31 | 0.10 |
| C4 | 45.77 | 117.75 | 0.39 | 0.15 |
| C5 | 15.81 | 39.62 | 0.40 | 0.16 |
| C6 | 17.47 | 45.27 | 0.39 | 0.15 |
| C7 | 4.34 | 37.86 | 0.11 | 0.01 |
| C8 | 6.39 | 82.75 | 0.08 | 0.01 |
| Sum | | | 2.25 | 0.74 |

### 8.1.4 Adjusted Demand-Minimum Local Price (Only PV Charging)

The pricing and adjusted demand formulation remained same from the section 8.1.3. That is, the total adjusted demand $e_{dnew,i,t}$ is 488.01 kWh (Table 8.9, Community Totals) with local price $p_{loc}$ between 14.8 cents to 21 cents (Fig.8.2, Pricing and Normalized Demand for 48 hours). The household results are summarized below in Table 8.14 (Individual Allocation).

Table 8.14: Individual Allocation

| **Allocation Variables/User** | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 |
|---|---|---|---|---|---|---|---|---|
| **Demand** | 38.14 | 44.55 | 82.09 | 117.75 | 39.62 | 45.27 | 37.86 | 82.75 |
| **Buy from Grid (kWh)** | 26.19 | 32.16 | 58.91 | 75.81 | 24.84 | 29.60 | 35.26 | 79.57 |
| **Buy Locally (kWh)** | 0.80 | 0.00 | 2.54 | 5.45 | 0.03 | 0.00 | 2.59 | 3.17 |
| **PV Sold Locally (kWh)** | 0.96 | 0.88 | 2.45 | 0.00 | 3.57 | 0.00 | 0.00 | 0.00 |
| **PV sold to Grid (kWh)** | 10.65 | 6.04 | 24.30 | 6.89 | 30.92 | 8.60 | 0.00 | 0.00 |
| **Use PV (kWh)** | 11.15 | 12.39 | 14.15 | 22.07 | 8.16 | 11.03 | 0.00 | 0.00 |
| **Use Battery (kWh)** | 0.00 | 0.00 | 6.49 | 14.42 | 6.59 | 4.64 | 0.00 | 0.00 |
| **Use PV Charging (kWh)** | 0.00 | 0.00 | 4.73 | 2.56 | 0.00 | 6.13 | 0.00 | 0.00 |
| **Sell Battery Locally (kWh)** | 0.00 | 0.00 | 0.00 | 1.47 | 3.84 | 1.40 | 0.00 | 0.00 |
| **Sell Battery to Grid (kWh)** | 0.00 | 0.00 | 13.51 | 4.11 | 1.57 | 15.96 | 0.00 | 0.00 |
| **Grid Buy cost($)** | 12.06 | 14.81 | 27.12 | 34.90 | 11.44 | 13.63 | 16.24 | 36.64 |
| **Local Buy Cost ($)** | 0.12 | 0.00 | 0.38 | 0.81 | 0.00 | 0.00 | 0.40 | 0.49 |
| **Grid Sales Revenue ($)** | 1.11 | 0.63 | 3.93 | 1.14 | 3.38 | 2.55 | 0.00 | 0.00 |
| Local Sales Revenue($) | 0.14 | 0.13 | 0.38 | 0.22 | 1.12 | 0.22 | 0.00 | 0.00 |
| **Net Purchase cost ($)** | 10.93 | 14.05 | 23.19 | 34.35 | 6.94 | 10.86 | 16.64 | 37.13 |
| **Conventional Bill ($)** | 17.56 | 20.51 | 37.79 | 54.21 | 18.24 | 20.84 | 17.43 | 38.10 |
| **Net savings ($)** | 6.63 | 6.46 | 14.60 | 19.86 | 11.30 | 9.98 | 0.79 | 0.97 |
| **%Savings** | 38% | 32% | 39% | 37% | 62% | 48% | 5% | 3% |

Table 8.15 (Community Trading Totals) and Table 8.16 (Community Closing Accounts)

summarize the community performance.

Table 8.15: Community Trading Totals

| Grid buy total (kWh) | Local Buy total (kWh) | Grid sell total (kWh) | Local sell total (kWh) | Use PV Total (kWh) | Use Battery Total (kWh) |
|---|---|---|---|---|---|
| 362.35 | 14.58 | 122.55 | 14.58 | 92.36 | 32.14 |

Table 8.16: Community Closing Accounts

| Total Purchase costs ($) | Total sales revenue ($) | Net Local($) | Conventional Bill ($) | Savings ($) | %Savings |
|---|---|---|---|---|---|
| 169.04 | 14.96 | 154.08 | 224.68 | 70.60 | 31% |

#### 8.1.4.1 Measurement Indices

Overall Community Self Sufficiency (SS) was noted to be 25.8% and Self consumption(SC) increased to 53.2% indicating increase in utilization higher portion of DERs through demand adjustment. Fairness Index ($F(X)$) did not show considerable improvement and was noted to be 0.816 (Table 8.17, Fairness Index Throughput).

Table 8.17: Fairness Index Throughput

| **User** | **Self Use/Local Buy or** $t_i$ | **Demand or** $o_i$ | $x_i$ | $x_i^2$ |
|---|---|---|---|---|
| C1 | 11.95 | 38.14 | 0.31 | 0.10 |
| C2 | 12.39 | 44.55 | 0.28 | 0.08 |
| C3 | 23.18 | 82.09 | 0.28 | 0.08 |
| C4 | 41.94 | 117.75 | 0.36 | 0.13 |
| C5 | 14.78 | 39.62 | 0.37 | 0.14 |
| C6 | 15.67 | 45.27 | 0.35 | 0.12 |
| C7 | 2.59 | 37.86 | 0.07 | 0.00 |
| C8 | 3.17 | 82.75 | 0.04 | 0.00 |
| Sum | | | 2.06 | 0.65 |

### 8.1.5 Vickrey Clarke Groves Auction Model

The Prosumer with surplus Storage or PV become suppliers after meeting their own demand with their respective generators if available. Thus, in the first step Prosumers meet their energy demand and check for surplus from their production. The Net self-usage and surplus of Prosumers that was calculated for 48 hours trading is summarized in Table 8.18 (Total Prosumer Self Usage and Net Demand).

Table 8.18: Total Prosumer Self Usage and Net Demand

| Prosumer | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 |
|---|---|---|---|---|---|---|---|---|
| Prosumer demand (kWh) | 33.4 | 70.6 | 89.4 | 115.6 | 34.2 | 36.2 | 29.5 | 77.4 |
| PV (kWh) | 22.8 | 19.3 | 45.6 | 31.5 | 42.7 | 25.8 | 0.0 | 0.0 |
| Net Demand (kWh) | 21.6 | 53.6 | 54.5 | 67.2 | 17.3 | 19.0 | 29.5 | 77.4 |
| Surplus PV (JkWh) | 10.9 | 2.3 | 19.7 | 0.0 | 33.6 | 13.0 | 0.0 | 0.0 |
| Use PV (kWh) | 11.8 | 17.0 | 23.3 | 28.8 | 8.1 | 8.9 | 0.0 | 0.0 |
| Use PV Charging (kWh) | 0.0 | 0.0 | 2.6 | 2.8 | 1.0 | 3.8 | 0.0 | 0.0 |
| Use Battery (kWh) | 0.0 | 0.0 | 11.6 | 19.6 | 8.8 | 8.4 | 0.0 | 0.0 |
| Battery Surplus (kWh) | 0.0 | 0.0 | 12.4 | 4.4 | 6.2 | 17.6 | 0.0 | 0.0 |

Community trading results are something we are looking to improve as well. The total demand and PV generation with trading totals and expenses is stated below. Trading totals in Table 8.19 (Community Usage and Trading Totals) and 8.20 (Community Closing Accounts) add up all the trading results for community.

Table 8.19: Community Usage and Trading Totals

| Demand kWh | PV kWh | Net Demand kWh | Use PV kWh | Use PV Charging kWh | Use Battery kWh | Local Sales kWh | Grid Sales kWh | Grid Buy kWh |
|---|---|---|---|---|---|---|---|---|
| 486.29 | 187.63 | 340.07 | 97.93 | 10.24 | 48.29 | 56.11 | 64.07 | 283.70 |

Table 8.20: Community Closing Accounts

| Conventional Cost ($) | Local Buy Cost ($) | Grid Buy Cost ($) | Grid Sales Revenue ($) | Local Sales Revenue ($) | Savings |
|---|---|---|---|---|---|
| 223.89 | 15.35 | 130.70 | 6.62 | 15.35 | 44% |

The Seller's Transaction includes all the sales to local market and Grid. The Buyer's Transaction involves total purchases made from the local market and the grid . Cumulative individual share of agents for 48 hours in the model is summarized in tables below. Table 8.21 (Prosumers Sales) gives total energy sold buy each prosumers to local market and grid. And Table 8.22 (Buyer Purchases) shows total buying transactions of all the users from the local market and grid. Individual Savings for each household is stated in Table 8.23 (Household Closing Accounts).

Table 8.21: Prosumer Sales

| Prosumer | C1 | C2 | C3 | C4 | C5 | C6 | Total kWh |
|---|---|---|---|---|---|---|---|
| Energy Sold kWh (Local +Grid) | 10.92 | 2.26 | 32.09 | 4.44 | 39.79 | 30.67 | 120.18 |

Table 8.22: Buyer Purchases

| Buyer | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | Total kWh |
|---|---|---|---|---|---|---|---|---|---|
| Energy Purchased kWh | 21.61 | 53.59 | 54.50 | 67.25 | 17.31 | 18.87 | 29.50 | 77.36 | 339.97 |

Table 8.23: Household Closing Accounts

| | **Total Grid Buy cost ($)** | **Total Local Buy cost ($)** | **Grid Sales Revenue($)** | **Local Sales Revenue($)** | **Net Local Cost ($)** | **Conventional Bill ($)** | **Savings ($)** | **%Savings** |
|---|---|---|---|---|---|---|---|---|
| C1 | 9.03 | 0.31 | 0.27 | 2.67 | 6.47 | 15.40 | 8.93 | 58% |
| C2 | 19.29 | 3.19 | 0.02 | 0.68 | 21.94 | 32.52 | 10.58 | 33% |
| C3 | 23.50 | 0.78 | 2.08 | 2.66 | 19.73 | 41.16 | 21.43 | 53% |
| C4 | 27.20 | 2.30 | 0.27 | 0.46 | 29.06 | 53.21 | 24.15 | 46% |
| C5 | 7.18 | 0.35 | 2.41 | 4.99 | 0.19 | 15.73 | 15.54 | 99% |
| C6 | 7.55 | 0.69 | 1.61 | 3.89 | 2.81 | 16.67 | 13.87 | 84% |
| C7 | 10.23 | 1.53 | 0.00 | 0 | 11.85 | 13.58 | 1.73 | 13% |
| C8 | 26.65 | 6.186 | 0.00 | 0 | 33.05 | 35.62 | 2.57 | 8% |

#### 8.1.5.1 Measurement Indices

Notably higher self sufficiency and self consumption was achieved with VCG trading as compared to previous MILP models. This makes sequential VCG model more simple and robust. Self Sufficiency (SS) was calculated to be 41.6% and Self-Consumption (SC) reached was 76.80%. The transaction works until entire demand in the pool is met or supply is finished, and any interaction with grid is initiated only after this local transaction is complete. The model ensured full utilization of PV/Battery in local market with grid purchases reducing considerably.

Fairness Index ($F(X)$) was calculated to be 0.936 which is close to the ideal value of 1. This means that the actual allocation to all the users was fair with respect to their optimal allocations (Table 8.24, Fairness Index Throughput).

Table 8.24: Fairness Index Throughput

Fairness Index Throughput

| **User** | **Self Use/Local Buy or** $t_i$ | **Demand or** $o_i$ | $x_i$ | $x_i^2$ |
|---|---|---|---|---|
| C1 | 13.82 | 33.45 | 0.41 | 0.17 |
| C2 | 28.72 | 70.63 | 0.41 | 0.17 |
| C3 | 38.35 | 89.40 | 0.43 | 0.18 |
| C4 | 56.34 | 115.57 | 0.49 | 0.24 |
| C5 | 18.56 | 34.17 | 0.54 | 0.29 |
| C6 | 19.81 | 36.22 | 0.55 | 0.30 |
| C7 | 7.26 | 29.50 | 0.25 | 0.06 |
| C8 | 19.46 | 77.36 | 0.25 | 0.06 |
| Sum | | | 3.32 | 1.48 |

### 8.1.6 Discussions

PV generation with respect to the demand profile and battery in the pool was noted to be less as seen in demand versus PV generation curve in Figure 4.1 (Cumulative Demand and PV Generation) of section 4.1, thereby indicating the need of bigger PV sizes for better share of PV charging and Local transactions. Table 8.25 (Dispatch Performance Summary) summarizes the outcomes of the dispatch mechanisms. It can be noted from the table that results stated for section 8.1.1 (Fixed Demand Variable Pricing) showed community percentage savings of 30% with maximum individual savings of 54% by Prosumer C5 and minimum savings of 5% (by consumers C7 and C8). Community percentage savings was less than half meant participants were still paying more for purchases. Significant amount of demand was being met by grid reducing self sufficiency to 34.55% which meant less amount of local energy was being traded, and surplus DER was significantly less than the demand in the local market. Self consumption share was lower (59.54%) because ratio of local market usage and total supply was higher and significant amount was sold to grid. This is because the demand was fulfilled by the local trading transactions for some instances and in many instances prosumers decided to sell generation to grid. Fairness Index achieved was moderate indicating inequality in the resource allocation between the households which means some members were well off from trading and some did not get sufficient dispatch. This Fixed Demand Variable Pricing model did not show optimum numbers as the battery charging with grid and local market came out to be more expensive for some time periods and when same charged battery was sold at lower rate in local market, the revenue earned by battery owners could not nullify the expenditure. Hence, cumulative performance numbers in this simulation stayed to lower values.

From first model it was noted that, transaction with grid and local market for charging batteries increased expenditure. The devised pricing strategy with same local prices was again tested with condition allowing use of only PV for charging batteries. The results from section 8.1.2 for Fixed Demand Variable Pricing (Only PV Charging) saw that the battery capacity reduced in the local energy pool as the surplus PV generation was not sufficient to charge batteries, reducing the participation of batteries in the local market and increasing dependency on grid purchases again (SS = 27.77% and SC = 55.8%). Thus, the lower performance numbers can be attributed to lesser supply in the pool during some time periods. Increase in savings by some prosumers (for C5, it rose from 54% to 62%) can be attributed to the decision of prosumer to use DERs themselves, rather than selling them and thus, some consumer lose out the savings (like C7's savings reduced to 2% from 5%). The increased self usage by prosumers rather than selling to grid also lead to increase in community savings (33%).

The model with Adjusted Demand-Minimum Local Price attempted to achieve a lowest local price for community welfare through demand adjustment to obtain lowest local price hoping to increase savings in the pool, but this lead to an increase in the cumulative community demand as adjusted demand for some users increased in particular hour and demand curve continued to show peaks. From results in section 8.1.3 for Adjusted Demand-Minimum Local Price model, lower local price was expected to bring down the expenditures as well, but the model simulation resulted in lowering of local market usage with respect to demand (SS = 28.5%) and supply (SC= 51.2%). The model indeed improved individual savings for some households (consumer C7 savings rose to 8%), however, overall performance was inferior MILP simulations for Fixed Demand Variable Pricing Models of sections 8.1.1 and 8.1.2.

Table 8.25: Dispatch Performance Summary

| Model | Local Pricing Range(cents) | Community Savings% | Maximum Individual Savings% | Minimum Individual Savings% | SS% | SC% | F(X) |
|---|---|---|---|---|---|---|---|
| Fixed Demand Variable Price | 19 to 30.5 | 30% | 54% | 5% | 34.55% | 59.54% | 0.852 |
| Fixed Demand Variable Price (Only PV Charging) | 19 to 30.5 | 33% | 62% | 2% | 27.77% | 55.80% | 0.816 |
| Adjusted Demand Minimum Local Price | 14.8 to 20 | 27% | 58% | 5% | 28.50% | 51.20% | 0.856 |
| Adjusted Demand Minimum Local Price (Only PV Charging) | 14.8 to 20 | 31% | 62% | 3% | 25.80% | 53.20% | 0.816 |
| VCG | Bid based | 44% | 99% | 8% | 41.6% | 76.80% | 0.936 |

The criteria of grid and local market purchases for battery charging was removed for the Adjusted Demand-Minimum Local Price model in section 8.1.4 to reduce additional expenditure. The model was implemented with same set of conditions with adjusted demand and local pricing as calculated in sub-section 8.1.3.1 to see, if the model can show improvements with combination of adjusted demand, a minimum local price, and reduced expenditure for battery charging. The consumers could not make much savings due adjusted demand for households(C7 saved 5% and C7 saved 3% only). The Self sufficiency index showed lower numbers due to reduction in overall supply (SS = 25.8%). Self consumption (SC = 53.2%) improved as more PV was used for charging the batteries due to adjusted demand than selling it to the local market or grid.

The auction model was adopted to tackle the discrepancies of MILP based models that used binary decision variables to set dispatch. Auction model was used as it is a cheaper option and the dispatch can be planned with simple algorithm. VCG model changed the pricing criteria slightly and rather than averaging all user prices, it applied individual prices from the consumers to set up a bidding market. The community savings showed improvement by resulting to 44% as compared to MILP Models. Highest Savings achieved by some prosumers was about 99% when compared with conventional bill, as profit was generated from local sales proving that pricing strategy and dispatch mechanism complemented each other. The model also performed better in terms of Self sufficiency numbers (41.6%) and self consumption numbers (76.80%) indicating DERs were well utilized in the local market with minimal waste. This model ensured that there is minimal interaction with grid and by very few prosumer households, after local transaction is fully finished in a hourly cycle. The savings of Prosumer and consumer improved with highest individual savings, for prosumers close to 99% (C5) and for consumers about 13% (C7). It shows that the dispatch mechanism attempted well to distribute savings and revenue between the households.

Fairness index was best achieved with VCG model close to about 0.936 indicating the distribution of resources was almost fair to all user with respect to their demand. The MILP based models performed moderately terms of fairness index by ranging between 0.816 to 0.856 only. The total supply was observed to be varying in each model, due to the battery charge and discharge process within the iterations. The models with only PV charging created lesser battery charging instances in the iterations, compared to the models where grid and local energy was used for charging.

## 8.2 Scenario Case-II

### 8.2.1 Fixed Demand-Variable Pricing

#### 8.2.1.1 Local Pricing Calculation

The pricing strategy was adopted from section 6.2 (Equation 6.2.1). The local price $p_{loc}$ was obtained to range between 17.6 cents to 28.1 cents (Fig.8.3, Pricing and Normalized Demand for 48 hours), which indicated lower demand patterns than the demand data set in scenario case-I. The community totals are summarized in Table 8.26 (Community Trading Totals) and Table 8.27 (Community Closing Accounts). Household savings are summarized in Table 8.28 (Individual Allocation).

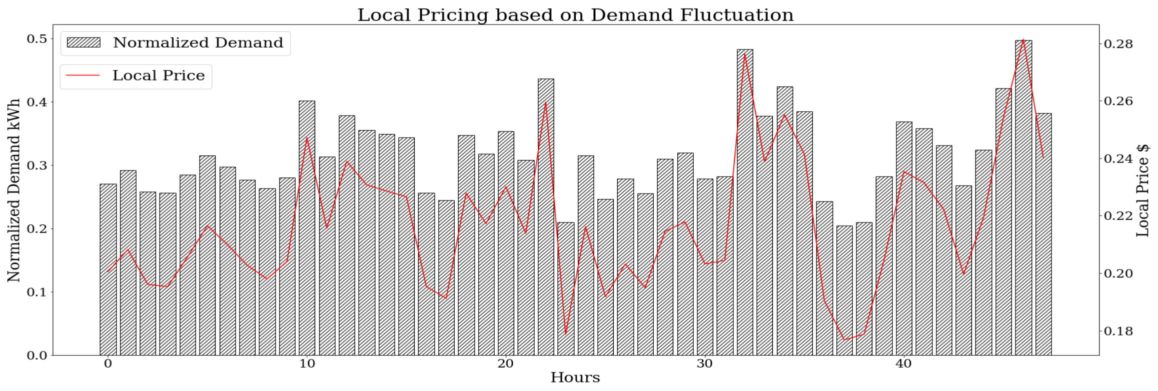

Figure 8.3: Pricing and Normalized Demand for 48 hours

Table 8.26: Community Trading Totals

| Grid Buy total (kWh) | Local Buy total (kWh) | Grid Sales total (kWh) | Local Sales total (kWh) | Use PV Total (kWh) | Use Battery Total (kWh) |
|---|---|---|---|---|---|
| 354.93 | 33.00 | 251.05 | 33.00 | 69.54 | 29.92 |

Table 8.27: Community Closing Accounts

| Total Purchase costs ($) | Total sales revenue ($) | Net Local ($) | Conventional Bill ($) | Savings ($) | %Savings |
|---|---|---|---|---|---|
| 170.02 | 32.72 | 137.30 | 189.45 | 52.14 | 28% |

Table 8.28: Individual Allocation

| **Allocation Variables/User** | H1 | H2 | H3 | H4 | H5 | H6 | H7 | H8 | H9 | H10 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Demand** | 38.53 | 26.71 | 75.17 | 30.93 | 33.23 | 25.12 | 44.38 | 20.69 | 38.59 | 78.13 |
| **Buy from Grid (kWh)** | 25.79 | 18.46 | 49.52 | 21.42 | 14.04 | 11.99 | 28.49 | 12.07 | 35.84 | 67.03 |
| **Buy Locally (kWh)** | 7.62 | 3.17 | 7.25 | 0.62 | 0.00 | 0.00 | 0.00 | 0.00 | 2.75 | 11.11 |
| **Buy Charging Locally (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.48 | 0.00 | 0.00 |
| **Buy Chargingfrom Grid (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 28.21 | 22.86 | 9.58 | 9.64 | 0.00 | 0.00 |
| **PV Sold Locally (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 7.71 | 0.00 | 0.03 | 0.00 | 0.00 |
| **PV sold to Grid (kWh)** | 7.46 | 12.42 | 29.85 | 10.03 | 17.18 | 21.68 | 6.99 | 19.12 | 0.00 | 0.00 |
| **Use PV (kWh)** | 5.12 | 5.08 | 18.40 | 8.89 | 7.82 | 4.04 | 10.04 | 5.02 | 0.00 | 0.00 |
| **Use Battery (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 11.37 | 9.09 | 5.85 | 3.61 | 0.00 | 0.00 |
| **Use PV Charging (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 1.49 | 0.24 | 0.33 | 3.08 | 0.00 | 0.00 |
| **Sell Battery Locally (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 13.96 | 5.84 | 2.54 | 2.92 | 0.00 | 0.00 |
| **Sell Battery to Grid (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 27.47 | 44.47 | 24.61 | 29.77 | 0.00 | 0.00 |
| **Grid Buy cost($)** | 11.88 | 8.50 | 22.80 | 9.86 | 6.46 | 5.52 | 13.12 | 5.78 | 16.50 | 30.86 |
| **Local Buy Cost ($)** | 1.50 | 0.65 | 1.54 | 0.12 | 0.00 | 0.00 | 0.00 | 0.09 | 0.53 | 2.18 |
| **Grid Sales Revenue ($)** | 0.78 | 1.29 | 3.10 | 1.04 | 4.64 | 6.88 | 3.29 | 5.08 | 0.00 | 0.00 |
| **Local Sales Revenue($)** | 0.00 | 0.00 | 0.00 | 0.00 | 2.87 | 2.56 | 0.49 | 0.69 | 0.00 | 0.00 |
| **Net Purchase cost ($)** | 12.60 | 7.86 | 21.23 | 8.94 | 0.00 | 0.00 | 9.34 | 0.09 | 17.03 | 33.04 |
| **Net Profit ($)** | 0.00 | 0.00 | 0.00 | 0.00 | 1.05 | 3.92 | 0.00 | 0.00 | 0.00 | 0.00 |
| **Conventional Bill ($)** | 17.74 | 12.30 | 34.61 | 14.24 | 15.30 | 11.56 | 20.43 | 9.53 | 17.77 | 35.97 |
| **Net savings ($)** | 5.14 | 4.44 | 13.38 | 5.31 | 15.30 | 11.56 | 11.09 | 9.43 | 0.73 | 2.94 |
| **%Savings** | 29% | 36% | 39% | 37% | 100% | 100% | 54% | 99% | 4% | 8% |

#### 8.2.1.2 Measurement Indices

Self Sufficiency was calculated as SS = 30.82% ans self consumption as SC = 34.54%. Fairness Index was calculated as $F(X) = 0.844$ (Table 8.29, Fairness Index Throughput).

Table 8.29: Fairness Index Throughput

| **User** | **Self Use/Local Buy or** $t_i$ | **Demand or** $o_i$ | $x_i$ | $x_i^2$ |
|---|---|---|---|---|
| H1 | 12.74 | 38.53 | 0.33 | 0.11 |
| H2 | 8.25 | 26.71 | 0.31 | 0.10 |
| H3 | 25.65 | 75.17 | 0.34 | 0.12 |
| H4 | 9.51 | 30.93 | 0.31 | 0.09 |
| H5 | 19.19 | 33.23 | 0.58 | 0.33 |
| H6 | 13.13 | 25.12 | 0.52 | 0.27 |
| H7 | 15.89 | 44.38 | 0.36 | 0.13 |
| H8 | 8.62 | 20.69 | 0.42 | 0.17 |
| H9 | 2.75 | 38.59 | 0.07 | 0.01 |
| H10 | 11.11 | 78.13 | 0.14 | 0.02 |
| Sum | | | 3.38 | 1.35 |

### 8.2.2 Fixed Demand-Variable Pricing (Only PV Charging)

The local pricing $p_{loc}$ was same ranging between 17.6 cents to 28.1 (Refer section 6.2 and Equation 6.2.1 for formula). The charging with only PV was considered to charge batteries, eliminating dependency on grid and local purchases to charge battery storage. Community totals is stated in Table 8.20 (Community Trading Totals) and Table (8.31, Community Closing Accounts). Household Summary is in Table 8.32 (Individual Allocation).

Table 8.30: Community Trading Totals

| Grid Buy total (kWh) | Local Buy total (kWh) | Grid Sales total (kWh) | Local Sales total (kWh) | Use PV Total (kWh) | Use Battery Total (kWh) |
|---|---|---|---|---|---|
| 309.72 | 17.97 | 207.35 | 17.97 | 89.71 | 15.67 |

Table 8.31: Community Closing Accounts

| Total Purchase costs ($) | Total sales revenue ($) | Net Local ($) | Conventional Bill ($) | Savings ($) | %Savings |
|---|---|---|---|---|---|
| 146.45 | 25.42 | 121.03 | 189.45 | 68.42 | 36% |

Table 8.32: Individual Allocation

| **Allocation Variables/User** | H1 | H2 | H3 | H4 | H5 | H6 | H7 | H8 | H9 | H10 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Demand** | 38.53 | 26.71 | 75.17 | 30.93 | 33.23 | 25.12 | 44.38 | 20.69 | 38.59 | 78.13 |
| **Buy from Grid (kWh)** | 33.34 | 21.63 | 56.77 | 21.67 | 20.22 | 15.51 | 28.62 | 12.71 | 32.41 | 66.84 |
| **Buy Locally (kWh)** | 0.07 | 0.00 | 0.00 | 0.37 | 0.00 | 0.05 | 0.00 | 0.00 | 6.18 | 11.29 |
| **PV Sold Locally (kWh)** | 0.00 | 0.00 | 8.93 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| **PV sold to Grid (kWh)** | 7.46 | 12.42 | 20.92 | 10.03 | 10.81 | 25.60 | 4.09 | 12.03 | 0.00 | 0.00 |
| **Use PV (kWh)** | 5.12 | 5.08 | 18.40 | 8.89 | 9.81 | 4.64 | 11.15 | 5.04 | 0.00 | 0.00 |
| **Use Battery (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 3.20 | 4.92 | 4.60 | 2.95 | 0.00 | 0.00 |
| **Use PV Charging (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 5.88 | 3.42 | 2.11 | 10.17 | 0.00 | 0.00 |
| **Sell Battery Locally (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 9.04 | 0.00 | 0.00 | 0.00 | 0.00 |
| **Sell Battery to Grid (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 26.50 | 25.64 | 21.80 | 30.05 | 0.00 | 0.00 |
| **Grid Buy cost($)** | 15.35 | 9.96 | 26.14 | 9.98 | 9.31 | 7.14 | 13.18 | 5.85 | 14.92 | 30.77 |
| **Local Buy Cost ($)** | 0.01 | 0.00 | 0.00 | 0.07 | 0.00 | 0.01 | 0.00 | 0.00 | 1.31 | 2.44 |
| **Grid Sales Revenue ($)** | 0.78 | 1.29 | 2.18 | 1.04 | 3.88 | 5.33 | 2.69 | 4.38 | 0.00 | 0.00 |
| **Local Sales Revenue($)** | 0.00 | 0.00 | 1.98 | 0.00 | 0.00 | 1.87 | 0.00 | 0.00 | 0.00 | 0.00 |
| **Net Purchase cost ($)** | 14.59 | 8.67 | 21.98 | 9.01 | 5.43 | -0.04 | 10.48 | 1.47 | 16.24 | 33.21 |
| **Net Profit ($)** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.04 | 0.00 | 0.00 | 0.00 | 0.00 |
| **Conventional Bill ($)** | 17.74 | 12.30 | 34.61 | 14.24 | 15.30 | 11.56 | 20.43 | 9.53 | 17.77 | 35.97 |
| **Net savings ($)** | 3.15 | 3.63 | 12.63 | 5.24 | 9.87 | 11.56 | 9.95 | 8.05 | 1.53 | 2.76 |
| **%Savings** | 18% | 30% | 36% | 37% | 65% | 100% | 49% | 85% | 9% | 8% |

#### 8.2.2.1 Measurement Indices

Self Sufficiency(SS) = 24.73%

Self Consumption(SC) = 37.30%

Fairness Index($F(X)$) = 0.876 (Table 8.33, Fairness Index Throughput)

Table 8.33: Fairness Index Throughput

| **User** | **Self Use/Local Buy or $t_i$** | **Demand or $o_i$** | $x_i$ | $x_i^2$ |
|---|---|---|---|---|
| C1 | 5.19 | 38.53 | 0.13 | 0.02 |
| C2 | 5.08 | 26.71 | 0.19 | 0.04 |
| C3 | 18.40 | 75.17 | 0.24 | 0.06 |
| C4 | 9.26 | 30.93 | 0.30 | 0.09 |
| C5 | 13.01 | 33.23 | 0.39 | 0.15 |
| C6 | 9.61 | 25.12 | 0.38 | 0.15 |
| C7 | 15.76 | 44.38 | 0.36 | 0.13 |
| C8 | 7.99 | 20.69 | 0.39 | 0.15 |
| C9 | 6.18 | 38.59 | 0.16 | 0.03 |
| C10 | 11.29 | 78.13 | 0.14 | 0.02 |
| Sum | | | 2.69 | 0.83 |

### 8.2.3 Adjusted Demand-Minimum Local Price

The demand was adjusted for the households to derive a minimum local price . One of the assumptions in coding was made here was, that allocated demand was set to a lower bound of 0.2 kWh for a particular hour for each household and upper bound within the total generation in the pool. That is, a minimum 0.2 kWh will always be set for each household for necessary equipments and the demand will never go zero. The lower bound helped generating a lower adjusted demand and hence, a lower local price range.

#### 8.2.3.1 Adjusted Demand and Local Price Calculation

The local price and adjusted demand was derived from section 6.5.1 (Equation 6.5.1). The local price $p_{loc}$ was obtained between 13.0 to 14.5 cents (Fig.8.4, Pricing and Normalized Demand for 48 hours). The total adjusted demand ($e_{dnew,i,t}$) was 406.16 kWh, which was slightly less than the actual demand of 411.48 kWh (Table 8.34, Community Totals). However, the adjusted demand curve still constituted some peaks. The community trading totals can referred from Table 8.35 (Community Trading Totals) and Table 8.36 (Community Closing Accounts). Household trading results are stated in Table 8.37 (Individual Allocation).

Table 8.34: Community Totals

| Hours | Total demand (kWh) | Total PV (kWh) |
|---|---|---|
| 48 | 406.16 | 202.00 |

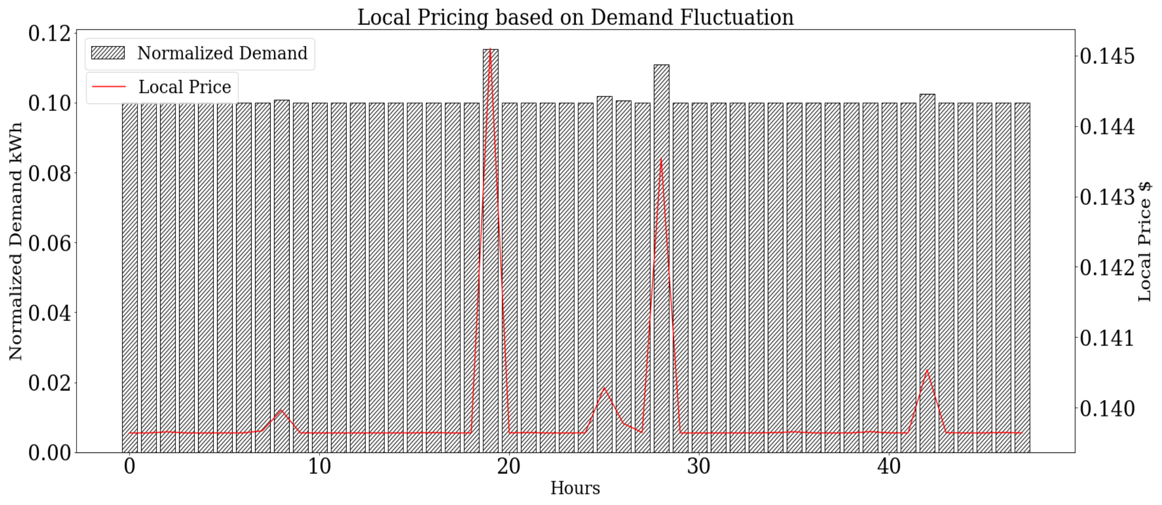

Figure 8.4: Pricing and Normalized Demand for 48 hours

Table 8.35: Community Trading Totals

| Grid Buy total (kWh) | Local Buy total (kWh) | Grid Sales total (kWh) | Local Sales total (kWh) | Use PV Total (kWh) | Use Battery Total (kWh) |
|---|---|---|---|---|---|
| 345.33 | 13.24 | 246.76 | 13.24 | 71.61 | 25.48 |

Table 8.36: Community Closing Accounts

| Total Purchase costs ($) | Total sales revenue ($) | Net Local ($) | Conventional Bill ($) | Savings ($) | %Savings |
|---|---|---|---|---|---|
| 160.84 | 27.51 | 133.32 | 187.00 | 53.67 | 29% |

Table 8.37: Individual Allocation

| **Allocation Variables/User** | H1 | H2 | H3 | H4 | H5 | H6 | H7 | H8 | H9 | H10 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Demand** | 52.38 | 24.83 | 86.57 | 24.79 | 31.54 | 30.44 | 24.80 | 30.02 | 28.00 | 72.79 |
| **Buy from Grid (kWh)** | 43.66 | 17.53 | 68.12 | 17.75 | 16.55 | 18.44 | 13.55 | 18.39 | 27.14 | 68.81 |
| **Buy Locally (kWh)** | 2.43 | 0.04 | 2.80 | 0.00 | 0.00 | 0.00 | 0.43 | 0.00 | 0.87 | 3.97 |
| **Buy Charging Locally (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.40 | 1.82 | 0.50 | 0.00 | 0.00 |
| **Buy Chargingfrom Grid (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 9.51 | 5.08 | 9.11 | 11.69 | 0.00 | 0.00 |
| **PV Sold Locally (kWh)** | 0.40 | 0.00 | 0.00 | 0.00 | 0.00 | 7.58 | 0.04 | 0.00 | 0.00 | 0.00 |
| **PV sold to Grid (kWh)** | 5.89 | 10.24 | 32.58 | 11.88 | 18.92 | 16.07 | 9.62 | 17.15 | 0.00 | 0.00 |
| **Use PV (kWh)** | 6.29 | 7.27 | 15.66 | 7.04 | 7.18 | 5.57 | 5.43 | 5.77 | 0.00 | 0.00 |
| **Use Battery (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 7.80 | 6.42 | 5.39 | 5.87 | 0.00 | 0.00 |
| **Use PV Charging (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 0.39 | 4.43 | 2.27 | 4.31 | 0.00 | 0.00 |
| **Sell Battery Locally (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 2.43 | 2.80 | 0.00 | 0.00 | 0.00 |
| **Sell Battery to Grid (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 25.20 | 37.35 | 28.11 | 33.73 | 0.00 | 0.00 |
| **Grid Buy cost($)** | 20.10 | 8.07 | 31.36 | 8.17 | 7.62 | 8.67 | 7.08 | 8.69 | 12.49 | 31.68 |
| **Local Buy Cost ($)** | 0.34 | 0.01 | 0.39 | 0.00 | 0.00 | 0.06 | 0.31 | 0.07 | 0.12 | 0.55 |
| **Grid Sales Revenue ($)** | 0.61 | 1.06 | 3.39 | 1.24 | 4.59 | 5.56 | 3.92 | 5.29 | 0.00 | 0.00 |
| **Local Sales Revenue($)** | 0.06 | 0.00 | 0.00 | 0.00 | 0.00 | 1.40 | 0.40 | 0.00 | 0.00 | 0.00 |
| **Net Purchase cost ($)** | 19.77 | 7.01 | 28.36 | 6.94 | 3.03 | 1.77 | 3.07 | 3.47 | 12.61 | 32.24 |
| **Net Profit ($)** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| **Conventional Bill ($)** | 24.12 | 11.43 | 39.86 | 11.42 | 14.52 | 14.01 | 11.42 | 13.82 | 12.89 | 33.51 |
| **Net savings ($)** | 4.34 | 4.42 | 11.49 | 4.48 | 11.49 | 12.24 | 8.35 | 10.35 | 0.28 | 1.27 |
| **%Savings** | 18% | 39% | 29% | 39% | 79% | 87% | 73% | 75% | 2% | 4% |

#### 8.2.3.2 Measurement Indices

Self Sufficiency(SS) = 23.69%, Self Consumption(SC) = 30.9%, Fairness Index ($F(X)$) = 0.773 (Table 8.38, Fairness Index Throughput).

Table 8.38: Fairness Index Throughput

| **User** | **Self Use/Local Buy or $t_i$** | **Demand or $o_i$** | $x_i$ | $x_i^2$ |
|---|---|---|---|---|
| H1 | 8.72 | 52.38 | 0.17 | 0.03 |
| H2 | 7.30 | 24.83 | 0.29 | 0.09 |
| H3 | 18.45 | 86.57 | 0.21 | 0.05 |
| H4 | 7.04 | 24.79 | 0.28 | 0.08 |
| H5 | 14.98 | 31.54 | 0.48 | 0.23 |
| H6 | 12.00 | 30.44 | 0.39 | 0.16 |
| H7 | 11.25 | 24.80 | 0.45 | 0.21 |
| H8 | 11.64 | 30.02 | 0.39 | 0.15 |
| H9 | 0.87 | 28.00 | 0.03 | 0.00 |
| H10 | 3.97 | 72.79 | 0.05 | 0.00 |
| Sum | | | 2.75 | 0.98 |

### 8.2.4 Adjusted Demand-Minimum Local Price (Only PV Charging)

The adjusted demand scenario was rechecked with only PV charging criteria for battery storage. Community Trading results (Table 8.39, Community Trading Results and Table 8.40, Community Closing Accounts) and Household results (Table 8.41, Individual Allocation) are stated for observations. The local price $p_{loc}$ remained same between 13.0 to 14.5 cents with total adjusted demand as 406.16 kWh.

Table 8.39: Community Trading Totals

| Grid Buy total (kWh) | Local Buy total (kWh) | Grid Sales total (kWh) | Local Sales total (kWh) | Use PV Total (kWh) | Use Battery Total (kWh) |
|---|---|---|---|---|---|
| 302.62 | 21.52 | 206.95 | 21.52 | 93.13 | 19.00 |

Table 8.40: Community Closing Accounts

| Total Purchase costs ($) | Total sales revenue ($) | Net Local ($) | Conventional Bill ($) | Savings ($) | %Savings |
|---|---|---|---|---|---|
| 142.33 | 24.53 | 117.80 | 187.00 | 69.19 | 37% |

Table 8.41: Individual Allocation

| **Allocation Variables/User** | H1 | H2 | H3 | H4 | H5 | H6 | H7 | H8 | H9 | H10 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Demand** | 52.38 | 24.83 | 86.57 | 24.79 | 31.54 | 30.44 | 24.80 | 30.02 | 28.00 | 72.79 |
| **Buy from Grid (kWh)** | 45.87 | 17.56 | 67.30 | 16.68 | 13.83 | 18.16 | 13.71 | 19.62 | 25.38 | 64.51 |
| **Buy Locally (kWh)** | 0.22 | 0.00 | 3.62 | 1.07 | 5.21 | 0.50 | 0.00 | 0.00 | 2.62 | 8.28 |
| **PV Sold Locally (kWh)** | 0.11 | 0.58 | 13.45 | 0.18 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| **PV sold to Grid (kWh)** | 6.18 | 9.66 | 19.14 | 11.70 | 9.43 | 21.11 | 5.43 | 11.90 | 0.00 | 0.00 |
| **Use PV (kWh)** | 6.29 | 7.27 | 15.66 | 7.04 | 7.99 | 6.10 | 6.55 | 6.13 | 0.00 | 0.00 |
| **Use Battery (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 4.50 | 5.68 | 4.55 | 4.27 | 0.00 | 0.00 |
| **Use PV Charging (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 9.07 | 6.45 | 5.38 | 9.21 | 0.00 | 0.00 |
| **Sell Battery Locally (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 5.43 | 1.77 | 0.00 | 0.00 | 0.00 |
| **Sell Battery to Grid (kWh)** | 0.00 | 0.00 | 0.00 | 0.00 | 28.50 | 31.80 | 23.38 | 28.73 | 0.00 | 0.00 |
| **Grid Buy cost($)** | 21.12 | 8.09 | 30.99 | 7.68 | 6.37 | 8.36 | 6.31 | 9.03 | 11.68 | 29.70 |
| **Local Buy Cost ($)** | 0.03 | 0.00 | 0.51 | 0.15 | 0.73 | 0.07 | 0.00 | 0.00 | 0.37 | 1.16 |
| **Grid Sales Revenue ($)** | 0.64 | 1.00 | 1.99 | 1.22 | 3.94 | 5.50 | 3.00 | 4.23 | 0.00 | 0.00 |
| **Local Sales Revenue($)** | 0.01 | 0.08 | 1.88 | 0.03 | 0.00 | 0.76 | 0.25 | 0.00 | 0.00 | 0.00 |
| **Net Purchase cost ($)** | 20.49 | 7.00 | 27.62 | 6.59 | 3.15 | 2.17 | 3.07 | 4.81 | 12.05 | 30.86 |
| **Net Profit ($)** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| **Conventional Bill ($)** | 24.12 | 11.43 | 39.86 | 11.42 | 14.52 | 14.01 | 11.42 | 13.82 | 12.89 | 33.51 |
| **Net savings ($)** | 3.62 | 4.43 | 12.23 | 4.83 | 11.37 | 11.84 | 8.35 | 9.01 | 0.84 | 2.66 |
| **%Savings** | 15% | 39% | 31% | 42% | 78% | 85% | 73% | 65% | 7% | 8% |

#### 8.2.4.1 Measurement Indices

Measurement Indices were calculated as: Self Sufficiency = 25.49%, Self Consumption = 39.24%, Fairness Index = 0.798 (Table 8.42, Fairness Index Throughput).

Table 8.42: Fairness Index Throughput

| **User** | **Self Use/Local Buy or $t_i$** | **Demand or $o_i$** | $x_i$ | $x_i^2$ |
|---|---|---|---|---|
| C1 | 6.51 | 52.38 | 0.12 | 0.02 |
| C2 | 7.27 | 24.83 | 0.29 | 0.09 |
| C3 | 19.27 | 86.57 | 0.22 | 0.05 |
| C4 | 8.11 | 24.79 | 0.33 | 0.11 |
| C5 | 17.71 | 31.54 | 0.56 | 0.32 |
| C6 | 12.28 | 30.44 | 0.40 | 0.16 |
| C7 | 11.09 | 24.80 | 0.45 | 0.20 |
| C8 | 10.40 | 30.02 | 0.35 | 0.12 |
| C9 | 2.62 | 28.00 | 0.09 | 0.01 |
| C10 | 8.28 | 72.79 | 0.11 | 0.01 |
| Sum | | | 2.93 | 1.08 |

### 8.2.5 Vickrey Clarke Groves Auction Model

Community and Households results for 48 hours duration through Table 8.43 (Community Usage and Trading Totals) and Table 8.44 (Community Closing Accounts). Prosumer usage and net demand is summarized in Table 8.45 (Total Prosumer Usage and Net Demand). Total Prosumer sales is stated in Table 8.26 (Prosumer Sales) and Buyers Transactions are collected in Table 8.47 (Buyers Purchases). Household savings is calculated in Table 8.48 (Household Closing Accounts)

Table 8.43: Community Usage and Trading Totals

| Demand kWh | PV kWh | Net Demand kWh | Use PV kWh | Use PV Charging kWh | Use Battery kWh | Local Sales kWh | Grid Sales kWh | Grid Buy kWh |
|---|---|---|---|---|---|---|---|---|
| 411.48 | 201.998 | 327.395 | 69.092 | 17.552 | 14.993 | 89.978 | 132.483 | 237.417 |

Table 8.44: Community Closing Accounts

| Conventional Cost ($) | Local Buy Cost ($) | Grid Buy Cost ($) | Grid Sales Revenue ($) | Local Sales Revenue ($) | Savings |
|---|---|---|---|---|---|
| 189.45 | 20.88 | 109.20 | 13.78 | 20.88 | 49.6% |

Table 8.45: Total Prosumer Usage and Net Demand

| Prosumer | H1 | H2 | H3 | H4 | H5 | H6 | H7 | H8 | H9 | H10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Prosumer demand (kWh) | 38.53 | 26.71 | 75.17 | 30.93 | 33.23 | 25.12 | 44.38 | 20.69 | 38.59 | 78.13 |
| PV (kWh) | 12.58 | 17.50 | 48.24 | 18.93 | 26.50 | 33.66 | 17.36 | 27.24 | 0.00 | 0.00 |
| Net Demand (kWh) | 33.41 | 21.63 | 56.77 | 22.04 | 20.00 | 15.35 | 28.91 | 12.55 | 38.59 | 78.13 |
| Surplus PV (JkWh) | 7.46 | 12.42 | 29.85 | 10.03 | 10.23 | 28.25 | 3.19 | 13.92 | 0.00 | 0.00 |
| Use PV (kWh) | 5.12 | 5.08 | 18.40 | 8.89 | 9.96 | 4.90 | 11.39 | 5.37 | 0.00 | 0.00 |
| Use PV Charging (kWh) | 0.00 | 0.00 | 0.00 | 0.00 | 6.31 | 0.51 | 2.78 | 7.95 | 0.00 | 0.00 |
| Use Battery (kWh) | 0.00 | 0.00 | 0.00 | 0.00 | 3.27 | 4.87 | 4.08 | 2.77 | 0.00 | 0.00 |
| Battery Surplus (kWh) | 0.00 | 0.00 | 0.00 | 0.00 | 26.43 | 31.43 | 22.32 | 26.93 | 0.00 | 0.00 |

Table 8.46: Prosumer Sales

| Prosumer | H1 | H2 | H3 | H4 | H5 | H6 | H7 | H8 | Total kWh |
|---|---|---|---|---|---|---|---|---|---|
| Energy Sold kWh (Local +Grid) | 7.46 | 12.42 | 29.85 | 10.03 | 36.65 | 59.68 | 25.52 | 40.85 | 222.46 |

Table 8.47: Buyers Purchases

| Buyer | H1 | H2 | H3 | H4 | H5 | H6 | H7 | H8 | H9 | H10 | Total kWh |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Energy Purchased kWh | 33.41 | 21.63 | 56.77 | 22.04 | 20.00 | 15.35 | 28.91 | 12.55 | 38.59 | 78.13 | 327.40 |

Table 8.48: Household Closing Accounts

| | H1 | H2 | H3 | H4 | H5 | H6 | H7 | H8 | H9 | H10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Total Grid Buy cost ($) | 12.34 | 8.06 | 20.19 | 9.02 | 8.89 | 7.07 | 12.39 | 5.66 | 8.83 | 16.83 |
| Total Local Buy cost ($) | 1.13 | 0.63 | 2.60 | 0.32 | 0.13 | 0.00 | 0.31 | 0.03 | 3.91 | 11.82 |
| Grid Sales Revenue($) | 0.31 | 0.26 | 2.52 | 0.45 | 2.12 | 4.06 | 0.61 | 3.43 | 0.00 | 0.00 |
| Local Sales Revenue($) | 1.28 | 2.89 | 1.24 | 1.77 | 3.15 | 4.67 | 4.43 | 1.45 | 0.00 | 0.00 |
| Net Local Cost ($) | 11.98 | 5.60 | 19.20 | 7.19 | 3.82 | 0.00 | 7.75 | 0.85 | 12.81 | 28.78 |
| Net Local Profit | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 1.62 | 0.00 | 0.00 | 0.00 | 0.00 |
| Conventional Bill ($) | 17.74 | 12.30 | 34.61 | 14.24 | 15.30 | 11.56 | 20.43 | 9.53 | 17.77 | 35.97 |
| Savings ($) | 5.76 | 6.70 | 15.41 | 7.05 | 11.49 | 11.56 | 12.68 | 8.68 | 4.96 | 7.19 |
| %Savings | 33% | 55% | 45% | 50% | 76% | 100% | 62% | 92% | 28% | 20% |

#### 8.2.5.1 Measurement Indices

Measurement Indices are summarized as:

Self Sufficiency = 42.3%, Self Consumption = 59.12%, Fairness Index $F(X) = 0.975$ (Table 8.49, Fairness Index Throughput).

Table 8.49: Fairness Index Throughput

| User | Self-Use/Local Buy or $t_i$ | Demand or $o_i$ | $x_i$ | $x_i^2$ |
|---|---|---|---|---|
| H1 | 11.72 | 38.53 | 0.30 | 0.09 |
| H2 | 9.20 | 26.71 | 0.34 | 0.12 |
| H3 | 31.30 | 75.17 | 0.42 | 0.17 |
| H4 | 11.34 | 30.93 | 0.37 | 0.13 |
| H5 | 13.41 | 33.23 | 0.40 | 0.16 |
| H6 | 9.77 | 25.12 | 0.39 | 0.15 |
| H7 | 17.46 | 44.38 | 0.39 | 0.15 |
| H8 | 8.39 | 20.69 | 0.41 | 0.16 |
| H9 | 19.41 | 38.59 | 0.50 | 0.25 |
| H10 | 41.47 | 78.13 | 0.53 | 0.28 |
| Sum | | | 4.06 | 1.69 |

### 8.2.6 Discussions

The overall performance for the models was predictable for this Scenario load set due to similarity in demand and generation patterns as in scenario case-I and can be referred in Table 8.50 (Dispatch Performance Summary). The Fixed Demand-Variable Price model results obtained in section 8.2.1 performed fairly for the second dataset, with moderate savings and performance index. The community savings was about 28% and Prosumer H5 and H6 generated profit from the transaction hence, their savings for recorded as 100%. Performance indices were average for the model.

The Fixed Demand-Variable Price model tested with only PV charging in section 8.2.2 and improved the savings of the consumer H9 from 4 % to 9% as the dispatch decisions of Prosumers changed and slight increase in local sales was observed. The overall supply in the pool was reduced as battery charging was dependent on the surplus PV only. Prosumer H5 got reduced savings of 65%. Community savings increased to 36%. The self sufficiency reduced to 24.73% and Self Consumption was better than previous fixed demand model with value increased to 37.30%.

Table 8.50: Dispatch Performance Summary

| Model | Local Pricing Range(cents) | Community Savings% | Maximum Individual Savings% | Minimum Individual Savings% | SS% | SC% | F(X) |
|---|---|---|---|---|---|---|---|
| Fixed Variable Local Price | 17.6 to 20 | 28% | 100% | 4% | 30.82% | 34.54% | 0.844 |
| Fixed Variable Local Price (Only PV Charging) | 17.6 to 20 | 36% | 100% | 8% | 24.73% | 37.30% | 0.876 |
| Adjusted Demand Minimum Local Pricing | 13.9 to 14 | 29% | 87% | 2% | 23.69% | 30.90% | 0.773 |
| Adjusted Demand Minimum Local Pricing (Only PV Charging) | 13.9 to 14 | 37% | 85% | 7% | 25.49% | 39.24% | 0.798 |
| VCG | Bid based | 49.60% | 100% | 20% | 42.30% | 59.12% | 0.975 |

The Adjusted Demand-Minimum Local Price model showed reduced performance with the new adjusted demand of 406.16 kWh. The local pricing achieved was low between 13 cents to 14.5 cents. The model did not perform well in terms of savings (community savings was to 29% and individual household savings was as low as 2%). Performance metrics were also less than satisfactory values (SS = 23.60%, SC = 30.9%, $F(X)$ = 0.773).

The Adjusted Demand-Minimum Local Price model tested with only PV Charging improved the community savings moderately with same set of adjusted demand (406.16 kWh) and pricing (between 13 cents to 14.5 cents). Performance metrics improved mildly with this model and community savings increasing to 37% compared to previous model having option of battery charging with grid and local market purchases.

VCG model for second load set showed promising results again with savings ranging between 62% to 100% for prosumers with PV and Battery (H5, H6, H7, H8) due to revenue obtained from the sales to grid and local market. The self sufficiency numbers showed improvement as the DERs met major share of the community demand as compared to MILP models. The slightly less self consumption value indicated less local usage as compared to total PV and Battery supply in pool.

The Fairness index in MILP models behaved similarly as in scenario case-I ranging in moderate numbers between 0.773 to 0.876. The Fairness index was 0.975 with VCG model indicating all users were receiving allocation close to their optimum and fairly equal with respect to it terms of their demand. The results were expected to be similar in pattern as in scenario case-I, because they showed similar demand and generation trends.

# CHAPTER 9: CONCLUSION

## 9.1 Observations and Conclusion

In this thesis, an important segment of the Local energy trading was studied which emphasized on the consumer welfare inference through pricing and dispatch schema devised for two sets of load scenarios for a local residential set up in New South Wales Market, Australia. All the Households in the optimization models considered utility function of increasing savings from the local market usage and reducing grid purchase costs. The benefits of local energy sharing models were evaluated and measured from the community's as well as individual household's perspective through performance indices. Fewer models have looked into this aspect in the literature review so far and did not focus much on the fair distribution of resources with respect to individual demand. It is to be noted that some factors like environmental benefits, battery and PV installation costs, and investment recovery etc, are not considered in the work, however, they can be included in the later works. Households having only battery storage were not considered to be feasible option in the trading models in thesis, as battery charging becomes dependent on purchases from the markets and may affect the savings or revenues. The simulation was performed for small time horizon of 48 hour due to hardware limitations.

Five dispatch mechanisms (four MILP and one Auction based) were implemented with a pricing strategy. Model performances and social welfare measures were computed to analyze weight of consumer and prosumers trading results. The simplistic approach for dispatch used initially was Mixed Integer Linear programming with local price varying between grid price and tariff price. The model results showed moderate amount of savings by individual households and community as whole. The dispatch was further modified by introducing an adjusted demand topology in which local pricing can be minimized each hour by allowing households to adjust demand within supply pool. However, the performance of this topology was not impressive, as it did use the DERs resources efficiently at the consumer level. The modified VCG mechanism performed well for both load case scenarios and provided promising dispatch structure for improving the self sufficiency and fairness in distribution of the DERs as compared to MILP based dispatch. Numerical results corroborate that the proposed mechanism was able to meet the requirement of optimal use of DERs in the local market pool and reduce grid dependency. The model results showed pricing strategy worked well for the local trading platform for both the MILP and Auction based model. The VCG model provided suitable savings to consumer and prosumers by setting priority to bidders and sellers, which distributed savings and incomes proportionally to satisfy their respective utility functions. Thus, with the given load sets, the market performed better with the auction model in terms of savings and performance indices. The goal of the local market is also to cut the peak demand patterns from the curve, a scenario where a consumer having the major share of the community can capture the entire generation from the Prosumers, but the higher pricing for such a consumer in the pool will not encourage the sentiment to have a higher demand and it is assumed that with auction system based on proposed pricing the consumers will be motivated to their manage demand within the surplus generation persisting in the pool.

The 100% self sufficiency is obtained if all the local generation is consumed in the local market, but 100% number is difficult to achieve especially in a model where trading is motioned for periodic cycles like 15 minute or 1 hour and this periodic synchronization of supply and demand is never available. To get surplus battery storage energy to fill in the demand supply mismatch can be difficult sometimes, due to discharging and charging cycles [19]. Similarly, the self consumption can be achieved as 100%, if all the generation is used up in the local market, but the continuous fluctuation in the demand and generation does not make it possible to meet this percentage. Because for particular time periods surplus demand is not fulfilled by local supply or surplus generation is not used up in local market and is sold to grid, hence, this periodic mismatch again reduces the cumulative results of self consumption. From demand versus generation curves in Figure 4.1 and 4.2 we can observe that, the demand is at peak in the night time and lower during day time. This can be stated as one of the reasons of lower Self sufficiency numbers in all the models because of the high stress on battery charge/discharge cycles and struggle to keep low expenditure for battery charging, making battery dispatch difficult in many hourly instances. Thus, this possibly calls for a higher battery size and higher PV size or changing the demand pattern, because surplus PV generated was unable to meet both local demand and battery charging in both the scenario cases. However, budget and space constraints require careful consideration for planning higher sizes. Changing demand patterns need detailed equipment schedules. As these details are unavailable for calculating intricate sizes for given datasets, this part not been considered in suggestion right now.

From above discussions it can be seen that, measurement indexes provide an idea of model performance in terms of demand and supply available in the pool, and are independent numbers. The numbers will vary for different load and generation patterns, thus comparison with previous proven test results may not feasible as the locations and conditions vary. Performance indices are good indicators to quantify improvement in consumption of households and generation capacities of the DERs and synchronize them suitably by changing consumption timings, manually planning battery dispatch or increasing or decreasing PV capacity etc. With additional details like equipment schedule, area of household etc., a trade-off between measurement indices like Self Sufficiency and Self Consumption can be effectively planned to implement equipment schedule, PV/battery sizes, and a suitable dispatch model can be finalized for a given community that can synchronize demand and DERs available at best capacity.

Social welfare or fairness in distribution is difficult to measure in a network distribution especially when the nodes are not homogeneous and have different optimals [84][85]. Similar structure stands for the energy trading mechanisms where each household has different optimal demand with limited generation in the local pool, it is not fair to divide the supply equally among households. One of the interpretation fairness index looks to solve is to divide the supply based on the required proportion of demand for each household. Fairness index can be used as a metric to ensure all users are well-off in the model and pricing and dispatch can be combined appropriately in the allocation model to develop a fair distribution of revenues and savings. It is understood from the given dataset and simulations that prosumers are likely to have more returns from the trading as they are sellers in the market, and consumers will have advantage to save expenditure. Thus, Fairness index can be looked upon as a common metric to ensure that, fair allocation is obtained between consumer and prosumers from the local market transactions.

Power generation with DERs like solar are highly unpredictable in nature due to intermittency in weather patterns [86]. Many models have come up previously, that look into demand response and load scheduling by smart appliances. But achieving adjusted demand could be difficult in local market conditions, as it is difficult to predict which household will produce what amount of energy each hour (from PV) or use what amount of Battery storage energy, and what percentage energy will be consumed or sold to other household. Hence, this raises new challenge in handling function and management of equipments, that is to make them adaptable to volatility, and ensure safety and stability through flexible measures. This includes integration of fast reacting demand response and storage systems [87], which need to be physically and computationally robust. But they can be subject to financial constraints for some residential set ups that have limited household budget, limited income, lesser credit etc. As a result, new and economic methods of energy dispatch and pricing mechanisms based on auctions, game theory, incentives and centrally controlled models can be explored for local markets which meet such constraints and can be improved further to provide flexible solutions suiting different local communities based on their characteristic usage patterns, physical layout, income, existing utility prices etc.

## 9.2 Future Work

The thesis used simple dispatch and pricing model in the system which are naive at this stage and intend to explore economic models of dispatch based on available data and resources. It does not consider complexity of physical constraints in the system like the transmission looses, grid congestion's, non-linear characteristics in battery charging/discharging process. Additional financial considerations like demand response charges, time of use tariff, rental cost of transmission services to utility company, policy, regulations, aspects have been skipped for now in order to see how the model performance achieved with simple numerical formulations. The future work can consider including the above constraints for refining the model, however, detailed information is required to realistically merge these constraints within trading model and needs real time data to simulate, as many such factors are highly volatile like transmission losses, demand response, etc., and will differ for each trading participant and for every dispatch node. The models can also be tested with different set of prices for peak and off-peak periods, and use them in pricing strategy to asses household and community gains. The time horizon can be increased to get monthly or yearly estimates of savings and verify model performance for more locations, if hardware requirements are met.

## REFERENCES

[1] T. Morstyn, N. Farrell, S. J. Darby, and M. D. McCulloch, “Using peer-to-peer energy-trading platforms to incentivize prosumers to form federated power plants,” *Nature Energy*, vol. 3, no. 2, pp. 94–101, 2018.

[2] “Peer-to-Peer Energy Trading Still Looks Like a Distant Prospect | Greentech Media.”

[3] Vandebron, “Duurzame energie van Nederlandse bodem,” 2019.

[4] Piclo, “Piclo - Building software for a smarter energy future,” 2019.

[5] B. Brandherm, J. Baus, and J. Frey, “Peer energy cloud-Civil marketplace for trading renewable energies,” in *Proceedings - 8th International Conference on Intelligent Environments, IE 2012*, pp. 375–378, 2012.

[6] R. Apel, J. Benze, K. Eger, S. Fries, A. Harner, K. Hemberger, R. Hoffmann, G. Kaestle, M. Kahmann, P. Kellendonk, H. Kerber, A. Kießling, S. Kosslers, S. Lehnhoff, A. Malina, W. Mohr, T. Müller, A. Nattrodt, T. Niemand, A. Probst, A. Schindler, B. Schulz, R. Sporer, M. Staubermann, J. Stein, H. Steusloff, A. Suhr, R. Tretter, L. Uhl, and M. Uslar, “The German Roadmap E-Energy/Smart Grid 2.0,” tech. rep., 2012.

[7] “sonnenCommunity.”

[8] C. Zhang, J. Wu, C. Long, and M. Cheng, “Review of Existing Peer-to-Peer Energy Trading Projects,” *Energy Procedia*, vol. 105, pp. 2563–2568, 2017.

[9] Amit Rosner, “Lights Out for Yeloha - Why We Shut Down the Solar Sharing Network | Amit Rosner | Pulse | LinkedIn.”

[10] “The LichtBlick company: Generation of pure energy.”

[11] “Brooklyn’s TransActive Grid Will Allow Neighbors to Share Local Solar-Powered Microgrid.”

[12] C. Zhang, J. Wu, M. Cheng, Y. Zhou, and C. Long, “A Bidding System for Peer-to-Peer Energy Trading in a Grid-connected Microgrid,” *Energy Procedia*, vol. 103, pp. 147–152, dec 2016.

[13] C. Zhang, J. Wu, Y. Zhou, M. Cheng, and C. Long, “Peer-to-Peer energy trading in a Microgrid,” *Applied Energy*, vol. 220, pp. 1–12, 2018.

[14] A. Goranovic, M. Meisel, L. Fotiadis, S. Wilker, A. Treytl, and T. Sauter, “Blockchain applications in microgrids: An overview of current projects and concepts,” *Proceedings IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, vol. 2017-Janua, no. October, pp. 6153–6158, 2017.

[15] Power Ledger, “WHITEPAPER We believe empowering individuals and communities to co-create their energy future will underpin the development of a power system that is resilient , low-cost , zero-carbon and owned by the people of the world .,” tech. rep., 2019.

[16] "Bankymoon - Blockchain enabled solutions and services."

[17] "Power Ledger Integrates Blockchain-Based Energy Auditing in Solar Power Asset."

[18] "Power Ledger Wallet - Secure your Power Ledger (POWR) assets | Ledger."

[19] E. Mengelkamp, P. Staudt, J. Garttner, and C. Weinhardt, "Trading on local energy markets: A comparison of market designs and bidding strategies," *International Conference on the European Energy Market, EEM*, 2017.

[20] N. Liu, X. Yu, C. Wang, C. Li, L. Ma, and J. Lei, "Energy-Sharing Model with Price-Based Demand Response for Microgrids of Peer-to-Peer Prosumers," *IEEE Transactions on Power Systems*, vol. 32, no. 5, pp. 3569–3583, 2017.

[21] C. Long, Y. Zhou, and J. Wu, "A game theoretic approach for peer to peer energy trading," *Energy Procedia*, vol. 159, pp. 454–459, 2019.

[22] C. Long, J. Wu, C. Zhang, M. Cheng, and A. Al-Wakeel, "Feasibility of Peer-to-Peer Energy Trading in Low Voltage Electrical Distribution Networks," *Energy Procedia*, vol. 105, pp. 2227–2232, 2017.

[23] C. Long, J. Wu, C. Zhang, L. Thomas, M. Cheng, and N. Jenkins, "Peer-to-peer energy trading in a community microgrid," *IEEE Power and Energy Society General Meeting*, vol. 2018-Janua, no. July, pp. 1–5, 2018.

[24] M. R. Alam, M. St-Hilaire, and T. Kunz, "An optimal P2P energy trading model for smart homes in the smart grid," *Energy Efficiency*, vol. 10, pp. 1475–1493, dec 2017.

[25] C. Long, J. Wu, Y. Zhou, and N. Jenkins, "Peer-to-peer energy sharing through a two-stage aggregated battery control in a community Microgrid," *Applied Energy*, vol. 226, no. June, pp. 261–276, 2018.

[26] A. Lüth, J. M. Zepter, P. Crespo del Granado, and R. Egging, "Local electricity market designs for peer-to-peer trading: The role of battery flexibility," *Applied Energy*, vol. 229, no. August, pp. 1233–1243, 2018.

[27] R. Jing, M. N. Xie, F. X. Wang, and L. X. Chen, "Fair P2P energy trading between residential and commercial multi-energy systems enabling integrated demand-side management," *Applied Energy*, vol. 262, p. 114551, mar 2020.

[28] J. M. Zepter, A. Lüth, P. Crespo del Granado, and R. Egging, "Prosumer integration in wholesale electricity markets: Synergies of peer-to-peer trade and residential storage," *Energy and Buildings*, vol. 184, pp. 163–176, 2019.

[29] S. Zhou, F. Zou, Z. Wu, W. Gu, Q. Hong, and C. Booth, "A smart community energy management scheme considering user dominated demand side response and P2P trading," *International Journal of Electrical Power and Energy Systems*, vol. 114, no. May 2019, p. 105378, 2020.

[30] D. H. Park, Y. G. Park, J. H. Roh, K. Y. Lee, and J. B. Park, "A Hierarchical Peer-to-Peer Energy Transaction Model Considering Prosumer's Renewable Energy Preference," *IFAC-PapersOnLine*, vol. 52, no. 4, pp. 312–317, 2019.

[31] A. De Paola, D. Angeli, and G. Strbac, "Price-Based Schemes for Distributed Coordination of Flexible Demand in the Electricity Market," *IEEE Transactions on Smart Grid*, vol. 8, no. 6, pp. 3104–3116, 2017.

[32] R. Jain, "A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems," no. June 2014, 1998.

[33] Energy Networks Australia, "Guide to Australia's energy networks," tech. rep.

[34] W. Wu, G. Quezada, E. Schleiger, A. Bratanova, P. Graham, and B. Spak, "THE FUTURE OF PEER TO PEER TRADING OF DISTRIBUTED," 2019.

[35] "Ausgrid - Home."

[36] "Solar home electricity data - Ausgrid."

[37] "SOLAR FEED-IN TARIFF BENCHMARK," tech. rep., 2019.

[38] L. P. Fee, D. Debit, D. Payment, C. Card, D. Payment, D. Fee, R. Fee, C. Payment, P. Fee, P. Energy, and P. Number, "Energy Market Offer - Fact Sheet Solar Feed-in Pool Automation & Services Offer - Fact Sheet Pool Services Offer - Charges."

[39] Y. Wang, K. Lai, F. Chen, Z. Li, and C. Hu, "Shadow price based co-ordination methods of microgrids and battery swapping stations," *Applied Energy*, vol. 253, no. July, p. 113510, 2019.

[40] R. Jain, D. Chiu, and W. Hawe, "A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems," *2015 IEEE Power and Energy Society Innovative Smart Grid Technologies Conference, ISGT 2015*, no. June, 1998.

[41] S. Nguyen, W. Peng, P. Sokolowski, D. Alahakoon, and X. Yu, "Optimizing rooftop photovoltaic distributed generation with battery storage for peer-to-peer energy trading," *Applied Energy*, vol. 228, no. September 2018, pp. 2567–2580, 2018.

[42] "1. Introduction to Computers and Python - Python for Programmers, First Edition."

[43] Gurobi, "Gurobi - The fastest solver."

[44] "Gurobi - Wikipedia."

[45] W. Tushar, S. Member, T. K. Saha, C. Yuen, S. Member, and H. V. Poor, "Peer-to-Peer Trading in Electricity Networks : An Overview," pp. 1–15.

[46] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," 2010.

[47] N. Programming, "Nonlinear programming 4," in *Mathematics and Computers in Simulation*, vol. 25, pp. 86–87, 1983.

[48] "Game Theory Definition."

[49] S. I.-p. Models, "Integer Programming 9," pp. 272–319.

[50] Gurobi Optimization, "Documentation - Gurobi," 2020.

[51] E. K. P. Chong, "An introduction to optimization."

[52] Gurobi, "Mixed-Integer Programming ( MIP ) - A Primer on the Basics Branch-and-Bound," 2018.

[53] J. Guerrero, A. C. Chapman, and G. Verbic, "Decentralized P2P Energy Trading under Network Constraints in a Low-Voltage Network," *IEEE Transactions on Smart Grid*, vol. 10, no. 5, pp. 5163–5173, 2018.

[54] V. Kenny, M. Nathal, and S. Saldana, "Heuristic algorithms - optimization," 2016.

[55] "Auctions - Econlib."

[56] K. Chen, J. Lin, and Y. Song, "Trading strategy optimization for a prosumer in continuous double auction-based peer-to-peer market: A prediction-integration model," *Applied Energy*, vol. 242, no. September 2018, pp. 1121–1133, 2019.

[57] M. Khorasany, Y. Mishra, and G. Ledwich, "Auction based energy trading in transactive energy market with active participation of prosumers and consumers," in *2017 Australasian Universities Power Engineering Conference (AUPEC)*, vol. 2017-Novem, pp. 1–6, IEEE, nov 2017.

[58] J. Bredin and D. C. Parkes, "Models for truthful online double auctions," *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence, UAI 2005*, pp. 50–59, 2005.

[59] P. R. Wurman, W. E. Walsh, and M. P. Wellman, "Flexible double auctions for electronic commerce: Theory and implementation," *Decision Support Systems*, vol. 24, no. 1, pp. 17–27, 1998.

[60] M. Khorasany, Y. Mishra, and G. Ledwich, "Design of auction-based approach for market clearing in peer-to-peer market platform," *The Journal of Engineering*, vol. 2019, no. 18, pp. 4813–4818, 2019.

[61] C. H. Leong, C. Gu, and F. Li, "Auction mechanism for P2P local energy trading considering physical constraints," *Energy Procedia*, vol. 158, pp. 6613–6618, 2019.

[62] P. Samadi, H. Mohsenian-Rad, R. Schober, and V. W. Wong, "Advanced demand side management for the future smart grid using mechanism design," *IEEE Transactions on Smart Grid*, vol. 3, no. 3, pp. 1170–1180, 2012.

[63] H. Leong, C. Gu, F. Li, and C. Gu, "ScienceDirect ScienceDirect ScienceDirect Auction Mechanism for P2P Local Energy Trading considering Auction Mechanism for P2P Local Energy Trading considering Physical Constraints Physical Constraints the feasibility of using the heat demand-outdoor Chou," *Energy Procedia*, vol. 158, pp. 6613–6618, 2019.

[64] "The min-max scaling method - Feature Engineering Made Easy."

[65] "sklearn.preprocessing.MinMaxScaler — scikit-learn 0.22.2 documentation."

[66] H. Huang, S. Nie, J. Lin, Y. Wang, and J. Dong, "Optimization of Peer-to-Peer Power Trading in a Microgrid with Distributed PV and Battery Energy Storage Systems," 2020.

[67] X. Yan, Y. Ozturk, Z. Hu, and Y. Song, "A review on price-driven residential demand response," *Renewable and Sustainable Energy Reviews*, vol. 96, no. August, pp. 411–419, 2018.

[68] I. Kalysh, A. Alimkhan, I. Temirtayev, H. S. Kumar Nunna, S. Doolla, and K. Vipin, "Dynamic programming based peer-to-peer energy trading framework for smart microgrids," *Proceedings - 2019 IEEE 13th International Conference on Compatibility, Power Electronics and Power Engineering, CPE-POWERENG 2019*, pp. 0–5, 2019.

[69] H. Mohsenian-rad, T. Saha, H. V. Poor, and K. L. Wood, "Networks via Peer-to-Peer," *IEEE Signal Processing Magazine*, vol. 35, no. July, pp. 90–111, 2018.

[70] A. Paudel, K. Chaudhari, C. Long, and H. B. Gooi, "Peer-to-peer energy trading in a prosumer-based community microgrid: A game-theoretic model," *IEEE Transactions on Industrial Electronics*, vol. 66, no. 8, pp. 6087–6097, 2019.

[71] J. Abdella and K. Shuaib, "Peer to peer distributed energy trading in smart grids: A survey," *Energies*, vol. 11, no. 6, 2018.

[72] Y. Wu, X. Sun, X. Tan, L. Meng, L. Yu, W. Z. Song, and D. H. K. Tsang, "Cooperative distributed energy generation and energy trading for future smart grid," *Proceedings of the 33rd Chinese Control Conference, CCC 2014*, pp. 8150–8157, 2014.

[73] N. Immorlica and D. R. Karger, "First-Price Procurement Auctions," *Small*.

[74] V. Krishna, *Auction Theory*. No. October, 2003.

[75] B. Snyder, "Exploring Auction Models for Peer-to-Peer Lending | Stanford Graduate School of Business," 2011.

[76] Y. Zhou, J. Wu, and C. Long, "Evaluation of peer-to-peer energy sharing mechanisms based on a multiagent simulation framework," *Applied Energy*, vol. 222, pp. 993–1022, jul 2018.

[77] S. Kuruseelan and C. Vaithilingam, "Peer-to-peer energy trading of a community connected with an AC and DC microgrid," *Energies*, vol. 12, no. 19, 2019.

[78] M. Dianati, X. Shen, and S. Naik, "A new fairness index for radio resource allocation in wireless networks," *IEEE Wireless Communications and Networking Conference, WCNC*, vol. 2, pp. 712–717, 2005.

[79] "Fairness measure - Wikipedia."

[80] T. Hobfeld, L. Skorin-Kapov, P. E. Heegaard, and M. Varela, "Definition of QoE Fairness in Shared Systems," *IEEE Communications Letters*, vol. 21, pp. 184–187, jan 2017.

[81] F. R. P. Cavalcanti, *Resource allocation and MIMO for 4G and beyond*, vol. 9781461480. 2013.

[82] C. K. Tham and T. Luo, "Fairness and social welfare in service allocation schemes for participatory sensing," *Computer Networks*, vol. 73, pp. 58–71, 2014.

[83] A. Ghosh, V. Aggarwal, and H. Wan, "Exchange of Renewable Energy among Prosumers using Blockchain with Dynamic Pricing," 2018.

[84] M. Zubeldía, A. Ferragut, and F. Paganini, "Neighbor selection for proportional fairness in P2P networks," *Computer Networks*, vol. 83, pp. 249–264, 2015.

[85] H. Shi, R. V. Prasad, E. Onur, and I. G. Niemegeers, "Fairness in wireless networks: Issues, measures and challenges," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 5–24, 2014.

[86] M. Andoni, V. Robu, D. Flynn, S. Abram, D. Geach, D. Jenkins, P. McCallum, and A. Peacock, "Blockchain technology in the energy sector: A systematic review of challenges and opportunities," *Renewable and Sustainable Energy Reviews*, vol. 100, no. February 2018, pp. 143–174, 2019.

[87] X. Luo, J. Wang, M. Dooner, and J. Clarke, "Overview of current development in electrical energy storage technologies and the application potential in power system operation," *Applied Energy*, vol. 137, pp. 511–536, 2015.

# APPENDIX : PROGRAM CODES

## MILP : Fixed Demand-Variable Pricing

```
#Importing libraries
import gurobipy as grb
from gurobipy import*
import pandas as pd
import numpy as np
import scipy
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sns
import sklearn
import random
import statsmodels.api as sm
from collections import OrderedDict
import collections, functools, operator
scipy.set_printoptions(precision = 4, suppress = True)
import matplotlib.pyplot as plt

price=[]
#seting up variable price model for each hour
#this calculates local market price for each hour
def price_model(load):
       peak_demand=[]
       from sklearn.preprocessing import MinMaxScaler
       # load data
       load=np.array(load)
       # creating scaler
       load=load.reshape(8,-1)
       scaler2 = MinMaxScaler(feature_range=(.104,.4604))
       scaler2.fit(load)
       \# applying transform
       normalized = scaler2.transform(load)
       normalized
       normalized_avg=sum(normalized)/8
       normalized_avg
       return(normalized_avg)

#Reading load data file
```

```
df=pd.read_csv('C:/Users/smipa/OneDrive/Documents/Scenario_Run/[3]_scenario_2_variable␣_rates/
    demand_data_input.csv')

#Setting up dataframe parameters for exporting output into a common csv /excel file after all
    iterations.
dx2=pd.DataFrame()
Hourly_total_transaction=pd.DataFrame()
col=['demand','buy␣from␣␣grid','buy␣locally','buy_charging_locally','buy_charging_from_grid','
    pv_sold_locally','pv_sold_to_grid',
'use_own_pv','use_own_battery','use_own_pv_charging','sell_battery_locally','sell_battery_to_grid',
'CHARGE_DECISION','DISCHARGE_DECISION','DECISION_TO_SELL','DECISION_TO_BUY','Battery␣Status␣after␣
    trading']
cx1=pd.DataFrame()
cx2=pd.DataFrame()
cx3=pd.DataFrame()
cx4=pd.DataFrame()
cx5=pd.DataFrame()
cx6=pd.DataFrame()
cx7=pd.DataFrame()
cx8=pd.DataFrame()

#Setting up group varaibles for optmization
Population=['C1','C2','C3','C4','C5','C6','C7','C8'] # all population
grpA=['C7','C8'] #Consumer (no PV or Battery)
grpB=['C1','C2'] #Only PV
grpC=['C3','C4','C5','C6'] #Battery+PV
grpAnB=['C7','C8','C1','C2']
grpBnC=['C1','C2','C3','C4','C5','C6']

#Prices
Pg=.4604 #grid price
Pt=.104 #price for selling to grid

#Setting constraint list ,Optmization model
#Also battery dictionary is set up to store battery status after optmization in each hour .
#the battery status is used as input in next iteration.
constraint=[]
opt_model= grb.Model(name="MIP␣Model")
Battery_status={(i):opt_model.addVars(("{0}".format(i) for i in grpC),vtype=grb.GRB.CONTINUOUS,lb
    =0,name="Bt_{0}".format(i)) for i in range(0,49) }
Battery_initial_status={'C3':20.5,'C4':22.5,'C5':15.8,'C6':21.5}
```

```

#Setting Battery initial status only for first iteration
for i in grpC:
       Battery_status[0][i]=Battery_initial_status[i]

capacity={'C3':2,'C4':2,'C5':1,'C6':2} #maximum charge and discharge rate possible from battery.
    kept it fixed for this program
Battery_Max={'C3':22.5,'C4':22.5,'C5':15.8,'C6':22.5} # Maximum Battery limit
Battery_Min={'C3':5,'C4':5,'C5':3,'C6':5} # Minimum Battery limit

#INITIATING PROGRAM LOOP TO OPTMIZE EACH HOUR
for q in range(0,48):
       Data=df.iloc[q] #READING ELEMENTS OF ROW NUMBER
       load =[Data[2],Data[3],Data[4],Data[5],Data[6],Data[7],Data[8],Data[9]]
       #Total Demand and PV specified
       total_demand=Data[2]+Data[3]+Data[4]+Data[5]+Data[6]+Data[7]+Data[8]+Data[9]
       total_pv=Data[10]+Data[11]+Data[12]+Data[13]+Data[14]+Data[15]
       #Calling fubction for price model for this iteration.
       Pl=price_model(load)
       #Setting demand and supply variables for use in optmization model
       P_demand ={'C1':Data[2],'C2':Data[3],'C3':Data[4],'C4':Data[5],'C5':Data[6],'C6':Data[7],'C7
           ':Data[8],'C8':Data[9]}
       grpA_demand={'C7':Data[8],'C8':Data[9]}
       grpB_demand={'C1':Data[2],'C2':Data[3]}
       grpC_demand={'C3':Data[4],'C4':Data[5],'C5':Data[6],'C6':Data[7]}
       demand_grpAnB={'C7':Data[8],'C8':Data[9],'C2':Data[2],'C3':Data[3]}
       demand_grpBnC={'C1':Data[2],'C2':Data[3],'C3':Data[4],'C4':Data[5],'C5':Data[6],'C6':Data
           [7]}
       grpB_supply={'C1':Data[10],'C2':Data[11]}
       grpC_supply={'C3':Data[12],'C4':Data[13],'C5':Data[14],'C6':Data[15]}
       supply_grpBnC={'C1':Data[10],'C2':Data[11],'C3':Data[12],'C4':Data[13],'C5':Data[14],'C6':
           Data[15]}

       #SETTING DECSION VARIABLES FOR ALLOCATION INTO EACH GROUP
       #BINARY VARIABLES ARE ALLOTTED 0 or 1 by SOLVER BASED ON DECISION
       buy_from_grid={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="buy_from_grid_{0}".
           format(i)) for i in Population}
       buy_locally={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="buy_locally_{0}".
           format(i)) for i in Population}
       pv_sold_locally={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="pv_sold_locally_
           {0}".format(i)) for i in grpBnC}
    pv_sold_to_grid={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="pv_sold_to_grid_{0}".format(i)) for i in grpBnC}
    use_own_pv={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="use_own_pv_{0}".format(i)) for i in grpBnC }
    use_own_battery={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="use_own_battery_{0}".format(i)) for i in grpC }
    buy_charging_locally={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="buy_charging_locally_{0}".format(i)) for i in grpC }
    buy_charging_from_grid={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="buy_charging_from_grid_{0}".format(i)) for i in grpC }
    sell_battery_locally={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="sell_local_locally_{0}".format(i)) for i in grpC }
    sell_battery_to_grid={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="sell_battery_to_grid_{0}".format(i)) for i in grpC }
    use_own_pv_charging={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="use_own_pv_charging_{0}".format(i)) for i in grpC }
    CHARGE_DECISION={(i):opt_model.addVar(vtype=grb.GRB.BINARY,name="CHARGE_DECISION_{0}".format(i)) for i in grpC }
    DISCHARGE_DECISION={(i):opt_model.addVar(vtype=grb.GRB.BINARY,name="DISCHARGE_DECISION_{0}".format(i)) for i in grpC }
    DECISION_TO_SELL={(i):opt_model.addVar(vtype=grb.GRB.BINARY,name="DECISION_TO_SELL_{0}".format(i)) for i in grpBnC }
    DECISION_TO_BUY={(i):opt_model.addVar(vtype=grb.GRB.BINARY,name="DECISION_TO_BUY_{0}".format(i)) for i in Population }

    #CONSTRAINTS FOR GROUP _A (ONLY CONSUMER)
    for i in grpA:
        constraint={(i):opt_model.addConstr(lhs=(grpA_demand[i]),
        sense=grb.GRB.EQUAL, rhs=(buy_locally[i]+buy_from_grid[i] ) , name="constraint_{0}".format(i))}

        constraint={(i):opt_model.addConstr(lhs=(DECISION_TO_BUY[i]),
        sense=grb.GRB.EQUAL, rhs=(1) , name="constraint_{0}".format(i))}

    #CONSTRAINTS FOR GROUP_B (PV ONLY)
    for i in grpB:
        constraint={(i):opt_model.addConstr(lhs=(grpB_demand[i]),
        sense=grb.GRB.EQUAL, rhs=(use_own_pv[i]+buy_from_grid[i]+ buy_locally[i] ) , name="constraint_{0}".format(i))}

        constraint={(i):opt_model.addConstr(lhs=(DECISION_TO_SELL[i] +DECISION_TO_BUY[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=(1) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(use_own_pv[i]+pv_sold_locally[i]+
                pv_sold_to_grid[i]),
            sense=grb.GRB.EQUAL, rhs=(grpB_supply[i]) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(pv_sold_locally[i]+pv_sold_to_grid[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=(grpB_supply[i]*(DECISION_TO_SELL[i])) , name="
                constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(buy_from_grid[i]+ buy_locally[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=(grpB_demand[i]*(DECISION_TO_BUY[i])) , name="
                constraint_{0}".format(i))}

      #CONSTRAINTS FOR GROUP C (PV+BATTERY)
      for i in grpC:
            constraint={(i):opt_model.addConstr(lhs=(grpC_demand[i]),
            sense=grb.GRB.EQUAL, rhs=(use_own_pv[i]+use_own_battery[i]+buy_locally[i]+
                buy_from_grid[i]) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(CHARGE_DECISION[i]+DISCHARGE_DECISION[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=(1) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(DECISION_TO_SELL[i] +DECISION_TO_BUY[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=(1) , name="constraint_{0}".format(i))}

            \#SETTING DECISIONS FOR SELL AND BUY TO VARIABLES:
            constraint={(i):opt_model.addConstr(lhs=(use_own_pv[i]+pv_sold_locally[i]+
                pv_sold_to_grid[i]+use_own_pv_charging[i]),
            sense=grb.GRB.EQUAL, rhs=(grpC_supply[i]) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(pv_sold_to_grid[i]+pv_sold_locally[i]+
                sell_battery_locally[i]+sell_battery_to_grid[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=((capacity[i]+grpC_supply[i])*DECISION_TO_SELL[i]) ,
                name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(buy_locally[i]+buy_from_grid[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=((grpC_demand[i])*(DECISION_TO_BUY[i]) ), name="
                constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(buy_charging_from_grid[i]+
                buy_charging_locally[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=(capacity[i]*DECISION_TO_BUY[i]) , name="constraint_{0}
                ".format(i))}

            #SETTING CHARGE AND DISCHARGE DECSIONS TO VARIABLES
            constraint={(i):opt_model.addConstr(lhs=(sell_battery_to_grid[i]+sell_battery_locally
                [i]+use_own_battery[i]),
            sense=grb.GRB.EQUAL, rhs=(capacity[i]*(DISCHARGE_DECISION[i]) ), name="constraint_{0}
                ".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(buy_charging_from_grid[i]+
                buy_charging_locally[i]+use_own_pv_charging[i]),
            sense=grb.GRB.EQUAL, rhs=(capacity[i]*(CHARGE_DECISION[i])) , name="constraint_{0}".
                format(i))}

            \#SETTING BATTERY MAXIMUM AND MINIMUM LIMITS
            constraint={(i):opt_model.addConstr(lhs=(Battery_status[q+1][i]),
            sense=grb.GRB.LESS_EQUAL, rhs=(Battery_Max[i]) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(Battery_status[q+1][i]),
            sense=grb.GRB.GREATER_EQUAL, rhs=(Battery_Min[i]) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(Battery_status[q+1][i]),
            sense=grb.GRB.EQUAL, rhs=((Battery_status[q][i] )+(buy_charging_locally[i]+
                buy_charging_from_grid[i]+use_own_pv_charging[i])-(sell_battery_locally[i]+
                sell_battery_to_grid[i]+use_own_battery[i])) , name="constraint_{0}".format(i))}

    #COMMON CONSTRAINTS FOR ALL GROUPS
    constraint={opt_model.addConstr(lhs=(grb.quicksum(buy_locally[i] for i in Population)+grb.
        quicksum(buy_charging_locally[i] for i in grpC)),
    sense=grb.GRB.EQUAL, rhs=(grb.quicksum(pv_sold_locally[i] for i in grpBnC)+grb.quicksum(
        sell_battery_locally[i] for i in grpC)) , name="constraint_{0}".format(i))}

    constraint={opt_model.addConstr(lhs=(total_demand),
    sense=grb.GRB.EQUAL, rhs=(grb.quicksum(buy_locally[i] for i in Population)+grb.quicksum(
        buy_from_grid[i] for i in Population)+grb.quicksum(use_own_pv[i] for i in grpBnC)+grb.
        quicksum(use_own_battery[i] for i in grpC) ), name="constraint_{0}".format(i))}

    constraint={opt_model.addConstr(lhs=(total_pv),
        sense=grb.GRB.EQUAL, rhs=(grb.quicksum(pv_sold_locally[i] for i in grpBnC)+grb.quicksum(
            pv_sold_to_grid[i] for i in grpBnC)+grb.quicksum(use_own_pv[i] for i in grpBnC)+grb.
            quicksum(use_own_pv_charging[i] for i in grpC)) , name="constraint_{0}".format(i))}

        #SETTING OBJECTIVE FUNCTION
        objective = grb.quicksum(Pg*buy_from_grid[i] for i in Population)+grb.quicksum(Pg*
            buy_charging_from_grid[i] for i in grpC)

        #SETTING OBJECTIVE
        opt_model.ModelSense = grb.GRB.MINIMIZE
        opt_model.optimize()
        status = opt_model.status

        # STANDARD OUTPUT DISPLAY
        print('Date␣and␣time' ,Data[0],':',Data[1],'\n\n')
        print('BUY␣FROM␣GRID␣TO␣USE:','\n\n',buy_from_grid,'\n\nBUY␣LOCAL␣FOR␣USE:','\n\n',
            buy_locally,'\n\n')
        print('BUY␣LOCAL␣CHARGE:','\n\n',buy_charging_locally,'\n\nBUY␣GRID␣CHARGE:','\n\n',
            buy_charging_from_grid,'\n\n')
        print('SELL␣PV␣TO␣GRID␣','\n\n',pv_sold_to_grid,'\n\nSELL␣PV␣LOCALLY␣:','\n\n',
            pv_sold_locally,'\n\n')
        print('USE␣OWN␣PV␣␣','\n\n',use_own_pv,'\n\n')
        print('USE␣BATTERY:','\n\n', use_own_battery,'\n\n␣USE␣PV␣CHARGE␣BATTERY:','\n\n',
            use_own_pv_charging,'\n\n')
        print('SELL␣BATTERY␣LOCALLY:','\n\n', sell_battery_locally,'\n\n␣SELL␣BATTERY␣TO␣GRID:','\n\
            n',sell_battery_to_grid,'\n\n')
        print('CHARGE␣DECSION:','\n\n', CHARGE_DECISION,'\n␣\nDISCHARGE␣DECSION','\n\n',
            DISCHARGE_DECISION,'\n\n')
        print('SELL␣DECISION:','\n\n', DECISION_TO_SELL,'\n␣\nBUY␣DECISION','\n\n',DECISION_TO_BUY,'
            \n\n')
        for i in grpC:
              print('BATTERY␣STATUS:',Battery_status[q+1][i])
              print('LOCAL␣PRICE:', Pl)
        # Setting variables for creating dataframe for output
        load=[Data[2],Data[3],Data[4],Data[5],Data[6],Data[7],Data[8],Data[9]]
        local_P=Pl
        space=[]*8

        # all decision variables converted to list
        m1=[buy_from_grid[a].x for a in Population]
        m2=[buy_locally[a].x for a in Population]
        m3=[buy_charging_locally[a].x for a in grpC]
        m4=[buy_charging_from_grid[a].x for a in grpC]
        m5=[pv_sold_locally[a].x for a in grpBnC]
        m6=[pv_sold_to_grid[a].x for a in grpBnC]
        m7=[use_own_pv[a].x for a in grpBnC]
        m8=[use_own_battery[a].x for a in grpC]
        m9=[use_own_pv_charging[a].x for a in grpC]
        m10=[sell_battery_locally[a].x for a in grpC]
        m11=[sell_battery_to_grid[a].x for a in grpC]
        m12=[CHARGE_DECISION[a].x for a in grpC]
        m13=[DISCHARGE_DECISION[a].x for a in grpC]
        m14=[DECISION_TO_SELL[a].x for a in grpBnC]
        m15=[DECISION_TO_BUY[a].x for a in Population]
        m16=[Battery_status[q+1][i].x for i in grpC]
        z=[0.0]

        # converting unequal rows to equal rows of grp C and grpBnC by putting zero in the missing
             location (size 8 for 8 households)
        for a in range(0,2):
               m3.extend(z)
               m3.insert(0,0.0)
               m4.extend(z)
               m4.insert(0,0.0)
               m8.extend(z)
               m8.insert(0,0.0)
               m9.extend(z)
               m9.insert(0,0.0)
               m10.extend(z)
               m10.insert(0,0.0)
               m11.extend(z)
               m11.insert(0,0.0)
               m12.extend(z)
               m12.insert(0,0.0)
               m13.extend(z)
               m13.insert(0,0.0)
               m16.extend(z)
               m16.insert(0,0.0)
               m5.extend(z)
               m6.extend(z)
               m7.extend(z)
               m14.extend(z)
```

```
#creating columns and index
    columns =['c1','c2','c3','c4','c5','c6','c7','c8']
    index = ['demand','buy␣from␣grid','buy␣locally','buy_charging_locally','
        buy_charging_from_grid','pv_sold_locally','pv_sold_to_grid',
    'use_own_pv','use_own_battery','use_own_pv_charging','sell_battery_locally','
        sell_battery_to_grid',
    'CHARGE_DECISION','DISCHARGE_DECISION','DECISION_TO_SELL','DECISION_TO_BUY','Battery␣Status␣
        after␣trading','Local␣Price','']
    #Combining lists in to a bigger list
    L=[load,m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,m15,m16,local_P,space]
    #creating dataframe for printing transactions in each hour.
    dx1=pd.DataFrame(L, columns = ['c1','c2','c3','c4','c5','c6','c7','c8'],index=index)

    #Creating another dataframe for calculating all totals for each iteration
    hourly_cumulative=pd.DataFrame()
    row_grid_buy=dx1.loc[["buy␣from␣grid","buy_charging_from_grid",]]
    row_grid_sell=dx1.loc[["pv_sold_to_grid","sell_battery_to_grid",]]
    row_buy_local= dx1.loc[["buy␣locally","buy_charging_locally"]]
    row_sell_local= dx1.loc[["pv_sold_locally","sell_battery_locally"]]
    row_use_pv=dx1.loc[["use_own_pv","use_own_pv_charging"]]
    row_use_battery=dx1.loc[["use_own_battery"]]
    dx2=dx2.append(dx1)
    self_gridbuy_total= row_grid_buy.sum(axis=1)
    self_localbuy_total= row_buy_local.sum(axis=1)
    self_gridsell_total=row_grid_sell.sum(axis=1)
    self_localsell_total= row_sell_local.sum(axis=1)
    use_pvtotal= row_use_pv.sum(axis=1)
    use_battery_total=row_use_battery.sum(axis=1)
    hourly_cumulative['Total␣demand']=[total_demand]
    hourly_cumulative['Total␣PV']=[total_pv]
    hourly_cumulative['Grid␣buy␣total']=[self_gridbuy_total.sum(axis=0)]
    hourly_cumulative['Local␣Buy␣total']=[self_localbuy_total.sum(axis=0)]
    hourly_cumulative['Grid␣sell␣total']=[self_gridsell_total.sum(axis=0)]
    hourly_cumulative['Local␣sell␣total']=[self_localsell_total.sum(axis=0)]
    hourly_cumulative['Use␣PV␣Total␣']=[use_pvtotal.sum(axis=0)]
    hourly_cumulative['Use␣Battery␣Total']=[ use_battery_total.sum(axis=0)]
    hourly_cumulative['Total␣Purchase␣costs␣']=(self_gridbuy_total.sum(axis=0)*.4604)+(
        self_localbuy_total.sum(axis=0)*local_P)
    hourly_cumulative['Total␣sales␣revenue']=(self_gridsell_total.sum(axis=0)*.104)+(
        self_localsell_total.sum(axis=0)*local_P)
        hourly_cumulative['Net␣Purchase␣costs␣after␣sales␣']=hourly_cumulative['Total␣Purchase␣costs␣'].values-hourly_cumulative['Total␣sales␣revenue'].values
        hourly_cumulative['Local␣Price␣']=Pl
        Hourly_total_transaction=Hourly_total_transaction.append(hourly_cumulative)

        \#Creating dataframe for summing the all iterations of each Household and a separate sum of all household transactions.
        for i in range(0,8):
                H=[load[i],m1[i],m2[i],m3[i],m4[i],m5[i],m6[i],m7[i],m8[i],m9[i],m10[i],m11[i],m12[i],m13[i],m14[i],m15[i],m16[i]]
                H=np.transpose(H)
                H=[H]
                if i==0:
                        cx1 = cx1.append(H)
                elif i==1:
                        cx2=cx2.append(H)
                elif i==2:
                        cx3=cx3.append(H)
                elif i==3:
                        cx4=cx4.append(H)
                elif i==4:
                        cx5=cx5.append(H)
                elif i==5:
                        cx6=cx6.append(H)
                elif i==6:
                        cx7=cx7.append(H)
                elif i==7:
                        cx8=cx8.append(H)
Cumulative=pd.DataFrame()
Cumulative=Cumulative.append(cx1.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx2.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx3.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx4.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx5.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx6.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx7.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx8.sum(axis=0),ignore_index=True)
Cumulative.columns=col
hours=pd.Series(range(0,48))
cx1.columns=col
cx1.index=hours
```

```
cx1.index.name='Hours'
cx2.columns=col
cx2.index=hours
cx2.index.name='Hours'
cx3.columns=col
cx3.index=hours
cx3.index.name='Hours'
cx4.columns=col
cx4.index=hours
cx4.index.name='Hours'
cx5.columns=col
cx5.index=hours
cx5.index.name='Hours'
cx6.columns=col
cx6.index=hours
cx6.index.name='Hours'
cx7.columns=col
cx7.index=hours
cx7.index.name='Hours'
cx8.columns=col
cx8.index=hours
cx8.index.name='Hours'
Cumulative.index=[Population]
Cumulative.index.name='Household'
\#converting to csv /excel
excelpath = 'C:/Users/smipa/OneDrive/Desktop/net_household.xlsx'
\# Write dataframes to different sheets
\# cx output is for transaction for each household ion the given hours row wise from sheet 1 to 8
\#sheet 9 sums the transaction of each house in all hours and presents them together in sheet 9 .

with pd.ExcelWriter(excelpath) as transaction:
        cx1.to_excel(transaction,sheet_name='Sheet1')
        cx2.to_excel(transaction,sheet_name='Sheet2')
        cx3.to_excel(transaction,sheet_name='Sheet3')
        cx4.to_excel(transaction,sheet_name='Sheet4')
        cx5.to_excel(transaction,sheet_name='Sheet5')
        cx6.to_excel(transaction,sheet_name='Sheet6')
        cx7.to_excel(transaction,sheet_name='Sheet7')
        cx8.to_excel(transaction,sheet_name='Sheet8')
        Cumulative.to_excel(transaction,sheet_name='Sheet9')
dx2.to_csv('C:/Users/smipa/OneDrive/Desktop/dx2.csv')
```

```
Hourly_total_transaction.index=hours
Hourly_total_transaction.index.name='Hours'
Net_Trading=Hourly_total_transaction.sum(axis=0)
Net_Trading.name='Total'
Hourly_total_transaction=Hourly_total_transaction.append(Net_Trading)
Hourly_total_transaction.to_csv('C:/Users/smipa/OneDrive/Desktop/Hourly_total_transaction.csv')

#dx2=csv file constains output allocation of hourly transactions
#net househod has 9 sheets that constains transactions of each household separately in each sheet
    their totals in sheet 9.#total transactons has all houshold (buy , sell , use records telling
    total local and grid trading penetation for all houses combined)
```

MILP : Fixed Demand-Variable Pricing (Only PV)

```
#Importing libraries
import gurobipy as grb
from gurobipy import*
import pandas as pd
import numpy as np
import scipy
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sns
import sklearn
import random
import statsmodels.api as sm
from collections import OrderedDict
import collections, functools, operator
scipy.set_printoptions(precision = 4, suppress = True)
import matplotlib.pyplot as plt

price=[]
#seting up variable price model for each hour
#this calculates local market price for each hour
def price_model(load):
       peak_demand=[]
       from sklearn.preprocessing import MinMaxScaler
       load=np.array(load)
       # creating scaler
       load=load.reshape(8,-1)
       scaler2 = MinMaxScaler(feature_range=(.104,.4604))
       scaler2.fit(load)
       # applying transform
       normalized = scaler2.transform(load)
       normalized
       normalized_avg=sum(normalized)/8
       normalized_avg
       return(normalized_avg)

#Reading load data file
df=pd.read_csv('C:/Users/smipa/OneDrive/Documents/Scenario_Run/[3]_scenario_2_variable␣_rates/
     demand_data_input.csv')
```

```
#Setting up dataframe parameters for exporting output into a common csv /excel file after all
    iterations.
dx2=pd.DataFrame()
Hourly_total_transaction=pd.DataFrame()
col=['demand','buy from  grid','buy locally','pv_sold_locally','pv_sold_to_grid',
'use_own_pv','use_own_battery','use_own_pv_charging','sell_battery_locally','sell_battery_to_grid',
'CHARGE_DECISION','DISCHARGE_DECISION','DECISION_TO_SELL','DECISION_TO_BUY','Battery Status after 
    trading']
cx1=pd.DataFrame()
cx2=pd.DataFrame()
cx3=pd.DataFrame()
cx4=pd.DataFrame()
cx5=pd.DataFrame()
cx6=pd.DataFrame()
cx7=pd.DataFrame()
cx8=pd.DataFrame()

#Setting up varaible for optmization
Population=['C1','C2','C3','C4','C5','C6','C7','C8'] # all population
grpA=['C7','C8'] #Consumer (no PV or Battery)
grpB=['C1','C2'] #Only PV
grpC=['C3','C4','C5','C6'] #Battery+PV
grpAnB=['C7','C8','C1','C2']
grpBnC=['C1','C2','C3','C4','C5','C6']

#Prices
Pg=.4604 #grid price
Pt=.104 #price for selling to grid

#Setting constraint list ,Optmization model
#Also battery dictionary is set up to store battery status after optmization in each hour .
#the battery status is used as input in next iteration.
constraint=[]
opt_model= grb.Model(name="MIP Model")
Battery_status={(i):opt_model.addVars(("{0}".format(i) for i in grpC),vtype=grb.GRB.CONTINUOUS,lb
    =0,name="Bt_{0}".format(i)) for i in range(0,49) }
Battery_initial_status={'C3':20.5,'C4':22.5,'C5':15.8,'C6':21.5}

#Setting Battery initial status only for first iteration
for i in grpC:
        Battery_status[0][i]=Battery_initial_status[i]
```

```

Battery_Max={'C3':22.5,'C4':22.5,'C5':15.8,'C6':22.5} # Maximum Battery limit
Battery_Min={'C3':5,'C4':5,'C5':3,'C6':5} # Minimum Battery limit
char_c={'C3':2,'C4':2,'C5':1,'C6':2}
def cap(cd3,cd4,cd5,cd6,cs3,cs4,cs5,cs6):
        if cs3-cd3>0 and cs3-cd3<2:
                cap3=cs3-cd3
        elif cs3-cd3>0 and cs3-cd3>=2:
                cap3=2
        else:
                cap3=0
        if cs4-cd4 and cs4-cd4<2:
                cap4=cs4-cd4
        elif cs4-cd4>0 and cs4-cd4>=2:
                cap4=2
        else:
                cap4=0
        if cs5-cd5>0 and cs5-cd5<1:
                cap5=cs5-cd5
        elif cs5-cd5>0 and cs5-cd5>=1:
                cap5=1
        else:
                cap5=0
        if cs6-cd6>0 and cs6-cd6<2:
                cap6=cs6-cd6
        elif cs6-cd6>0 and cs6-cd6>=2:
                cap6=2
        else:
                cap6=0
        return cap3,cap4,cap5,cap6

#INITIATING FOR LOOP TO OPTMIZE EACH HOUR
for q in range(0,48):
        Data=df.iloc[q] #READING ELEMENTS OF ROW NUMBER
        load =[Data[2],Data[3],Data[4],Data[5],Data[6],Data[7],Data[8],Data[9]]
        #Total Demand and PV specified
        total_demand=Data[2]+Data[3]+Data[4]+Data[5]+Data[6]+Data[7]+Data[8]+Data[9]
        total_pv=Data[10]+Data[11]+Data[12]+Data[13]+Data[14]+Data[15]
        cap3,cap4,cap5,cap6=cap(Data[4],Data[5],Data[6],Data[7],Data[12],Data[13],Data[14],Data[15])
        capacity={'C3':cap3,'C4':cap4,'C5':cap5,'C6':cap6}
        #Calling function for price calculation
```

```
    Pl=price_model(load)

    #Setting demand and supply variables for use in optmization model
    P_demand ={'C1':Data[2],'C2':Data[3],'C3':Data[4],'C4':Data[5],'C5':Data[6],'C6':Data[7],'C7
        ':Data[8],'C8':Data[9]}
    grpA_demand={'C7':Data[8],'C8':Data[9]}
    grpB_demand={'C1':Data[2],'C2':Data[3]}
    grpC_demand={'C3':Data[4],'C4':Data[5],'C5':Data[6],'C6':Data[7]}
    demand_grpAnB={'C7':Data[8],'C8':Data[9],'C2':Data[2],'C3':Data[3]}
    demand_grpBnC={'C1':Data[2],'C2':Data[3],'C3':Data[4],'C4':Data[5],'C5':Data[6],'C6':Data
        [7]}
    grpB_supply={'C1':Data[10],'C2':Data[11]}
    grpC_supply={'C3':Data[12],'C4':Data[13],'C5':Data[14],'C6':Data[15]}
    supply_grpBnC={'C1':Data[10],'C2':Data[11],'C3':Data[12],'C4':Data[13],'C5':Data[14],'C6':
        Data[15]}

    #SETTING DECSION VARIABLES FOR ALLOCATION INTO EACH GROUP
    #BINARY VARIABLES ARE ALLOTTED 0 or 1 by SOLVER BASED ON DECISION
    buy_from_grid={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="buy_from_grid_{0}".
        format(i)) for i in Population}
    buy_locally={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="buy_locally_{0}".
        format(i)) for i in Population}
    pv_sold_locally={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="pv_sold_locally_
        {0}".format(i)) for i in grpBnC}
    pv_sold_to_grid={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="pv_sold_to_grid_
        {0}".format(i)) for i in grpBnC}
    use_own_pv={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="use_own_pv_{0}".format(
        i)) for i in grpBnC }
    use_own_battery={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="use_own_battery_
        {0}".format(i)) for i in grpC }
    #buy_charging_locally={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="
        buy_charging_locally_{0}".format(i)) for i in grpC }
    #buy_charging_from_grid={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="
        buy_charging_from_grid_{0}".format(i)) for i in grpC }
    sell_battery_locally={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="
        sell_local_locally_{0}".format(i)) for i in grpC }
    sell_battery_to_grid={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="
        sell_battery_to_grid_{0}".format(i)) for i in grpC }
    use_own_pv_charging={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="
        use_own_pv_charging_{0}".format(i)) for i in grpC }
    CHARGE_DECISION={(i):opt_model.addVar(vtype=grb.GRB.BINARY,name="CHARGE_DECISION_{0}".format
        (i)) for i in grpC }
    DISCHARGE_DECISION={(i):opt_model.addVar(vtype=grb.GRB.BINARY,name="DISCHARGE_DECISION_{0}".
        format(i)) for i in grpC }
    DECISION_TO_SELL={(i):opt_model.addVar(vtype=grb.GRB.BINARY,name="DECISION_TO_SELL_{0}".
        format(i)) for i in grpBnC }
    DECISION_TO_BUY={(i):opt_model.addVar(vtype=grb.GRB.BINARY,name="DECISION_TO_BUY_{0}".format
        (i)) for i in Population }

    #CONSTRAINTS FOR GROUP _A (ONLY CONSUMER)
    for i in grpA:
        constraint={(i):opt_model.addConstr(lhs=(grpA_demand[i]),
        sense=grb.GRB.EQUAL, rhs=(buy_locally[i]+buy_from_grid[i] ) , name="constraint_{0}".
            format(i))}

        constraint={(i):opt_model.addConstr(lhs=(DECISION_TO_BUY[i]),
        sense=grb.GRB.EQUAL, rhs=(1) , name="constraint_{0}".format(i))}

    #CONSTRAINTS FOR GROUP_B (PV ONLY)
    for i in grpB:
        constraint={(i):opt_model.addConstr(lhs=(grpB_demand[i]),
        sense=grb.GRB.EQUAL, rhs=(use_own_pv[i]+buy_from_grid[i]+ buy_locally[i] ) , name="
            constraint_{0}".format(i))}

        constraint={(i):opt_model.addConstr(lhs=(DECISION_TO_SELL[i] +DECISION_TO_BUY[i]),
        sense=grb.GRB.LESS_EQUAL, rhs=(1) , name="constraint_{0}".format(i))}

        constraint={(i):opt_model.addConstr(lhs=(use_own_pv[i]+pv_sold_locally[i]+
            pv_sold_to_grid[i]),
        sense=grb.GRB.EQUAL, rhs=(grpB_supply[i]) , name="constraint_{0}".format(i))}

        constraint={(i):opt_model.addConstr(lhs=(pv_sold_locally[i]+pv_sold_to_grid[i]),
        sense=grb.GRB.LESS_EQUAL, rhs=(grpB_supply[i]*(DECISION_TO_SELL[i])) , name="
            constraint_{0}".format(i))}

        constraint={(i):opt_model.addConstr(lhs=(buy_from_grid[i]+ buy_locally[i]),
        sense=grb.GRB.LESS_EQUAL, rhs=(grpB_demand[i]*(DECISION_TO_BUY[i])) , name="
            constraint_{0}".format(i))}

    #CONSTRAINTS FOR GROUP C (PV+BATTERY)
    for i in grpC:
            constraint={(i):opt_model.addConstr(lhs=(grpC_demand[i]),
            sense=grb.GRB.EQUAL, rhs=(use_own_pv[i]+use_own_battery[i]+buy_locally[i]+
                buy_from_grid[i]) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(CHARGE_DECISION[i]+DISCHARGE_DECISION[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=(1) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(DECISION_TO_SELL[i] +DECISION_TO_BUY[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=(1) , name="constraint_{0}".format(i))}

            #SETTING DECISIONS FOR SELL AND BUY TO VARIABLES:
            constraint={(i):opt_model.addConstr(lhs=(use_own_pv[i]+pv_sold_locally[i]+
                pv_sold_to_grid[i]+use_own_pv_charging[i]),
            sense=grb.GRB.EQUAL, rhs=(grpC_supply[i]) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(pv_sold_to_grid[i]+pv_sold_locally[i]+
                sell_battery_locally[i]+sell_battery_to_grid[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=((char_c[i]+grpC_supply[i])*DECISION_TO_SELL[i]) , name
                ="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(buy_locally[i]+buy_from_grid[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=((grpC_demand[i])*(DECISION_TO_BUY[i]) ), name="
                constraint_{0}".format(i))}

            #SETTING CHARGE AND DISCHARGE DECSIONS TO VARIABLES
            constraint={(i):opt_model.addConstr(lhs=(sell_battery_to_grid[i]+sell_battery_locally
                [i]+use_own_battery[i]),
            sense=grb.GRB.EQUAL, rhs=(char_c[i]*(DISCHARGE_DECISION[i]) ), name="constraint_{0}".
                format(i))}

            constraint={(i):opt_model.addConstr(lhs=(use_own_pv_charging[i]),
            sense=grb.GRB.EQUAL, rhs=(capacity[i]*(CHARGE_DECISION[i])) , name="constraint_{0}".
                format(i))}

            #SETTING BATTERY MAXIMUM AND MINIMUM LIMITS

            constraint={(i):opt_model.addConstr(lhs=(Battery_status[q+1][i]),
            sense=grb.GRB.LESS_EQUAL, rhs=(Battery_Max[i]) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(Battery_status[q+1][i]),
            sense=grb.GRB.GREATER_EQUAL, rhs=(Battery_Min[i]) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(Battery_status[q+1][i]),
            sense=grb.GRB.EQUAL, rhs=((Battery_status[q][i] )+use_own_pv_charging[i]-(
                sell_battery_locally[i]+sell_battery_to_grid[i]+use_own_battery[i])) , name="
                constraint_{0}".format(i))}

        #COMMON CONSTRAINTS FOR ALL GROUPS
        constraint={opt_model.addConstr(lhs=grb.quicksum(buy_locally[i] for i in Population),
        sense=grb.GRB.EQUAL, rhs=(grb.quicksum(pv_sold_locally[i] for i in grpBnC)+grb.quicksum(
            sell_battery_locally[i] for i in grpC)) , name="constraint_{0}".format(i))}

        constraint={opt_model.addConstr(lhs=(total_demand),
        sense=grb.GRB.EQUAL, rhs=(grb.quicksum(buy_locally[i] for i in Population)+grb.quicksum(
            buy_from_grid[i] for i in Population)+grb.quicksum(use_own_pv[i] for i in grpBnC)+grb.
            quicksum(use_own_battery[i] for i in grpC) ), name="constraint_{0}".format(i))}

        constraint={opt_model.addConstr(lhs=(total_pv),
        sense=grb.GRB.EQUAL, rhs=(grb.quicksum(pv_sold_locally[i] for i in grpBnC)+grb.quicksum(
            pv_sold_to_grid[i] for i in grpBnC)+grb.quicksum(use_own_pv[i] for i in grpBnC)+grb.
            quicksum(use_own_pv_charging[i] for i in grpC)) , name="constraint_{0}".format(i))}

        #SETTING OBJECTIVE FUNCTION
        objective = grb.quicksum(Pg*buy_from_grid[i] for i in Population)
        #SETTING OBJECTIVE
        opt_model.optimize()
        status = opt_model.status

        # STANDARD OUTPUT DISPLAY
        print('Date␣and␣time' ,Data[0],':',Data[1],'\n\n')
        print('BUY␣FROM␣GRID␣TO␣USE:','\n\n',buy_from_grid,'\n\nBUY␣LOCAL␣FOR␣USE:','\n\n',
            buy_locally,'\n\n')
        # print('BUY LOCAL CHARGE:','\n\n',buy_charging_locally,'\n\nBUY GRID CHARGE:','\n\n',
            buy_charging_from_grid,'\n\n')
        print('SELL␣PV␣TO␣GRID␣','\n\n',pv_sold_to_grid,'\n\nSELL␣PV␣LOCALLY␣:','\n\n',
            pv_sold_locally,'\n\n')
        print('USE␣OWN␣PV␣␣','\n\n',use_own_pv,'\n\n')
        print('USE␣BATTERY:','\n\n', use_own_battery,'\n\n␣USE␣PV␣CHARGE␣BATTERY:','\n\n',
            use_own_pv_charging,'\n\n')
        print('SELL␣BATTERY␣LOCALLY:','\n\n', sell_battery_locally,'\n\n␣SELL␣BATTERY␣TO␣GRID:','\n\
            n',sell_battery_to_grid,'\n\n')
        print('CHARGE␣DECSION:','\n\n', CHARGE_DECISION,'\n␣\nDISCHARGE␣DECSION','\n\n',
             DISCHARGE_DECISION,'\n\n')
        print('SELL␣DECISION:','\n\n', DECISION_TO_SELL,'\n␣\nBUY␣DECISION','\n\n',DECISION_TO_BUY,'
             \n\n')
        for i in grpC:
               print('BATTERY␣STATUS:',Battery_status[q+1][i])
        print('LOCAL␣PRICE:', Pl)

        # Setting variables for creating dataframe for output
        load=[Data[2],Data[3],Data[4],Data[5],Data[6],Data[7],Data[8],Data[9]]
        local_P=Pl
        space=[]*8

        # all decision variables converted to list
        m1=[buy_from_grid[a].x for a in Population]
        m2=[buy_locally[a].x for a in Population]
        m5=[pv_sold_locally[a].x for a in grpBnC]
        m6=[pv_sold_to_grid[a].x for a in grpBnC]
        m7=[use_own_pv[a].x for a in grpBnC]
        m8=[use_own_battery[a].x for a in grpC]
        m9=[use_own_pv_charging[a].x for a in grpC]
        m10=[sell_battery_locally[a].x for a in grpC]
        m11=[sell_battery_to_grid[a].x for a in grpC]
        m12=[CHARGE_DECISION[a].x for a in grpC]
        m13=[DISCHARGE_DECISION[a].x for a in grpC]
        m14=[DECISION_TO_SELL[a].x for a in grpBnC]
        m15=[DECISION_TO_BUY[a].x for a in Population]
        m16=[Battery_status[q+1][i].x for i in grpC]
        z=[0.0]

        # converting unequal rows to equal rows of grp C and grpBnC by putting zero in the missing
             location (size 8 for 8 households)
        for a in range(0,2):
        m8.extend(z)
        m8.insert(0,0.0)
        m9.extend(z)
        m9.insert(0,0.0)
        m10.extend(z)
        m10.insert(0,0.0)
        m11.extend(z)
        m11.insert(0,0.0)
        m12.extend(z)
        m12.insert(0,0.0)
        m13.extend(z)
        m13.insert(0,0.0)
        m16.extend(z)
        m16.insert(0,0.0)
        m5.extend(z)
        m6.extend(z)
        m7.extend(z)
        m14.extend(z)
        #creating columns and index
        columns =['c1','c2','c3','c4','c5','c6','c7','c8']
        index = ['demand','buy␣from␣grid','buy␣locally','pv_sold_locally','pv_sold_to_grid',
        'use_own_pv','use_own_battery','use_own_pv_charging','sell_battery_locally','
            sell_battery_to_grid',
        'CHARGE_DECISION','DISCHARGE_DECISION','DECISION_TO_SELL','DECISION_TO_BUY','Battery␣Status␣
            after␣trading','Local␣Price','']
        #Combining lists in to a bigger list
        L=[load,m1,m2,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,m15,m16,local_P,space]
        #creating dataframe for printing transactions in each hour.
        dx1=pd.DataFrame(L, columns = ['c1','c2','c3','c4','c5','c6','c7','c8'],index=index)

        #Creating another dataframe for calculating all totals for each iteration
        hourly_cumulative=pd.DataFrame()
        row_grid_buy=dx1.loc[["buy␣from␣grid"]]
        row_grid_sell=dx1.loc[["pv_sold_to_grid","sell_battery_to_grid",]]
        row_buy_local= dx1.loc[["buy␣locally"]]
        row_sell_local= dx1.loc[["pv_sold_locally","sell_battery_locally"]]
        row_use_pv=dx1.loc[["use_own_pv","use_own_pv_charging"]]
        row_use_battery=dx1.loc[["use_own_battery"]]
        dx2=dx2.append(dx1)
        self_gridbuy_total= row_grid_buy.sum(axis=1)
        self_localbuy_total= row_buy_local.sum(axis=1)
        self_gridsell_total=row_grid_sell.sum(axis=1)
        self_localsell_total= row_sell_local.sum(axis=1)
        use_pvtotal= row_use_pv.sum(axis=1)
        use_battery_total=row_use_battery.sum(axis=1)
        hourly_cumulative['Total␣demand']=[total_demand]
        hourly_cumulative['Total␣PV']=[total_pv]
        hourly_cumulative['Grid␣buy␣total']=[self_gridbuy_total.sum(axis=0)]
        hourly_cumulative['Local␣Buy␣total']=[self_localbuy_total.sum(axis=0)]
        hourly_cumulative['Grid␣sell␣total']=[self_gridsell_total.sum(axis=0)]
        hourly_cumulative['Local␣sell␣total']=[self_localsell_total.sum(axis=0)]
        hourly_cumulative['Use␣PV␣Total␣']=[use_pvtotal.sum(axis=0)]
        hourly_cumulative['Use␣Battery␣Total']=[ use_battery_total.sum(axis=0)]
        hourly_cumulative['Total␣Purchase␣costs␣']=(self_gridbuy_total.sum(axis=0)*.4604)+(
            self_localbuy_total.sum(axis=0)*local_P)
        hourly_cumulative['Total␣sales␣revenue']=(self_gridsell_total.sum(axis=0)*.104)+(
            self_localsell_total.sum(axis=0)*local_P)
        hourly_cumulative['Net␣Purchase␣costs␣after␣sales␣']=hourly_cumulative['Total␣Purchase␣costs
            ␣'].values-hourly_cumulative['Total␣sales␣revenue'].values
        hourly_cumulative['Local␣Price␣']=Pl
        Hourly_total_transaction=Hourly_total_transaction.append(hourly_cumulative)

        #Creatng dataframe for summing the all iterations of each Household and a separate sum of
            all household transactions.
        for i in range(0,8):
        H=[load[i],m1[i],m2[i],m5[i],m6[i],m7[i],m8[i],m9[i],m10[i],m11[i],m12[i],m13[i],m14[i],m15[
            i],m16[i]]
        H=np.transpose(H)
        H=[H]
        if i==0:
                cx1 = cx1.append(H)
        elif i==1:
                cx2=cx2.append(H)
        elif i==2:
                cx3=cx3.append(H)
        elif i==3:
                cx4=cx4.append(H)
        elif i==4:
                cx5=cx5.append(H)
        elif i==5:
                cx6=cx6.append(H)
        elif i==6:
                cx7=cx7.append(H)
        elif i==7:
                cx8=cx8.append(H)
Cumulative=pd.DataFrame()
Cumulative=Cumulative.append(cx1.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx2.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx3.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx4.sum(axis=0),ignore_index=True)
```

```
Cumulative=Cumulative.append(cx5.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx6.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx7.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx8.sum(axis=0),ignore_index=True)
Cumulative.columns=col
hours=pd.Series(range(0,48))
cx1.columns=col
cx1.index=hours
cx1.index.name='Hours'
cx2.columns=col
cx2.index=hours
cx2.index.name='Hours'
cx3.columns=col
cx3.index=hours
cx3.index.name='Hours'
cx4.columns=col
cx4.index=hours
cx4.index.name='Hours'
cx5.columns=col
cx5.index=hours
cx5.index.name='Hours'
cx6.columns=col
cx6.index=hours
cx6.index.name='Hours'
cx7.columns=col
cx7.index=hours
cx7.index.name='Hours'
cx8.columns=col
cx8.index=hours
cx8.index.name='Hours'
Cumulative.index=[Population]
Cumulative.index.name='Household'

#converting to csv /excel
excelpath = 'C:/Users/smipa/OneDrive/Desktop/net_household.xlsx'

# Write your dataframes to different sheets
# cx output is for transaction for each household ion the given hours row wise from sheet 1 to 8
#sheet 9 sums the transaction of each house in all hours and presents them together in sheet 9 .

with pd.ExcelWriter(excelpath) as transaction:
        cx1.to_excel(transaction,sheet_name='Sheet1')
        cx2.to_excel(transaction,sheet_name='Sheet2')
        cx3.to_excel(transaction,sheet_name='Sheet3')
        cx4.to_excel(transaction,sheet_name='Sheet4')
        cx5.to_excel(transaction,sheet_name='Sheet5')
        cx6.to_excel(transaction,sheet_name='Sheet6')
        cx7.to_excel(transaction,sheet_name='Sheet7')
        cx8.to_excel(transaction,sheet_name='Sheet8')
        Cumulative.to_excel(transaction,sheet_name='Sheet9')
dx2.to_csv('C:/Users/smipa/OneDrive/Desktop/dx2.csv')
Hourly_total_transaction.index=hours
Hourly_total_transaction.index.name='Hours'
Net_Trading=Hourly_total_transaction.sum(axis=0)
Net_Trading.name='Total'
Hourly_total_transaction=Hourly_total_transaction.append(Net_Trading)
Hourly_total_transaction.to_csv('C:/Users/smipa/OneDrive/Desktop/Hourly_total_transaction.csv')

#dx2=csv file constains output allocation of hourly transactions
#net househod has 9 sheets that constains transactions of each household separately in each sheet
     their totals in sheet 9.#total transactons has all houshold (buy , sell , use records telling
     total local and grid trading penetation for all houses combined)
```

MILP : Adjusted Demand-Minimum Local Price

```
#Importing libraries
import gurobipy as grb
from gurobipy import*
import pandas as pd
import numpy as np
import scipy
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sns
import sklearn
import random
import statsmodels.api as sm
from collections import OrderedDict
import collections, functools, operator
scipy.set_printoptions(precision = 4, suppress = True)
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from sklearn.preprocessing import MinMaxScaler
from scipy import*

price=[]
response_load=[]
#Simple Demand Adjustment
def demand_response(x,Supply):
       load_i=x
       constraints = ({'type':'ineq','fun': lambda load:Supply-load[0]+load[1]+load[2]+load[3]+load
            [4]+load[5]+load[6]+load[7]},
       {'type':'ineq','fun': lambda load: load[0]},
       {'type':'ineq','fun': lambda load: load[1]},
       {'type':'ineq','fun': lambda load: load[2]},
       {'type':'ineq','fun': lambda load: load[3]},
       {'type':'ineq','fun': lambda load: load[4]},
       {'type':'ineq','fun': lambda load: load[5]},
       {'type':'ineq','fun': lambda load: load[6]},
       {'type':'ineq','fun': lambda load: load[7]}
       )
       res = minimize(eqn, load_i,constraints=constraints)
       return res.fun,res.x

```

```
def eqn(load):
        price=[]
        load=np.array(load)
        # create scaler
        load=load.reshape(8,-1)
        from sklearn.preprocessing import StandardScaler
        scaler2 = MinMaxScaler(feature_range=(.104,.4604))
        scaler2.fit(load)
        normalized = scaler2.transform(load)
        normalized_avg=sum(normalized)/8
        price.append(normalized_avg)
        return normalized_avg
#Reading load data file
df=pd.read_csv('C:/Users/smipa/OneDrive/Documents/Scenario_Run/[4]
     _scenario_2_variable_rates_demand_change/demand_data_input.csv')

#Setting up dataframe parameters for exporting output into a common csv /excel file after all
     iterations.
dx2=pd.DataFrame()
Hourly_total_transaction=pd.DataFrame()
col=['demand','buy from  grid','buy locally','buy_charging_locally','buy_charging_from_grid','
     pv_sold_locally','pv_sold_to_grid',
'use_own_pv','use_own_battery','use_own_pv_charging','sell_battery_locally','sell_battery_to_grid',
'CHARGE_DECISION','DISCHARGE_DECISION','DECISION_TO_SELL','DECISION_TO_BUY','Battery Status after 
     trading']
cx1=pd.DataFrame()
cx2=pd.DataFrame()
cx3=pd.DataFrame()
cx4=pd.DataFrame()
cx5=pd.DataFrame()
cx6=pd.DataFrame()
cx7=pd.DataFrame()
cx8=pd.DataFrame()

#Setting up varaible for optmization
Population=['C1','C2','C3','C4','C5','C6','C7','C8'] # all population
grpA=['C7','C8'] #Consumer (no PV or Battery)
grpB=['C1','C2'] #Only PV
grpC=['C3','C4','C5','C6'] #Battery+PV
grpAnB=['C7','C8','C1','C2']
grpBnC=['C1','C2','C3','C4','C5','C6']
```

```

#Prices
Pg=.4604 #grid price
Pt=.104 #price for selling to grid

#Setting constraint list ,Optmization model
#Also battery dictionary is set up to store battery status after optmization in each hour .
#the battery status is used as input in next iteration.
constraint=[]
opt_model= grb.Model(name="MIP␣Model")
Battery_status={(i):opt_model.addVars(("{0}".format(i) for i in grpC),vtype=grb.GRB.CONTINUOUS,lb
     =0,name="Bt_{0}".format(i)) for i in range(0,49) }
Battery_initial_status={'C3':20.5,'C4':22.5,'C5':15.8,'C6':21.5}

#Setting Battery initial status only for first iteration
for i in grpC:
        Battery_status[0][i]=Battery_initial_status[i]

capacity={'C3':2,'C4':2,'C5':1,'C6':2} #maximum charge and discharge rate possible from battery.
     kept it fixed for this program
Battery_Max={'C3':22.5,'C4':22.5,'C5':15.8,'C6':22.5} # Maximum Battery limit
Battery_Min={'C3':5,'C4':5,'C5':3,'C6':5} # Minimum Battery limit
Data=pd.DataFrame()

#INITIATING FOR LOOP TO OPTMIZE EACH HOUR
for q in range(0,48):
        Data2=df.iloc[q] #READING ELEMENTS OF ROW NUMBER

        total_pv=Data2[10]+Data2[11]+Data2[12]+Data2[13]+Data2[14]+Data2[15]
        x=[Data2[2],Data2[3],Data2[4],Data2[5],Data2[6],Data2[7],Data2[8],Data2[9]]
        Supply=total_pv+7
        norm,arr=demand_response(x,Supply)#Calling function to adjust demand
        new_load=arr
        price.append(norm)
        response_load.append(arr)
        df2=pd.DataFrame()
        df2=Data2
        df2
        df2.iloc[2:10, ] = new_load
        Data=df2
        total_demand=Data[2]+Data[3]+Data[4]+Data[5]+Data[6]+Data[7]+Data[8]+Data[9]
        Pl=norm
        #Setting demand and supply variables for use in optmization model
        P_demand ={'C1':Data[2],'C2':Data[3],'C3':Data[4],'C4':Data[5],'C5':Data[6],'C6':Data[7],'C7':Data[8],'C8':Data[9]}
        grpA_demand={'C7':Data[8],'C8':Data[9]}
        grpB_demand={'C1':Data[2],'C2':Data[3]}
        grpC_demand={'C3':Data[4],'C4':Data[5],'C5':Data[6],'C6':Data[7]}
        demand_grpAnB={'C7':Data[8],'C8':Data[9],'C2':Data[2],'C3':Data[3]}
        demand_grpBnC={'C1':Data[2],'C2':Data[3],'C3':Data[4],'C4':Data[5],'C5':Data[6],'C6':Data[7]}
        grpB_supply={'C1':Data[10],'C2':Data[11]}
        grpC_supply={'C3':Data[12],'C4':Data[13],'C5':Data[14],'C6':Data[15]}
        supply_grpBnC={'C1':Data[10],'C2':Data[11],'C3':Data[12],'C4':Data[13],'C5':Data[14],'C6':Data[15]}

        #SETTING DECSION VARIABLES FOR ALLOCATION INTO EACH GROUP
        #BINARY VARIABLES ARE ALLOTTED 0 or 1 by SOLVER BASED ON DECISION

        buy_from_grid={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="buy_from_grid_{0}".format(i)) for i in Population}
        buy_locally={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="buy_locally_{0}".format(i)) for i in Population}
        pv_sold_locally={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="pv_sold_locally_{0}".format(i)) for i in grpBnC}
        pv_sold_to_grid={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="pv_sold_to_grid_{0}".format(i)) for i in grpBnC}
        use_own_pv={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="use_own_pv_{0}".format(i)) for i in grpBnC }
        use_own_battery={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="use_own_battery_{0}".format(i)) for i in grpC }
        buy_charging_locally={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="buy_charging_locally_{0}".format(i)) for i in grpC }
        buy_charging_from_grid={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="buy_charging_from_grid_{0}".format(i)) for i in grpC }
        sell_battery_locally={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="sell_local_locally_{0}".format(i)) for i in grpC }
        sell_battery_to_grid={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="sell_battery_to_grid_{0}".format(i)) for i in grpC }
        use_own_pv_charging={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="use_own_pv_charging_{0}".format(i)) for i in grpC }
        CHARGE_DECISION={(i):opt_model.addVar(vtype=grb.GRB.BINARY,name="CHARGE_DECISION_{0}".format
            (i)) for i in grpC }
        DISCHARGE_DECISION={(i):opt_model.addVar(vtype=grb.GRB.BINARY,name="DISCHARGE_DECISION_{0}".
            format(i)) for i in grpC }
        DECISION_TO_SELL={(i):opt_model.addVar(vtype=grb.GRB.BINARY,name="DECISION_TO_SELL_{0}".
            format(i)) for i in grpBnC }
        DECISION_TO_BUY={(i):opt_model.addVar(vtype=grb.GRB.BINARY,name="DECISION_TO_BUY_{0}".format
            (i)) for i in Population }

        #CONSTRAINTS FOR GROUP _A (ONLY CONSUMER)
        for i in grpA:
                    constraint={(i):opt_model.addConstr(lhs=(grpA_demand[i]),
                    sense=grb.GRB.EQUAL, rhs=(buy_locally[i]+buy_from_grid[i] ) , name="
                        constraint_{0}".format(i))}

                    constraint={(i):opt_model.addConstr(lhs=(DECISION_TO_BUY[i]),
                    sense=grb.GRB.EQUAL, rhs=(1) , name="constraint_{0}".format(i))}

        #CONSTRAINTS FOR GROUP_B (PV ONLY)
        for i in grpB:
              constraint={(i):opt_model.addConstr(lhs=(grpB_demand[i]),
              sense=grb.GRB.EQUAL, rhs=(use_own_pv[i]+buy_from_grid[i]+ buy_locally[i] ) , name="
                  constraint_{0}".format(i))}

              constraint={(i):opt_model.addConstr(lhs=(DECISION_TO_SELL[i] +DECISION_TO_BUY[i]),
              sense=grb.GRB.LESS_EQUAL, rhs=(1) , name="constraint_{0}".format(i))}

              constraint={(i):opt_model.addConstr(lhs=(use_own_pv[i]+pv_sold_locally[i]+
                  pv_sold_to_grid[i]),
              sense=grb.GRB.EQUAL, rhs=(grpB_supply[i]) , name="constraint_{0}".format(i))}

              constraint={(i):opt_model.addConstr(lhs=(pv_sold_locally[i]+pv_sold_to_grid[i]),
              sense=grb.GRB.LESS_EQUAL, rhs=(grpB_supply[i]*(DECISION_TO_SELL[i])) , name="
                  constraint_{0}".format(i))}

              constraint={(i):opt_model.addConstr(lhs=(buy_from_grid[i]+ buy_locally[i]),
              sense=grb.GRB.LESS_EQUAL, rhs=(grpB_demand[i]*(DECISION_TO_BUY[i])) , name="
                  constraint_{0}".format(i))}

        #CONSTRAINTS FOR GROUP C (PV+BATTERY)
        for i in grpC:
            constraint={(i):opt_model.addConstr(lhs=(grpC_demand[i]),
            sense=grb.GRB.EQUAL, rhs=(use_own_pv[i]+use_own_battery[i]+buy_locally[i]+
                buy_from_grid[i]) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(CHARGE_DECISION[i]+DISCHARGE_DECISION[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=(1) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(DECISION_TO_SELL[i] +DECISION_TO_BUY[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=(1) , name="constraint_{0}".format(i))}

            #SETTING DECISIONS FOR SELL AND BUY TO VARIABLES:
            constraint={(i):opt_model.addConstr(lhs=(use_own_pv[i]+pv_sold_locally[i]+
                pv_sold_to_grid[i]+use_own_pv_charging[i]),
            sense=grb.GRB.EQUAL, rhs=(grpC_supply[i]) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(pv_sold_to_grid[i]+pv_sold_locally[i]+
                sell_battery_locally[i]+sell_battery_to_grid[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=((capacity[i]+grpC_supply[i])*DECISION_TO_SELL[i]) ,
                name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(buy_locally[i]+buy_from_grid[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=((grpC_demand[i])*(DECISION_TO_BUY[i]) ), name="
                constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(buy_charging_from_grid[i]+
                buy_charging_locally[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=(capacity[i]*DECISION_TO_BUY[i]) , name="constraint_{0}
                ".format(i))}


            #SETTING CHARGE AND DISCHARGE DECSIONS TO VARIABLES
            constraint={(i):opt_model.addConstr(lhs=(sell_battery_to_grid[i]+sell_battery_locally
                [i]+use_own_battery[i]),
            sense=grb.GRB.EQUAL, rhs=(capacity[i]*(DISCHARGE_DECISION[i]) ), name="constraint_{0}
                ".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(buy_charging_from_grid[i]+
                buy_charging_locally[i]+use_own_pv_charging[i]),
            sense=grb.GRB.EQUAL, rhs=(capacity[i]*(CHARGE_DECISION[i])) , name="constraint_{0}".
                format(i))}

            #SETTING BATTERY MAXIMUM AND MINIMUM LIMITS
            constraint={(i):opt_model.addConstr(lhs=(Battery_status[q+1][i]),
            sense=grb.GRB.LESS_EQUAL, rhs=(Battery_Max[i]) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(Battery_status[q+1][i]),
            sense=grb.GRB.GREATER_EQUAL, rhs=(Battery_Min[i]) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(Battery_status[q+1][i]),
            sense=grb.GRB.EQUAL, rhs=((Battery_status[q][i] )+(buy_charging_locally[i]+
                buy_charging_from_grid[i]+use_own_pv_charging[i])-(sell_battery_locally[i]+
                sell_battery_to_grid[i]+use_own_battery[i])) , name="constraint_{0}".format(i))}

        #COMMON CONSTRAINTS FOR ALL GROUPS
        constraint={opt_model.addConstr(lhs=(grb.quicksum(buy_locally[i] for i in Population)+grb.
            quicksum(buy_charging_locally[i] for i in grpC)),
        sense=grb.GRB.EQUAL, rhs=(grb.quicksum(pv_sold_locally[i] for i in grpBnC)+grb.quicksum(
            sell_battery_locally[i] for i in grpC)) , name="constraint_{0}".format(i))}

        constraint={opt_model.addConstr(lhs=(total_demand),
        sense=grb.GRB.EQUAL, rhs=(grb.quicksum(buy_locally[i] for i in Population)+grb.quicksum(
            buy_from_grid[i] for i in Population)+grb.quicksum(use_own_pv[i] for i in grpBnC)+grb.
            quicksum(use_own_battery[i] for i in grpC) ), name="constraint_{0}".format(i))}

        constraint={opt_model.addConstr(lhs=(total_pv),
        sense=grb.GRB.EQUAL, rhs=(grb.quicksum(pv_sold_locally[i] for i in grpBnC)+grb.quicksum(
            pv_sold_to_grid[i] for i in grpBnC)+grb.quicksum(use_own_pv[i] for i in grpBnC)+grb.
            quicksum(use_own_pv_charging[i] for i in grpC)) , name="constraint_{0}".format(i))}

        #SETTING OBJECTIVE FUNCTION
        objective = grb.quicksum(Pg*buy_from_grid[i] for i in Population)
        #SETTING OBJECTIVE
        opt_model.ModelSense = grb.GRB.MINIMIZE
        opt_model.optimize()
        #opt_model.printQuality()
        status = opt_model.status

        # STANDARD OUTPUT DISPLAY
        print('Date␣and␣time' ,Data[0],':',Data[1],'\n\n')
        print('BUY␣FROM␣GRID␣TO␣USE:','\n\n',buy_from_grid,'\n\nBUY␣LOCAL␣FOR␣USE:','\n\n',
            buy_locally,'\n\n')
        print('BUY␣LOCAL␣CHARGE:','\n\n',buy_charging_locally,'\n\nBUY␣GRID␣CHARGE:','\n\n',buy_charging_from_grid,'\n\n')
        print('SELL␣PV␣TO␣GRID␣','\n\n',pv_sold_to_grid,'\n\nSELL␣PV␣LOCALLY␣:','\n\n',pv_sold_locally,'\n\n')
        print('USE␣OWN␣PV␣␣','\n\n',use_own_pv,'\n\n')
        print('USE␣BATTERY:','\n\n', use_own_battery,'\n\n␣USE␣PV␣CHARGE␣BATTERY:','\n\n',use_own_pv_charging,'\n\n')
        print('SELL␣BATTERY␣LOCALLY:','\n\n', sell_battery_locally,'\n\n␣SELL␣BATTERY␣TO␣GRID:','\n\n',sell_battery_to_grid,'\n\n')
        print('CHARGE␣DECSION:','\n\n', CHARGE_DECISION,'\n␣\nDISCHARGE␣DECSION','\n\n',DISCHARGE_DECISION,'\n\n')
        print('SELL␣DECISION:','\n\n', DECISION_TO_SELL,'\n␣\nBUY␣DECISION','\n\n',DECISION_TO_BUY,'\n\n')
        for i in grpC:
        print('BATTERY␣STATUS:',Battery_status[q+1][i])
        print('LOCAL␣PRICE:', Pl)

        # Setting variables for creating dataframe for output
        load=[Data[2],Data[3],Data[4],Data[5],Data[6],Data[7],Data[8],Data[9]]
        local_P=[Pl]
        space=[]*8

        # all decision variables converted to list
        m1=[buy_from_grid[a].x for a in Population]
        m2=[buy_locally[a].x for a in Population]
        m3=[buy_charging_locally[a].x for a in grpC]
        m4=[buy_charging_from_grid[a].x for a in grpC]
        m5=[pv_sold_locally[a].x for a in grpBnC]
        m6=[pv_sold_to_grid[a].x for a in grpBnC]
        m7=[use_own_pv[a].x for a in grpBnC]
        m8=[use_own_battery[a].x for a in grpC]
        m9=[use_own_pv_charging[a].x for a in grpC]
        m10=[sell_battery_locally[a].x for a in grpC]
        m11=[sell_battery_to_grid[a].x for a in grpC]
        m12=[CHARGE_DECISION[a].x for a in grpC]
        m13=[DISCHARGE_DECISION[a].x for a in grpC]
        m14=[DECISION_TO_SELL[a].x for a in grpBnC]
        m15=[DECISION_TO_BUY[a].x for a in Population]
        m16=[Battery_status[q+1][i].x for i in grpC]
        z=[0.0]

        # converting unequal rows to equal rows of grp C and grpBnC by putting zero in the missing
            location (size 8 for 8 households)
        for a in range(0,2):
               m3.extend(z)
               m3.insert(0,0.0)
               m4.extend(z)
               m4.insert(0,0.0)
               m8.extend(z)
               m8.insert(0,0.0)
               m9.extend(z)
               m9.insert(0,0.0)
               m10.extend(z)
               m10.insert(0,0.0)
               m11.extend(z)
               m11.insert(0,0.0)
               m12.extend(z)
               m12.insert(0,0.0)
               m13.extend(z)
               m13.insert(0,0.0)
               m16.extend(z)
               m16.insert(0,0.0)
               m5.extend(z)
               m6.extend(z)
               m7.extend(z)
               m14.extend(z)
        #creating columns and index
        columns =['c1','c2','c3','c4','c5','c6','c7','c8']
        index = ['demand','buy␣from␣grid','buy␣locally','buy_charging_locally','
            buy_charging_from_grid','pv_sold_locally','pv_sold_to_grid',
        'use_own_pv','use_own_battery','use_own_pv_charging','sell_battery_locally','
            sell_battery_to_grid',
        'CHARGE_DECISION','DISCHARGE_DECISION','DECISION_TO_SELL','DECISION_TO_BUY','Battery␣Status␣
            after␣trading','Local␣Price','']
        #Combining lists in to a bigger list
        L=[load,m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,m15,m16,local_P,space]
        #creating dataframe for printing transactions in each hour.
        dx1=pd.DataFrame(L, columns = ['c1','c2','c3','c4','c5','c6','c7','c8'],index=index)

        #Creating another dataframe for calculating all totals for each iteration
        hourly_cumulative=pd.DataFrame()
        row_grid_buy=dx1.loc[["buy␣from␣grid","buy_charging_from_grid",]]
        row_grid_sell=dx1.loc[["pv_sold_to_grid","sell_battery_to_grid",]]
        row_buy_local= dx1.loc[["buy␣locally","buy_charging_locally"]]
        row_sell_local= dx1.loc[["pv_sold_locally","sell_battery_locally"]]
        row_use_pv=dx1.loc[["use_own_pv","use_own_pv_charging"]]
        row_use_battery=dx1.loc[["use_own_battery"]]
        self_gridbuy_total= row_grid_buy.sum(axis=1)
        self_localbuy_total= row_buy_local.sum(axis=1)
        self_gridsell_total=row_grid_sell.sum(axis=1)
        self_localsell_total= row_sell_local.sum(axis=1)
        use_pvtotal= row_use_pv.sum(axis=1)
        use_battery_total=row_use_battery.sum(axis=1)
        hourly_cumulative['Total␣demand']=[total_demand]
        hourly_cumulative['Total␣PV']=[total_pv]
        hourly_cumulative['Grid␣buy␣total']=[self_gridbuy_total.sum(axis=0)]
        hourly_cumulative['Local␣Buy␣total']=[self_localbuy_total.sum(axis=0)]
        hourly_cumulative['Grid␣sell␣total']=[self_gridsell_total.sum(axis=0)]
        hourly_cumulative['Locall␣sell␣total']=[self_localsell_total.sum(axis=0)]
        hourly_cumulative['Use␣PV␣Total␣']=[use_pvtotal.sum(axis=0)]
        hourly_cumulative['Use␣Battery␣Total']=[ use_battery_total.sum(axis=0)]
        hourly_cumulative['Total␣Purchase␣costs␣']=(self_gridbuy_total.sum(axis=0)*.4604)+(self_localbuy_total.sum(axis=0)*Pl)
        hourly_cumulative['Total␣sales␣revenue']=(self_gridsell_total.sum(axis=0)*.104)+(self_localsell_total.sum(axis=0)*Pl)
        hourly_cumulative['Net␣Purchase␣costs␣after␣sales␣']=hourly_cumulative['Total␣Purchase␣costs␣'].values-hourly_cumulative['Total␣sales␣revenue'].values
        hourly_cumulative['Local␣Price␣']=Pl
        Hourly_total_transaction=Hourly_total_transaction.append(hourly_cumulative)
        dx2=dx2.append(dx1)

        #Creatng dataframe for summing the all iterations of each Household and a separate sum of all household transactions.
        for i in range(0,8):
            H=[load[i],m1[i],m2[i],m3[i],m4[i],m5[i],m6[i],m7[i],m8[i],m9[i],m10[i],m11[i],m12[i],m13[i],m14[i],m15[i],m16[i]]
            H=np.transpose(H)
            H=[H]
            if i==0:
                cx1 = cx1.append(H)
            elif i==1:
                cx2=cx2.append(H)
            elif i==2:
```

```
                        cx3=cx3.append(H)
                elif i==3:
                        cx4=cx4.append(H)
                elif i==4:
                        cx5=cx5.append(H)
                elif i==5:
                        cx6=cx6.append(H)
                elif i==6:
                        cx7=cx7.append(H)
                elif i==7:
                        cx8=cx8.append(H)

Cumulative=pd.DataFrame()
Cumulative=Cumulative.append(cx1.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx2.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx3.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx4.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx5.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx6.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx7.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx8.sum(axis=0),ignore_index=True)
Cumulative.columns=col
hours=pd.Series(range(0,48))
cx1.columns=col
cx1.index=hours
cx1.index.name='Hours'
cx2.columns=col
cx2.index=hours
cx2.index.name='Hours'
cx3.columns=col
cx3.index=hours
cx3.index.name='Hours'
cx4.columns=col
cx4.index=hours
cx4.index.name='Hours'
cx5.columns=col
cx5.index=hours
cx5.index.name='Hours'
cx6.columns=col
cx6.index=hours
cx6.index.name='Hours'
```

```
cx7.columns=col
cx7.index=hours
cx7.index.name='Hours'
cx8.columns=col
cx8.index=hours
cx8.index.name='Hours'
Cumulative.index=[Population]
Cumulative.index.name='Household'

#converting to csv /excel
excelpath = 'C:/Users/smipa/OneDrive/Desktop/net_household.xlsx'
# Write your dataframes to different sheets
# cx output is for transaction for each household ion the given hours row wise from sheet 1 to 8
#sheet 9 sums the transaction of each house in all hours and presents them together in sheet 9 .
with pd.ExcelWriter(excelpath) as transaction:
        cx1.to_excel(transaction,sheet_name='Sheet1')
        cx2.to_excel(transaction,sheet_name='Sheet2')
        cx3.to_excel(transaction,sheet_name='Sheet3')
        cx4.to_excel(transaction,sheet_name='Sheet4')
        cx5.to_excel(transaction,sheet_name='Sheet5')
        cx6.to_excel(transaction,sheet_name='Sheet6')
        cx7.to_excel(transaction,sheet_name='Sheet7')
        cx8.to_excel(transaction,sheet_name='Sheet8')
        Cumulative.to_excel(transaction,sheet_name='Sheet9')

dx2.to_csv('C:/Users/smipa/OneDrive/Desktop/dx2.csv')
Hourly_total_transaction.index=hours
Hourly_total_transaction.index.name='Hours'
Net_Trading=Hourly_total_transaction.sum(axis=0)
Net_Trading.name='Total'
Hourly_total_transaction=Hourly_total_transaction.append(Net_Trading)
Hourly_total_transaction.to_csv('C:/Users/smipa/OneDrive/Desktop/Hourly_total_transaction.csv')
updated_demand=pd.DataFrame(np.vstack(response_load))
updated_demand.to_csv('C:/Users/smipa/OneDrive/Desktop/update_demand.csv')
#dx2=csv file constains output allocation of hourly transactions
#net househod has 9 sheets that constains transactions of each household separately in each sheet
     their totals in sheet 9.#total transactons has all houshold (buy , sell , use records telling
     total local and grid trading penetation for all houses combined)
```

MILP : Adjusted Demand-Minimum Local Price (Only PV Charging)

```
 ##Importing libraries
import gurobipy as grb
from gurobipy import*
import pandas as pd
import numpy as np
import scipy
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sns
import sklearn
import random
import statsmodels.api as sm
from collections import OrderedDict
import collections, functools, operator
scipy.set_printoptions(precision = 4, suppress = True)
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from sklearn.preprocessing import MinMaxScaler
from scipy import*
#setting up variable price model for each hour
#this calculates local market price for each hour
price=[]
response_load=[]

def demand_response(x,Supply):
       load_i=x
       constraints = ({'type':'ineq','fun': lambda load:Supply-load[0]+load[1]+load[2]+load[3]+load
            [4]+load[5]+load[6]+load[7]},
       {'type':'ineq','fun': lambda load: load[0]},
       {'type':'ineq','fun': lambda load: load[1]},
       {'type':'ineq','fun': lambda load: load[2]},
       {'type':'ineq','fun': lambda load: load[3]},
       {'type':'ineq','fun': lambda load: load[4]},
       {'type':'ineq','fun': lambda load: load[5]},
       {'type':'ineq','fun': lambda load: load[6]},
       {'type':'ineq','fun': lambda load: load[7]})
       res = minimize(eqn, load_i,constraints=constraints)
       return res.fun,res.x

```

```
def eqn(load):
        price=[]
        load=np.array(load)
        # create scaler
        load=load.reshape(8,-1)
        from sklearn.preprocessing import StandardScaler
        scaler2 = MinMaxScaler(feature_range=(.104,.4604))
        scaler2.fit(load)
        normalized = scaler2.transform(load)
        normalized_avg=sum(normalized)/8
        price.append(normalized_avg)
        return normalized_avg

#Reading load data file
df=pd.read_csv('C:/Users/smipa/OneDrive/Documents/Scenario_Run/[4]
     _scenario_2_variable_rates_demand_change/demand_data_input.csv')
l_ref=pd.read_csv('C:/Users/smipa/OneDrive/Documents/Scenario_Run/[4]
     _scenario_2_variable_rates_demand_change/demand_reference.csv')

#Setting up dataframe parameters for exporting output into a common csv /excel file after all
     iterations.
dx2=pd.DataFrame()
Hourly_total_transaction=pd.DataFrame()
col=['demand','buy␣from␣␣grid','buy␣locally','pv_sold_locally','pv_sold_to_grid',
'use_own_pv','use_own_battery','use_own_pv_charging','sell_battery_locally','sell_battery_to_grid',
'CHARGE_DECISION','DISCHARGE_DECISION','DECISION_TO_SELL','DECISION_TO_BUY','Battery␣Status␣after␣
     trading']
cx1=pd.DataFrame()
cx2=pd.DataFrame()
cx3=pd.DataFrame()
cx4=pd.DataFrame()
cx5=pd.DataFrame()
cx6=pd.DataFrame()
cx7=pd.DataFrame()
cx8=pd.DataFrame()

#Setting up varaible for optmization
Population=['C1','C2','C3','C4','C5','C6','C7','C8'] # all population
grpA=['C7','C8'] #Consumer (no PV or Battery)
grpB=['C1','C2'] #Only PV
grpC=['C3','C4','C5','C6'] #Battery+PV
```

```
grpAnB=['C7','C8','C1','C2']
grpBnC=['C1','C2','C3','C4','C5','C6']



#Prices
Pg=.4604 #grid price
Pt=.104 #price for selling to grid

#Setting constraint list ,Optmization model
#Also battery dictionary is set up to store battery status after optmization in each hour .
#the battery status is used as input in next iteration.

constraint=[]
opt_model= grb.Model(name="MIP␣Model")
Battery_status={(i):opt_model.addVars(("{0}".format(i) for i in grpC),vtype=grb.GRB.CONTINUOUS,lb
    =0,name="Bt_{0}".format(i)) for i in range(0,49) }
Battery_initial_status={'C3':20.5,'C4':22.5,'C5':15.8,'C6':21.5}

#Setting Battery initial status only for first iteration
for i in grpC:
      Battery_status[0][i]=Battery_initial_status[i]

char_c={'C3':2,'C4':2,'C5':1,'C6':2} #maximum charge and discharge rate possible from battery.kept
    it fixed for this program
Battery_Max={'C3':22.5,'C4':22.5,'C5':15.8,'C6':22.5} # Maximum Battery limit
Battery_Min={'C3':5,'C4':5,'C5':3,'C6':5} # Minimum Battery limit
Data=pd.DataFrame()

char_c={'C3':2,'C4':2,'C5':1,'C6':2}
def cap(cd3,cd4,cd5,cd6,cs3,cs4,cs5,cs6):
if cs3-cd3>0 and cs3-cd3<2:
      cap3=cs3-cd3
elif cs3-cd3>0 and cs3-cd3>=2:
      cap3=2
else:
      cap3=0
if cs4-cd4 and cs4-cd4<2:
      cap4=cs4-cd4
elif cs4-cd4>0 and cs4-cd4>=2:
cap4=2
      else:
```

```
cap4=0
if cs5-cd5>0 and cs5-cd5<1:
       cap5=cs5-cd5
elif cs5-cd5>0 and cs5-cd5>=1:
       cap5=1
else:
       cap5=0
if cs6-cd6>0 and cs6-cd6<2:
       cap6=cs6-cd6
elif cs6-cd6>0 and cs6-cd6>=2:
       cap6=2
else:
       cap6=0
return cap3,cap4,cap5,cap6

#INITIATING FOR LOOP FOR OPTIMIZING AT EACH HOUR
for q in range(0,48):
       Data2=df.iloc[q] #READING ELEMENTS OF ROW NUMBER
       ref=l_ref.iloc[q]
       total_pv=Data2[10]+Data2[11]+Data2[12]+Data2[13]+Data2[14]+Data2[15]
       x=[Data2[2],Data2[3],Data2[4],Data2[5],Data2[6],Data2[7],Data2[8],Data2[9]]
       y=ref[0]
       Supply=total_pv+7
       norm,arr=demand_response(x,Supply)
       new_load=arr
       new_load
       price.append(norm)
       response_load.append(arr)
       df2=pd.DataFrame()
       df2=Data2
       df2
       df2.iloc[2:10, ] = new_load
       Data=df2
       total_demand=Data[2]+Data[3]+Data[4]+Data[5]+Data[6]+Data[7]+Data[8]+Data[9]
       #Calling fubction for price model for this iteration.
       Pl=norm
       cap3,cap4,cap5,cap6=cap(Data[4],Data[5],Data[6],Data[7],Data[12],Data[13],Data[14],Data[15])
       capacity={'C3':cap3,'C4':cap4,'C5':cap5,'C6':cap6}
       #Setting demand and supply variables for use in optmization model
       P_demand ={'C1':Data[2],'C2':Data[3],'C3':Data[4],'C4':Data[5],'C5':Data[6],'C6':Data[7],'C7
            ':Data[8],'C8':Data[9]}
```

```
    grpA_demand={’C7’:Data[8],’C8’:Data[9]}
    grpB_demand={’C1’:Data[2],’C2’:Data[3]}
    grpC_demand={’C3’:Data[4],’C4’:Data[5],’C5’:Data[6],’C6’:Data[7]}
    demand_grpAnB={’C7’:Data[8],’C8’:Data[9],’C2’:Data[2],’C3’:Data[3]}
    demand_grpBnC={’C1’:Data[2],’C2’:Data[3],’C3’:Data[4],’C4’:Data[5],’C5’:Data[6],’C6’:Data[7]}
    grpB_supply={’C1’:Data[10],’C2’:Data[11]}
    grpC_supply={’C3’:Data[12],’C4’:Data[13],’C5’:Data[14],’C6’:Data[15]}
    supply_grpBnC={’C1’:Data[10],’C2’:Data[11],’C3’:Data[12],’C4’:Data[13],’C5’:Data[14],’C6’:Data[15]}

    #SETTING DECSION VARIABLES FOR ALLOCATION INTO EACH GROUP
    #BINARY VARIABLES ARE ALLOTTED 0 or 1 by SOLVER BASED ON DECISION
    buy_from_grid={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="buy_from_grid_{0}".format(i)) for i in Population}
    buy_locally={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="buy_locally_{0}".format(i)) for i in Population}
    pv_sold_locally={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="pv_sold_locally_{0}".format(i)) for i in grpBnC}
    pv_sold_to_grid={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="pv_sold_to_grid_{0}".format(i)) for i in grpBnC}
    use_own_pv={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="use_own_pv_{0}".format(i)) for i in grpBnC }
    use_own_battery={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="use_own_battery_{0}".format(i)) for i in grpC }
    sell_battery_locally={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="sell_local_locally_{0}".format(i)) for i in grpC }
    sell_battery_to_grid={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="sell_battery_to_grid_{0}".format(i)) for i in grpC }
    use_own_pv_charging={(i):opt_model.addVar(vtype=grb.GRB.CONTINUOUS,lb=0,name="use_own_pv_charging_{0}".format(i)) for i in grpC }
    CHARGE_DECISION={(i):opt_model.addVar(vtype=grb.GRB.BINARY,name="CHARGE_DECISION_{0}".format(i)) for i in grpC }
    DISCHARGE_DECISION={(i):opt_model.addVar(vtype=grb.GRB.BINARY,name="DISCHARGE_DECISION_{0}".format(i)) for i in grpC }
    DECISION_TO_SELL={(i):opt_model.addVar(vtype=grb.GRB.BINARY,name="DECISION_TO_SELL_{0}".format(i)) for i in grpBnC }
    DECISION_TO_BUY={(i):opt_model.addVar(vtype=grb.GRB.BINARY,name="DECISION_TO_BUY_{0}".format(i)) for i in Population }

    #CONSTRAINTS FOR GROUP _A (ONLY CONSUMER)
    for i in grpA:
            constraint={(i):opt_model.addConstr(lhs=(grpA_demand[i]),
            sense=grb.GRB.EQUAL, rhs=(buy_locally[i]+buy_from_grid[i] ) , name="constraint_{0}".
                format(i))}

            constraint={(i):opt_model.addConstr(lhs=(DECISION_TO_BUY[i]),
            sense=grb.GRB.EQUAL, rhs=(1) , name="constraint_{0}".format(i))}

    #CONSTRAINTS FOR GROUP_B (PV ONLY)
    for i in grpB:
            constraint={(i):opt_model.addConstr(lhs=(grpB_demand[i]),
            sense=grb.GRB.EQUAL, rhs=(use_own_pv[i]+buy_from_grid[i]+ buy_locally[i] ) , name="
                constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(DECISION_TO_SELL[i] +DECISION_TO_BUY[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=(1) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(use_own_pv[i]+pv_sold_locally[i]+
                pv_sold_to_grid[i]),
            sense=grb.GRB.EQUAL, rhs=(grpB_supply[i]) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(pv_sold_locally[i]+pv_sold_to_grid[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=(grpB_supply[i]*(DECISION_TO_SELL[i])) , name="
                constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(buy_from_grid[i]+ buy_locally[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=(grpB_demand[i]*(DECISION_TO_BUY[i])) , name="
                constraint_{0}".format(i))}

    #CONSTRAINTS FOR GROUP C (PV+BATTERY)

    for i in grpC:
            constraint={(i):opt_model.addConstr(lhs=(grpC_demand[i]),
            sense=grb.GRB.EQUAL, rhs=(use_own_pv[i]+use_own_battery[i]+buy_locally[i]+
                buy_from_grid[i]) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(CHARGE_DECISION[i]+DISCHARGE_DECISION[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=(1) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(DECISION_TO_SELL[i] +DECISION_TO_BUY[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=(1) , name="constraint_{0}".format(i))}

            #SETTING DECISIONS FOR SELL AND BUY TO VARIABLES:
            constraint={(i):opt_model.addConstr(lhs=(use_own_pv[i]+pv_sold_locally[i]+
                pv_sold_to_grid[i]+use_own_pv_charging[i]),
            sense=grb.GRB.EQUAL, rhs=(grpC_supply[i]) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(pv_sold_to_grid[i]+pv_sold_locally[i]+
                sell_battery_locally[i]+sell_battery_to_grid[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=((char_c[i]+grpC_supply[i])*DECISION_TO_SELL[i]) , name
                ="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(buy_locally[i]+buy_from_grid[i]),
            sense=grb.GRB.LESS_EQUAL, rhs=((grpC_demand[i])*(DECISION_TO_BUY[i]) ), name="
                constraint_{0}".format(i))}

            #SETTING CHARGE AND DISCHARGE DECSIONS TO VARIABLES
            constraint={(i):opt_model.addConstr(lhs=(sell_battery_to_grid[i]+sell_battery_locally
                [i]+use_own_battery[i]),
            sense=grb.GRB.EQUAL, rhs=(char_c[i]*(DISCHARGE_DECISION[i]) ), name="constraint_{0}".
                format(i))}

            constraint={(i):opt_model.addConstr(lhs=(use_own_pv_charging[i]),
            sense=grb.GRB.EQUAL, rhs=(capacity[i]*(CHARGE_DECISION[i])) , name="constraint_{0}".
                format(i))}

            #SETTING BATTERY MAXIMUM AND MINIMUM LIMITS
            constraint={(i):opt_model.addConstr(lhs=(Battery_status[q+1][i]),
            sense=grb.GRB.LESS_EQUAL, rhs=(Battery_Max[i]) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(Battery_status[q+1][i]),
            sense=grb.GRB.GREATER_EQUAL, rhs=(Battery_Min[i]) , name="constraint_{0}".format(i))}

            constraint={(i):opt_model.addConstr(lhs=(Battery_status[q+1][i]),
            sense=grb.GRB.EQUAL, rhs=((Battery_status[q][i] )+(use_own_pv_charging[i])-(
                sell_battery_locally[i]+sell_battery_to_grid[i]+use_own_battery[i])) , name="
                constraint_{0}".format(i))}

        #COMMON CONSTRAINTS FOR ALL GROUPS
        constraint={opt_model.addConstr(lhs=(grb.quicksum(buy_locally[i] for i in Population)),
        sense=grb.GRB.EQUAL, rhs=(grb.quicksum(pv_sold_locally[i] for i in grpBnC)+grb.quicksum(
            sell_battery_locally[i] for i in grpC)) , name="constraint_{0}".format(i))}
```

```

    constraint={opt_model.addConstr(lhs=(total_demand),
    sense=grb.GRB.EQUAL, rhs=(grb.quicksum(buy_locally[i] for i in Population)+grb.quicksum(buy_from_grid[i] for i in Population)+grb.quicksum(use_own_pv[i] for i in grpBnC)+grb.quicksum(use_own_battery[i] for i in grpC) ), name="constraint_{0}".format(i))}

    constraint={opt_model.addConstr(lhs=(total_pv),
    sense=grb.GRB.EQUAL, rhs=(grb.quicksum(pv_sold_locally[i] for i in grpBnC)+grb.quicksum(pv_sold_to_grid[i] for i in grpBnC)+grb.quicksum(use_own_pv[i] for i in grpBnC)+grb.quicksum(use_own_pv_charging[i] for i in grpC)) , name="constraint_{0}".format(i))}

    #SETTING OBJECTIVE FUNCTION
    objective = grb.quicksum(Pg*buy_from_grid[i] for i in Population)
    #SETTING OBJECTIVE
    opt_model.ModelSense = grb.GRB.MINIMIZE
    opt_model.optimize()
    status = opt_model.status

    # STANDARD OUTPUT DISPLAY
    print('Date and time' ,Data[0],':',Data[1],'\n\n')
    print('BUY FROM GRID TO USE:','\n\n',buy_from_grid,'\n\nBUY LOCAL FOR USE:','\n\n',buy_locally,'\n\n')
    print('USE OWN PV  ','\n\n',use_own_pv,'\n\n')
    print('USE BATTERY:','\n\n', use_own_battery,'\n\n USE PV CHARGE BATTERY:','\n\n',use_own_pv_charging,'\n\n')
    print('SELL BATTERY LOCALLY:','\n\n', sell_battery_locally,'\n\n SELL BATTERY TO GRID:','\n\n',sell_battery_to_grid,'\n\n')
    print('CHARGE DECSION:','\n\n', CHARGE_DECISION,'\n \nDISCHARGE DECSION','\n\n',DISCHARGE_DECISION,'\n\n')
    print('SELL DECISION:','\n\n', DECISION_TO_SELL,'\n \nBUY DECISION','\n\n',DECISION_TO_BUY,'\n\n')
    for i in grpC:
        print('BATTERY STATUS:',Battery_status[q+1][i])
    print('LOCAL PRICE:', Pl)

    # Setting variables for creating dataframe for output
    load=[Data[2],Data[3],Data[4],Data[5],Data[6],Data[7],Data[8],Data[9]]
    local_P=[Pl]
    space=[]*8

    # all decision variables converted to list
```

```
        m1=[buy_from_grid[a].x for a in Population]
        m2=[buy_locally[a].x for a in Population]
        m5=[pv_sold_locally[a].x for a in grpBnC]
        m6=[pv_sold_to_grid[a].x for a in grpBnC]
        m7=[use_own_pv[a].x for a in grpBnC]
        m8=[use_own_battery[a].x for a in grpC]
        m9=[use_own_pv_charging[a].x for a in grpC]
        m10=[sell_battery_locally[a].x for a in grpC]
        m11=[sell_battery_to_grid[a].x for a in grpC]
        m12=[CHARGE_DECISION[a].x for a in grpC]
        m13=[DISCHARGE_DECISION[a].x for a in grpC]
        m14=[DECISION_TO_SELL[a].x for a in grpBnC]
        m15=[DECISION_TO_BUY[a].x for a in Population]
        m16=[Battery_status[q+1][i].x for i in grpC]
        z=[0.0]

        # converting unequal rows to equal rows of grp C and grpBnC by putting zero in the missing
             location (size 8 for 8 households)
        for a in range(0,2):
        m8.extend(z)
        m8.insert(0,0.0)
        m9.extend(z)
        m9.insert(0,0.0)
        m10.extend(z)
        m10.insert(0,0.0)
        m11.extend(z)
        m11.insert(0,0.0)
        m12.extend(z)
        m12.insert(0,0.0)
        m13.extend(z)
        m13.insert(0,0.0)
        m16.extend(z)
        m16.insert(0,0.0)
        m5.extend(z)
        m6.extend(z)
        m7.extend(z)
        m14.extend(z)

        #creating columns and index
        columns =['c1','c2','c3','c4','c5','c6','c7','c8']
        index = ['demand','buy␣from␣grid','buy␣locally','pv_sold_locally','pv_sold_to_grid',
        'use_own_pv','use_own_battery','use_own_pv_charging','sell_battery_locally','sell_battery_to_grid',
        'CHARGE_DECISION','DISCHARGE_DECISION','DECISION_TO_SELL','DECISION_TO_BUY','Battery␣Status␣after␣trading','Local␣Price',''] #Combining lists in to a bigger list

        L=[load,m1,m2,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,m15,m16,local_P,space]
        #creating dataframe for printing transactions in each hour.
        dx1=pd.DataFrame(L, columns = ['c1','c2','c3','c4','c5','c6','c7','c8'],index=index)

        #Creating another dataframe for calculating all totals for each iteration
        hourly_cumulative=pd.DataFrame()
        row_grid_buy=dx1.loc[["buy␣from␣grid"]]
        row_grid_sell=dx1.loc[["pv_sold_to_grid","sell_battery_to_grid",]]
        row_buy_local= dx1.loc[["buy␣locally"]]
        row_sell_local= dx1.loc[["pv_sold_locally","sell_battery_locally"]]
        row_use_pv=dx1.loc[["use_own_pv","use_own_pv_charging"]]
        row_use_battery=dx1.loc[["use_own_battery"]]
        self_gridbuy_total= row_grid_buy.sum(axis=1)
        self_localbuy_total= row_buy_local.sum(axis=1)
        self_gridsell_total=row_grid_sell.sum(axis=1)
        self_localsell_total= row_sell_local.sum(axis=1)
        use_pvtotal= row_use_pv.sum(axis=1)
        use_battery_total=row_use_battery.sum(axis=1)
        hourly_cumulative['Total␣demand']=[total_demand]
        hourly_cumulative['Total␣PV']=[total_pv]
        hourly_cumulative['Grid␣buy␣total']=[self_gridbuy_total.sum(axis=0)]
        hourly_cumulative['Local␣Buy␣total']=[self_localbuy_total.sum(axis=0)]
        hourly_cumulative['Grid␣sell␣total']=[self_gridsell_total.sum(axis=0)]
        hourly_cumulative['Locall␣sell␣total']=[self_localsell_total.sum(axis=0)]
        hourly_cumulative['Use␣PV␣Total␣']=[use_pvtotal.sum(axis=0)]
        hourly_cumulative['Use␣Battery␣Total']=[ use_battery_total.sum(axis=0)]
        hourly_cumulative['Total␣Purchase␣costs␣']=(self_gridbuy_total.sum(axis=0)*.4604)+(self_localbuy_total.sum(axis=0)*Pl)
        hourly_cumulative['Total␣sales␣revenue']=(self_gridsell_total.sum(axis=0)*.104)+(self_localsell_total.sum(axis=0)*Pl)
        hourly_cumulative['Net␣Purchase␣costs␣after␣sales␣']=hourly_cumulative['Total␣Purchase␣costs␣'].values-hourly_cumulative['Total␣sales␣revenue'].values
        hourly_cumulative['Local␣Price␣']=Pl
        Hourly_total_transaction=Hourly_total_transaction.append(hourly_cumulative)
        dx2=dx2.append(dx1)

        #Creatng dataframe for summing the all iterations of each Household and a separate sum of all household transactions.
        for i in range(0,8):
        H=[load[i],m1[i],m2[i],m5[i],m6[i],m7[i],m8[i],m9[i],m10[i],m11[i],m12[i],m13[i],m14[i],m15[i],m16[i]]
        H=np.transpose(H)
        H=[H]
        if i==0:
                cx1 = cx1.append(H)
        elif i==1:
                cx2=cx2.append(H)
        elif i==2:
                cx3=cx3.append(H)
        elif i==3:
                cx4=cx4.append(H)
        elif i==4:
                cx5=cx5.append(H)
        elif i==5:
                cx6=cx6.append(H)
        elif i==6:
                cx7=cx7.append(H)
        elif i==7:
                cx8=cx8.append(H)
Cumulative=pd.DataFrame()
Cumulative=Cumulative.append(cx1.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx2.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx3.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx4.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx5.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx6.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx7.sum(axis=0),ignore_index=True)
Cumulative=Cumulative.append(cx8.sum(axis=0),ignore_index=True)
Cumulative.columns=col
hours=pd.Series(range(0,48))
cx1.columns=col
cx1.index=hours
cx1.index.name='Hours'
cx2.columns=col
cx2.index=hours
cx2.index.name='Hours'
cx3.columns=col
```

```
cx3.index=hours
cx3.index.name='Hours'
cx4.columns=col
cx4.index=hours
cx4.index.name='Hours'
cx5.columns=col
cx5.index=hours
cx5.index.name='Hours'
cx6.columns=col
cx6.index=hours
cx6.index.name='Hours'
cx7.columns=col
cx7.index=hours
cx7.index.name='Hours'
cx8.columns=col
cx8.index=hours
cx8.index.name='Hours'
Cumulative.index=[Population]
Cumulative.index.name='Household'

#converting to csv /excel
excelpath = 'C:/Users/smipa/OneDrive/Desktop/net_household.xlsx'
# Write your dataframes to different sheets
# cx output is for transaction for each household ion the given hours row wise from sheet 1 to 8
#sheet 9 sums the transaction of each house in all hours and presents them together in sheet 9 .
with pd.ExcelWriter(excelpath) as transaction:
        cx1.to_excel(transaction,sheet_name='Sheet1')
        cx2.to_excel(transaction,sheet_name='Sheet2')
        cx3.to_excel(transaction,sheet_name='Sheet3')
        cx4.to_excel(transaction,sheet_name='Sheet4')
        cx5.to_excel(transaction,sheet_name='Sheet5')
        cx6.to_excel(transaction,sheet_name='Sheet6')
        cx7.to_excel(transaction,sheet_name='Sheet7')
        cx8.to_excel(transaction,sheet_name='Sheet8')
        Cumulative.to_excel(transaction,sheet_name='Sheet9')
dx2.to_csv('C:/Users/smipa/OneDrive/Desktop/dx2.csv')
Hourly_total_transaction.index=hours
Hourly_total_transaction.index.name='Hours'
Net_Trading=Hourly_total_transaction.sum(axis=0)
Net_Trading.name='Total'
Hourly_total_transaction=Hourly_total_transaction.append(Net_Trading)
```

```
Hourly_total_transaction.to_csv('C:/Users/smipa/OneDrive/Desktop/Hourly_total_transaction.csv')
updated_demand=pd.DataFrame(np.vstack(response_load))
updated_demand.to_csv('C:/Users/smipa/OneDrive/Desktop/update_demand.csv')

#dx2=csv file constains output allocation of hourly transactions
#net househod has 9 sheets that constains transactions of each household separtely in each sheet
     their totals in sheet 9.#total transactons has all houshold (buy , sell , use records telling
     total local and grid trading penetation for all houses combined)
```

VCG Auction

```
import gurobipy as grb
import pandas as pd
import numpy as np
import itertools
import scipy
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sns
import sklearn
import random
import statsmodels.api as sm
from collections import OrderedDict
import collections, functools, operator
scipy.set_printoptions(precision = 4, suppress = True)
from itertools import zip_longest
from collections import OrderedDict
import collections, functools, operator
from itertools import zip_longest
from collections import deque
#Setting up Group Transactions for supply and demand surplus
def GrpA(demand):
net_demand=demand
return net_demand

def GrpB(demand,pv):
if pv==0:
       surplus_pv=0
       net_demand=demand
       use_own_pv=0
       return demand,pv,net_demand,surplus_pv,use_own_pv


elif pv>0 and pv>demand:
       use_own_pv=demand
       surplus_pv=pv-demand
       net_demand=0
       return demand,pv,net_demand,surplus_pv,use_own_pv


```

```
elif pv>0 and pv<=demand:
        use_own_pv=pv
        net_demand=demand-pv
        surplus_pv=0
        return demand,pv,net_demand,surplus_pv,use_own_pv

def GrpC(demand,pv,status,minimum):
#when no battery no PV
        if pv==0 and status<=2:
                surplus_pv=0
                net_demand=demand
                use_own_pv=0
                use_own_pv_charging=0
                use_own_battery=0
                battery_surplus=0
                status=status
                return  demand,pv,net_demand,surplus_pv,use_own_pv,use_own_pv_charging,
                     use_own_battery,battery_surplus,status

        #PV>0 , demand is more than PV and battery under minimum limit . No buying or selling as
             demand is met
        elif pv>0 and demand-pv>0 and status<=2:
                use_own_pv=pv
                net_demand=demand-pv
                surplus_pv=0
                use_own_battery=0
                status=status
                battery_surplus=0
                use_own_pv_charging=0
                return demand,pv,net_demand,surplus_pv,use_own_pv,use_own_pv_charging,use_own_battery
                     ,battery_surplus,status

        #surplus pv and demand is less than PV
        elif pv>0 and pv-demand>0:
                surplus_pv=pv-demand
                use_own_pv=demand
                net_demand=0 # all demand is met by PV

        # Battery needs charging and surplus_Pv> charge limit of 2 kWh
                if surplus_pv>0 and status<=2 and surplus_pv>=minimum:
                        status=status+minimum
```

```
            use_own_pv_charging=minimum
            surplus_pv=surplus_pv-minimum
            use_own_battery=0
            battery_surplus=0
            return demand,pv,net_demand,surplus_pv,use_own_pv,use_own_pv_charging,use_own_battery,battery_surplus,status
    # Battery needs charging and surplus_Pv < charge limit of 2 kWh
        elif surplus_pv>0 and status<=2 and surplus_pv<minimum: #charging pv <2
            status=status+surplus_pv
            use_own_pv_charging=surplus_pv
            surplus_pv=0 #all PV used up here , demand also fulfilled , no role in market
            use_own_battery=0
            battery_surplus=0
            return demand,pv,net_demand,surplus_pv,use_own_pv,use_own_pv_charging,use_own_battery,battery_surplus,status


    #surplus pv and battery dos not need charging ,Prosumer is seller here
        elif surplus_pv>0 and status>2 :
            surplus_pv=surplus_pv
            use_own_pv_charging=0
            use_own_battery=0
            status=status
            battery_surplus=0
            return demand,pv,net_demand,surplus_pv,use_own_pv,use_own_pv_charging,use_own_battery,battery_surplus,status

    #demand is more than pv , PV all used up to meet demand.
    elif pv>0 and demand-pv>0:
        net_demand= demand-pv # still some demand to be met
        surplus_pv=0
        use_own_pv=pv

    #demand to be met by battery if available , no charging
        if status>2 and net_demand<minimum: #when demand is less than discharge limit
            use_own_battery=net_demand
            battery_surplus=minimum-net_demand
            status=status-(minimum)
            net_demand=0
            use_own_pv_charging=0
```

```
                    return demand,pv,net_demand,surplus_pv,use_own_pv,use_own_pv_charging,
                        use_own_battery,battery_surplus,status

            elif status>2 and net_demand>=minimum: #when demand is more than discharge limit
                    net_demand=net_demand-minimum
                    use_own_battery=minimum
                    battery_surplus=0
                    status=status-minimum
                    use_own_pv_charging=0
                    return demand,pv,net_demand,surplus_pv,use_own_pv,use_own_pv_charging,
                        use_own_battery,battery_surplus,status


        #demand to be met by battery only as No PV is available
        elif pv==0 and status>2 and demand<minimum: #when demand is less than discharge limit
            pv=pv
            use_own_pv=pv
            surplus_pv=0
            battery_surplus=minimum-demand
            net_demand=0
            use_own_battery=demand
            status=status-(minimum)
            net_demand=0
            use_own_pv_charging=0
            return demand,pv,net_demand,surplus_pv,use_own_pv,use_own_pv_charging,use_own_battery
                ,battery_surplus,status

        elif pv==0 and status>2 and demand>=minimum: #when demand is more than discharge limit
            pv=pv
            use_own_pv=pv
            surplus_pv=0
            net_demand=demand-minimum
            battery_surplus=0
            use_own_battery=minimum
            status=status-minimum
            use_own_pv_charging=0
            return demand,pv,net_demand,surplus_pv,use_own_pv,use_own_pv_charging,use_own_battery
                ,battery_surplus,status

dg = pd.DataFrame(columns=['Supplier','supply','demand','Buyer','net␣supply','net␣demand','Energy␣
    sold','price','revenue/cost'])
```

```
acc=pd.DataFrame(columns=['Cost','Revenue','local_sell'])
grid=pd.DataFrame(pd.DataFrame(columns=['Cost','Revenue','grid_sell']))
#Battery Initial status
prev_status=[22.5,22.5,15.8,22.5]

#Setting up varaibles for grouping assignment
Population=['C1','C2','C3','C4','C5','C6','C7','C8'] # all population
grpA=['C7','C8'] #Consumer (no PV or Battery)
grpB=['C1','C2'] #Only PV
grpC=['C3','C4','C5','C6'] #Battery+PV
grpAnB=['C7','C8','C1','C2']
grpBnC=['C1','C2','C3','C4','C5','C6']

#Prices
Pg=.4604 #grid price
Pt=.104 #price for selling to grid

#All self transactions defined in fucntions
df=pd.read_csv('C:/Users/smipa/OneDrive/Documents/Scenario_Run/[3]_scenario_2_variable␣_rates/
    demand_data_input.csv')
dc1=pd.DataFrame()
dc2=pd.DataFrame()
dc3=pd.DataFrame()
dc4=pd.DataFrame()
dc5=pd.DataFrame()
dc6=pd.DataFrame()
dc7=pd.DataFrame()
dc8=pd.DataFrame()

for q in range(0,48):
#Reading data
      Data=df.iloc[q]
      DataGrpA_demand=[Data[8],Data[9]]
      DataGrpB_demand=[Data[2],Data[3]]
      DataGrpB_supply=[Data[10],Data[11]]
      DataGrpC_demand=[Data[4],Data[5],Data[6],Data[7]]
      DataGrpC_supply=[Data[12],Data[13],Data[14],Data[15]]
      limit=[2,2,1,2]
      #Total Demand and PV specified
      total_demand=Data[2]+Data[3]+Data[4]+Data[5]+Data[6]+Data[7]+Data[8]+Data[9]
      total_pv=Data[10]+Data[11]+Data[12]+Data[13]+Data[14]+Data[15]
    countB=0
    for (i,j) in itertools.zip_longest(DataGrpB_demand, DataGrpB_supply):
    demand,pv,net_demand,surplus_pv,use_own_pv=GrpB(i,j)
    LB=[demand,pv,net_demand,surplus_pv,use_own_pv]
    if countB==0:
    dc1 = dc1.append([LB],ignore_index=True)
    countB=countB+1
    elif countB==1 :
    dc2=dc2.append([LB],ignore_index=True)

    countC=0
    for (i,j,k,l) in zip(DataGrpC_demand, DataGrpC_supply,prev_status,limit):
    demand,pv,net_demand,surplus_pv,use_own_pv,use_own_pv_charging,use_own_battery,
        battery_surplus,status=GrpC(i,j,k,l)
    LC=[ demand,pv,net_demand,surplus_pv,use_own_pv,use_own_pv_charging,use_own_battery,
        battery_surplus,status]

    if countC==0:
           dc3 = dc3.append([LC],ignore_index=True)
           prev_status[0]=status
           countC=countC+1
    elif countC==1 :
           dc4=dc4.append([LC],ignore_index=True)
           prev_status[1]=status
           countC=countC+1
    elif countC==2 :
           dc5=dc5.append([LC],ignore_index=True)
           prev_status[2]=status
           countC=countC+1
           prev_status[2]=status
           elif countC==3 :
           dc6=dc6.append([LC],ignore_index=True)
           prev_status[3]=status

    countA=0
    for i in itertools.zip_longest(DataGrpA_demand):
           net_demand=GrpA(i)
           LA=[net_demand]
           if countA==0:
           dc7 = dc7.append(LA,ignore_index=True)
           countA=countA+1
```

```
            elif countA==1 :
            dc8=dc8.append(LA,ignore_index=True)

    col1=['demand','pv','net_demand','surplus_pv','use_own_pv']
    col2=['demand','pv','net_demand','surplus_pv','use_own_pv','use_own_pv_charging','
        use_own_battery','battery_surplus','status']
    col3=['net_demand']
    dc1.columns=col1
    dc2.columns=col1
    dc3.columns=col2
    dc4.columns=col2
    dc5.columns=col2
    dc6.columns=col2
    dc7.columns=col3
    dc8.columns=col3
    excelpath = 'C:/Users/smipa/OneDrive/Desktop/self_transaction.xlsx'
    with pd.ExcelWriter(excelpath) as trans:
            dc1.to_excel(trans,sheet_name='Sheet1')
            dc2.to_excel(trans,sheet_name='Sheet2')
            dc3.to_excel(trans,sheet_name='Sheet3')
            dc4.to_excel(trans,sheet_name='Sheet4')
            dc5.to_excel(trans,sheet_name='Sheet5')
            dc6.to_excel(trans,sheet_name='Sheet6')
            dc7.to_excel(trans,sheet_name='Sheet7')
            dc8.to_excel(trans,sheet_name='Sheet8')

for n in range(0,48):
    row1=dc1.iloc[n]
    row2=dc2.iloc[n]
    row3=dc3.iloc[n]
    row4=dc4.iloc[n]
    row5=dc5.iloc[n]
    row6=dc6.iloc[n]
    row7=dc7.iloc[n]
    row8=dc8.iloc[n]
    total_net_demand=row1[2]+row2[2]+row3[2]+row4[2]+row5[2]+row6[2]+row7[0]+row8[0]
    total_net_supply=row1[3]+row2[3]+row3[3]+row3[7]+row4[3]+row4[7]+row5[3]+row5[7]+row6[3]+
        row6[7]
    total_net_demand
    total_net_supply
    #buyer/seller classification and buyers bid
```

```
        buyer={}
        seller={}
        bid={}
        #c1
        if row1[2]>0 and row1[3]==0:
                buyer.update({'buyer_c1':row1[2]})
        elif row1[2]==0 and row1[3]>0:
                seller.update({'seller_c1':row1[3]})
        #c2
        if row2[2]>0 and row2[3]==0:
                buyer.update({'buyer_c2':row2[2]})
        elif row2[2]==0 and row2[3]>0:
                seller.update({'seller_c2':row2[3]})
        #c3
        if row3[2]>0 and row3[3]==0 and row3[7]==0:
                buyer.update({'buyer_c3':row3[2]})
        if row3[2]==0 and row3[3]>0 or row3[7]>0:
                seller.update({'seller_c3':(row3[3]+row3[7])})
        #c4
        if row4[2]>0 and row4[3]==0 and row4[7]==0:
                buyer.update({'buyer_c4':row4[2]})
        if row4[2]==0 and row4[3]>0 or row4[7]>0:
                seller.update({'seller_c4':(row4[3]+row4[7])})
        #c5
        if row5[2]>0 and row5[3]==0 and row5[7]==0:
                buyer.update({'buyer_c5':row5[2]})
        if row5[2]==0 and row5[3]>0 or row5[7]>0:
                seller.update({'seller_c5':(row5[3]+row5[7])})
        #c6
        if row6[2]>0 and row6[3]==0 and row6[7]==0:
                buyer.update({'buyer_c6':row6[2]})
        if row6[2]==0 and row6[3]>0 or row6[7]>0:
                seller.update({'seller_c6':(row6[3]+row6[7])})
        #c7
        buyer.update({'buyer_c7':row7[0]})
        #c8
        buyer.update({'buyer_c8':row8[0]})

        #bid price
        for i,j in buyer.items():
                price=.4604-(((total_net_demand-j)/(total_net_demand))*(.4604-.104))
        bid.update({i:(price)})
    cost=0
    #Arranging bid and supplies
    seller_list = sorted(seller.items(), key=operator.itemgetter(1))
    s=list(i[1] for i in seller_list)
    name_s=list(i[0] for i in seller_list)
    buyer_list =sorted(buyer.items(), key=operator.itemgetter(1),reverse=True)
    c=list(i[1] for i in buyer_list)
    name_c=list(i[0] for i in buyer_list)
    bid_list=sorted(bid.items(), key=operator.itemgetter(1),reverse=True)
    rate= list(i[1] for i in bid_list)
    rate_n=list(i[0] for i in bid_list)
    cost=0
    revenue=0
    local_buy=0
    local_sell=0
    grid_buy=0
    grid_sell=0
    #Local Transactions
    while s and c:
        if s[0]>c[0]:
            dg=dg.append({'Supplier':name_s[0],'supply':s[0],'demand':c[0],'Buyer':name_c[0],'net␣supply':s[0]-c[0],'net␣demand':0,'Energy␣sold':c[0],'price':rate[0],'revenue/cost':c[0]*rate[0]},ignore_index=True)
            s[0]=s[0]-c[0]
            cost=cost+c[0]*rate[0]
            revenue=revenue+c[0]*rate[0]
            local_sell=local_sell+c[0]
            del c[0]
            del name_c[0]
            del rate[0]
            del rate_n[0]

        elif c[0]>s[0]:
            dg=dg.append({'Supplier':name_s[0],'supply':s[0],'demand':c[0],'Buyer':name_c[0],'net␣supply':0,'net␣demand':c[0]-s[0],'Energy␣sold':s[0],'price':rate[0],'revenue/cost':s[0]*rate[0]},ignore_index=True)
            c[0]=c[0]-s[0]
            cost=cost+(s[0])*rate[0]
            revenue=revenue+(s[0])*rate[0]
            local_sell=local_sell+s[0]
            del s[0]
            del name_s[0]
        acc=acc.append({'Cost':cost,'Revenue':revenue,'local_sell':local_sell},ignore_index=True)
        cost_g=0
        revenue_g=0
    #Grid transactions
    if s:
        for i in range(0,len(s)):
            revenue_g=revenue_g+(s[i])*.104
            grid_sell=grid_sell+s[i]
            dg=dg.append({'Supplier':name_s[i],'supply':s[i],'demand':0,'Buyer':'grid','net␣supply':s[i],'net␣demand':0,'Energy␣sold':s[i],'price':.104,'revenue/cost':s[i]*.104},ignore_index=True)
        dg=dg.append({'Supplier':'','supply':'','demand':'','Buyer':'','net␣supply':'','net␣demand':'','Energy␣sold':'','price':'','revenue/cost':''},ignore_index=True)
        grid=grid.append({'Revenue':revenue_g,'grid_sell':grid_sell},ignore_index=True)
    if c:
        for i in range(0,len(c)):
            cost_g=cost_g+(c[i])*.4604
            dg=dg.append({'Supplier':'grid','supply':0,'demand':c[i],'Buyer':name_c[i],'net␣supply':0,'net␣demand':0,'Energy␣sold':c[i],'price':.4604,'revenue/cost':c[i]*.4604},ignore_index=True)
            dg=dg.append({'Supplier':'','supply':'','demand':'','Buyer':'','net␣supply':'','net␣demand':'','Energy␣sold':'','price':'','revenue/cost':''},ignore_index=True)
            grid=grid.append({'Cost':cost_g},ignore_index=True)
        dg.to_csv('C:/Users/smipa/OneDrive/Desktop/dg.csv')
        acc.to_csv('C:/Users/smipa/OneDrive/Desktop/acc.csv')
        grid.to_csv('C:/Users/smipa/OneDrive/Desktop/grid.csv')

#dg.csv is the log of hourly transaction output for each hour
#acc.csv is the total of the local transactions like revenue for each hour
#grid.csv is total of the trandsaction with grid for each hour
```