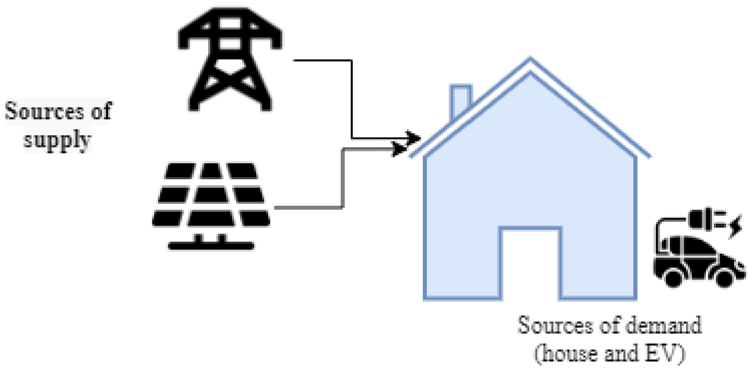
MJ2505- Optimization Tutorial



In this tutorial, we will optimize the schedule of meeting the demand of electricity for meeting the demand of a house and an electric vehicle. Some key points to understand

1. There are two sources of supply: grid and solar PV
2. There is demand from residential consumption and EV charging

We will use pulp (https://coin-or.github.io/pulp/) package in python for this tutorial. In the link, you will find the detailed information about the several parts of the package. There are several other packages in python with a difference in syntax and structure.

Input data

1. The input data is prepared in excel.
2. There are 5 columns in excel "Hour", "EV", "Demand", "Price" and "Solar PV"
3. We have data for one week (7 days)

Steps in the code

The first step is to install the packages needed for the algorithm.

1. Install pulp, numpy, pandas and xlswriter

Installing required libraries

```
[ ]: pip install pulp
[ ]: pip install numpy
[ ]: pip install xlswriter
[ ]: pip install pandas
```

Remember- You can skip this, if you have already installed the packages. Even after you have installed it, you can run this again, does not change anything.

2. Import the packages that have been installed

Import the required libraries

[2]:
```python
import pandas as pd
import pulp as pl
import numpy as np
import xlsxwriter as xl
```

3. Import the data from input excel sheet

We can define days and select columns from the imported excel sheet. Here, [.iloc function]() is used to select the columns from the excel.

- Solar PV data is solar generation profile.
- Demand is hourly demand of the house
- Price is wholesale electricity price from Nordpool
- EV is the binary value showing if the vehicle is available for charging or not ('0' means not available and '1' means available)

Hour	EV	Demand (in kWh)	Price (in SEK/kWh)	Solar PV (in kWh)
0	1	3.620145	0.03639	0
1	1	4.1733	0.03654	0
2	1	0.42471	0.03097	0
3	1	5.53704	0.01998	0
4	1	1.65735	0.0216	0.045
5	1	0.34746	0.0384	0.105
6	1	4.08489	0.04104	0.21
7	1	0.447555	0.04492	0.36
8	0	0.653925	0.04905	0.54
9	0	4.496865	0.05126	0.72
10	0	2.6601	0.05122	0.72
11	0	8.007645	0.04904	0.675
12	0	0.819495	0.0481	0.585
13	0	4.301985	0.04702	0.54
14	0	1.01637	0.04606	0.465
15	0	1.72107	0.05029	0.375
16	0	2.668665	0.05257	0.3
17	0	3.736185	0.05837	0.18
18	0	2.00229	0.05779	0.09
19	1	4.337445	0.05736	0.015

Import the data for the optimization from excel sheet

Price data in SEK/kWh Demand in kWh EV_ availability is binary value

[3]:
```python
#input data reading from excel
days = 7 # number of days of simulation
worksheet = pd.read_excel('Input_data.xlsx')

#hourly electricity price
EV_available = worksheet.iloc[0:days*24, 1]
Demand = worksheet.iloc[0:days*24, 2]
Price_data = worksheet.iloc[0:days*24, 3]
SolarPV_generation = worksheet.iloc[0:days*24,4]
```

4. Defining the parameters

Here we define the parameters. These are the initial values, which is based on specifics of the technologies considered. As you see, we have considered on charging and discharging losses. Other losses in battery like storage losses, calendar ageing etc. are ignored. Here we have defined Charger power also.

Parameters for optimization

[4]:
```
Battery_capacity = 40 #Size of the EV battery in kWh
C_Rating = 1 #The rate at which the battery charges.
# The capacity of a battery is commonly rated at 1C, meaning that a fully charged battery rated at 1Ah should provide 1A for one h
Ptrafo = 1200 #The capacity is in kW

CS_losses = 0.96 #charging losses
DS_losses = 1.041 #discharging losses


Start_storage = 0.2*Battery_capacity # Initial Battery Capacity to start with

#We have one EV charger installed in a house
Charger_Power = 7.2 #Rated EV charger in kW- each charge will be able to supply this capacity
```

5. Defining the sets- the time index

This is one of the most critical step and utmost care must be taken while defining this. As you see in the code, function range is used. It defines the indexes. Also sets “I” and “J” are defined. You can observe that “J” has 25 indexes, one more than “I”. This is because, it will be used to store the values for next day from previous days results.

Defining sets

[5]:
```
# time series parameter
#This is important for assigning the variables that are used in the optimization.
#In python, you have to initialize lists/tuples/dictionaries first if they are used at some later point in the code.

Price = {x:1 for x in range(25)}
Demand_hourly = {x:1 for x in range(25)}
EV_availability = {x:1 for x in range(25)}
SolarPV = {x:1 for x in range(25)}

#sets initialization
I = list(range(24))
J = list(range(25)) #We have this until 25 , one  extra hour so as to ensure that the the storage level at the end of the day, act
print(I)
print(J)
```
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
```

6. Here we define the excel sheet where the results will be stored

```
[6]: workbook = xl.Workbook('Results_optimization.xlsx')
worksheet = workbook.add_worksheet()

worksheet.write(0, 0, 'Days')
worksheet.write(0, 1, 'Hour')
worksheet.write(0, 2, 'Demand')
worksheet.write(0, 3, 'SolarPV')
worksheet.write(0, 4, 'EV Available')
worksheet.write(0, 5, 'Charging power') #Total charging in an hour
worksheet.write(0, 6, 'Storage Level') #Total storage level in the battery of EV
worksheet.write(0, 7, 'Purchase from the grid') #Total amount of electricity purchased from the grid
worksheet.write(0, 8, 'SolarPV_consumption')
worksheet.write(0, 9, 'Price')
worksheet.write(0, 10, 'Cost of supply from grid') #Total of price*purchased electricity from the grid in an hour
```

7. As we are running the model for 7 days, we define a function to run the model each day. Functions are small tools designed to perform specific functions.

```
[7]:
#Here we write a function which can be called repeatedly where needed in the code
#This function is meant to fetch the data for each 24 hours of the optimization window.
#We run the optimization for every 24 hours and then it runs for the whole week

def assign(d):
    for y in I:
        Price[y] = Price_data[d*24 + y]
        Demand_hourly[y] = Demand[d*24 + y]
        EV_availability[y] = EV_available[d*24 + y]
        SolarPV[y] = SolarPV_generation[d*24 + y]
```

8. We define the model with constraints as a function. Comments are written in the code. You can refer to pulp documentation also to see the syntax and more functionalities.

```python
# Optimization model- Here we define another function
#Global variables are those values which are constant and will be accessed by the function. Therefore, it is important to assign the
#You can try by removing one of them and the function will start producing an error

def Optimize():
    global Start_storage #Initiating variable for starting the storage of charged units
    global Battery_capacity #This is parameter as defined in the beginning
    global Charger_Power # This is parameter defined above
    global Ptrafo


    model = pl.LpProblem("Cost_minimize_charging_problem", pl.LpMinimize) # Here we define if the problem is minimization or maximiz

    # Decision variables- The ones which offer flexibility and decision making  capability to optimizer

    Grid_purchase = pl.LpVariable.dicts('Grid_purchase', I, cat='Continuous', lowBound=0) # Syntax to specify the characteristics o
    Storage_level = pl.LpVariable.dicts('Storage_level', J,cat='Continuous', lowBound=0.2*Battery_capacity, upBound=0.8*Battery_capa
    Charge_hourly = pl.LpVariable.dicts('Charge_hourly', I, cat='Continuous', lowBound=0) #Amount of charge every hour, CS
    EV_charging_power = pl.LpVariable.dicts('EV_charging_power', I,cat='Continuous', lowBound=0)

    # Constraints

    #Storage level constraint
    model += Storage_level[0] == Start_storage       #Start of level of storage

    #in every hour
    for i in I:

        model += Grid_purchase[i] - Charge_hourly[i] - Demand_hourly[i] + SolarPV[i] == 0 # Energy Balance Constraint
        model += Grid_purchase[i] <= Ptrafo # Grid limit constraint

        model += Charge_hourly[i] <= C_Rating*Charger_Power*EV_availability[i] # Maximum charging limit in an hour

        if EV_availability[i] == 1 and EV_availability[i+1] == 0: # If EV leaves the house, the storage level
            model += Storage_level[i+1] == 0.8*Battery_capacity

        if EV_availability[i] == 0: #If EV is not available for charging
            model += Storage_level[i+1] == 0.2* Battery_capacity

        if EV_availability[i] == 1: #If EV is avialable for charging
            model += Storage_level[i+1] == Storage_level[i] + Charge_hourly[i]*CS_losses


  # Objective Function
    model += pl.lpSum (Grid_purchase[i]*Price[i] for i in I) #Defining the objective function
    model.solve()

    Start_storage = Storage_level[24].varValue     #for the next day we have to store the storage level of last hour of previous da

   #Writing results
    for y in I:

        worksheet.write(x*24+y+1, 0, x+1) # Print number of days in excel
        worksheet.write(x*24+y+1, 1, y) # Print hours
        worksheet.write(x*24+y+1, 2, Demand_hourly[y])
        worksheet.write(x*24+y+1, 3, SolarPV[y])
        worksheet.write(x*24+y+1, 4, EV_availability[y])
        worksheet.write(x*24+y+1, 5, Charge_hourly[y].varValue)
        worksheet.write(x*24+y+1, 6, Storage_level[y].varValue)
        worksheet.write(x*24+y+1, 7, Grid_purchase[y].varValue)
        #worksheet.write(x*24+y+1, 8, Demand_hourly[y].varValue-Grid_purchase[y].varValue)
        worksheet.write(x*24+y+1, 9, Price[y])
        worksheet.write(x*24+y+1, 10, Grid_purchase[y].varValue*Price[y])

  #Running the model code

    model.solve()
    #Printing some results, not necessary though
    print('Day:')
    print(x+1)
    print (pl.LpStatus[model.status])
    print (pl.value(model.objective))
```

9. We run the model for 7 days by calling the "assign" and "optimize" function as defined and close the excel after data is saved. The results are saved in a new excel file with name "Results_optimize.xlsx"

```python
#Using the function optimize to run the model for assigned set of data, here in our case it
# is one week as provided in the input data sheet

for x in range(days):
    assign(x) #Calling the data assignmment function
    Optimize()# Calling the optimize function
workbook.close()
```

10. The results are now printed in “Results_optimize.xlsx” in the same folder where the code and inputs files are located.

	A	B	C	D	E	F	G	H	I	J
1	Days	Hour	Demand	SolarPV	EV Available	Charging power	Storage Level	Purchase from the grid	Price	Cost of supply from grid
2	1	0	3.620145	0	1	3.4	8	7.020145	0.03639	0.255463077
3	1	1	4.1733	0	1	0	11.264	4.1733	0.03654	0.152492382
4	1	2	0.42471	0	1	7.2	11.264	7.62471	0.03097	0.236137269
5	1	3	5.53704	0	1	7.2	18.176	12.73704	0.01998	0.254486059
6	1	4	1.65735	0.045	1	7.2	25.088	8.81235	0.0216	0.19034676
7	1	5	0.34746	0.105	1	0	32	0.24246	0.0384	0.009310464
8	1	6	4.08489	0.21	1	0	32	3.87489	0.04104	0.159025486
9	1	7	0.447555	0.36	1	0	32	0.087555	0.04492	0.003932971
10	1	8	0.653925	0.54	0	0	32	0.113925	0.04905	0.005588021
11	1	9	4.496865	0.72	0	0	8	3.776865	0.05126	0.1936021
12	1	10	2.6601	0.72	0	0	8	1.9401	0.05122	0.099371922
13	1	11	8.007645	0.675	0	0	8	7.332645	0.04904	0.359592911
14	1	12	0.819495	0.585	0	0	8	0.234495	0.0481	0.01127921
15	1	13	4.301985	0.54	0	0	8	3.761985	0.04702	0.176888535

11. Use the results in excel to plot the graphs to visualize and also calculate values
    a. What is the total optimized electricity cost that the consumer needs to pay?
    b. What is the total amount of charging done at 1:00 am?