



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

UNIVERSITA' DEGLI STUDI DI PADOVA

Dipartimento di Ingegneria Industriale DII

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria dell'Energia Elettrica

Optimizing battery energy storage system for photovoltaic power generation in buildings: a Python-based approach to evaluate economic profitability and energy performance

Nicola Trivellin

Giuseppe Fumarola

Anno Accademico 2022/2023

TABLE OF CONTENTS

| | |
|--|----|
| INTRODUCTION..... | 3 |
| ABSTRACT | 3 |
| 1. PROBLEM STATEMENT | 4 |
| 2. ENERGY MISMATCH EVALUATION..... | 4 |
| 2.1 ENERGY MISMATCH | 4 |
| 2.2 ENERGY FLATNESS | 5 |
| 2.3 KEY PERFORMANCE INDICATORS (KPI'S) | 6 |
| 3. BATTERY SIZING..... | 11 |
| 4. PERFORMANCE ANALYSIS | 13 |
| 5. INCREASING MISMATCH | 15 |
| 6. ECONOMIC FEASIBILITY | 16 |
| 7. RESUMING CHART FLOW | 17 |
| 8. LABORATORY CASE STUDY | 17 |
| 8.1 LIVING LABORATORY FEATURES | 17 |
| 8.2 MISMATCH EVALUATION | 18 |
| 9. BUILDING CONSUMPTION | 24 |
| 9.1 BELGIUM ANNUAL OFFICE BUILDING CONSUMPTIONS..... | 24 |
| 9.2 OFFICE BUILDING CONSUMPTION PROFILE..... | 28 |
| 10 SYNTHETIC MODEL OF PHOTOVOLTAIC PRODUCTION | 30 |
| 10.1 ARRAY SIZE ESTIMATION | 31 |
| 10.2 PVWatt MODEL..... | 32 |
| 11. MAIN CODE | 37 |
| 11.1 "BESTBATTERY" FUNCTION | 39 |
| 11.2 BATTERY OPTIMIZATION | 49 |
| 11.3 PERFORMANCE ANALYSIS..... | 52 |
| 12. WORKING EXAMPLE..... | 53 |
| 13. INPUT DATA INFLUENCE ANALYSIS | 55 |
| CONCLUSIONS..... | 61 |
| REFERENCES..... | 76 |

INTRODUCTION

Energy efficiency in buildings is an increasingly important issue in addressing the challenges of climate change and sustainable energy supply. In this context, the use of renewable energy generation technologies, such as solar photovoltaic energy, is increasingly widespread. However, the problem of intermittent solar production makes it necessary to integrate electrical energy storage systems, such as battery packs, to ensure a balance between production and building energy consumption.

In this thesis, a Python code was developed to assess the energy mismatch between production and consumption of a building, and to study the optimal choice of battery pack size to improve the building's self-consumption and energy performance. In particular, the objective was to maximise the self-consumption of the energy produced by the photovoltaic system in order to reduce the dependency on the electricity grid and make the investment in the battery pack plus photovoltaics more profitable than a photovoltaic-only system. By analysing the energy consumption and solar production data (if provided) of a building, Python code was used to calculate the optimal battery pack size and load/unload strategy to maximise the self-sufficiency and self-consumption of the solar energy produced. The results show that the integration of an appropriately sized battery pack can significantly improve the building's self-consumption and energy efficiency, reducing energy costs and grid dependency.

In conclusion, this thesis demonstrates the importance of an accurate analysis of the energy mismatch between the production and consumption of a building, and the optimisation of the size and charge/discharge strategies of the battery pack to improve the building's self-consumption and energy efficiency. This work has implications for the design of photovoltaic systems integrated with energy storage systems to maximise the utilisation of renewable energy sources and reduce the environmental impact of traditional energy sources.

This thesis was carried out at the Faculty of Engineering Technology of the University of Leuven (KU Leuven) within the ELECTA-Ghent research group, under the supervision of KU Leuven post-doctoral researcher Bert Herteleer and the University of Padua's academic supervisor Prof. Nicola Trivellin.

ABSTRACT

This thesis work focuses on optimizing the Battery Energy Storage System (BESS) size for photovoltaic (PV) power generation in buildings. A literature review is conducted on the definition of mismatch, its calculation, methods of reducing it, and how to evaluate energy performance and economic profitability. A raw Python code is developed and tested on the SOLARISE Living Laboratory battery at the Ghent Technology Campus. The final code takes input data and provides results such as the size of the battery capacity that maximises investment profitability and the optimal size that makes the investment in the building's photovoltaic plant + BESS more profitable than the investment of the PV plant alone. The final code was applied to a case study of an office building in Ghent, Belgium. The results show that the final code approach optimizes the energy performance and economic profitability of buildings with PV plant + BESS system and achieving the highest possible economic return from the investment while minimizing its cost requires a balance between the size of the optimal battery capacity and the IRR of the investment. Therefore, the developed final code provides a tool for designing PV systems with battery storage and analysing their economic viability, facilitating the transition towards more sustainable energy systems. Future research may investigate the applicability of the proposed method to different case studies and further optimizing the method to improve its accuracy and efficiency.

1. PROBLEM STATEMENT

To properly size the battery storage system is necessary to first define what the energy mismatch is. The difference between energy demand and energy supply might result in a lack of electricity grid request in summer or at daytime, or a lack of energy supply in winter, cloudy days or at night, causing a mismatch between energy demand and energy supply.

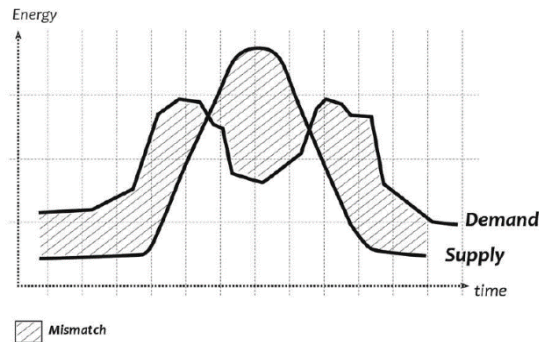


Figure 1: mismatch [1]

This mismatch should be solved to avoid energy grid dependence and therefore to reach Energy flatness.

2. ENERGY MISMATCH EVALUATION

2.1 ENERGY MISMATCH

The aim of avoiding energy mismatch requires a deeper understanding of what a mismatch is and what factors contribute to its occurrence.

The energy demand of a building is unpredictable [1] in short time intervals of hours because every building is different and its energy demand depends on many variables such as the occupant behaviour, the physical properties of the building, the function and the external variables such as outdoor temperature and location.

In the following figure is shown a daily common energy demand profile:

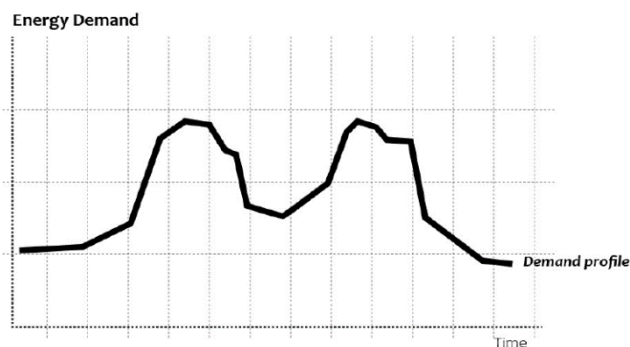


Figure 2: energy demand profile curve [1]

Considering a photovoltaic plant as the renewable energy sources integrated into the built environment leads us to face the problem that photovoltaic energy supply is weather dependant. The following figure shows the average power output of a solar energy production system in hours in the 12 months of the year.

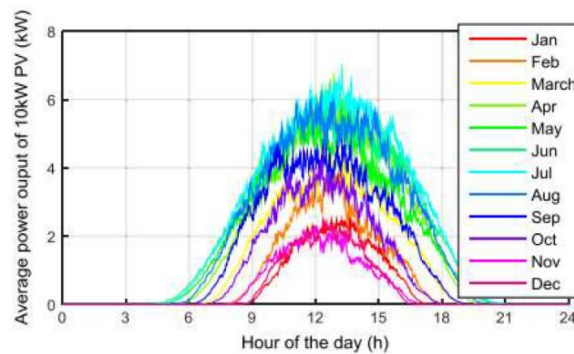


Figure 3: average daily PV production for each month [2]

The difference between the unpredictable and weather-dependent demand and the weather-dependent supply results in a mismatch in short time periods of hours (next figure) as well as over the year:

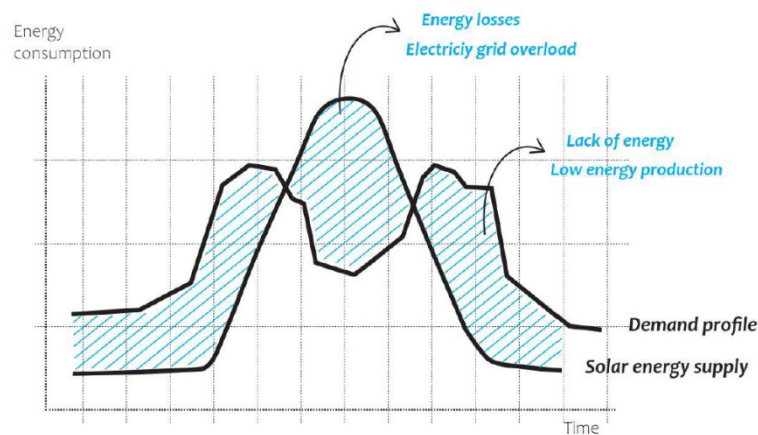


Figure 4: energy mismatch [1]

2.2 ENERGY FLATNESS

There are different states of a building in terms of how it deals with energy mismatch and in terms of its energy performance such as the that Zero Energy Buildings (ZEB) and the Energy Flat Buildings.

Research based on the mismatch compensation factor in zero energy buildings stated that Zero Energy Buildings (ZEB) are the ones that minimize heating and electricity demand with on-site renewable energy generation. There are on-grid ZEB and off-grid ZEB, and the difference is that the on-grid has the ability to purchase energy from the grid when the energy produced by itself is not enough, or can sell the excess energy to the same grid when there is surplus [3]

Energy flatness is a state of a building's energy performance. Is called "Energy-flat" a building in which the difference between the final energy demand, and the energy supplied to the building with an on-site energy generation is zero at any time. This concept has been developed in the thesis "Energy-flat housing"[4]. Different concepts complement and help to understand the term of energy flatness, but do not mean the same.

The complete energy flatness accomplishes the requirement for an energy-neutral building by balancing the demand and supply in a yearly basis. Furthermore, this concept goes beyond energy neutrality because it aims to achieve energy balance in a yearly, monthly, daily and hourly basis.

When aiming for Zero Energy Buildings (ZEB) the scope is not clear, as it can include either the thermal balance, or the electricity balance or it can allow for energy inefficient buildings to achieve the status of ZEB by having an oversized PV energy supply and without energy saving measures [5].

The complete energy flatness happens when the final energy and the energy supply match at any time of the year.

If a complete match cannot be achieved, there are two possible mismatches. Negative mismatch indicates the moments of shortage of energy, which means that the building will need to import energy from the external grid. Positive mismatch indicates a surplus of energy; the building can store the extra energy or export it to the shared grid. Both exporting and importing energy are not the aim of the energy flatness, because the building should be able to be self-energy sustainable.

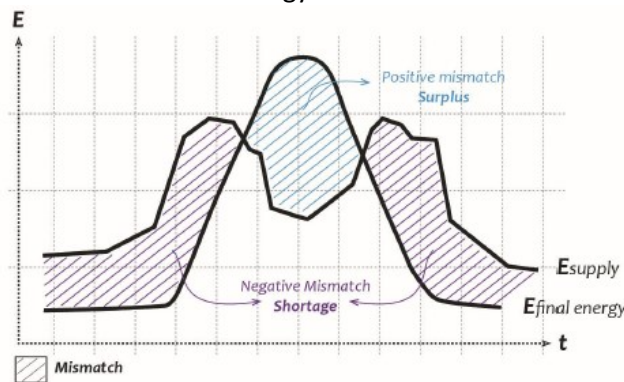


Figure 5: positive and negative mismatch [1]

Energy Storage technologies have great potential for smoothing out the electricity and thermal energy, ensuring that the onsite energy produced matches the needed energy demand of the building. Energy storage is well known for its rapid response if the system is located within the building boundary, and it aims to balance the energy within the building.

The capacity and power of the storage can be sized according to the needs of the building, making it possible to respond quickly, which is important to ensure energy balance at any hour of year, thus achieving Energy flatness.

2.3 KEY PERFORMANCE INDICATORS (KPI'S)

After understanding what a mismatch is and setting the goal of achieving energy flatness in buildings, which means compensating the mismatch at any time of the year, the next step consists of choosing the best way to evaluate the mismatch.

Key Performance Indicators (KPI) aim to quantify the mismatch, because the smaller the mismatch, the closer the building is to being energy flat. Three KPIs are analysed based on the KPI proposed by V. Höfte [4], H. Lund et al. [3] and Ala-Juusela et al [6].

In order to define the KPIs, an analysis of the main characteristics of the mismatch was developed.

- The first KPI aims to calculate the absolute energy flatness or the electricity mismatch, which is the difference between the final energy and the energy supply during the 8760 hours of the year.
- The second KPI measures the peak of the mismatch in terms of positive mismatch (surplus) and negative mismatch (shortage) to understand the hours of the year when the difference between final energy and energy supply is at its peak.
- The third KPI calculates the cumulative surplus and the energy shortage before it can be compensated with the storage energy system. The difference between the maximum cumulative positive and maximum cumulative negative mismatch represents the size of the energy storage system in order to balance the energy system.

KPI 1 – ABSOLUTE ENERGY FLATNESS (AEF)

An Energy-flat building is a building in which the difference between the final energy demand and the energy supply to the building services is zero at any time of the year.

$$AEF = \sum_{t=1}^{t=8760} |E_{final\ energy}(t) - E_{energy\ supply}(t)| \text{ [kW]} \quad (1)$$

The energy flatness can be calculated with the previous formula, where the sum of the energy delivered ($E_{final\ energy}(t)$) in a time step (t) in kWh, minus the sum of the energy supply ($E_{energy\ supply}(t)$) at the same time step (t) in kWh is equal to Zero at any time step. The (t) refers to the time steps in hours from the energy simulation that can be between 1 hour and 8760 hours in a year.

The result of this equation does not differentiate the positive or negative mismatch but aims to show the absolute total mismatch during a t time. The perfect energy flat building will reach a result equal to zero at any time. In this equation, it is necessary to have clear boundary lines for the system and balance boundary because the complementary energy system plays an important role to achieve complete flatness.

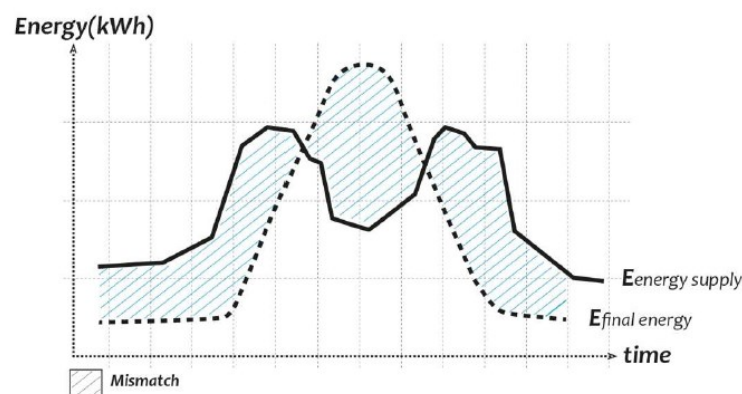


Figure 6: energy mismatch [1]

KPI 2 – MAXIMUM MISMATCH PEAK (MMP)

The second KPI shows the peak of the mismatch. To understand what the maximum peak of the mismatch is, it is necessary to analyse it in terms of the highest positive and negative mismatch that can occur between the final energy and the energy supply during a year.

$$MMP = \max_{0 \leq t \leq 8760} |E_{final\ energy}(t) - E_{energy\ supply}(t)| [kW] \quad (2)$$

The formula describes a time (t) that is between hour 0 and the hour 8760, when the difference between the final energy and the energy supply is on its maximum peak compared to the other mismatches during the year. Therefore, compared to the measurement KPI 1, the unit of KPI 2 is in KW (kilowatts). Furthermore, with this KPI it is possible to calculate the maximum positive mismatch peak (peak of the surplus) and the maximum negative mismatch peak (peak of the shortage).

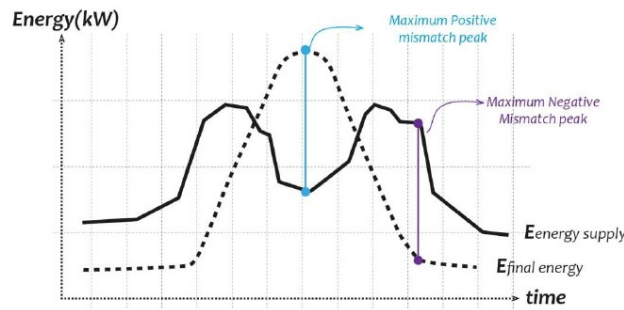


Figure 7: maximum mismatch peak [1]

KPI 3 – MAXIMUM CUMULATIVE ENERGY MISMATCH (MCEM)

The energy mismatch can be positive or negative, and a zero mismatch is equal to energy flatness. When the periods of shortage are repetitive within the hours, there is a cumulative negative mismatch (CEM_negative). This is calculated by the sum of the negative mismatches of the previous time steps before there is an on-site renewable energy production available to compensate for the shortage.

The cumulative positive mismatch (CEM_positive) is calculated by the sum of all the positive mismatches of the previous time steps before there is a period of shortage that needs to be compensated with the renewable energy production.

After the cumulative positive and negative mismatches are calculated, the difference between the maximum of the cumulative positive mismatch that happened at a (a) time and the maximum of the cumulative negative mismatch that happened at another (b) time will be the answer to KPI 3 as well as the required size of the energy storage system.

$$MCEM = \max_{0 \leq a \leq 8760} \text{positive}(CEM_{positive}(a)) - \max_{0 \leq b \leq 8760} \text{negative}(CEM_{negative}(b)) [kWh] \quad (3)$$

The figure below shows a graph of the mismatch where it is possible to see the positive cumulative mismatch and negative cumulative mismatch.

KPI 3 shows how large the surplus of energy (potential storage energy) could be when it is cumulated within the hours because it was not used by the building energy balance. The negative cumulative mismatch comprises energy through the hours where there was not enough final onsite energy production to compensate for the energy need by the complementary energy system in the building.

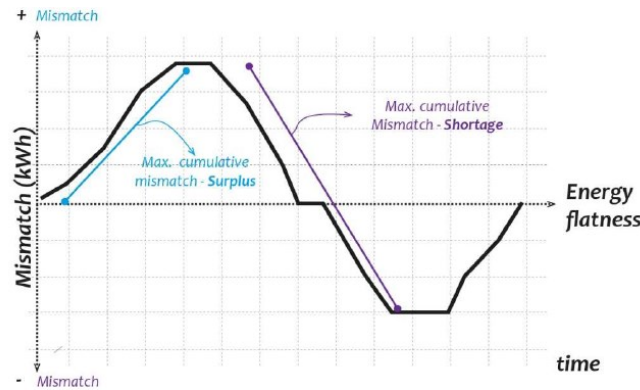


Figure 8: cumulative mismatch [1]

The difference between the maximum cumulative positive and the maximum cumulative negative is the approximate size of the complementary energy storage system, without counting the energy losses through conversion or distribution.

KPI 3 – Example

In order to make the KPIs clear, an example of the analysis of the mismatch in an office building with solar panels and only one person working for one day is displayed in the next table[1].

| Time (hour) | Final energy (kW) | Energy supply (kW) | Energy Mismatch (kW) | Cumulative mismatch (kWh) |
|-------------|-------------------|--------------------|----------------------|---------------------------|
| 1 | 0 | 0,04 | -0,04 | -0,04 |
| 2 | 0 | 0,04 | -0,04 | -0,08 |
| 3 | 0 | 0,04 | -0,04 | -0,12 |
| 4 | 0 | 0,04 | -0,04 | -0,16 |
| 5 | 0 | 0,04 | -0,04 | -0,2 |
| 6 | 0,15 | 0,15 | 0 | -0,2 |
| 7 | 0,25 | 0,3 | -0,05 | -0,25 |
| 8 | 0,3 | 0,3 | 0 | -0,25 |
| 9 | 0,35 | 0,3 | 0,05 | -0,2 |
| 10 | 0,55 | 0,6 | -0,05 | -0,25 |
| 11 | 0,75 | 0,6 | 0,15 | -0,1 |
| 12 | 0,85 | 0,75 | 0,1 | 0 |
| 13 | 0,9 | 0,75 | 0,15 | 0,15 |
| 14 | 0,85 | 0,6 | 0,25 | 0,4 |
| 15 | 0,65 | 0,5 | 0,15 | 0,55 |
| 16 | 0,45 | 0,5 | -0,05 | 0,5 |
| 17 | 0,35 | 0,5 | -0,15 | 0,35 |
| 18 | 0,25 | 0,3 | -0,05 | 0,3 |
| 19 | 0,1 | 0,1 | 0 | 0,3 |
| 20 | 0 | 0,1 | -0,1 | 0,2 |
| 21 | 0 | 0,06 | -0,06 | 0,14 |
| 22 | 0 | 0,06 | -0,06 | 0,08 |
| 23 | 0 | 0,04 | -0,04 | 0,04 |
| 24 | 0 | 0,04 | -0,04 | 0 |
| Total | 6,75 | 6,75 | 0 | 0 |

Figure 9: building energy mismatch timetable [1]

In this example the total mismatch in the 24 hours of the day is equal to zero, which means that the daily basis is solved, but the building itself is not energy flat, because in the hourly basis the mismatch is not solved.

Nonetheless, there are three hours in the day when a perfect flatness is reached, at 6, 8 and 19 hours. During those hours the building was energy flat because the final energy use of the building is perfectly correlated with the energy supply.

In the graph below, one can see the mismatch period that happens during 24 hours between final energy and energy supply. Energy flatness is reached in a period of 24 hours because the sum of the final energy that happens during the 24 hours minus the sum of the energy supply during those 24 hours is the same.

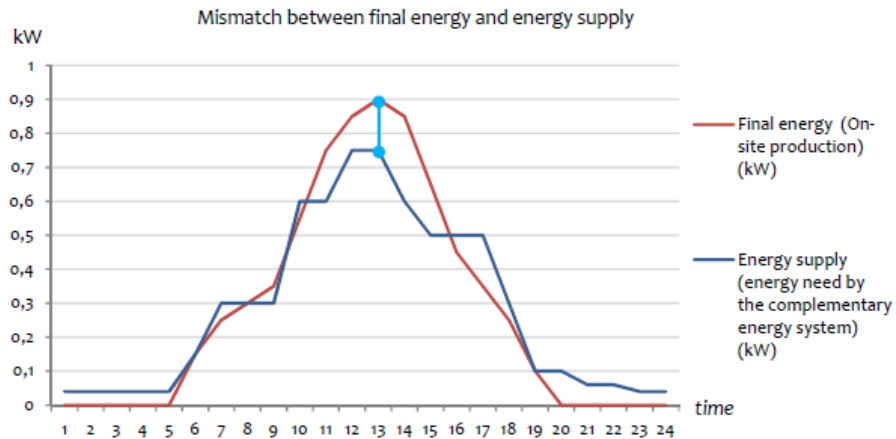


Figure 10: building hourly energy mismatch over a day [1]

$$KPI\ 1 = \sum_{t=1}^{t=8760} |6,75\ kW_{(24h)} - 6,75\ kW_{(24h)}| \quad (4)$$

Even though the building in energy flat during that day, in the hourly basis it is not energy flat. The maximum mismatch happens during the hour 14 ($t = 14$). During this hour the difference between final energy and energy supply is 0,25 kw of positive mismatch (energy surplus) that can be store in order to compensate the energy shortage. During the hour 22 the difference was 0.06 kW of the negative mismatch (shortage of energy) that needs to be supplied from the energy surplus.

$$KPI\ 2 = \max_{t=14} \text{positive } |0,85\ kW_{(t)} - 0,6\ kW_{(t)}| = 0,25\ kW \quad (5)$$

$$KPI\ 2 = \max_{t=22} \text{negative } |0\ kW_{(t)} - 0,06\ kW_{(t)}| = -0,06\ kW \quad (6)$$

KPI 3 account for the maximum of the cumulative negative and positive mismatches. Therefore, during the hour 8 the maximum cumulative negative mismatch takes place, which is the sum of all the shortage hours between hour 1 and 8, which means that a shortage of 0,25 kWh needs to be compensated by the complementary energy system.

On the other hand, the maximum cumulative positive mismatch happens in the hour 15, and is the sum of the positive mismatches between hour 13 and 15, when 0,55 kWh represent the cumulative surplus of energy that can help to compensate the energy shortage period.

$$KPI\ 3 = 0,55\ kWh - (-0,25\ kWh) = \mathbf{0,8\ kWh} \quad (7)$$

The difference between the maximum cumulative mismatches represents the KPI 3.

In conclusion, this means that the size of the battery energy storage system should at least have a capacity to store 0.8 kWh to be able to charge and discharge during the maximum cumulative surplus and the maximum cumulative shortage and during the peaks of the mismatch, in order to achieve the Energy flatness. The periods when there is a positive cumulative mismatch, the battery energy storage system will be charged, while in the periods when the cumulative mismatch is negative, the battery energy storage system will discharge the stored energy.

3. BATTERY SIZING

After having defined the most accurate way to quantify the energy mismatch that could provide us with the minimum optimal value of the battery storage system capacity (in order to achieve Energy flatness), the following step consists of defining the ultimate final size of the Battery Energy Storage System (BESS).

To find the final size of the energy storage system, it is necessary to consider other constraints that affect the battery during its work leading to inefficiency thus resulting in an underestimated size of the battery.

The Storage Systems Working Group of the IEEE Standards Coordinating Committee 21 on Fuel Cells, Photovoltaics, Dispersed Generation, and Energy Storage (SCC21) developed a recommended practice method [7] for sizing both vented and valve-regulated lead-acid batteries used in terrestrial photovoltaic (PV) systems. This procedure aims to size a battery bank to have sufficient capacity to provide the required energy over the autonomy period initially defined. We will extend this recommended procedure for the sizing of every kind of battery.

After initially choosing a trial battery capacity, the most important adjustment required for the minimum optimal battery capacity, provided by the maximum cumulative energy mismatch (MCEM) calculation, are as follows:

- Discharge adjustments

The unadjusted capacity (C_{unj}) should be modified to assure satisfactory battery cycle life. Battery manufacturers rate cells for maximum depth of discharge (MDOD), maximum daily depth of discharge (MDDOD) and end-of-life (EOL) capacity. The battery capacity should be adjusted in the following ways:

- a) The capacity adjusted for MDOD is obtained by dividing the unadjusted capacity by MDOD (in percent).
- b) The capacity adjusted for MDDOD is obtained by dividing the maximum daily ampere hours by MDDOD (in percent).

c) The capacity adjusted for life is obtained by dividing the unadjusted capacity by the end-of-life capacity expressed in percent of the rated capacity, commonly 80%.

$$C_{adj,1} = \text{MAX} \left\{ \frac{C_{unj}}{\text{MDOD}}; \frac{C_{unj}}{\text{MDDOD}}; \frac{C_{unj}}{\text{EOL}} \right\} \quad (8)$$

The largest of these three capacities will satisfy the depth-of-discharge and end-of-life adjustments.

- Temperature adjustment

The available capacity of a battery is affected by its operating temperature. Cell capacity ratings are generally standardized at 25 °C. Capacity increases at temperatures above 25 °C and decreases at temperatures below 25 °C. Capacity is rarely adjusted for warm temperature operation, but adjustments are routinely made for cold weather applications. Refer to the battery manufacturer's literature for temperature correction factors. The adjusted capacity determined in 5.3.3.1 should be corrected by this factor to yield capacity adjusted for temperature.

$$C_{adj,2} = K_T \cdot C_{adj,1} \quad (9)$$

- Design margin adjustment

It is prudent design practice to provide a capacity margin to allow for uncertainties in the load determination, e.g., less-than-optimum conditions and load growth. A common practice to provide this design margin is to add 10–25% to the capacity as determined in 5.3.3.2

$$C_{adj,3} = C_{adj,2}(1.1 \div 1.25) \quad (10)$$

In order to quantify the effectiveness of adding a battery energy storage system to the pre-existing photovoltaic plant (PV), several analyses need to be conducted in terms of building energy performance and economic profitability of the investment.

4. PERFORMANCE ANALYSIS

The matching potential is frequently expressed using load matching indicators such as self-sufficiency and self-consumption. In the paper 'Graphical analysis of photovoltaic generation and load matching in buildings' [8], Luthander R. et al. introduced the energy matching chart which consists of a novel graphical approach to visualise PV load matching.

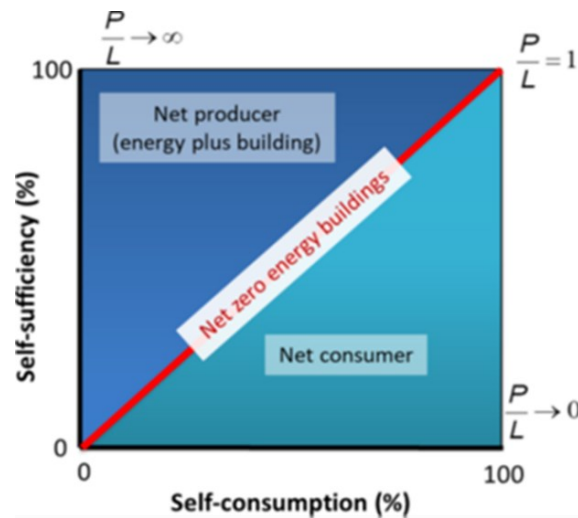


Figure 11: Energy mismatch chart [8]

The above chart uses self-sufficiency and self-consumption to provide information regarding the matching in both size and time.

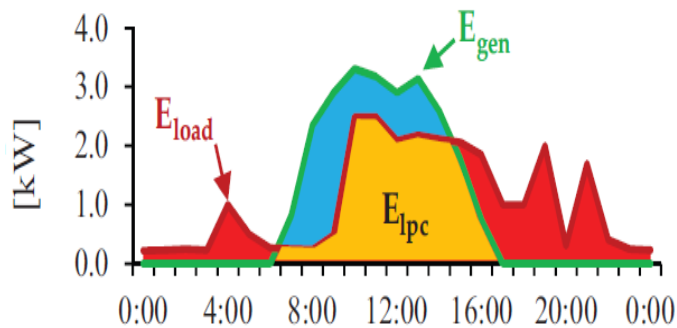


Figure 12: Example of daily profile with hourly time step source [9]

The figure above represents an example of a daily energy photovoltaic production and the relative building daily energy hourly consumption.

The Self-sufficiency is defined [9] the ratio between the energy locally produced and immediately used by the consumer (E_{lpc}) and the total load energy (E_{load}):

$$SS = \frac{E_{lpc}}{E_{load}} \quad (11)$$

The Self-consumption, instead, is defined [9] the ratio between the energy locally produced and immediately used by the consumer (E_{lpc}) and the total generation (E_{gen}):

$$SC = \frac{E_{lpc}}{E_{gen}} \quad (12)$$

The relationship between self-consumption and self-sufficiency is therefore $\frac{SS}{SC} = \frac{E_{gen}}{E_{load}}$ (13) .

The overlapping energy E_{lpc} between the total net electricity demand and PV generation, represents the PV power that each instant is directly used to fulfil the building energy requirement at that specific time. Perfect matching is achieved in the top right corner, where both the self-consumption and self-sufficiency is 100%. Poor matching in time gives a result in the lower left corner, with both low self-consumption and low self-sufficiency. Net zero energy buildings, which produce as much electricity as they consume on an annual basis, will be on the diagonal red line where $SC=SS$ and thus $P/L=1$

When evaluating and interpreting metrics for PV self-consumption[10], it is important to be aware of how a couple of factors affect the results:

- Relative sizes of PV power generation and power demand:
Increasing the PV generation relative to the demand will always decrease the self-consumption while self-sufficiency will be increased or remain unchanged.

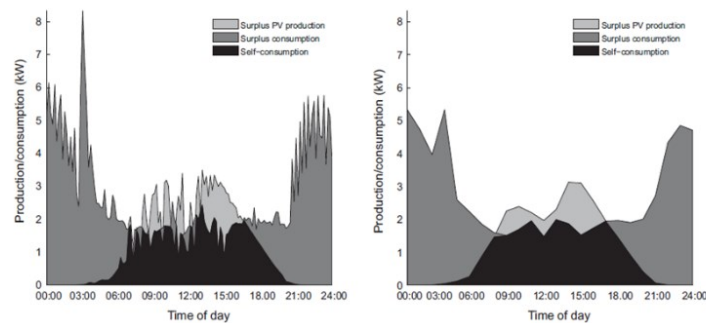


Figure 13: Example of the difference in PV self-consumption when using 10-min (left) and hourly (right) data, source [10]

- Time resolution

A lower resolution will always lead to an overestimation of the self-consumption since fluctuations causing mismatch between the generation and load profiles are evened out by averaging.

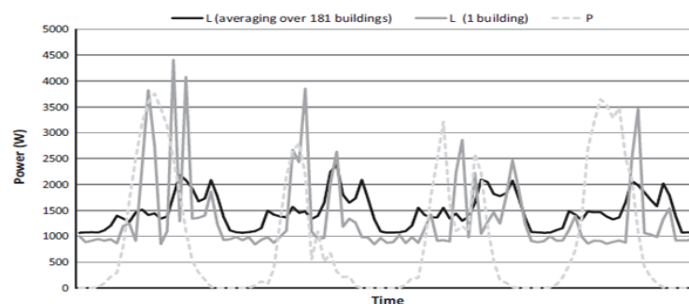


Figure 14: Example of the smoothing effect on the load (L) from averaging over 181 buildings compared to one, source [10]

- Number of buildings
load profiles can be expected to be more affected by the number of buildings.

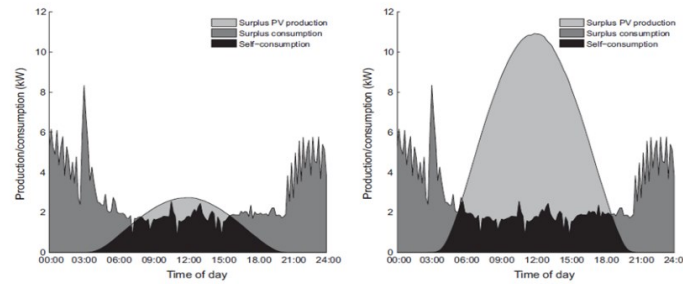


Figure 15: Example of a PV system with a low (left) and a high (right) rated power supplying the same building load, source [10]

5. INCREASING MISMATCH

There are two common ways to increase self-consumption and self-sufficiency [10]: they are demand side management (DSM) and electrochemical energy storage in batteries.

There are several meanings of the concept of demand side management (DSM), where the common denominator is to improve the energy system at the side of consumption. In the paper the term is used for load shifting which can be used to shift the power demands of the loads in a household, for example washing machine and heating, ventilation, and air-conditioning (HVAC) systems, from time periods with surplus consumption to periods with surplus PV production. Therefore, the periods with highest interaction with the power distribution grid (feed in and out) can therefore be decreased. Load shifting can be achieved either manually, where persons switch on electric devices when the sun is shining, or automatically, which requires control algorithms and devices, and sometimes also weather forecasts of ambient temperature and solar irradiation.

Most of the papers examine PV-battery systems, sometimes combined with DSM. The results show that it is possible to increase relative self-consumption by 13–24% points with a battery storage capacity of 0.5–1 kWh per installed kW of PV power and between 2% and 15% points with DSM, both compared to the original rate of self-consumption.

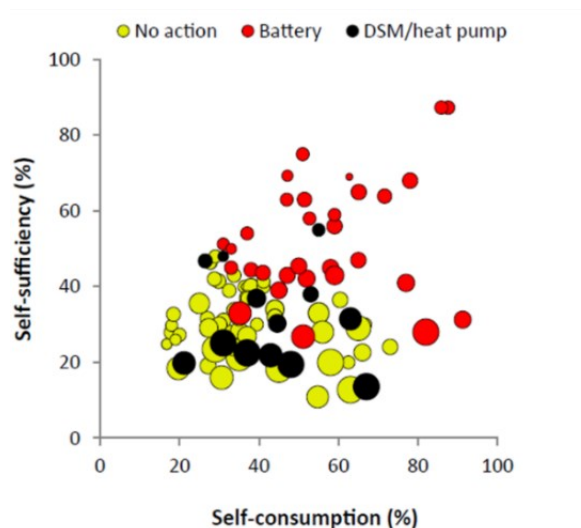


Figure 16: The Energy matching chart showing the self-consumption and self-sufficiency reported in the paper, source [9]

6. ECONOMIC FEASIBILITY

There are different criteria to analyse the economic feasibility of an investment. The expected net present value (NPV) and the economic internal rate of return (IRR) should be calculated for all projects in which benefits can be valued. The main difference between the NPV and IRR is that the NPV provides the economic return of the investment instead the IRR is much more useful to compare different kind of investments. The general criterion for accepting a project is to achieve a positive NPV discounted at the minimum required IRR or to achieve the minimum required IRR of 9% [11].

The results of the reference scenario “Solar energy storage in German “[12] show positive net present values (NPV) for PV systems of approx. 500–1,800 EUR/kWp and NPV for BESS of approx. 150–500 EUR/kWh (including cabins and installation).

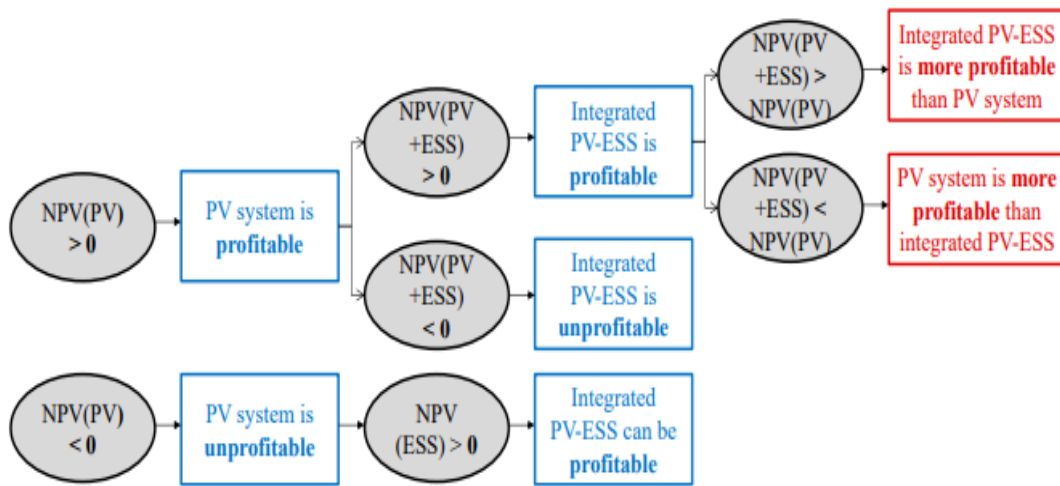


Figure 17: Net Present Value profitability scheme source [11]

To properly understand the expected profitability of the investment, it's needed to take into account one of the most important key assumptions in the cost-benefit analysis of the battery energy storage system (BESS) investment is the evolution of the batteries prices.

In fact, lithium-ion cell prices are expected to continue to fall in the next few years as manufacturing capacity increases [11].

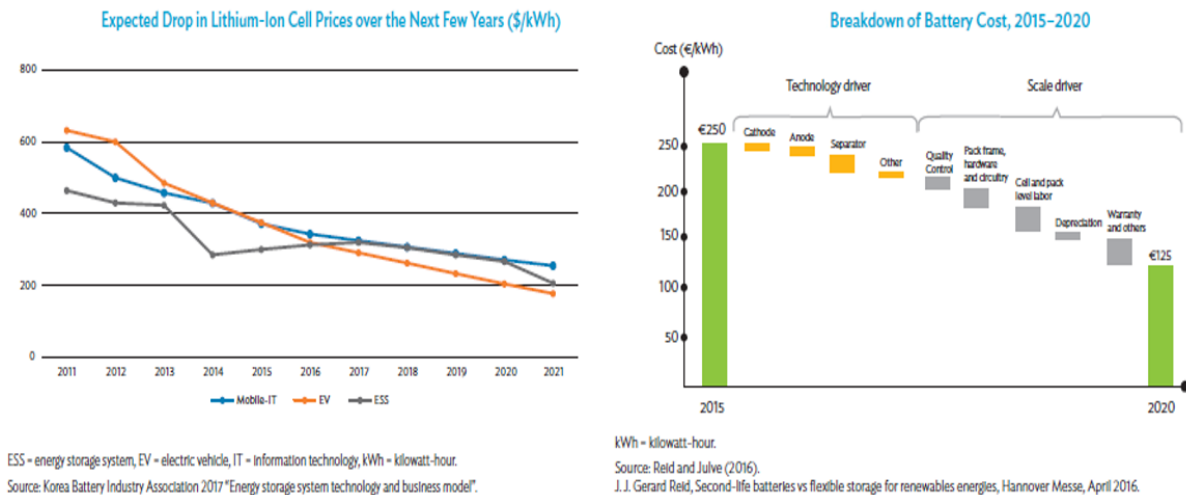


Figure 18: Lithium battery prices charts

7. RESUMING CHART FLOW

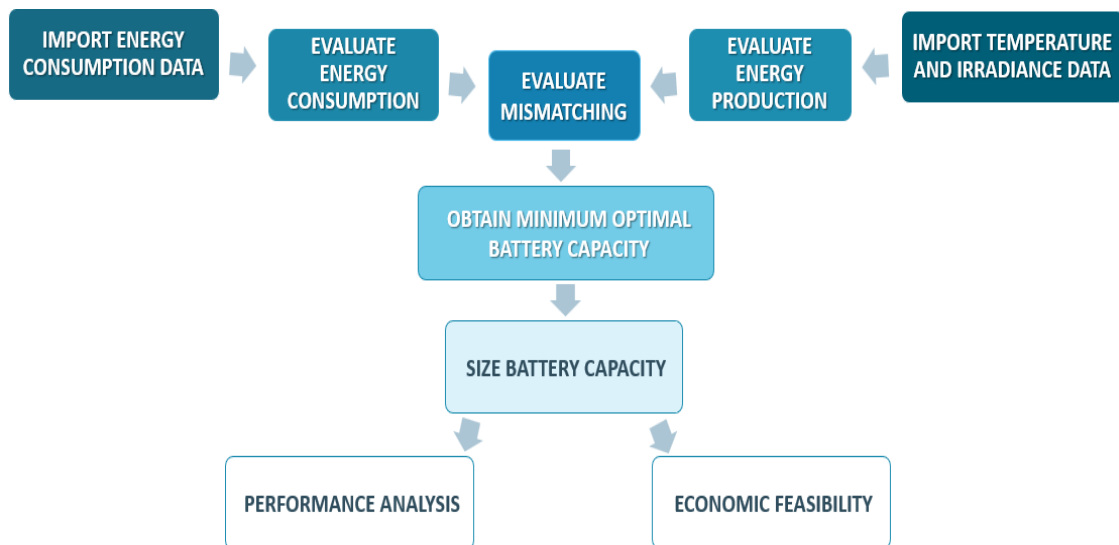


Figure 19: Resuming chart flow

After reconstructing, from several sources in the literature, a method for properly size the battery energy storage system, in order to validate the effectiveness of this whole literature-based approach (resumed into the figure above) it is necessary to test it on real building-like consumptions models and analyse the energy performances and the economical profitability with the related literature-provided tools.

8. LABORATORY CASE STUDY

8.1 LIVING LABORATORY FEATURES

As part of the SOLARISE project, KU Leuven proposed to build a living laboratory at the Technology Campus Ghent, demonstrating a variety of solar technologies. The KU Leuven SOLARISE Living Lab is an extremely useful tool that allows us to test the battery pack sizing procedure and, thanks to its real photovoltaic energy production and load consumption profile, is able to give us realistic feedback on the feasibility of the procedure we are testing.

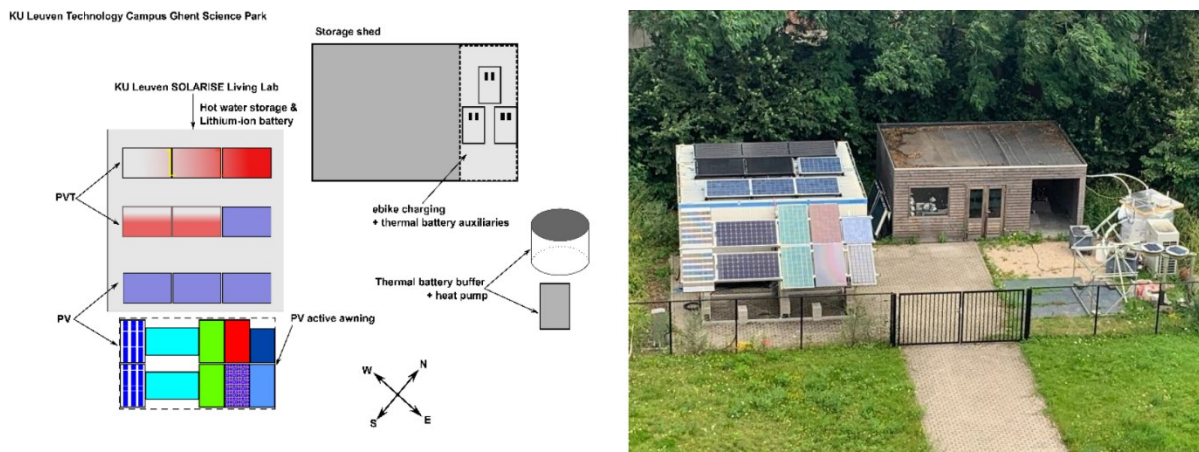


Figure 20: Bird's eye view of the Science Park and indicative placement of SOLARISE components (left), with the corresponding photograph (right). [13]

The overall photovoltaic power installed is 4.665 kWp and the nominal battery capacity installed is 5.76 kWh.

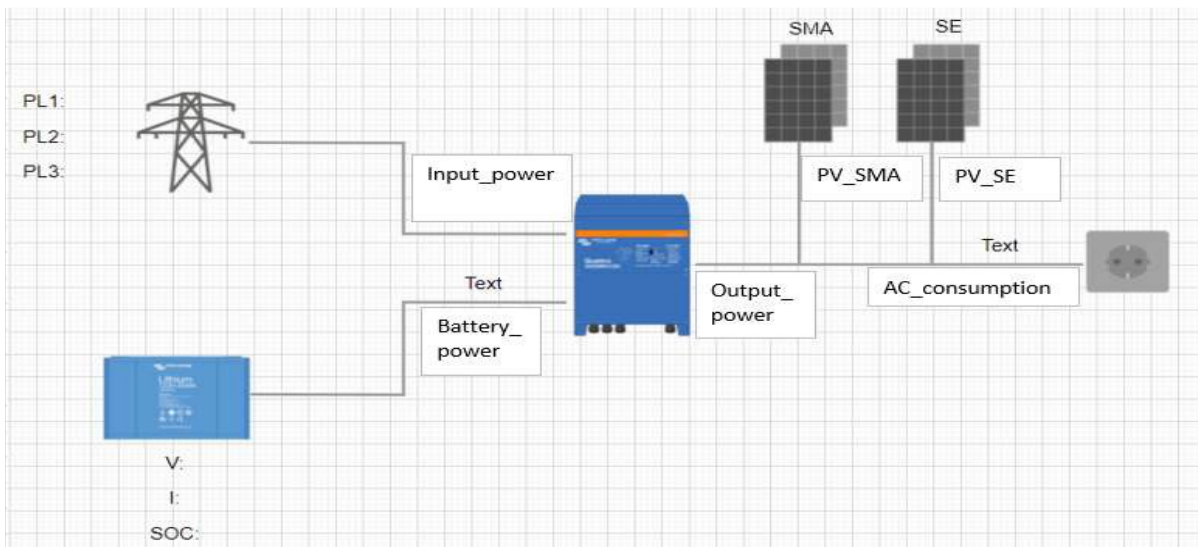


Figure 21: SOLARISE Living lab simplified grid connection scheme. Source: KU Leuven Excel sheet

All the data are collected over a period of a year (from 01/07/2021 until 30/06/2022) and they have 1-minute time interval.

8.2 MISMATCH EVALUATION

Since the measured time interval is 1 minute, all data are grouped together every 60 minutes, so as to obtain a 1-hour time interval, according to two different criteria. The grouping criterion is based on choosing either the average production and consumption or the maximum value, for both load and supply, within each 60-minute time interval.

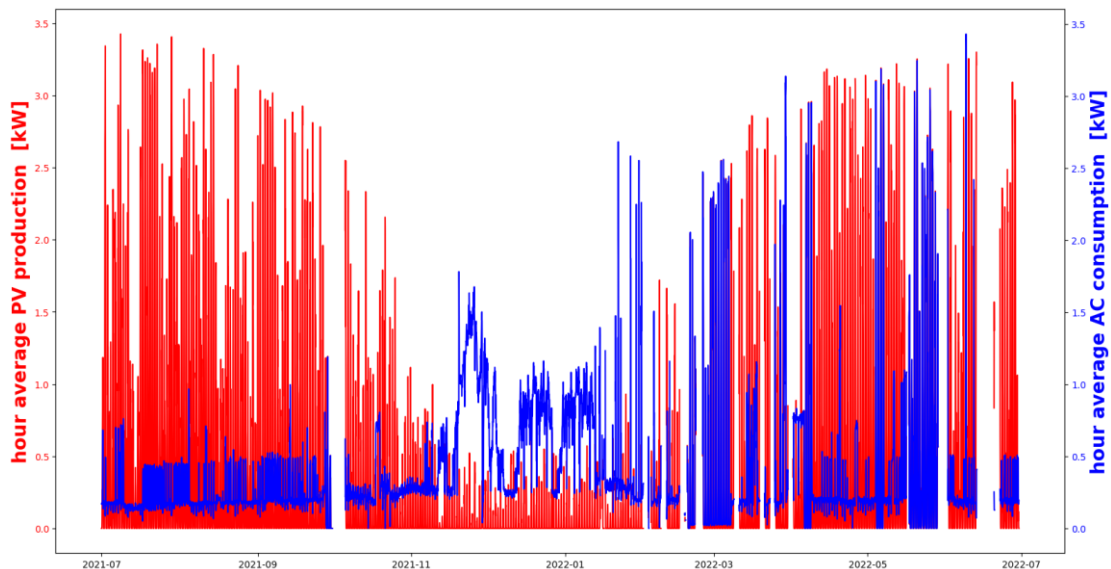


Figure 22: Hourly production and consumption considering their average values per each hour time slot.

```

#evaluate HOUR-AVERAGE cumulative mismatch and add the relative column
new_solaris['(hour-av)_cumul_mismatch'] = hour_mean_mismatch_final["h_av_mismatch"].cumsum()

# find MCEM positive and negative HOUR-AVERAGE Cumulative Mismatch
max_av_positive_cem= max(new_solaris['(hour-av)_cumul_mismatch'])
max_av_negative_cem= min(new_solaris['(hour-av)_cumul_mismatch'])
max_av_CEM= max_av_positive_cem-max_av_negative_cem # [Wh]
max_av_CEM_kWh=(max_av_CEM)/1000 #[kWh]

#find max average cum mismatch between positive max and negative max
max_av_positive_cem_abs=abs(max_av_positive_cem)
max_av_negative_cem_abs=abs(max_av_negative_cem)
if max_av_positive_cem_abs > max_av_negative_cem_abs:
|   MAXIMUM_AV_CEM=max_av_positive_cem_abs
else:
|   MAXIMUM_AV_CEM=max_av_negative_cem_abs
#print('MAXIMUM_AV_CEM :',MAXIMUM_AV_CEM) # KPI 3'

##evaluate HOUR-MAX cumulative mismatch and add the relative column
new_solaris['(hour-max)_cumul_mismatch'] = hour_max_mismatch["h_max_mismatch"].cumsum()

#find MCEM positive and negative HOUR-AVERAGE Cumulative Mismatch
max_max_positive_cem= max(new_solaris['(hour-max)_cumul_mismatch'])
max_max_negative_cem= min(new_solaris['(hour-max)_cumul_mismatch'])
max_max_CEM= max_max_positive_cem-max_max_negative_cem # [Wh]
max_max_CEM_kWh=(max_max_CEM)/1000 #[kWh]

#find max MAX cum mismatch between positive max and negative max
max_max_positive_cem_abs=abs(max_max_positive_cem)
max_max_negative_cem_abs=abs(max_max_negative_cem)
if max_max_positive_cem_abs > max_max_negative_cem_abs:
|   MAXIMUM_MAX_CEM=max_max_positive_cem_abs
else:
|   MAXIMUM_MAX_CEM=max_max_negative_cem_abs
#print('MAXIMUM_MAX_CEM:',MAXIMUM_MAX_CEM) #KPI 3"

```

For each criterion, the average hour mismatch, the maximum hour mismatch, the cumulative average mismatch and the cumulative maximum mismatch are calculated using a testing raw Python code and shown in the pictures below:



Figure 23: mismatch and cumulative mismatch in both case of average and maximum hour value

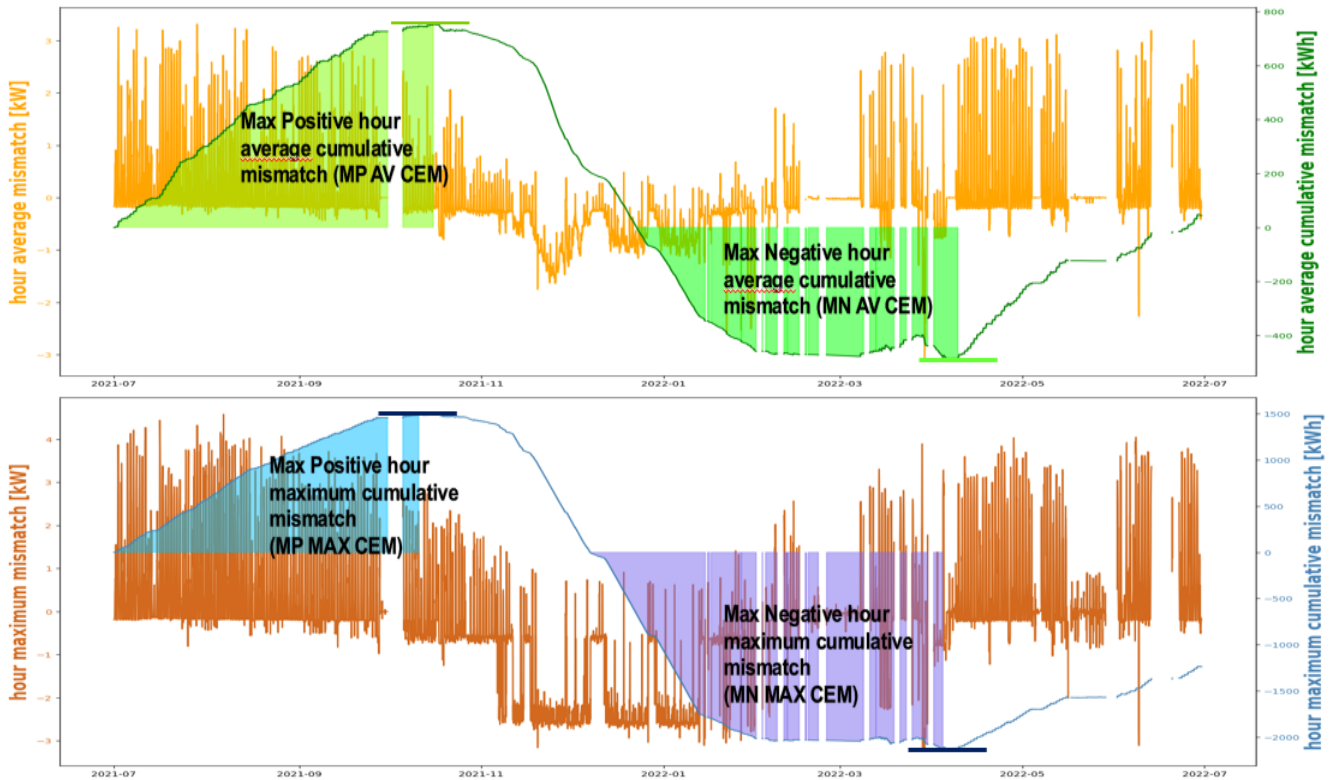


Figure 24: average and maximum mismatch and max positive and negative cumulative mismatch in case of average and maximum hour value

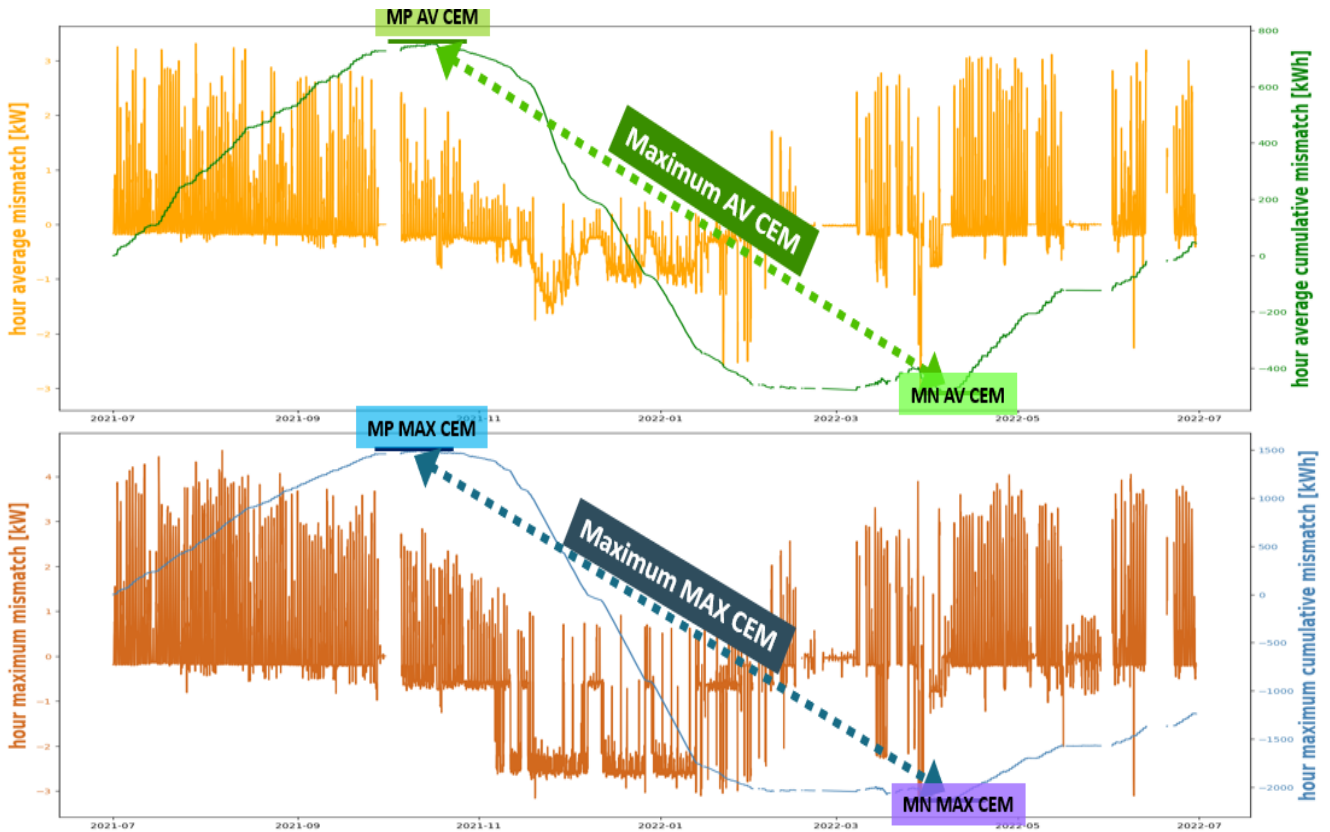


Figure 25: average and maximum mismatch and maximum cumulative mismatch in case of average and maximum hour value

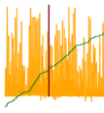
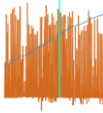
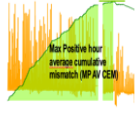

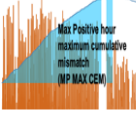
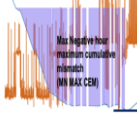
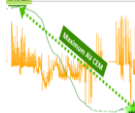

| MAX HOUR AVERAGE MISMATCH [KW] | MAX HOUR MAXIMUM MISMATCH [KW] | MAX POSITIVE HOUR AVERAGE CUMULATIVE MISMATCH [KWh] | MAX NEGATIVE HOUR AVERAGE CUMULATIVE MISMATCH [KWh] | MAX POSITIVE HOUR MAXIMUM CUMULATIVE MISMATCH [KWh] | MAX NEGATIVE HOUR MAXIMUM CUMULATIVE MISMATCH [KWh] | TOTAL MAX HOUR AVERAGE CUMULATIVE MISMATCH [KWh] | TOTAL MAX HOUR MAXIMUM CUMULATIVE MISMATCH [KWh] |
|---|---|---|---|---|--|---|---|
|  |  |  |  |  |  |  |  |
| 3,317 | 4,582 | 753,283 | - 487,015 | 1496,226 | - 2131,762 | 1240,298 | 3627,989 |

Figure 26: table of results

For the battery pack sizing procedure, it has been selected a li-ion battery, from ECO-LFP51.2- ECO ESS energy storage catalogue 2022 [14], with the following main features: a nominal voltage of 51.2 V, a capacity of 300 Ah, physical dimensions of 482 mm * 570 mm * 270 mm and a total weight of approximately 112 Kg. The sizing procedure already described in the previous chapters has been implemented in Python as shown in the following figure:

```

battery_voltage=48          # [v]
battery_capacity=150       # [Ah]
EOL=0.8                   # End Of Life typically expressed in percent of the rated capacity,
                           # commonly 80%.
MDOD=0.78                 # Battery manufacturers rate cells for maximum
MDDOD=0.79                # maximum daily depth of discharge
Kt=1                      # temperature coefficient
EOD=37.5                  #End Of Discharge voltage [V]
cell_recharge_voltage=55.5 # [V]voltage per cell
MAX_discharge_current=100 #max discharge countinuous current

####calculate I_max
I_max=solarise2_df['output_current'].max()
place_I_max=solarise2_df[solarise2_df['output_current']== I_max]
#print(place_I_max)
#at I_max corresponds a load ac consumption of 2604.75 W

####calculate I_min
positive_solarise2=solarise2_df.loc[solarise2_df['output_current']>0.1]
I_min=positive_solarise2['output_current'].min()

place_I_min=positive_solarise2[positive_solarise2['output_current']== I_min]
#print(place_I_min)
#at I_min corresponds a load ac consumption of 766.74 W

AC_load_max=solarise2_df['ac_consumption'].max() # max load power [W]
AC_load_min=solarise2_df['ac_consumption'].min() # min load power [W]

####calculate V max load ---not exactly
V_max_load=766.74/I_min
V_max_load_oversized=AC_load_max/I_min

####calculate V min load ---not exactly
V_min_load=2604.75/I_max
V_min_load_oversized=AC_load_min/I_max

#max_av_CEM
#MAXIMUM_AV_CEM
#max_max_CEM
#MAXIMUM_MAX_CEM
#Max_av_mism_daily

choise_wh=7076   ####----Min_av_mism_daily----###

print('1) For the calucaltion it is choosen [kWh]:',round(choise_wh/1000,2))
C_unj=(choise_wh)/battery_voltage      #[Ah]

max_discharge_adjustment=max(EOL,MDOD,MDDOD)

C_adj1=C_unj/max_discharge_adjustment      #Discharge adjustments

C_adj2=C_adj1*Kt                          #temperature adjustment

C_adj3=C_adj1*(1.2)                       #Design margin factor adjustment

T_functional=C_adj3/I_max                  #functional hour rate

maximum_number_cells=V_max_load/cell_recharge_voltage #max allowed number of cell in series
EOD_calculated=V_min_load/maximum_number_cells      #EOD calculated

#if EOD_calculated > EOD:
| # print('CORRECT!: EOD_calculated >EOD')
#else:
| # print('ERROR! : EOD_calculated < EOD => reduce number series cells')

I_dicharge_T_functional_current=battery_capacity/T_functional

#if I_dicharge_T_functional_current < MAX_discharge_current:
| #print('CORRECT!: I_dicharge_T_functional_current < MAX_discharge_current')
#else:
| # print('ERROR! : I_dicharge_T_functional_current > MAX_discharge_current => reduce battery capacity')

number_parallel_battery = round(C_adj3/battery_capacity) #300Ah rated capacity battery
total_kwh_energy_bess=(battery_voltage*battery_capacity*number_parallel_battery)/1000 #[kWh]

print('PV production per year [kWh]:',round(energy_per_year_PV_kWh,2))
print('ac cons per year [kWh]:',round(ac_cons_per_year_PV_kWh,2))
print('2) number of batteries:',number_parallel_battery)
print('3) BESS energy [kWh]:',total_kwh_energy_bess)

```

Although it was chosen the biggest battery in terms both of voltage and capacity on the catalogue, the cumulative mismatch evaluation provided a too high energy value that would have required an enormous sized battery pack.

In the picture below, the amount of battery needed to fulfil each type of calculated cumulative mismatch. In order to give more sensitivity about the size of the battery pack, the figure also shows the amount of energy of each cumulative mismatch expressed in terms of the equivalent hour during which the batteries must continuously provide an amount of energy equal to the average hourly mismatch and the maximum hourly mismatch.

| | | NUMBER OF BATTERIES | MAX HOUR AVERAGE MISMATCH [KW] MAX HOUR MAXIMUM MISMATCH [KW] | EQUIVALENT NUMBER OF HOUR | MONTHS / DAYS / HOURS | BESS ENERGY [kWh] |
|---|----------|---------------------|--|---------------------------|--------------------------|-------------------|
| TOTAL MAX HOUR AVERAGE CUMULATIVE MISMATCH [KWh] | 1240,298 | 121 | 3,317 | 374 | 15 days 14 hours | 1858,56 |
| | | | 4,582 | 271 | 11 days 7 hours | |
| TOTAL MAX HOUR MAXIMUM CUMULATIVE MISMATCH [KWh] | 3627,989 | 354 | 3,317 | 1094 | 1 month 15 days 14 hours | 5437,44 |
| | | | 4,582 | 792 | 1 month 3 days | |
| MAX ABSOLUTE HOUR AVERAGE CUMULATIVE MISMATCH [KWh] | 753,283 | 74 | 3,317 | 227 | 9 days 11 hours | 1136,64 |
| | | | 4,582 | 164 | 6 days 20 hours | |
| MAX ABSOLUTE HOUR MAXIMUM CUMULATIVE MISMATCH [KWh] | 2131,762 | 208 | 3,317 | 643 | 26 days 19 hours | 3194,88 |
| | | | 4,582 | 465 | 19 days 9 hours | |

Figure 27: detailed table of results including the needed number of batteries

On one hand, this large number of batteries is theoretically capable of guaranteeing full independence from the grid throughout the year, but on the other hand it proves to be an economically unfeasible investment, so a different approach to battery sizing is required than in the literature.

The goal is now to move away from the living lab model to be able to work with a real building consumption profile and a real PV energy production profile, thus finding the optimal minimum size of battery capacity that still improves the energy performance of the building (leading to an increase in both self-sufficiency and self-consumption), but this time respecting the economic constraint whereby the storage system together with the PV plant is a more profitable investment than the PV plant alone.

9. BUILDING CONSUMPTION

The consumption profile of buildings varies widely depending on whether they are residential buildings or office buildings. Initially, the thesis project was designed to provide a real estate company with results on the optimal size of the battery pack for the office buildings they planned to build.

This reason led to the thesis's work focusing on a five floors Belgian office building and using load profiles relating to this, nevertheless the developed python code is capable of working with any type of load profile regardless of the building type.

After quantifying the average annual office building consumptions in Belgium, daily profiles and hourly values for a whole year will be obtained.

9.1 BELGIUM ANNUAL OFFICE BUILDING CONSUMPTIONS

The BPIE (leading independent centre of expertise on the energy performance of buildings-building stock) states that considering only buildings built after 2010 [15], the main energy consumption items of the Belgian office buildings are given as following:

| | |
|---------------------------|---------------------------|
| Space heating | 142,76 kWh/m2/year |
| Domestic hot water | 14,40 kWh/m2/year |
| Space cooling | 20,46 kWh/m2/year |
| Lighting | 41,41 kWh/m2/year |
| TOTAL | 219,03 kWh/m2/year |

The analysis of building consumption presupposes the classification of consumption among the numerous end uses of energy.

A deeper sight through all the items that makes up the final energy consumption of the Belgian office buildings is provided in the report "Benchmark di consumo energetico degli edifici per uffici in Italia"[16].

This survey it has been compiled in May 2019 by *ENEA, Agenzia nazionale per le nuove tecnologie, l'energia e lo sviluppo economico sostenibile*, the Italian energy research and development agency.

The study is based on the actual consumption of 123 energy diagnoses of office buildings located throughout the country with a total annual consumption of approximately 40,000 toe. The data used was extracted by ENEA from the Audit 102 Portal (<http://www.efficienzaenergetica.enea.it/per-le-imprese/diagnosi-energetiche/portale-audit102>), which collects the results of energy diagnoses submitted by large companies.

The energy consumption distribution of the facilities always presents within a building can be carried out into General Services and Auxiliary Services[16]:

General services include:

- Lighting system
- Air conditioning system
- Lifting facilities (lifts, hoists, escalators)
- Domestic hot water system

Auxiliary services, on the other hand, include:

- IT infrastructure, i.e. PCs, printers and other equipment functional to office activities
- Other electrical utilities ancillary to office activities (coffee machines and other equipment in break rooms, etc.)
- Other additional services, i.e. mechanical ventilation, canteens, garage, outdoor lighting
- Server farm, often consist of thousands of computers which require a large amount of power to run and to keep cool.

The percentage consumption distribution for the three main utilities for office buildings without server farms is :

- Air conditioning: 57%
- Lighting: 17%
- FEM utilities (including IT infrastructures and other utilities such as food service floors, electric boilers, etc.): 25%
- Other consumptions: 1%

Domestic hot water is mostly included in air conditioning (if supplied by thermal utilities) or in the so-called FEM utilities if supplied by electric boilers.

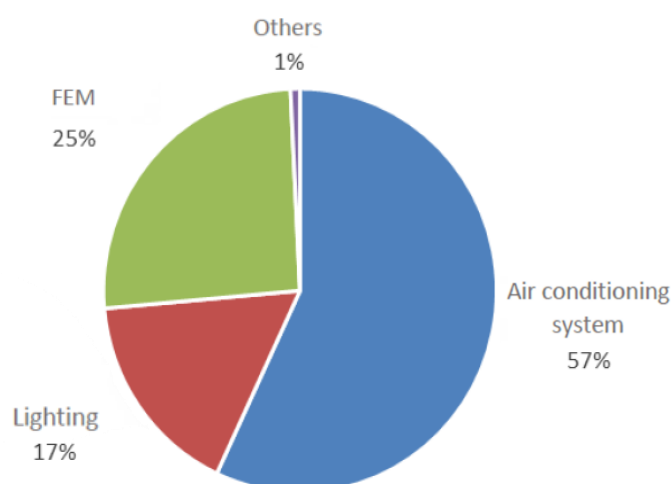


Figure 28: Percentage consumption distribution for the three main utilities for office buildings without server farms [16]

Adding the FEM utilities consumption and the other consumptions to the already known electricity building load (according to the previous chart), it is possible to conclude that the average yearly office building consumptions per square meter in Belgium are:

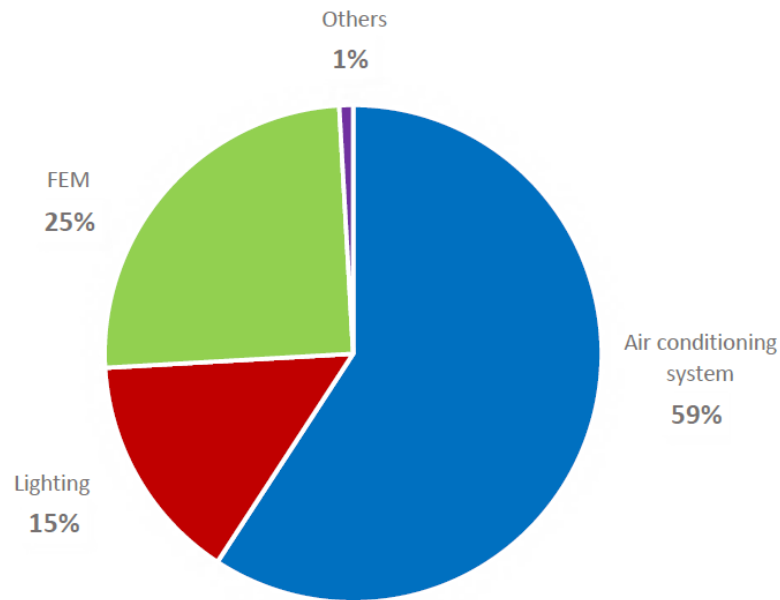


Figure 29: Final percentage consumption distribution

| | | | | |
|--------------------------------|--------------|---------------------------|---------------------------|--------------|
| Air conditioning system | 59,2% | Space heating | 142,76 kWh/m2/year | 51,6% |
| | | Space cooling | 20,46 kWh/m2/year | 7,6% |
| | | Lighting | 41,41 kWh/m2/year | 14,8% |
| FEM | 25% | Domestic hot water | 14,40 kWh/m2/year | 5,2% |
| | | Other FEM | 54,85 kWh/m2/year | 19,8% |
| | | Other consumptions | 2,77 kWh/m2/year | 0,9% |
| | | TOTAL | 277 kWh/m2/year | |

Knowing the average yearly office building consumptions per square meter in Belgium, the next step is to find the average floor area of an office building in order to estimate its consumptions.

Most commercial buildings in Belgium are reported to have 2-3 floors [17]. The following are given as stats for commercial buildings:

| Number of floors | Number of buildings | Proportion of buildings |
|------------------|---------------------|-------------------------|
| 1 floor | 21,383 | 14% |
| 2 - 3 floors | 120,994 | 79% |
| 4 - 5 floors | 10,202 | 7% |
| Over 5 floors | 102 | 0.1% |
| Total | 152,681 | 100% |

Figure 30: Number of floors. Source: Statistics Belgium [17]

The average floor area for the commercial buildings taken into account is [17]:

| Area | Commercial buildings | Proportion |
|------------------------|----------------------|------------|
| Under 45m ² | 4412 | 2.9% |
| 45 - 64m ² | 10287 | 6.7% |
| 65 - 104m ² | 34534 | 22.6% |
| Over 104m ² | 103448 | 67.8% |
| Total | 152,681 | 100% |

Figure 31: Floor area. Source: Statistics Belgium [17]

Given the company's desire to build a building complex consisting of two office buildings, considering also that an office building in Belgium almost always has 2-3 floors and that each floor has, in most cases, an area greater than 104 m², it is possible to synthesize the consumption of the two buildings by assuming a total of 5 floors. The first building will have a total of 2 floors, while the second building will have 3 floors. Using the 360-degree visualization tool that the company provides on their official website to virtually visit their type of office, it is possible to conclude that for a 105 m² office floor there are about 5 employee workstations, about 10 PCs, 1 common TV or projector, 1 coffee machine, 1 printer, and a common meeting table.

Considering the previously found average annual consumption of office buildings of 277 kWh/m²/year, it can be concluded that each floor (with an area of 105 m²) will consume 29 085 kWh/year, for a total consumption of the building complex of 145 450 kWh/year.

Having defined the total annual consumption for the Belgian office building, it is now necessary to assess the hourly consumption behaviour over the course of the year, thus the consumption profile of the office building.

9.2 OFFICE BUILDING CONSUMPTION PROFILE

The review "Towards data-driven energy communities: a review of open-source datasets, models and tools"[18] provides a detailed overview of these open-source datasets, models and tools and the many ways they can be used to optimally design and manage real-world energy communities. Specifically, the table below provides open-source models that can be used to create electricity demand, i.e., load profiles for residential and commercial building users.

| Models | |
|---|--|
| Domain | Sources |
| Synthetic demand data | Load Profile Generator (LPG) [54, 84], Artificial Load Profile Generator (ALPG) [85, 55], House Load Electricity [86, 56], Office Load MATLAB application [87, 57], demandlib [58] |
| Generation potential of renewables (PV) | Sandia Labs PV Performance model Program (PVPMP) [59], PVLIB- Python [60], NREL PVWatts [62], Renewables.ninja [63], feedinlib, atlite |
| Storage models | QuEST [66, 67], OSESMO [68], EnergyBoost [69], Geotab-Sandbox, emobpy [71], vencopy [72], SUMO [73] |
| Power grids and systems | System Advisory Model (SAM) PowerGenome Project [76], Open Energy Modeling Framework (OEMOF) [77, 88], Multi-Vector Simulator [78], renpassGIS [79] |
| Building systems simulation | TEASER, RC BuildingSimulation, City Learn |
| Power systems simulation | Pandapower, MATPOWER, OpenDSS |

Figure 32: Open-source datasets, models and tools [18]

In the current case study, the Office Load MATLAB Application tool is used to generate synthetic load profiles for office buildings.

The paper "MATLAB Applications to Generate Synthetic Electricity Load Profiles of Office Buildings and Detached Houses" [19] presents an overview of the Office Load MATLAB application functionalities, code design, assumptions and limitations, and examples of their potential use in power system education and research. More in detail, the paper presents in the table below the model parameters, their values, and the relative correspondence of the variables.

| Model param. | Categ. value | Quant. values | Var. corresp. |
|--------------------------|--------------|---------------|--------------------|
| Building floors | Few | 1-3 | N_{off} |
| | Average | 4-6 | |
| | Many | 7-9 | |
| Floor size | Small | 440 sqm | A_{floor} |
| | Average | 640 sqm | |
| | Large | 840 sqm | |
| Occupants/floor | Few | 5 | $N_{occ, persons}$ |
| | Average | 15 | |
| | Many | 25 | |
| Thermal inertia | Low | 1200e3 min | C_{in} |
| | | 80e3 qtr | |
| | | 40e3 hhr | |
| | Moderate | 20e3 hrs | |
| | | 3600e3 min | |
| | | 240e3 qtr | |
| | High | 120e3 hhr | |
| | | 60e3 hrs | |
| | | 7200e3 min | |
| Heating/cooling capacity | Low | 6 kW | Q_{hp}^{max} |
| | Moderate | 10 kW | |
| | High | 14 kW | |

Figure 33: Office load application model parameters [19]

Using the collected information regarding the number of company employees in their existing office, the number of floors and the area of each floor, it is possible to generate the consumption profile of the Belgian office building during a year with a time resolution of 1 hour, taking care to respect the annual building consumption constraint previously defined.

This is because the total annual consumption of the office building in output from the Office Load MATLAB application is referenced to the city of Stockholm, Sweden, thus with irradiance and temperatures that may vary from those in Belgium and thus may affect its consumption. This aspect will lead to rescale the data in output from the application to have an annual consumption equal to what we had previously calculated.



Figure34: Office Load MATLAB application screen interface

The output results can be saved to the MATLAB workspace and then it is possible to download them as an Excel file.

The Python code it is written so that the only key input without which the program cannot run is the Excel datasheet of the hourly consumption of a building.

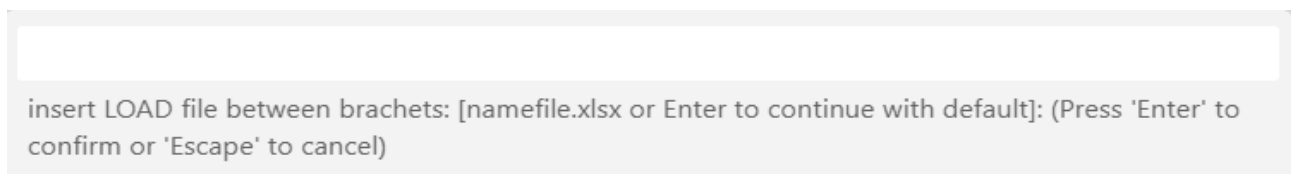


Figure35: Main Code data request of building load Excel file

This is because the code it is designed in such a way as to allow its use for any type of building consumption, including residential one, without limiting its use only to office buildings consumption profiles thus allowing any user to enter their own specific load profile. If the aim of the code had been limited to just the current case study, thus to the consumption of the office buildings, it would have been possible to modify the code

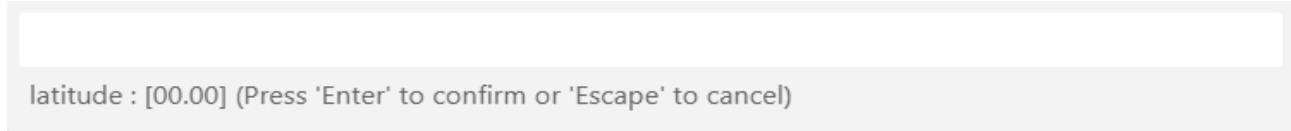
by allowing it to automatically work with the MATLAB application during the Python script execution through the MATLAB Engine API for Python.

The MATLAB Engine API for Python works by providing a Python interface to the MATLAB engine. This engine is a standalone process that runs in the background and can be controlled through the API. The engine is initialized using the "matlab.engine" module in Python, which starts a MATLAB session in the background and returns a reference to the engine. Once the MATLAB engine is running, Python can communicate with it using various API functions. For example, the "matlab.engine.eval()" function allows Python to execute MATLAB commands and retrieve the results. Similarly, the "matlab.engine.workspace" function provides a way to access MATLAB variables from Python. The MATLAB Engine API for Python also provides support for passing data between Python and MATLAB. For example, Python data can be converted to MATLAB arrays using the "matlab.double()" function, and MATLAB arrays can be converted to Python arrays using the "numpy.array()" function. Overall, the MATLAB Engine API for Python provides a powerful and flexible way to combine the capabilities of MATLAB and Python in a single application.

After entering the building consumption profile, it is necessary to evaluate the production profile to provide results about the minimum optimal battery that makes the investment profitable and energy performance analysis.

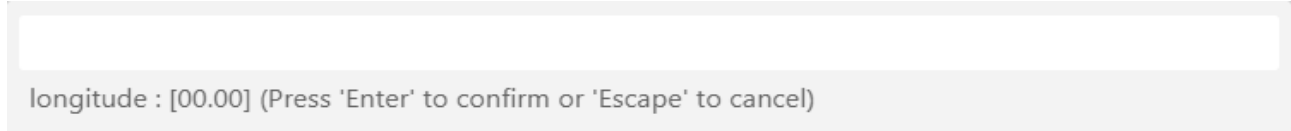
10 SYNTHETIC MODEL OF PHOTOVOLTAIC PRODUCTION

In order to evaluate the production profile, it is necessary to enter some basic fundamental information about the PV plant: latitude, longitude, tilt angle, azimuth angle and the size of the PV array.



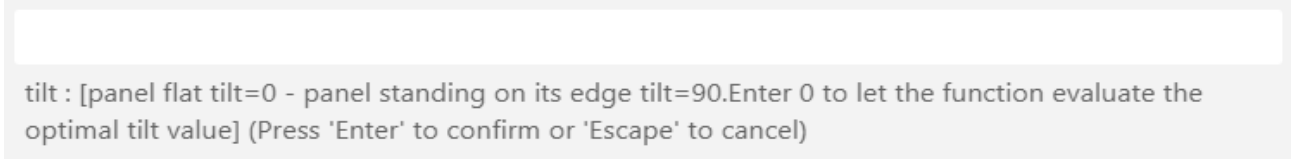
```
latitude : [00.00] (Press 'Enter' to confirm or 'Escape' to cancel)
```

Figure 36: Main Code data request of latitude



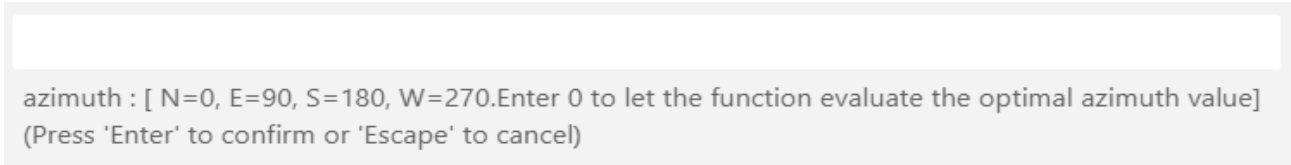
```
longitude : [00.00] (Press 'Enter' to confirm or 'Escape' to cancel)
```

Figure 37: Main Code data request of longitude



```
tilt : [panel flat tilt=0 - panel standing on its edge tilt=90.Enter 0 to let the function evaluate the optimal tilt value] (Press 'Enter' to confirm or 'Escape' to cancel)
```

Figure 38: Main Code data request of tilt angle



```
azimuth : [ N=0, E=90, S=180, W=270.Enter 0 to let the function evaluate the optimal azimuth value] (Press 'Enter' to confirm or 'Escape' to cancel)
```

Figure 39: Main Code data request of azimuth angle



```
PV size : [110000 Wp] (Press 'Enter' to confirm or 'Escape' to cancel)
```

Figure 40: Main Code data request of PV plant size

After entering these data, the code will automatically call a Python function that returns as output a synthetic-estimated AC hourly energy production during a year throughout the NREL's PV models implemented by pvlib named 'PVWatts'[20].

Before explaining in detail how the photovoltaic production model works, it is necessary to make a clarification concerning all the input data needed to model the energy production.

10.1 ARRAY SIZE ESTIMATION

The first four input are essential and must be entered by the user if a production profile is not initially entered. The last data concerning the array size can be omitted and, in this case, the code will automatically call another function that, following the step provided by Aurora Solar, is able to estimate the size of the PV plant basing on the office building energy consumptions previously mentioned and an additional entered input data of the annual average sun hours per day.

The PV size estimation steps are the following:

1. The code takes as input the total annual consumption [Wh].
2. The current result is obtained dividing the total annual consumption by 365 days [Wh/day].
3. The current result is obtained dividing the previous result by the annual daily sun hours per day of the considered location [Wp]
4. The final estimated PV size is obtained dividing the previous result by a typical derate factor of 0.8, according to the National Renewable Energy Laboratory's PVWatts calculator [Wp].

This estimated array size will be entered as input of the model if no PV size value is given by the user.

Concerning the current case study, the estimated array size, basing on the annual office building consumption of 145 450 kWh, is 107.9 kWp. For this calculation it has been entered the input value of 4.6 as the annual average sun hour per day, according to the average of the data provided by World Weather Online[21].

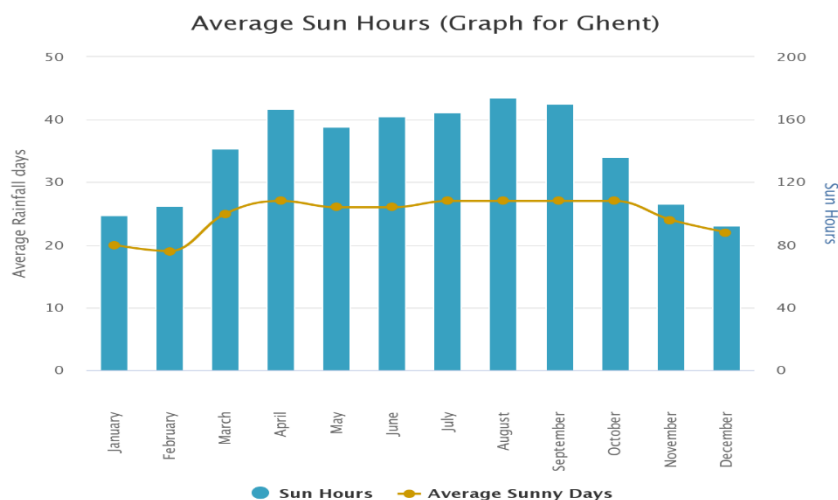


Figure 41: monthly average amount of sun hours in Ghent, Belgium [21]

For a better production modelling is obviously recommended to run again the code entering as PV size data an array size as close as possible to the estimated one but obtained considering a commercial power panel size.

10.2 PVWatt MODEL

Several research[22] referenced PV_LIB[23] as a very useful tool for synthetic models of photovoltaic energy systems.

The PV_LIB Toolbox, originally ported from the PVLIB MATLAB toolbox developed at Sandia National Laboratories, provides a set of well-documented functions for simulating the performance of photovoltaic energy systems. Currently there are two distinct versions (pvlib-python and PVLIB for Matlab) that differ in both structure and content.

In 2019, pvlib python became an Affiliated Project with NumFOCUS, a non-profit organization in the United States that aim to promote sustainable high-level programming languages and open code development through several educational programs such as the so called 'PyData'.

The educational program 'Pydata2021: Solar PV Modeling' provides a well detailed procedure to model a photovoltaic plant using the pvlib python tool, considering step by step all the factors that affect the photovoltaic plant production:

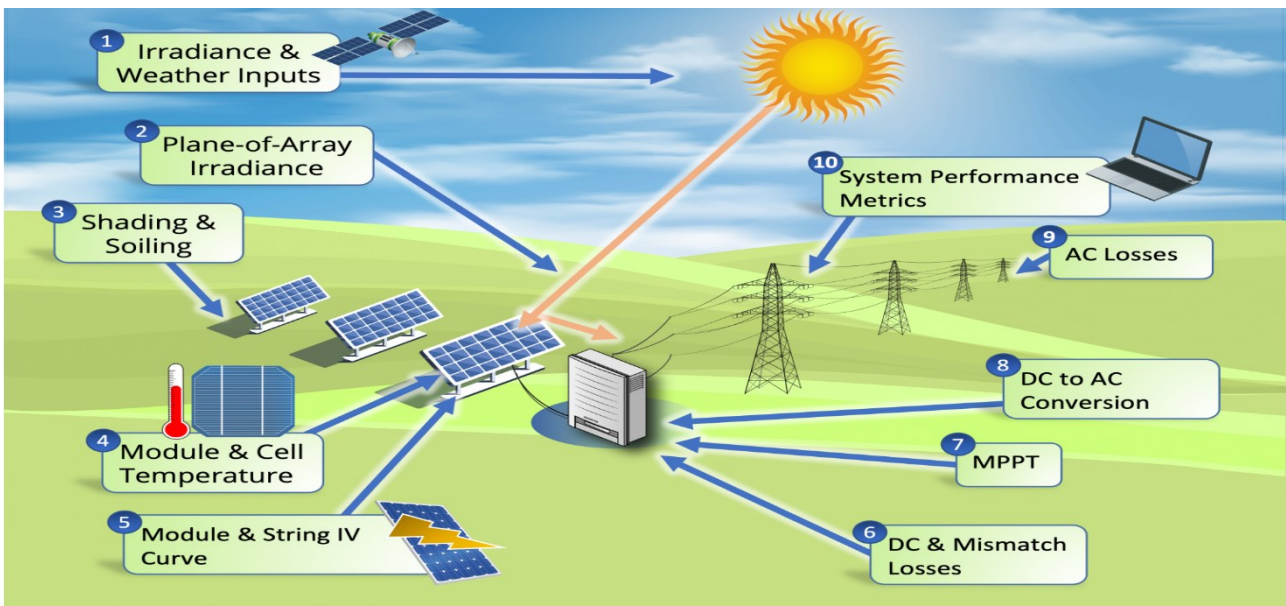


Figure 42: Sandia modeling steps[24]

The *Tutorial 3* of the 'Pydata2021: Solar PV Modeling' [25] shows how to use pvlib to model an array's output power given the POA irradiance and the cell temperature. This is possible due to pvlib's implementation of the NREL model called "PVWatts".

The PVWatts model requires only two array parameters that are the array size (nameplate capacity) and the array's efficiency change with cell temperature.

The cell temperature response parameter, often called the module's temperature coefficient, determines the efficiency loss for a temperature increase. Typical temperature coefficients range from $-0.5\%/^{\circ}\text{C}$ to $-0.2\%/^{\circ}\text{C}$ [25].

After importing pvlib python libraries, the first most important modelling step consist of reading the Typical Meteorological Year (TMY) datasets, that are intended to represent the weather for a typical year at a given location (Ghent, Belgium, latitude=51.034, longitude=3.695).

TMY datasets provide hourly solar irradiance, air temperature, wind speed, and other weather measurements for a hypothetical year (built, in this case, from yearly data that range from 2005 up to 2020) that represents a “median year” for solar resource.

```
import os # for getting environment variables
import pathlib # for finding the example dataset
import pvlib
import pandas as pd # for data wrangling
import matplotlib.pyplot as plt # for visualization
print(pvlib.__version__)

import numpy as np

TmY_df = pd.read_csv('tmy_51.034_3.695_2005_2020 (1).csv')
datastrana= pd.to_datetime(TmY_df['time(UTC)'], format = '%Y%m%d:%H%M')
TmY_df=pd.DataFrame(TmY_df)
newfakeindex=pd.date_range(start='2019-01-01 00:00', end='2019-12-31 23:00', freq='H')

#real fictious values from 2005 to 2020
TmY_df.set_index(datastrana,inplace=True)

#time index resample to obtain a horly year values, for ex. jan 2019-jan 2020
TmY_df_2019=TmY_df.copy()
TmY_df_2019.set_index(newfakeindex,inplace=True)

from pvlib import location
from pvlib import irradiance

from matplotlib import pyplot as plt

tz = 'GMT'
lat, lon = 51.034, 3.695

# Create location object to store lat, lon, timezone
site = location.Location(lat, lon, tz=tz)
```

In order to model the effective irradiance incident on the plane of the array (POA), that is dependent upon several factors including the sun position, the array Orientation (fixed or tracking) and the irradiance components (direct and diffuse), will be used the convenient wrapper function “pvlib.irradiance.get_total_irradiance”.

```
def get_irradiance(site_location, tilt, surface_azimuth):

    times = TmY_df.index
    solar_position = site.get_solarposition(times)

    POA_irradiance = irradiance.get_total_irradiance(
        surface_tilt=tilt,
        surface_azimuth=surface_azimuth,
        dni=TmY_df['Gb(n)'],
        ghi=TmY_df['G(h)'],
        dhi=TmY_df['Gd(h)'],
        solar_zenith=solar_position['apparent_zenith'],
        solar_azimuth=solar_position['azimuth'])
    # Return DataFrame with POA
    return pd.DataFrame({'POA': POA_irradiance['poa_global']})

# Get irradiance data for summer and winter solstice, assuming 40 degree tilt
# and a south-est facing array (generally south is 180° but on PVGIS is 0 so optimum is
# slope 40, azimuth -2 (sud-est) so --> azimuth 178)

total_irradiance = get_irradiance(site,40, 178)
```

The TMY irradiance effectively hitting the plane of the array is shown in the figure below:

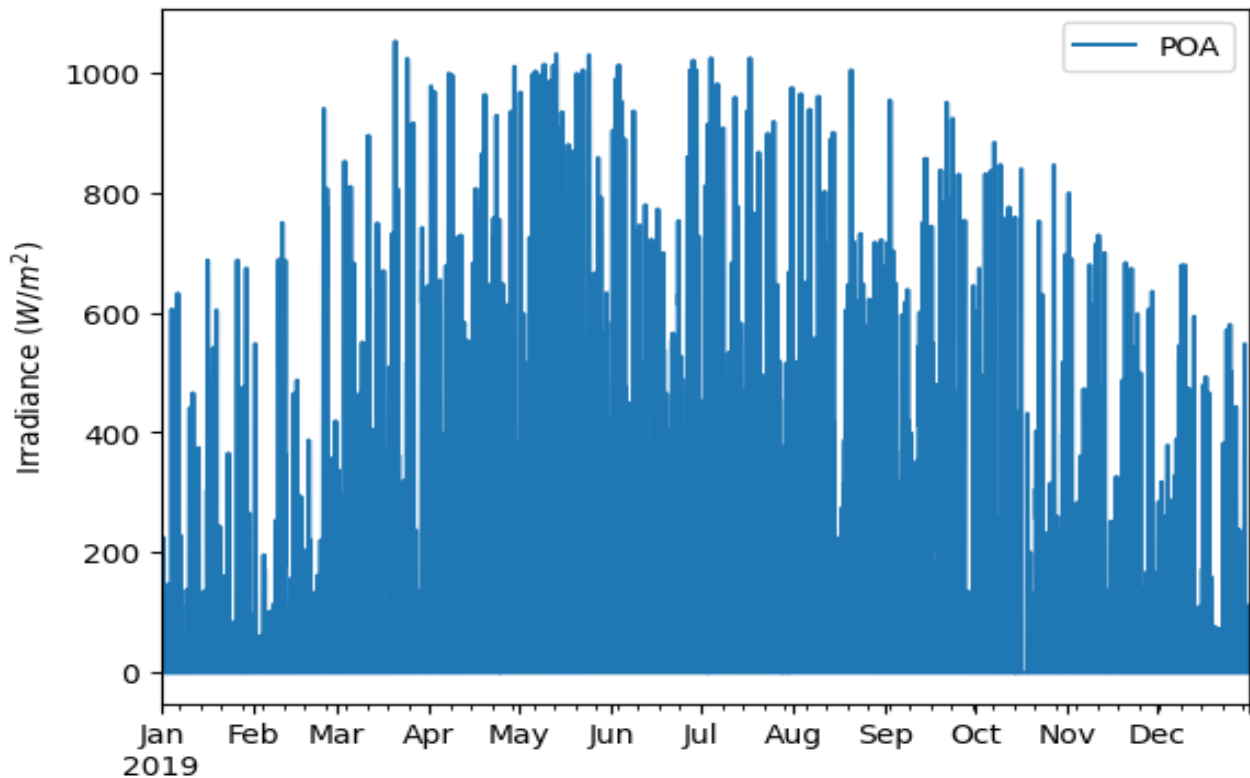


Figure 43: TMY plane of array irradiance

The PVWatts model use a simplified inverter model `pvlb.inverter.pvwatts(pdc, pdc0)` to return the AC output given DC output (`pdc`) and the DC limit (`pdc0`), which is the AC rated power over the nominal inverter efficiency.

```
#Recall we assumed a 110kW array, so we'll continue the hypothetical case and assume an
# AC size of 91.6 kW, a DC/AC ratio of 1.2. The default PVWatts nominal inverter efficiency
# is 0.96 which we use to get pdc0.

pdc0 = 91600/0.96 # W
ac = pvlb.inverter.pvwatts(array_power, pdc0)
PV_prod_ac_2019=pd.DataFrame(ac, columns=['PV_prod_ac_model'])
plt.rcParams['font.size'] = 14
#ax = ac.resample('D').sum().plot(figsize=(15, 10), label='AC')
ac_kw=ac/1000
ax = ac_kw.plot(figsize=(15, 10), label='AC',color='red')
#array_power.resample('D').sum().plot(ax=ax, label='DC')
#array_power.plot(ax=ax, label='DC')
plt.title('AC Power')
plt.ylabel('[kW]')
plt.ylim([0,110])
minor_ticks_upest=np.linspace(0,110,45)

plt.yticks(minor_ticks_upest, minor=True)

plt.grid(True,which='minor' ,color='grey',linewidth=0.2)
plt.legend()
```

The model gives us as output the synthetic AC hourly energy production during a year (the yearly production is about 117 MWh) of the 110 kWp estimated array size:

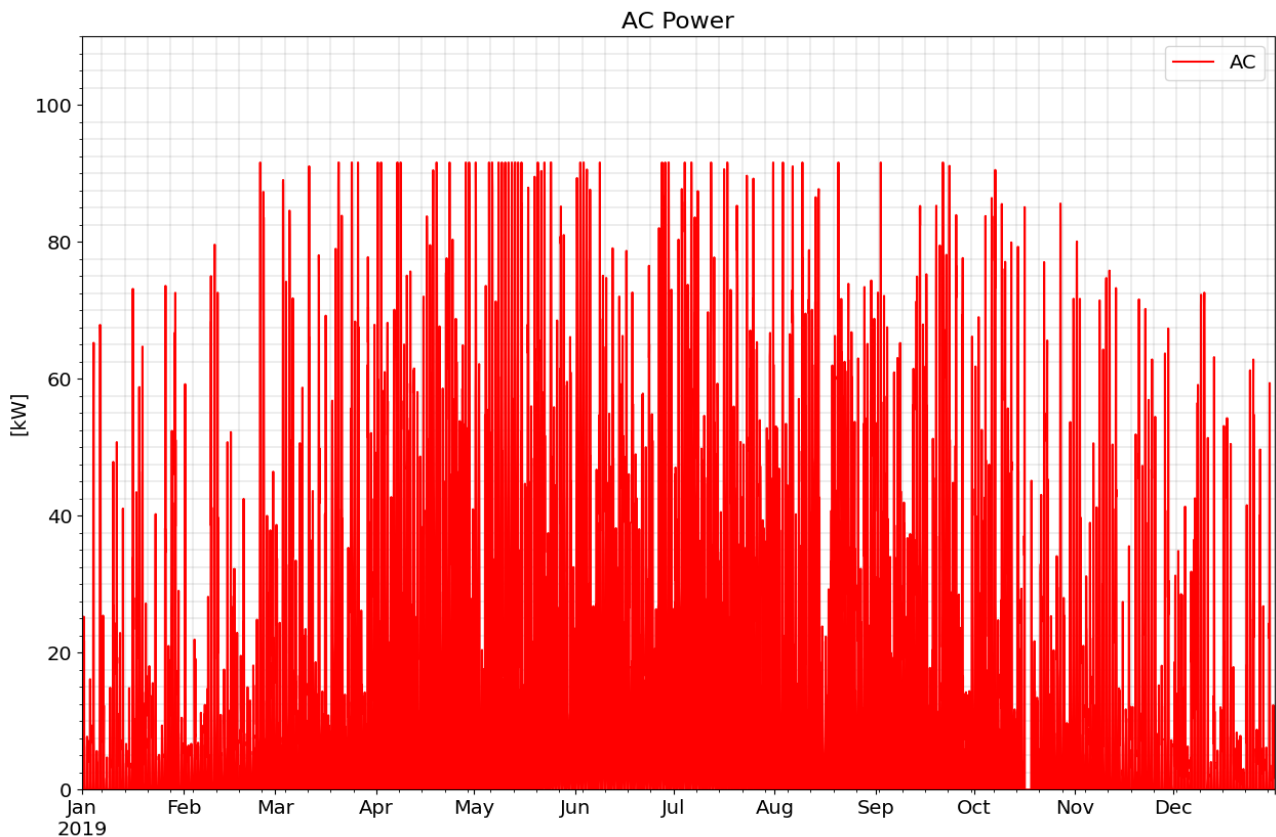


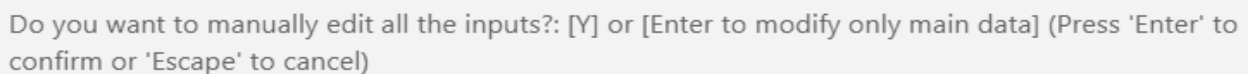
Figure 45: AC power generated by the PV plant.

After having understood the importance of the building hourly consumption Excel file as an essential input data for the code to work and how this allows us to obtain the estimated hourly production of the PV plant, it is now necessary to understand the actual operation of the main core of the Python code once the above-mentioned building production and consumption data are received as input.

11. MAIN CODE

The working principle of the code (APPENDIX 1) is simple and intuitive. When the code is launched, it is necessary only to enter some required data as input, and afterwards the code will be able to make its evaluations and provide results without the user having to interact with the terminal by intervening in any way on the already started code.

The first real request from the code, even before inserting the excel file of the building's consumption, concerns the user's ability to choose whether to insert only the essential data truly necessary for calculating the results or even modify other minor data (such as the annual degradation of the photovoltaic panel, the annual operation and maintenance costs, etc...) in case the user is aware of these more specific information and intends to improve the accuracy of the final result.



```
Do you want to manually edit all the inputs?: [Y] or [Enter to modify only main data] (Press 'Enter' to confirm or 'Escape' to cancel)
```

Figure 46: Main Code choice request of entering just main or all the input data

Whatever the user's choice, it is always possible to either press the Enter key to confirm the entered value or to skip the input request if nothing is entered. In this last case, the code will use the input data already set as default. These default data are referred to the city of Ghent, Belgium, as it represents the initial target of the thesis project for which the code was developed.

By pressing the Esc key, on the other hand, the user can decide to disable the input requests. This led to exit the code and generating the error screen shown below:

```
An exception has occurred, use %tb to see the full traceback.
```

```
SystemExit
```

This choice option is achieved through the following “get_input” function:

```
import keyboard
import sys

def get_input(prompt, default_value):
    value = default_value
    while True:
        try:
            value = float(input(prompt) or default_value)
            break # exit loop if valid input is provided
        except keyboard.is_pressed('Esc'):
            sys.exit()
        except ValueError:
            print("Invalid input. Please enter a valid number.")
    if keyboard.is_pressed('Esc'):
        sys.exit()
    return value
```

After entering the Excel file of consumption and other input data, the code calls an external function called "Simple_PVWatts_array_power" (APPENDIX 2) that returns as output the DataFrame (a two-dimensional tabular data structure with labelled rows and columns, which is provided by a software library written for the Python programming language called Pandas) of the hourly output of the PV system over a year, the DataFrame of the hourly building consumption over a year, the total amount of energy produced over a year, the total amount of energy consumed over a year, and the size of the PV plant used for the calculations. The latter output could have been chosen by the user as the initial input or estimated by the function mentioned earlier.

This "Simple_PVWatts_array_power" Python function, among other minor manipulation data tasks, is mainly responsible for generating the synthetic profile of PV production using the "PVWatts" model extensively described earlier. The output DataFrame of production and consumption will be entered as inputs in the "bestbattery" function, as they are essential for the evaluation of the State of Charge (SOC) of the battery and the economic analysis conducted on it. The "bestbattery" function takes several data as inputs, including the size of the battery storage system, and provides as output the DataFrame of the hourly SOC of the battery over a year and some economic indicators such as the Net Present Value (NPV), the Internal Rate of Return (IRR) and the Pay Back Time (PBT).

More In detail, there are four outcomes: the IRR of the investment including both the PV plant and the battery energy storage system (BESS), the IRR of the only PV plant investment, an array of together NPV/IRR/PBT of the only PV plant investment and NPV/IRR/PBT of the BESS+PV investment ($[[NPV_onlyPV, IRR_onlyPV, PBT_onlyPV], [NPV_BESS+PV, IRR_BESS+PV, PBT_BESS+PV]]$) and the DataFrame of the battery SOC.

Next, the code searches for the battery size that maximizes the internal rate of return (IRR) value of the PV + Battery Energy Storage System (BESS) investment and, moreover, the code uses a binary search to find the minimum battery capacity needed to achieve the IRR that makes the investment of PV + BESS either at least equally or more profitable than the PV plant alone. The code then returns results for the optimal battery size and the battery size that maximizes the IRR.

The last analysis of the code is related to the evaluation of the energy performance of the PV + battery system, providing, with the help of a graph, the self-sufficiency and self-consumption values of the building with integrated PV plant before and after the addition of the battery storage system. The battery size chosen for the performance analysis is the minimum battery capacity needed to achieve the IRR that makes the investment at least equally or more profitable than the PV plant alone.

Another flexibility feature of the code includes the possibility for the user to choose the size of a test battery and perform the economic and energy performance analyses on it.

The choice of test battery can be made by entering the selected size as a value in the initial input data requests as shown in the figure below:

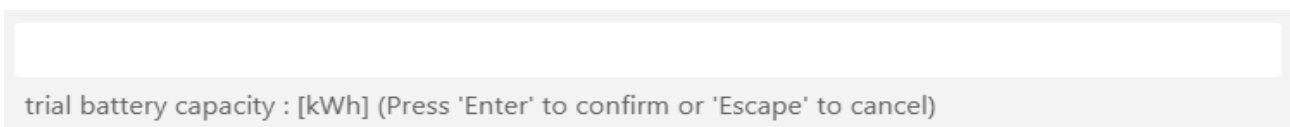


Figure 47: Main Code choice request to run the code with a test battery capacity

In this case, the results and the energy performance chart will refer to the test battery size instead of the minimum capacity that makes the investment profitable. If no test battery value is entered, the graph and the self-consumption and self-sufficiency values will be referenced as default to the minimum capacity that makes the investment profitable. All data obtained as results during code simulation are saved in a user-downloadable Excel file.

11.1 "BESTBATTERY" FUNCTION

The first part of the python function "bestbattery" concerns the construction of an algorithm capable of evaluating the SOC for each time step of the production and consumption DataFrames, thus capable of reconstructing the hourly profile of the SOC over a year.

Specifically, the first part of the code focuses on creating lists and initializing some variables, such as the initial energy stored in the battery (Eb_tot0) and the total battery capacity (Eb_tot) expressed in Wh. Eb_tot0 is defined multiplying Eb_tot with the initial SOC of the battery (SOC0).

```
Eb_tot0=SOC0*Bess_tot_size*1000 #[Wh] battery size
Eb_tot=Bess_tot_size*1000
SOC=[SOC0]
Eb_new=[Eb_tot0]
Eg=[0]
check_batt_en_supplied=[0]
```

The SOC0, the minimum SOC (SOC_min) and the maximum SOC (SOC_max) can be modified entering their value in the initial input data requests, otherwise the code will accept as default SOC0=90%, SOC_min=20% and SOC_max=100%.

Immediately after the definition of the variables, the code is responsible for evaluating the SOC of the battery for each time interval using the following algorithm.

```
for ehm in Em:
    SOC_in=SOC[-1]
    Eb=(SOC_in*Eb_tot)+ ehm
    if ehm>=0:
        if (Eb/Eb_tot)< SOC_max:
            Eb_new.append(Eb)
            SOC.append(Eb/Eb_tot)
            Eg.append(0)
            check_batt_en_supplied.append(Eb_new[-1]-Eb_new[-2])
        else:
            x_ehm=(SOC_max-SOC_in)*Eb_tot
            Eb_new.append(SOC_max*Eb_tot)
            SOC.append(SOC_max)
            Eg.append(ehm-x_ehm)
            check_batt_en_supplied.append(Eb_new[-1]-Eb_new[-2])
    else:
        if (Eb/Eb_tot) > SOC_min:
            Eb_new.append(Eb)
            SOC.append(Eb/Eb_tot)
            Eg.append(0)
            check_batt_en_supplied.append(Eb_new[-1]-Eb_new[-2])
        else:
            x_ehm=(SOC_min-SOC_in)*Eb_tot
            Eb_new.append(SOC_min*Eb_tot)
            SOC.append(SOC_min)
            Eg.append(ehm-x_ehm)
            check_batt_en_supplied.append(Eb_new[-1]-Eb_new[-2])
```

The involvement of the battery storage system is intended so that the energy mismatch between production and consumption of the building could be managed with the aim of ensuring a better self-consumption and self-sufficiency. Due to this reason, for each hourly energy mismatch (ehm) item within the list of all hourly energy mismatches during the year (Em), the code evaluates the current battery charge (Eb) by taking into

account its initial charge (referring to its previous value obtained by considering the SOC value of the previous time step) and add to it the mismatch value of the current time step (represented by ehm).

The value of ehm can be either positive or negative depending on whether, of course, the output of the PV system is respectively greater or smaller than the energy consumption of the building and, for this reason, the current charge of the battery can be greater than zero or less than zero. However, the fact that ehm is less than zero does not necessarily determine the fact that Eb is less than zero.

In the case where ehm is greater than zero, Eb is obviously positive but the battery charge may remain below the maximum SOC or there is a risk that it will exceed it.

```
if ehm>=0:
    if (Eb/Eb_tot)< SOC_max:
        Eb_new.append(Eb)
        SOC.append(Eb/Eb_tot)
        Eg.append(0)
        check_batt_en_supplied.append(Eb_new[-1]-Eb_new[-2])
    else:
        x_ehm=(SOC_max-SOC_in)*Eb_tot
        Eb_new.append(SOC_max*Eb_tot)
        SOC.append(SOC_max)
        Eg.append(ehm-x_ehm)
        check_batt_en_supplied.append(Eb_new[-1]-Eb_new[-2])
```

If the maximum SOC is not exceeded (actual Eb/Eb_tot < SOC_max), then the battery will be assigned the current charge (in this case Eb_new =Eb), while the new SOC will be assigned the value of the current SOC.

```
if (Eb/Eb_tot)< SOC_max:
    Eb_new.append(Eb)
    SOC.append(Eb/Eb_tot)
    Eg.append(0)
    check_batt_en_supplied.append(Eb_new[-1]-Eb_new[-2])
```

Eg represents the gain of energy from the grid. Eg can be positive, in the case where the positive mismatch is so large that the battery is charged up to SOC_max and there is still a surplus of energy that is fed into the grid; or it can be negative, in the case where the negative mismatch is so large that the battery is discharged up to SOC_min and there is still a deficit of energy that needs to be taken from the grid.

When, therefore, Eb and ehm are both positive and the maximum SOC is not exceeded then Eg will be zero since all the positive mismatch energy will be used to charge the battery. The check_batt_en_supplied provides a double confirmation regarding the energy exchanged to the grid by representing the amount of energy actually absorbed or given up by the battery and is calculated by simply subtracting the actual battery charge value to the previous time step battery charge value. In this way the sum of ehm,Eb,Eg and check_batt_en_supplied is always equal to zero.

In the case where Eb and ehm are both positive but the maximum SOC is exceeded (actual Eb/Eb_tot > SOC_max) the battery will be assigned the maximum charge (in this case Eb_new =Eb_tot*SOC_max), while the new SOC will be assigned the value of the maximum SOC.

```
else:
    x_ehm=(SOC_max-SOC_in)*Eb_tot
    Eb_new.append(SOC_max*Eb_tot)
    SOC.append(SOC_max)
    Eg.append(ehm-x_ehm)
    check_batt_en_supplied.append(Eb_new[-1]-Eb_new[-2])
```

The `x_ehm` item represents the amount of energy mismatch used to charge the battery up to the maximum SOC, and its value is obtained by simply subtracting the current capacity of the battery from its maximum capacity. Consequently, the energy surplus fed into the grid (`Eg`) will be obtained by subtracting `x_ehm` from the current mismatch value `ehm`. The `check_batt_en_supplied` value could be defined by simply assigning it the value of `x_ehm` but for convenience and invariance in the result it was calculated in the same way as described above.

In the case where `ehm` is negative, `Eb` could be positive and, in this case, the battery charge may remain within the range between the minimum SOC and the maximum SOC, or it may fall below the minimum SOC. If `ehm` and `Eb` are both negative the current SOC will necessarily be less than the minimum SOC

```

else:
    if (Eb/Eb_tot) > SOC_min:
        Eb_new.append(Eb)
        SOC.append(Eb/Eb_tot)
        Eg.append(0)
        check_batt_en_supplied.append(Eb_new[-1]-Eb_new[-2])
    else:
        x_ehm=(SOC_min-SOC_in)*Eb_tot
        Eb_new.append(SOC_min*Eb_tot)
        SOC.append(SOC_min)
        Eg.append(ehm-x_ehm)
        check_batt_en_supplied.append(Eb_new[-1]-Eb_new[-2])

```

In the case where `ehm` is negative but `Eb` is positive and the current SOC value is not less than the minimum SOC (actual `Eb/Eb_tot > SOC_min`), then every battery characteristic data will be calculated as in the case shown earlier where `ehm` is positive and the maximum SOC is not exceeded.

On the other hand, when `ehm` is negative but `Eb` is positive and the current SOC value goes below than the minimum SOC, all the battery characteristic data will be calculated as done in the case previously shown where `ehm` is positive and the maximum SOC is exceeded.

The reason it was not necessary to define a third case relating to the condition where both `ehm` and `Eb` are less than zero is due to the fact that since the battery charge of the previous time stamp cannot fall below the minimum value defined by the `SOC_minimum` (because of the way the code was constructed), if we add `ehm < 0` to this minimum charge such that `Eb < 0` then the condition (actual `Eb/Eb_tot > SOC_min`) will never be verified. This happens obviously because `Eb` is negative and therefore the entire `Eb/Eb_tot` ratio is while `SOC > 0`. This result causes the study of battery charging to fall into the previously discussed case where `ehm` is negative but `Eb` is positive and the current value of SOC is less than the minimum SOC. This algorithm leads to the creation of the DataFrame of the battery SOC for each hour of the year.

Before it was finally used, the code was subjected to a validation test by comparing its behaviour with that related to the actual SOC profile of the SOLARISE Living Lab battery at the Ghent Technology Campus.

The two main indicators used to assess the error between the actual measured value of the SOLARISE Living Lab battery SOC and the value of the battery SOC estimated by this algorithm are the Mean Absolute Error (MAE) and the Mean Absolute Percentage Error (MAPE). These error measurements were made using the functions of scikit-learn [26](a free software machine learning library for the Python programming language).

In statistics, Mean Absolute Error (MAE) is a measure of errors calculated as the sum of absolute errors divided by the sample size and therefore returns results in the same scale as the measured data:

$$MAE = \frac{\sum_{i=1}^n |x_i - y_i|}{n} \quad (14)$$

The Mean Absolute Percentage Error (MAPE) is a measure of errors calculated as the sum of absolute errors each divided by the true value and the entire sum of the relative errors is divided by the sample size as in the MAE. For this reason, it returns results as percentage error of the true value.

$$MAPE = \frac{1}{n} \sum_{i=1}^n \left| \frac{x_i - y_i}{x_i} \right| \quad (15)$$

In both equations the true value is represented by x_i , the predicted value is y_i and n represent the total number of data points.

The error was evaluated for each week of the year and each for different arbitrary minimum SOC values of 20%, 30%, 40% and 50%. as shown in the figures below:

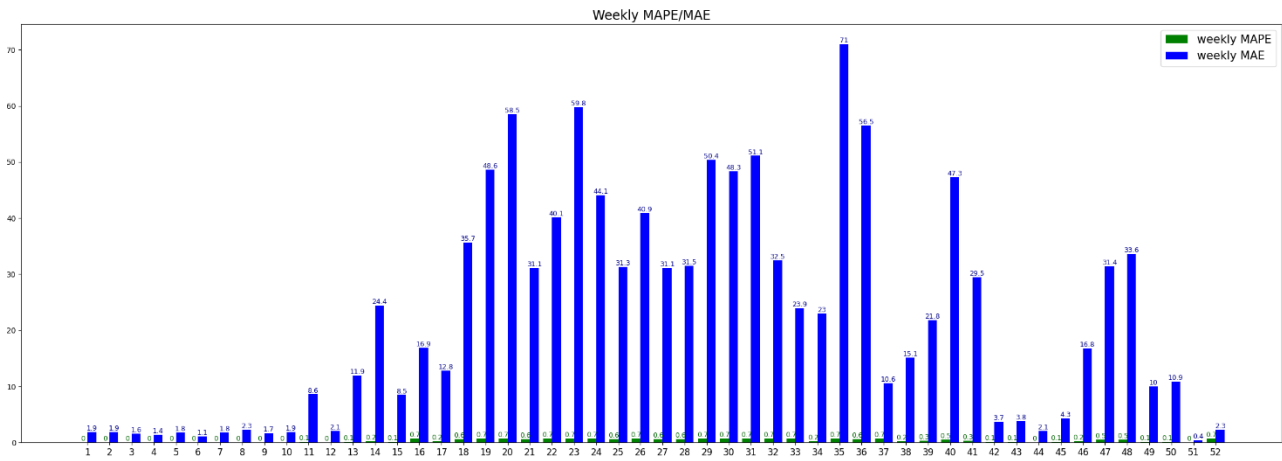


Figure 48: SOC_min= 20.0 %, total MAE: 22.22, total MAPE: 34.62 %

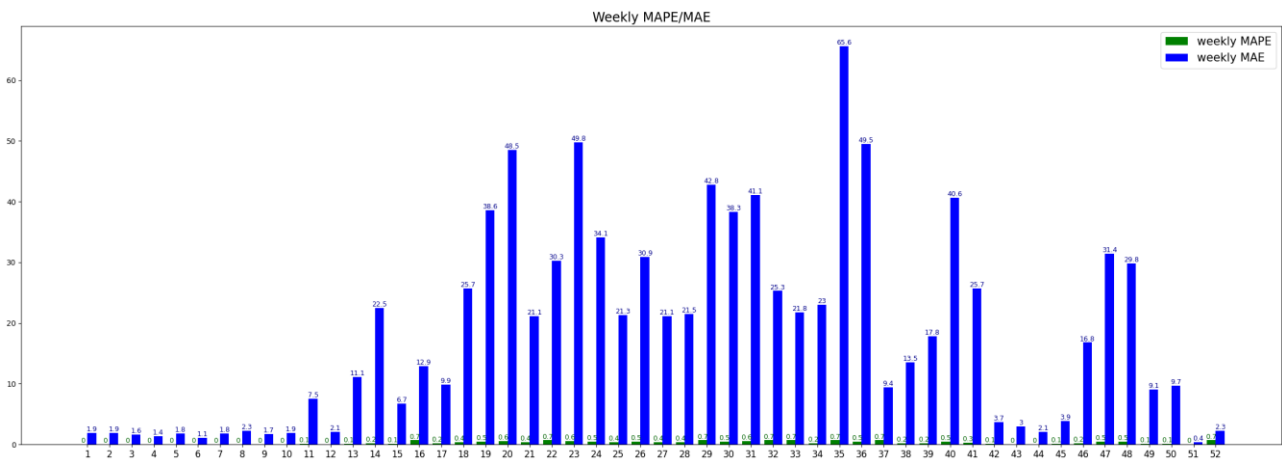


Figure 49: SOC_min= 30.0 %, total MAE: 18.45, total MAPE: 30 %

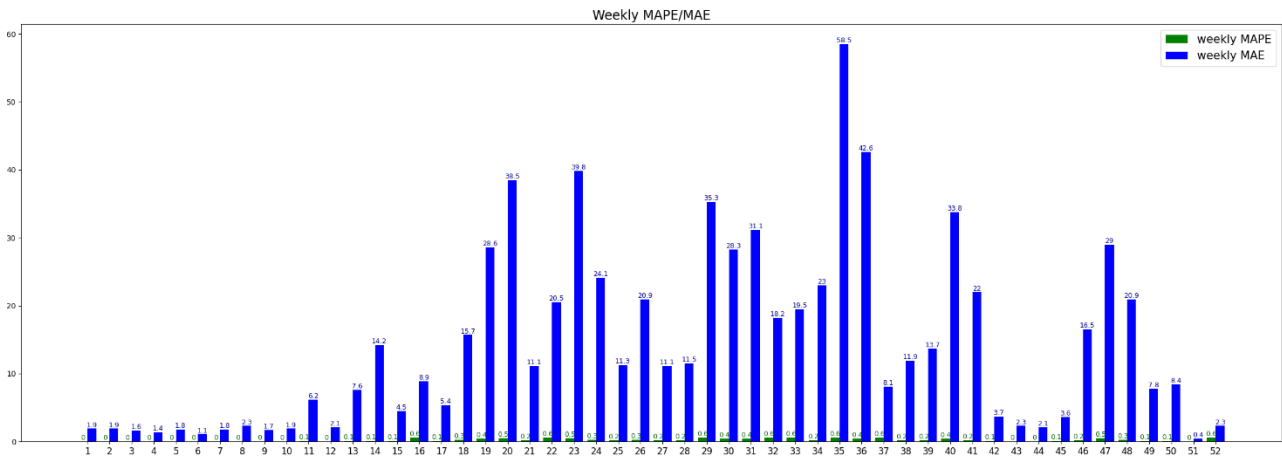


Figure 50: SOC_min= 40.0 %, total MAE: 14.27, total MAPE: 23.46 %

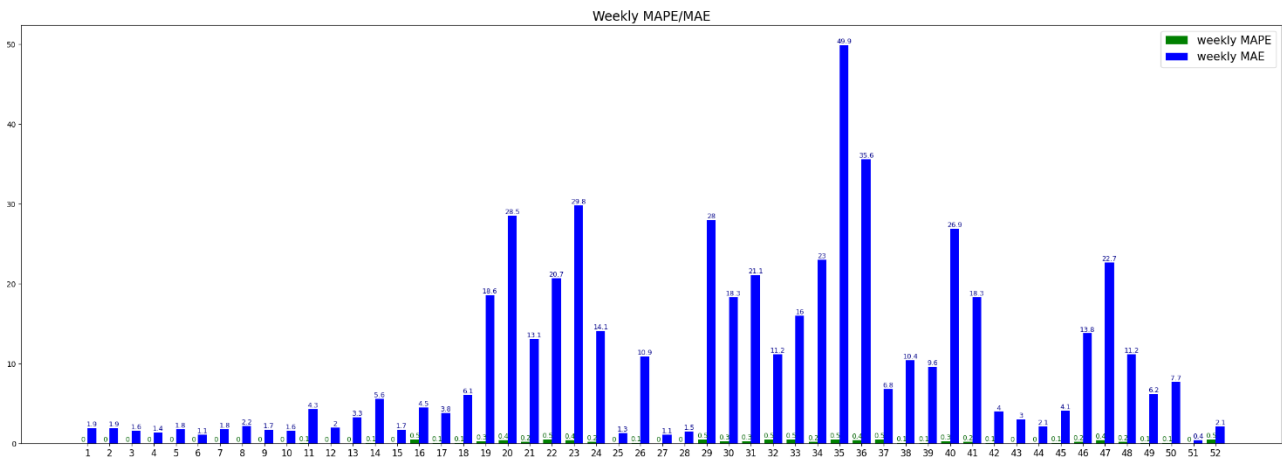


Figure 51: SOC_min= 50.0 %, total MAE: 10.39, total MAPE: 17.5%

Although the size of the images does not allow the error values for each individual week to be identified very clearly, it is still possible to note their qualitative trends. Regarding their overall behavior we can see that, in the case of SOC_min=20%, the total MAPE stands at 22.22 percentage points of SOC. In the first weeks related to the summer months (the calendar starts from 01/07 and ends on 31/06 of the following year) the absolute error is in the range of 1.5 percentage points of SOC while in the summer months it rises around 50 points with peaks of 71. MAPE, on the other hand, in the case of SOC_min=20% overall keeps during the year a relative error in the range of 0.1% to 10%.

By increasing the value of the minimum SOC from 20% to 50%, it is possible to see a reduction in the values of both MAE and MAPE especially in winter weeks whose values were quite high, thus allowing an overall flattening of the error throughout the year.

Comparing the actual measured SOC profile (blue line) with the estimated one (orange line) by examining a week in which the MAE and MAPE values are very low, for example the week 6, it can be seen that by moving from the case with a SOC_min=20% up to SOC_max=50% the curves, already almost overlapping with SOC_min=20%, still maintain the same profile with SOC_min=50%.

That is because the real profile of the SOC and the estimated one are already practically identical and cannot improve further since in both cases, they are already equal to: MAE=1.1 and MAPE=0%.

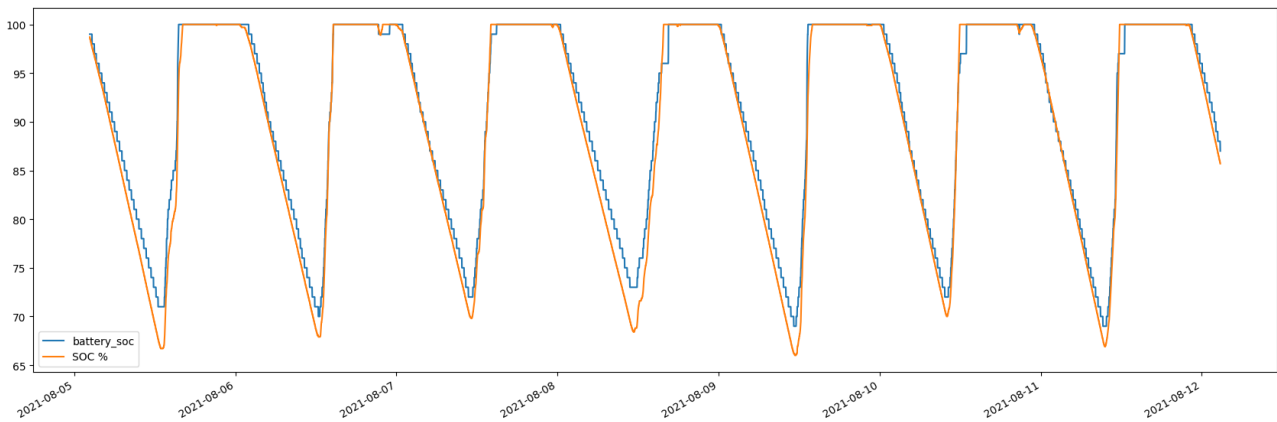


Figure 52: Week 6th, SOC_min= 20.0 %, MAE: 1.1, MAPE: 0%

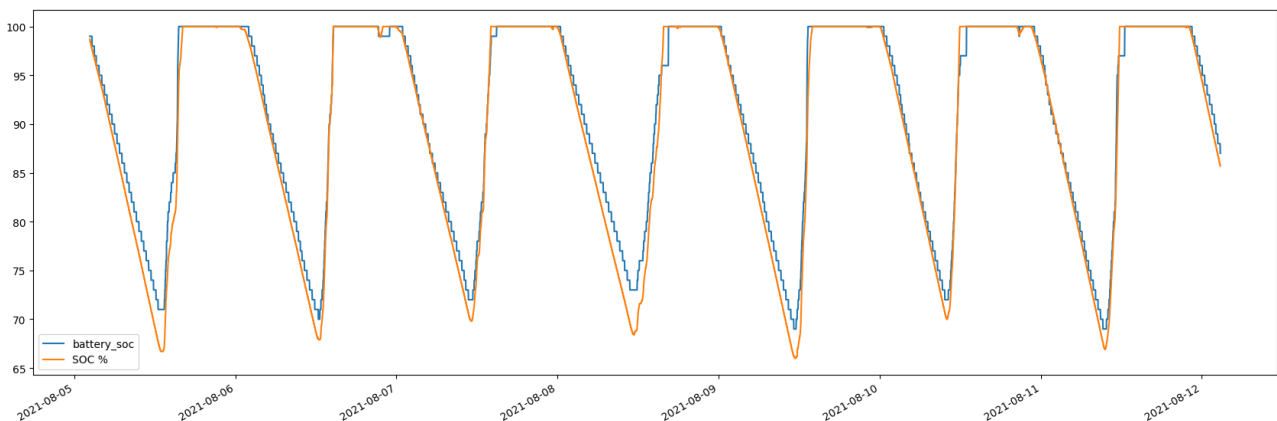


Figure 53: Week 6th, SOC_min= 50.0 %, MAE: 1.1, MAPE: 0%

On the other hand, comparing the actual SOC profile with the estimated profile by examining a week in which the MAE and MAPE values are higher instead, for example the week 27, it can be seen that going from the case with SOC_min=20% to SOC_max=50% the curves have in both situations a substantially identical profile trend but the estimated profile is shifted lower than the actual profile in terms of battery SOC.

It is possible to see, in the figures below, how increasing the minimum SOC from 20% to 50% has the effect of bringing the estimated SOC curve closer to the actual measured curve, thus reducing the MAE from 31.1 to 1.1 and the MAPE from 0.6% to 0%.

The errors, therefore, do not show any computational error made by the SOC measurement code, but these are due exclusively to a different position of the profiles in the SOC scale.

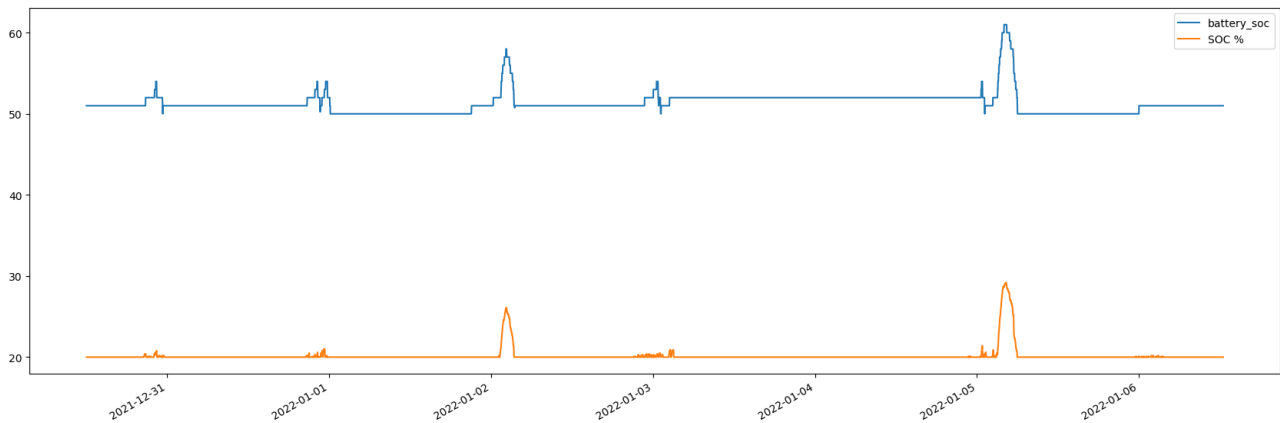


Figure 54: Week 27th, SOC_{min}= 20.0 %, MAE: 31.1, MAPE: 0.6%

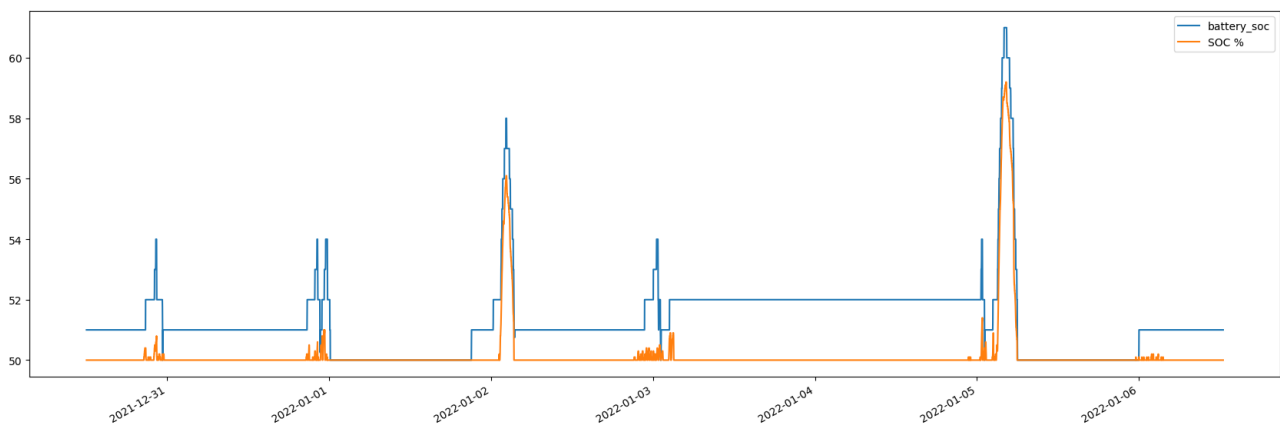


Figure 55: Week 27th, SOC_{min}= 50.0 %, MAE: 1.1, MAPE: 0%

The difference in the shift in the SOC scale between the actual measured battery SOC profile and the estimated one, especially during some weeks in the winter period, is due to the fact that the software used to manage the SOLARISE Living Lab battery performs an active control capable of dynamically changing over time the SOC_{min} and SOC_{max} values of the battery. This is done autonomously according to some established criteria with the aim of improving self-consumption and self-sufficiency.

This software SOC management feature of the SOLARISE Living Lab battery allows us to consider as minor the errors due to the shifting of the two SOC profiles in the SOC scale during the winter months and validate the effectiveness of the code operation in measuring the state of the battery charge.

Having measured the SOC of the battery for all hours of the year and having calculated for each of those hours the relative amount of energy to be exchanged with the grid and having created a list of those elements (Eg), the "bestbattery" function will now perform the economic analysis.

The first step is to estimate the annual expense of the bill for purchasing from the grid all the energy needed to meet the building's energy needs if there is no photovoltaic plant. Assuming the building is equipped with a photovoltaic plant, it is necessary to calculate the sum of all positive energy mismatches during the year and similarly estimate the sum of all negative energy mismatches during the year.

By multiplying the sum of the positive and negative mismatches with respectively the selling price of energy when it is fed into the grid by the photovoltaic plant and the purchase price of energy when it is absorbed by the grid, it is possible to estimate the annual amount of revenues and losses in economic terms related to the photovoltaic plant alone.

The difference between the annual amount of income and loss in economic terms defines the annual savings, which may be positive or negative. Before explaining how the cash flow list is made from these savings and what factors affects its creation, it is first necessary to specify that the selling and buying prices on which the following economic analyses are built were taken directly from the VREG, the independent authority of the Flemish energy market.

According to the data provided in their dashboards, the selling price of energy is assumed to be equal to that of January 2023, thus equal to 0.15 €/kWh[27].



Figure 56: selling price dashboard[27]

The purchase price of energy, on the other hand, is the one applicable to a company with a daily consumption of 30 000 kWh since it is the only choice that can be made as it represents the smallest of the options made available to a company. The chosen purchase price is referred to January 2023 and it is 0.51 €/kWh[27].

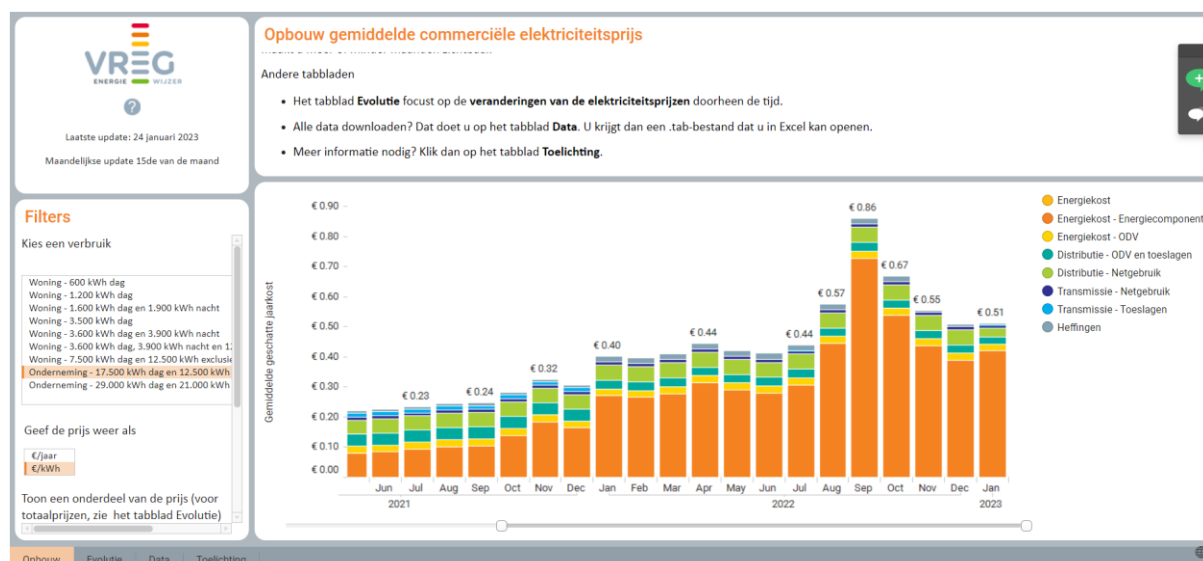


Figure 57: selling price dashboard[27]

Returning to the creation of the cash flow list concerning the PV plant only investment, this is essentially obtained by considering constant savings for each year, thus repeating the value of annual savings as many times as the number of years of the PV panel's lifespan. However, In order for this cash flow to be as realistic as possible, it is necessary to consider other factors that may affect income or losses annually and thus the savings themselves.

Considering an annual PV panel degradation value of 0.5 % [28]and assuming that the PV production decreases by the same amount, the annual revenues will decrease by the same amount and so will the savings:

```
#degradation affects production thus positive income
pos_inc_list_bpv = [year_gain_pos_income]
for i in range(year_life_pvplant):
    new_posinc_bpv = pos_inc_list_bpv[-1] * (1 - annual_module_degradation)
    pos_inc_list_bpv.append(new_posinc_bpv)

neg_inc_list_bpv = [year_gain_neg_income] * year_life_pvplant
final_bill_list_bpv = [abs(x + y) for x, y in zip(pos_inc_list_bpv, neg_inc_list_bpv)]
savings_listas_bpv = [year_energy_bill - x for x in final_bill_list_bpv]
```

Also considering the addition of annual operation and maintenance costs [29] of 9,4 €/kWp/year and adding the initial investment as first item with the negative sign (needed operation for the properly work of the economical functions) the cash flow list looks as follows:

```
cashflows_onlyPV=[-pv_investment]
for irisp_pv in savings_listas:
    cashflows_onlyPV.append(irisp_pv - OeM_tot_cost)
```

The latest change to the cash flow list takes into account the replacement of the inverter in the 15th year of operation of the photovoltaic plant. Generally, the replacement cost is assumed to be 12% [29] [30] of the total initial investment of the plant.

Having created the cash flow list, consisting of the first item related to the initial investment identified with a negative sign and next by the savings for each year, it is now possible to perform an economic analysis of the investment using the previously mentioned NPV, IRR and PBT indicators.

The Net Present Value is the result of calculations that find the current value of a future stream of payments, using the proper discount rate. In general, projects with a positive NPV are worth undertaking while those with a negative NPV are not. The calculation for the NPV of the project is as follows:

$$NPV = -I + \sum_{t=0}^n \frac{C}{(1+i)^t} \quad (16)$$

Where I is the initial investment, C is the cash flow, t is the number of time period and i is the discount rate.

The Internal Rate of Return is a metric used in financial analysis to estimate the profitability of potential investments. IRR is a discount rate that makes the net present value of all cash flows equal to zero in a discounted cash flow analysis.

$$-I + \sum_{t=0}^n \frac{C}{(1+IRR)^t} = 0 \quad (17)$$

The term Pay Back Time, instead, refers to the amount of time it takes to recover the cost of an investment.

$$PBT = \frac{I}{\text{Average Annual Cash Flow}} \quad (18)$$

The first two financial analyses related to NPV and IRR are done through the already implemented functions importable from the python library “numpy.financial”. The discount rate chosen for the NPV calculation is assumed to be 5% [31].

The Pay Back Time, on the other hand, was calculated through the following function:

```
def payback(cashflow):
    investment, cashflow = cashflow[0], cashflow[1:]
    if investment < 0 : investment = -investment
    return payback_of_investment(investment, cashflow)

def payback_of_investment(investment, cashflows):
    total, years, cumulative = 0.0, 0, []
    if not cashflows or (sum(cashflows) < investment):
        raise Exception("insufficient cashflows")
    for cashflow in cashflows:
        total += cashflow
        if total < investment:
            years += 1
            cumulative.append(total)
    A = years
    B = investment - cumulative[years-1]
    C = cumulative[years] - cumulative[years-1]
    return A + (B/C)
```

Regarding the economic analysis related to the investment of the photovoltaic plant plus the battery storage system, the steps for creating the cash flow list and the actual economic evaluation using the financial instruments of NPV, IRR and PBT are basically the same as those used for the study of the economic profitability of the photovoltaic plant alone.

The only differences concern firstly the evaluation of annual revenues and losses, obtained, this time, by not considering the positive and negative mismatch but only the positive and negative amount of energy exchanged with the grid (Eg), and secondly the possibility to face at 10th and 20th year of plant operation, the cost of battery pack replacement. The latter was assumed to be 10% of the total investment cost (seen from IRENA chart[29]).

11.2 BATTERY OPTIMIZATION

The main purpose of the code is to find the best size of battery energy storage system that has an IRR capable of ensuring that the investment is more profitable than the one related to the photovoltaic plant alone.

Specifically, the code focuses on both finding the minimum battery size that can make the IRR at least equal to or greater than the IRR of the PV plant-only investment and finding the battery size at which corresponds the maximum IRR. In either case, manually searching for such results carried out by trial and error would be time-consuming and, for this reason, it is necessary to use optimization algorithms that can optimize and speed up the calculation.

For what concerns finding the battery capacity capable of maximizing the IRR, the SciPy library[32] (a scientific computing library for Python that provides additional functionality for scientific computing, including optimization, linear algebra, signal processing, and statistics) provides several optimization algorithms that can be used to find the maximum or minimum of a function, such as the “minimize” and “fmin”.

The “minimize” function provides a general interface for solving unconstrained or constrained optimization problems and supports several optimization methods, including BFGS, Nelder-Mead, and Powell while the “fmin” function is a derivative-free optimization method and finds the minimum of a scalar function using a downhill simplex algorithm.

Both “fmin” and “minimize” can be used to minimize a function. However, “minimize” is more versatile and can handle a wider range of optimization problems than “fmin”. The “minimize” function also provides additional features such as support for bounds, constraints, and callback functions. On the other hand, “fmin” is a derivative-free optimization algorithm that works well for low-dimensional problems but does not support bounds or constraints and can be slower than gradient-based methods for some problems.

In general, “fmin” may be sufficient when dealing with simple optimization problem that does not involve constraints or bounds and is low-dimensional. However, for more complex optimization problems or high-dimensional problems, “minimize” is usually the better choice.

In the specific context of finding the battery capacity that maximizes the IRR, either “fmin” or “minimize” could have been used since the problem is relatively computationally simple but, in this case, given the need to define a lower limit for the battery size so that it is never zero, the choice of the optimization algorithm to be used for this analysis falls on the “minimize” function.

Since the “bestbattery” function takes multiple data inputs and provides multiple output results, it is necessary to manipulate the function in such a way that it is able to accept as input the unknown value of our interest, in our case the battery size, and return as output only one result, which is the IRR of the investment related to the photovoltaic plant plus the battery pack.

This is achieved by constructing a function called “objective_func_IRR_bpv(x)” as shown in the figure:

```
# define the objective function for optimization IRR_bpv
def objective_func_IRR_bpv(x):
    return bestbattery(x,energy_cost_vendita_input,energy_cost_acquisto_input,year_life_pvplant_input,real_discount_rate_input,
    BESS_cost_input,annual_module_degradation_input,year_first_replacement_battery_input,
    year_second_replacement_battery_input,year_replacement_inverters_input,pv_investment_input,inverters_tot_cost_input,
    inverters_tot_cost_bpv_input,OeM_tot_cost_input,SOC_max_input,SOC_min_input,SOC0_input,Em_input,av_mism_hour_upest_1)[0]
```

After having defined the objective function, it is now possible to use the “minimize” function on it, in order to find the value of the battery capacity that maximizes the IRR.

It is possible to notice that the “objective_func_IRR_bpv(x)” function is used as an argument to the “minimize” function with a negative sign, since we want to maximize the IRR.

```
# Find battery capacity that maximizes IRR with constraint and multiple starting points
max_irr = 0
best_capacity_kwh = None #[kwh]
for initial_guess in start_points:
    res = minimize(lambda x: -objective_func_IRR_bpv(x[0]), x0=np.array([initial_guess]), bounds=[(0.001, None)], options={'gtol': 1e-4, 'disp': True})
    if -res.fun > max_irr:
        max_irr = -res.fun
        best_capacity_kwh = res.x[0]
```

The use of the for loop is due to the need to cyclically repeat the optimization using different capacity values as multiple starting points within a predetermined range and check which result corresponds to the maximum IRR throughout the inner “if” condition.

More in detail, for each optimization named “res”, “res.fun” is the value of the objective function (in this case, the negative IRR) at the optimal point, and “res.x[0]” is the value of the battery capacity at the optimal point.

After having initialized “max_irr” to zero, in order to ensure that any result with a larger IRR will be greater than “max_irr”, it has been also initialized the “best_capacity_kWh” to None with the aim to ensure that it is assigned the battery capacity that corresponds to the maximum IRR. The line “if -res.fun > max_irr” checks if the current result has a larger IRR than the current maximum. If it does, the “max_irr” is updated to be the IRR of the current result (“-res.fun”) and “best_capacity_kWh” to be the corresponding battery capacity (“res.x[0]”). After the loop, “max_irr” contains the maximum IRR found among all the starting points, and “best_capacity_kWh” contains the corresponding battery capacity.

The choice of using multiple starting points is due to the need to be sure that the minimum of the “minimize” function is actually reached and therefore that the IRR value obtained is actually the maximum value.

In fact, when optimisation algorithms are used, it is often difficult to be sure that the minimum has been reached with absolute certainty. However, there are several ways to increase confidence in the results such as, for example, checking for convergence by examining the final value of the objective function and the number of iterations required for convergence, analysing the sensitivity of results by varying the input parameters and observing the resulting change in the objective function (If the results are sensitive to small changes in the input parameters, this may indicate that the optimization problem is ill-conditioned or that the algorithm has not converged to a minimum) and use multiple starting points.

Since the code was designed to make it usable for the widest category of users (who, for that reason, do not necessarily possess the skills to evaluate by themselves the accuracy of the results and, if necessary, repeat the calculation by changing some data), the interaction with the code was minimized by providing results only by entering some initial input data.

The goal of simplifying the user-code interaction leads to the need to find a solution that independently ensures that the result is the desired one. The strategy of using multiple starting points is the only one that allows to ensure that the global maximum has been found with no user involvement.

```
# Set up multiple starting points
start_points=[]
for sp in range(1,int(round(pv_size)),10):
    start_points.append(sp)
    start_points[0]=0.001
```

For what concerns finding the minimum battery size that can make the IRR at least equal to or greater than the IRR of the PV plant-only investment, it was used an algorithm approach called binary search.

A binary search is a search algorithm that is used to find the position of a target value in a sorted list or array. The algorithm works by repeatedly dividing the search interval in half, based on whether the target value is less than or greater than the middle value of the interval. This process is repeated until the target value is found or the search interval is empty.

In the code shown below, binary search is used to find the minimum battery capacity needed to achieve the Internal Rate of Return (IRR) that makes the investment of PV + BESS either at least equally or more profitable than the PV plant alone.

The function “find_min_battery_capacity” takes as input a battery capacity range, two functions that calculate the IRR of the PV+BESS system and the PV system alone, and a tolerance value that specifies the desired precision of the solution. The battery capacity range is divided into two halves, and the IRR of the PV and BESS investment and the IRR of the PV alone investment are calculated for the midpoint of the range. Based on the comparison of these IRR values and the maximum capacity that maximizes the IRR of the PV and BESS investment, the if statements in the code determines the direction in which to move the upper and lower bounds of the battery capacity range.

This process is repeated until the desired level of precision is achieved and, finally, the function returns the upper bound of the range as the minimum battery capacity needed to achieve the desired IRR.

```
def find_min_battery_capacity(battery_capacity_range, capacity_maximize_IRR_bpv, IRR_bess_PV_func, IRR_onlyPV_func, tolerance=0.0001):
    # battery_capacity_range: intervallo di ricerca di battery_capacity
    # IRR_bess_PV_func: funzione che calcola IRR_bess+PV dato battery_capacity
    # IRR_onlyPV_func: funzione che calcola IRR_onlyPV dato battery_capacity
    # tolerance: precisione richiesta

    # applica la ricerca binaria all'intervallo di ricerca
    lower, upper = battery_capacity_range
    while upper - lower > tolerance:
        mid = (lower + upper) / 2
        IRR_bess_PV = IRR_bess_PV_func(mid)
        IRR_onlyPV = IRR_onlyPV_func(mid)
        if capacity_maximize_IRR_bpv < mid:
            if IRR_bess_PV < IRR_onlyPV:
                upper = mid
            else:
                lower = mid
        else:
            if IRR_bess_PV < IRR_onlyPV:
                lower = mid
            else:
                upper = mid

    # restituisci il valore di battery_capacity che soddisfa la condizione
    return upper
```

Using a binary search allows the algorithm to quickly converge to the minimum battery capacity needed to achieve the desired IRR, because it repeatedly halves the battery capacity range and evaluates the IRR at the midpoint. This approach is much faster than other methods such as the linear search.

A linear search, in fact, involves iterating over all possible battery capacity values in the given range and evaluating the IRR function at each of these values. This means that the algorithm would have been compared the IRR of the PV+BESS system and the PV system alone at each battery capacity value and return the minimum battery capacity that satisfies the IRR condition.

Although is simple to implement and conceptually straightforward, the linear search can be very slow for large ranges or fine tolerances, as it requires evaluating the IRR function at a large number of points. This can make it impractical for real-world applications with large ranges of battery capacity and high precision requirements.

In summary, the linear search is conceptually simple but it can be slow and computationally expensive for large ranges or fine tolerances. A binary search is used in this case because is a more efficient and practical approach for finding the minimum battery capacity needed to achieve a desired IRR in a sorted range of possible battery capacities.

11.3 PERFORMANCE ANALYSIS

The last analysis concerns the evaluation of the energy performance of the PV + BESS system and is carried out by the Python function called "perf_analisy" (APPENDIX 3). This function takes as inputs the DataFrames of the hourly PV plant production, the hourly building consumption and the hourly battery SOC.

The initial part of the function performs a data manipulation similar to what is done for the same data in the "bestbattery" function. In this case, however, the sum of the positive and negative hourly mismatch is not done on the year time interval but is calculated on a daily basis. The same thing is done for the condition in which the battery pack is added to the photovoltaic plant. In this case, in fact, the sum of only the positive and negative amount of energy exchanged with the grid (Eg) is done on 24-hour time intervals.

To study self-sufficiency and self-consumption, it is necessary to calculate the "Elpc" area related to the amount of energy that is instantaneously produced by the photovoltaic plant and consumed by the building. In the case of the photovoltaic plant alone, the "Elpc" area is calculated by subtracting the positive mismatch portion of energy from the entire daily production.

```
#### sottraggio alla produzione fotovoltaica il positive mismatch---->area Elpc #####
PV_minus_posmism_dy=(pv_sum_d['PV_tot_days']-pos_hourmism_dy['pos_hourmism_daily'])
Elpc_day=pd.DataFrame(PV_minus_posmism_dy,columns=['Elpc_daily'])
```

By the same principle, in the case of photovoltaic plant+BESS, the "Elpc" area is expected to be larger and obtained by subtracting the portion of positive Eg energy fed into the grid from the total daily production.

```
#### sottraggio alla produzione fotovoltaica il positive gain totale giornaliero -> area Elpc per SC #####
PV_minus_posgain=(pv_sum_d['PV_tot_days']-pos_gain_day['pos_gain_day'])
ElpcSC_day_bess=pd.DataFrame(PV_minus_posgain,columns=['ElpcSC_day_bess'])
cons_minus_neggain=(cons_sum_d['cons_tot_days']+neg_gain_day['neg_gain_day'])
ElpcSS_day_bess=pd.DataFrame(cons_minus_neggain,columns=['ElpcSS_day_bess'])

Elpc_day_bess=pd.concat([ElpcSC_day_bess['ElpcSC_day_bess'],ElpcSS_day_bess['ElpcSS_day_bess']],axis=1)
date_d=pd.date_range(start='2019-07-01 00:00',end='2020-06-30 23:59:59',periods=365)
Elpc_day_bess.set_index(date_d,inplace=True)
```

By definition, "Elpc" should be the same whether it is calculated by subtracting a positive energy mismatch from total production or by subtracting a negative energy mismatch from total consumption. This is actually true only theoretically because in calculation practice the data after a mathematical operation are affected by algorithmic error due to approximation. For this reason, although the error between the two measurements is almost always small, MAPE measurement was still provided for "Elpc" areas calculated

according to the two possible ways in order to be able to inform the user of a possible low accuracy of the building energy performance data. The result is then printed on the screen. Generally, a MAPE over 50 % defines the measurement as not accurate compared with the reference data [33]. The “perf_analysys” function returns as results the value of the self-sufficiency and the self-consumption with and without the integration of the battery energy storage system and the relative chart.

12. WORKING EXAMPLE

The following images represent an example of the code operation and show the results under the assumption of considering as input data those set as default for the city of Ghent, Belgium.

The optimal azimuth for a fixed tilt of 40.0 degrees is 180.0 degrees.

yearly production: 116338.04 kWh
 yearly consumption: 145157.74 kWh
 PV_SIZE: 110.0 kWp

PVonly_NPV: 412686.1 euro
 PVonly_IRR: 44.228 %
 PVonly_PBT: 2.26 years

Minimum profitable battery capacity: 0.108 KWh
 NPV_bpv_optimal: 416870.7822386649 euro
 IRR_bpv_optimal: 44.218 %
 PBT_bpv_optimal: 2.2564597808259164 years

Battery capacity that maximizes IRR: 0.001 kWh
 max_IRR: 44.247 %

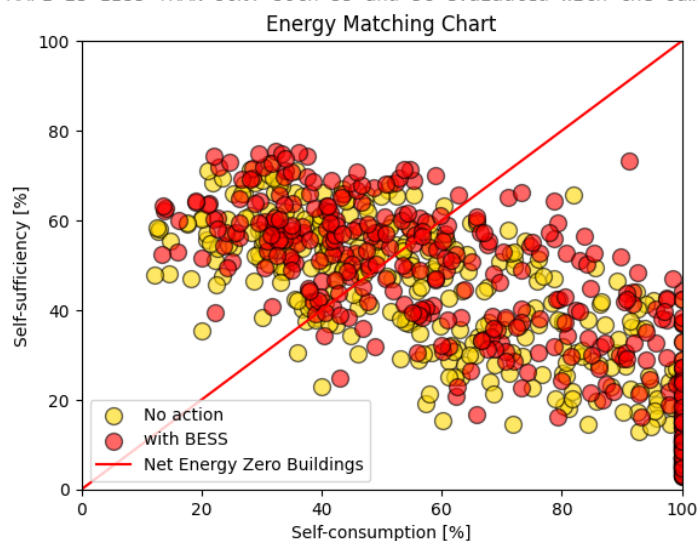
Trial battery capacity: 10.0 kWh
 Trial Battery: BESS+PV investment is LESS profitable than only PV

TrialBatt_NPV: 417006.68836897664 euro
 TrialBatt_IRR: 41.527 %
 TrialBatt_PBT: 2.396138246588303 years

---TRIAL BATTERY CAPACITY CHART---

MAPE_Elpc_day_bess: 0.01 %

MAPE IS LESS THAN 50%: both SS and SC evaluated with the same [ElpcSC_day_bess]



Figure(640x480)

| | SS % | SS bess % | SC % | SC bess % |
|-----|-------|-----------|-------|-----------|
| 50% | 43.57 | 46.58 | 57.91 | 60.82 |

Getting more specific about all the input data required by the code, the user is prompted to enter the name of a LOAD file in .xlsx format. The default value for this input is 'datiloadmodelUpest.xlsx'. The file contains hourly data of the electrical load demand of the residential building. Next, the user is prompted to enter the latitude and longitude of the building location. The default values are 51.034 and 3.695, respectively. These values are used to calculate the solar irradiance on the PV panels.

The tilt and azimuth angles of the PV panels are also important inputs for the analysis. The default value for the tilt angle is 40 degrees while the default azimuth angle is 180 degrees (PVGIS's optimum angle values), which corresponds to the panels facing south in the northern hemisphere. When 0 is entered as input data for one or both of the tilt and azimuth values, a function is called to determine an estimation of the input data that received the zero-input value. The optimal tilt angle is approximated with the value of the latitude data (a common rule of thumb for fixed arrays also cited by NREL's PVWatts Calculator) while the optimal azimuth value is found by estimating the azimuth angle between 0° and 360°, which, with the optimal tilt angle, maximizes the total annual amount of irradiance received by the panel.

```

optimal_tilt=latitude_def
azimuth_range = np.arange(0, 360, 10) # degrees
# Calculate the solar irradiance on the panel for each azimuth angle
irradiance = []
for azimuth_opt in azimuth_range:
    total_irradiance_opt = get_irradiance( optimal_tilt, azimuth_opt)
    irradiance.append(sum(total_irradiance_opt))

# Find the optimal azimuth that maximizes the POA irradiance
optimal_azimuth = azimuth_range[np.argmax(irradiance)]

```

The latitude of a location can be a good approximation for the tilt angle of a solar panel. This is because solar panels are most efficient when they are pointed directly at the Sun, and the angle of the panel affects the amount of sunlight it receives. By tilting a solar panel at an angle equal to the latitude of the location, the panel can be pointed more directly at the Sun during peak sunlight hours, which can help to maximize energy production. Additionally, the latitude provides a simple and easily accessible way to approximate the tilt angle, making it easy for people to determine the optimal tilt angle without needing to perform complex calculations. While the latitude is not necessarily the optimal tilt angle in every location, it provides a good starting point for further optimization by adjusting the tilt angle based on other factors such as the time of year, weather conditions, and the specific orientation of the solar panel.

The PV size is another crucial input, which is set to 0 Wp by default. If nothing is entered (in the working example it is chosen 110 kWp), this zero-value led to the estimation of the PV size by calling a function. The following required input data is the annual average sun hour per day of the considered location. The Global Solar Atlas provides a peak sun hours map. It can be noticed that both maps' legends are given in units of kWh/m².

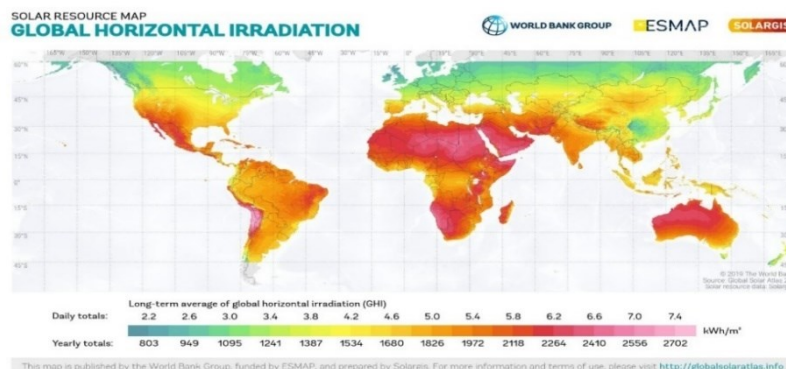


Figure 58: global map of average daily peak sun hours

Recalling that 1 peak sun hour is equal to 1 hour of sunlight at 1 kW/m², meaning these values are numerically identical. So, although neither map says “peak sun hours” anywhere on it, they are in fact both maps of average peak sun hours. From this map it is possible to approximate the global average daily sunshine hours on Earth is about 5.4 hours per day. However, it is important to note that this value is only a rough estimation and that the actual amount of daily sunshine hours can vary greatly depending on geographic location, altitude, weather conditions and seasons. In some regions of the world, such as the equator, daily sunshine hours may exceed 12 hours per day, while in other regions, such as polar areas, they may be less than 4 hours per day.

The trial battery capacity is also an input, with a default value of 0 kWh. The electricity selling and purchasing prices are essential parameters for the economic analysis. The default value for the selling price is 0.15 €/kWh, and the default purchasing price is 0.51 €/kWh. The user can enter different values based on their local prices. The installation cost of the PV panels is another important input, with a default value of 732.44 €/kWp. This value is based on the IRENA Renewable power generation costs 2021 report. The OeM cost is set to 9.4 €/kWp/y for Europe. The lifespan of the PV plant is set to 25 years by default. The nominal discount rate is set to 0.05, and the expected inflation rate is set to 0.00. The real discount rate is calculated based on these values. The BESS cost is an important input, with a default value of 758.39 €/kWh[29]. The inverter costs for PV and BESS are also inputs, with default values of 12% of the “pv_installation_cost” and 10% of the “pv_installation_cost+BESS_cost_input”, respectively. The annual module degradation rate is set to 0.005, and the years of the first and second replacement of batteries are set to 10 and 20 years, respectively. The year of replacement of inverters is set to 15 years.

Finally, the maximum and minimum state of charge of the battery and the initial state of charge are also inputs, with default values of 1, 0.2, and 0.9, respectively.

13. INPUT DATA INFLUENCE ANALYSIS

The optimal battery capacity size that ensures the economic profitability of the investment of the photovoltaic plant plus battery pack can vary greatly depending on the data entered by the user.

In particular, two different analyses were conducted focusing on what are the data mainly affecting the variations in the optimal battery capacity as they are most susceptible to sudden changes in their value over time. The first analysis concerns the influence on the optimal battery size due to the change in the investment cost of both the photovoltaic plant and the battery pack. The second analysis, on the other hand, concerns the influence on the optimal capacity related to the change in the selling price and purchase price of energy from the grid (these values are selected within the range provided by the VREG dashboards).

Focusing on the first analysis, the four tests performed are described below with their corresponding graphs:

The first test related to the first analysis is obtained by varying the investment cost of the battery pack in the range of 100 €/kWh to 1000 €/kWh while holding the investment cost of the photovoltaic plant constant at the default value:

| lat/lon | tilt/azimut | Year_prod | Year_cons | PVsize_KW | IRR_onlyPV | BestBatter | IRR_best% | OptimalBa | IRR_opt% | TrialBatter | IRR_trial% | electr_sell | electr_puri | PV_cost_ei | BESS_cost | SS % | SS bess % | SC % | SC bess % |
|--------------|--------------|-----------|-----------|-----------|------------|------------|-----------|-----------|----------|-------------|------------|-------------|-------------|------------|-----------|-------|-----------|-------|-----------|
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 43.904 | 101 | 47.739 | 0.053707 | 44.568 | 0 | 0 | 0.15 | 0.51 | 732.44 | 100 | 43.43 | 43.44 | 58.73 | 58.76 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 43.904 | 11 | 44.752 | 60.45596 | 43.903 | 0 | 0 | 0.15 | 0.51 | 732.44 | 200 | 43.43 | 55.68 | 58.73 | 76.46 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 43.904 | 0.001 | 44.562 | 15.28604 | 43.902 | 0 | 0 | 0.15 | 0.51 | 732.44 | 300 | 43.43 | 46.87 | 58.73 | 63.36 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 43.904 | 0.001 | 44.562 | 7.379984 | 43.9 | 0 | 0 | 0.15 | 0.51 | 732.44 | 400 | 43.43 | 45.3 | 58.73 | 60.97 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 43.904 | 0.001 | 44.562 | 4.691926 | 43.899 | 0 | 0 | 0.15 | 0.51 | 732.44 | 500 | 43.43 | 44.6 | 58.73 | 60.28 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 43.904 | 0.001 | 44.562 | 3.37425 | 43.903 | 0 | 0 | 0.15 | 0.51 | 732.44 | 600 | 43.43 | 44.07 | 58.73 | 59.58 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 43.904 | 0.001 | 44.562 | 2.372816 | 43.891 | 0 | 0 | 0.15 | 0.51 | 732.44 | 758.39 | 43.43 | 43.78 | 58.73 | 59.29 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 43.904 | 0.001 | 44.562 | 2.161988 | 43.9 | 0 | 0 | 0.15 | 0.51 | 732.44 | 800 | 43.43 | 43.75 | 58.73 | 59.25 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 43.904 | 0.001 | 44.562 | 1.845746 | 43.893 | 0 | 0 | 0.15 | 0.51 | 732.44 | 900 | 43.43 | 43.71 | 58.73 | 59.19 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 43.904 | 0.001 | 44.562 | 1.582211 | 43.899 | 0 | 0 | 0.15 | 0.51 | 732.44 | 1000 | 43.43 | 43.67 | 58.73 | 59.14 |

Figure 59: Excel sheet of Main Code data result obtained by varying BESS investment cost

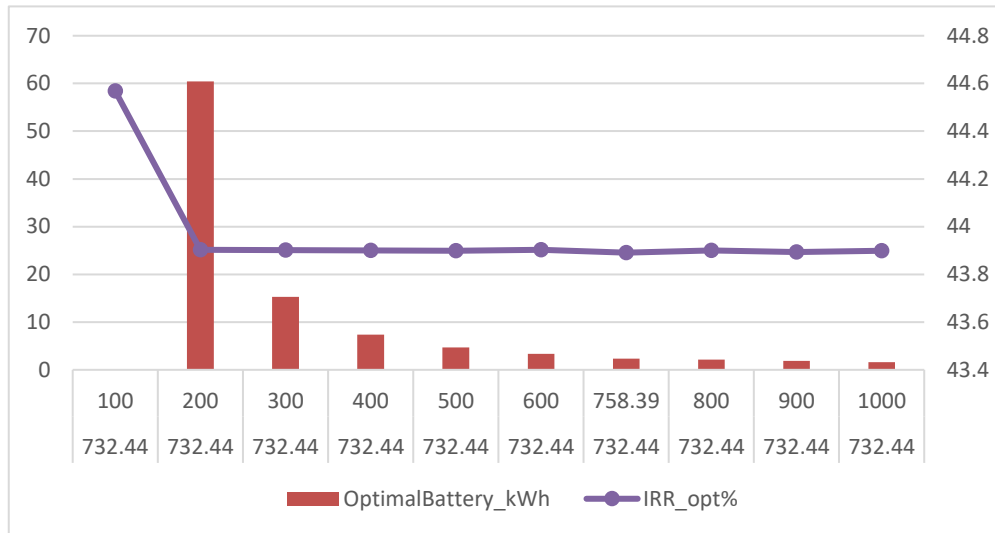


Figure 60: Excel chart obtained by varying BESS investment cost while keeping constant PV plant investment cost

The second test related to the first analysis is obtained this time by holding the investment cost of the battery pack constant at the default value while varying the investment cost of the photovoltaic plant in the range of 100 €/kWh to 1000 €/kWh:

| lat/lon | tilt/azimut | Year_prod | Year_cons | PVsize_KW | IRR_onlyPI | BestBatter | IRR_best% | OptimalBa | IRR_opt% | TrialBatter | IRR_trial% | electr_sell | electr_pur | PV_cost_ei | BESS_cost | SS % | SS bess % | SC % | SC bess % |
|--------------|--------------|-----------|-----------|-----------|------------|------------|-----------|-----------|----------|-------------|------------|-------------|------------|------------|-----------|-------|-----------|-------|-----------|
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 322.616 | 0.001 | 327.227 | 0.264535 | 321.574 | 0 | 0 | 0.15 | 0.51 | 100 | 758.39 | 43.43 | 43.47 | 58.73 | 58.85 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 161.241 | 0.001 | 163.555 | 0.475363 | 161.139 | 0 | 0 | 0.15 | 0.51 | 200 | 758.39 | 43.43 | 43.5 | 58.73 | 58.92 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 107.449 | 0.001 | 108.995 | 0.738898 | 107.415 | 0 | 0 | 0.15 | 0.51 | 300 | 758.39 | 43.43 | 43.54 | 58.73 | 58.97 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 80.552 | 0.001 | 81.715 | 1.055141 | 80.515 | 0 | 0 | 0.15 | 0.51 | 400 | 758.39 | 43.43 | 43.59 | 58.73 | 59.03 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 64.411 | 0.001 | 65.346 | 1.371383 | 64.403 | 0 | 0 | 0.15 | 0.51 | 500 | 758.39 | 43.43 | 43.64 | 58.73 | 59.09 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 53.646 | 0.001 | 54.432 | 1.740332 | 53.645 | 0 | 0 | 0.15 | 0.51 | 600 | 758.39 | 43.43 | 43.69 | 58.73 | 59.17 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 43.904 | 0.001 | 44.562 | 2.372816 | 43.891 | 0 | 0 | 0.15 | 0.51 | 732.44 | 758.39 | 43.43 | 43.78 | 58.73 | 59.29 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 40.171 | 0.001 | 40.784 | 2.689059 | 40.167 | 0 | 0 | 0.15 | 0.51 | 800 | 758.39 | 43.43 | 43.83 | 58.73 | 59.35 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 35.666 | 0.001 | 36.228 | 3.268836 | 35.663 | 0 | 0 | 0.15 | 0.51 | 900 | 758.39 | 43.43 | 44.03 | 58.73 | 59.52 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 32.051 | 0.001 | 32.578 | 3.954027 | 32.048 | 0 | 0 | 0.15 | 0.51 | 1000 | 758.39 | 43.43 | 44.3 | 58.73 | 59.89 |

Figure 61: Excel sheet of Main Code data result obtained by varying BESS investment cost

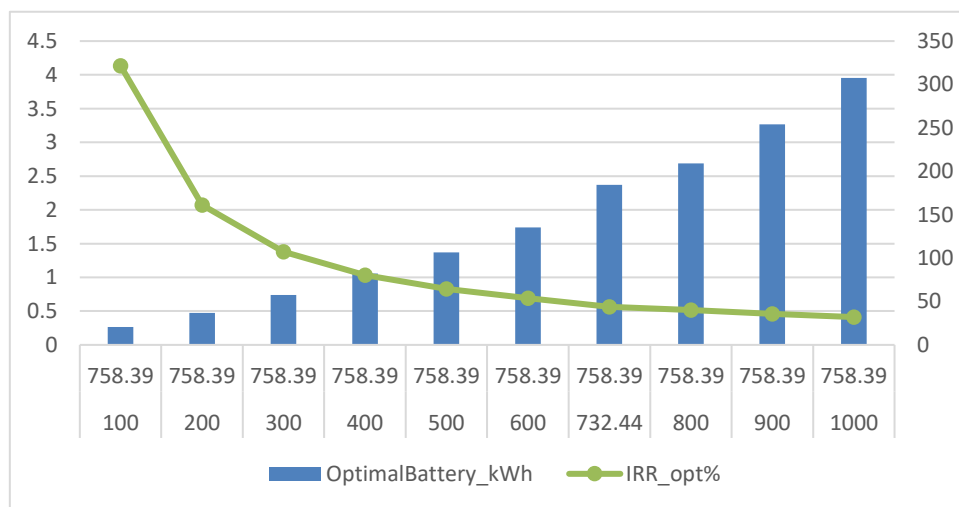


Figure 62: Excel chart obtained by varying PV plant investment cost while keeping constant BESS investment cost

The third test related to the first analysis is obtained this time by varying the investment cost of both the PV plant and the battery pack in a range between 100 €/kWh to 1000 €/kWh in a mutual opposite way:

| lat/lon | tilt/azimut | Year_prod | Year_cons | PVsize_KW | IRR_onlyPV | BestBattery | IRR_best% | OptimalBa | IRR_opt% | TrialBattery | IRR_trial% | electr_sell | electr_pur | PV_cost_e | BESS_cost | SS % | SS bess % | SC % | SC bess % |
|--------------|--------------|-----------|-----------|-----------|------------|-------------|-----------|-----------|----------|--------------|------------|-------------|------------|-----------|-----------|-------|-----------|-------|-----------|
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 32.051 | 101 | 35.961 | 0.053707 | 32.583 | 0 | 0 | 0.15 | 0.51 | 1000 | 100 | 43.43 | 43.44 | 58.73 | 58.76 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 35.666 | 31 | 36.696 | 107.945 | 35.881 | 0 | 0 | 0.15 | 0.51 | 900 | 200 | 43.43 | 63.02 | 58.73 | 89.17 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 40.171 | 0.001 | 40.784 | 19.55531 | 40.169 | 0 | 0 | 0.15 | 0.51 | 800 | 300 | 43.43 | 47.4 | 58.73 | 64.69 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 43.904 | 0.001 | 44.562 | 7.379984 | 43.9 | 0 | 0 | 0.15 | 0.51 | 732.44 | 400 | 43.43 | 45.3 | 58.73 | 60.97 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 53.646 | 0.001 | 54.432 | 3.321543 | 53.635 | 0 | 0 | 0.15 | 0.51 | 600 | 500 | 43.43 | 44.05 | 58.73 | 59.55 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 64.411 | 0.001 | 65.346 | 1.898453 | 64.397 | 0 | 0 | 0.15 | 0.51 | 500 | 600 | 43.43 | 43.71 | 58.73 | 59.2 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 80.552 | 0.001 | 81.715 | 1.055141 | 80.515 | 0 | 0 | 0.15 | 0.51 | 400 | 758.39 | 43.43 | 43.59 | 58.73 | 59.03 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 107.449 | 0.001 | 108.995 | 0.686191 | 107.433 | 0 | 0 | 0.15 | 0.51 | 300 | 800 | 43.43 | 43.53 | 58.73 | 58.96 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 161.241 | 0.001 | 163.554 | 0.422656 | 160.964 | 0 | 0 | 0.15 | 0.51 | 200 | 900 | 43.43 | 43.5 | 58.73 | 58.91 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 322.616 | 0.001 | 327.22 | 0.211828 | 321.187 | 0 | 0 | 0.15 | 0.51 | 100 | 1000 | 43.43 | 43.46 | 58.73 | 58.83 |

Figure 63: Excel sheet of Main Code data result obtained by varying both PV and BESS investment cost in a mutual opposite way

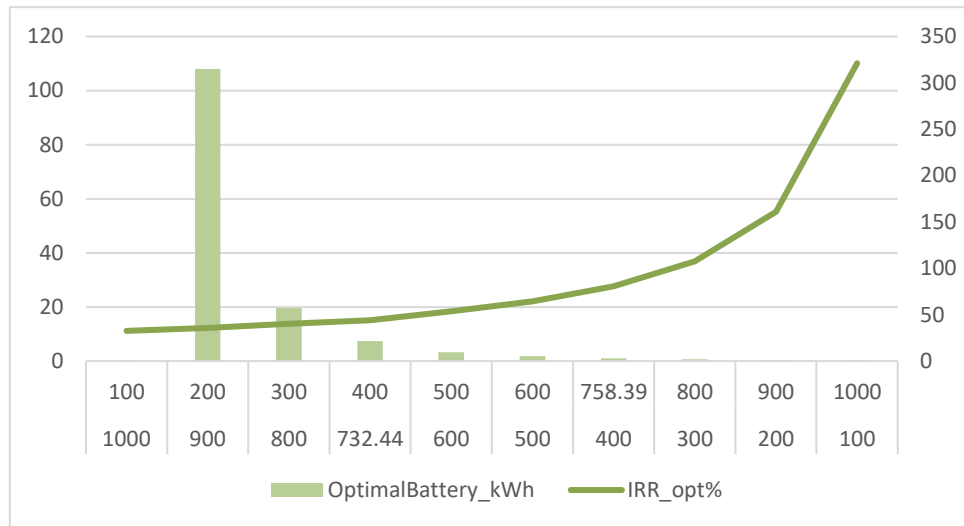


Figure 64: Excel chart obtained by varying both PV and BESS investment cost in a mutual opposite way

The fourth test related to the first analysis is obtained this time by varying the investment cost of both the photovoltaic plant and the battery pack in the range of 100 €/kWh to 1000 €/kWh using the same value for both:

| lat/lon | tilt/azimut | Year_prod | Year_cons | PVsize_KW | IRR_onlyPV | BestBattery | IRR_best% | OptimalBa | IRR_opt% | TrialBattery | IRR_trial% | electr_sell | electr_pur | PV_cost_e | BESS_cost | SS % | SS bess % | SC % | SC bess % |
|--------------|--------------|-----------|-----------|-----------|------------|-------------|-----------|-----------|----------|--------------|------------|-------------|------------|-----------|-----------|-------|-----------|-------|-----------|
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 322.616 | 0.001 | 327.247 | 2.47823 | 322.557 | 0 | 0 | 0.15 | 0.51 | 100 | 100 | 43.43 | 43.8 | 58.73 | 59.31 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 161.241 | 0.001 | 163.559 | 2.47823 | 161.215 | 0 | 0 | 0.15 | 0.51 | 200 | 200 | 43.43 | 43.8 | 58.73 | 59.31 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 107.449 | 0.001 | 108.997 | 2.47823 | 107.433 | 0 | 0 | 0.15 | 0.51 | 300 | 300 | 43.43 | 43.8 | 58.73 | 59.31 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 80.552 | 0.001 | 81.715 | 2.47823 | 80.54 | 0 | 0 | 0.15 | 0.51 | 400 | 400 | 43.43 | 43.8 | 58.73 | 59.31 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 64.411 | 0.001 | 65.346 | 2.47823 | 64.4 | 0 | 0 | 0.15 | 0.51 | 500 | 500 | 43.43 | 43.8 | 58.73 | 59.31 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 53.646 | 0.001 | 54.432 | 2.47823 | 53.636 | 0 | 0 | 0.15 | 0.51 | 600 | 600 | 43.43 | 43.8 | 58.73 | 59.31 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 43.904 | 0.001 | 44.562 | 2.372816 | 43.891 | 0 | 0 | 0.15 | 0.51 | 732.44 | 758.39 | 43.43 | 43.78 | 58.73 | 59.29 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 40.171 | 0.001 | 40.784 | 2.47823 | 40.167 | 0 | 0 | 0.15 | 0.51 | 800 | 800 | 43.43 | 43.8 | 58.73 | 59.31 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 35.666 | 0.001 | 36.228 | 2.530938 | 35.656 | 0 | 0 | 0.15 | 0.51 | 900 | 900 | 43.43 | 43.81 | 58.73 | 59.32 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 32.051 | 0.001 | 32.578 | 2.530938 | 32.05 | 0 | 0 | 0.15 | 0.51 | 1000 | 1000 | 43.43 | 43.81 | 58.73 | 59.32 |

Figure 65: Excel sheet of Main Code data result obtained by varying both PV and BESS investment cost with the same value

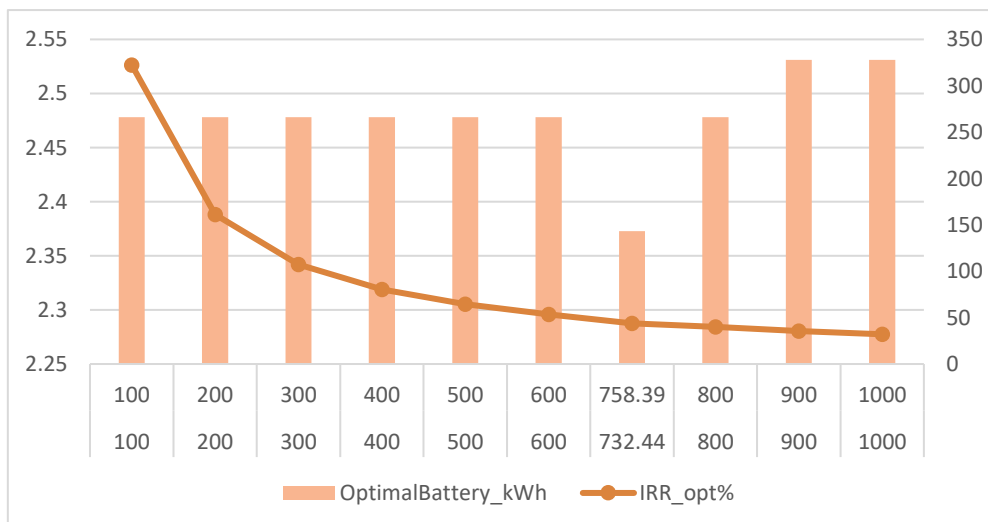


Figure 66: obtained by varying both PV and BESS investment cost with the same value

These tests related to the first analysis show that an increase in the investment cost of the battery pack is linked to a decrease in the size of the optimal battery. In addition, an increase in the investment cost of the photovoltaic plant is also linked to a decrease in the IRR (decreases the economic return).

Focusing now on the second analysis, other four tests are performed and described below with the corresponding graphs.

The first test related to the second analysis is obtained by varying the energy purchase price in a range from 0.23 €/kWh to 0.86 €/kWh while keeping the energy selling price constant at the default value:

| lat/lon | tilt/azimut | Year_prod | Year_cons | PVsize_KW | IRR_onlyPV | BestBatter | IRR_best% | OptimalBa | IRR_opt% | TrialBatter | IRR_trial% | electr_sell | electr_pur | PV_cost_ei | BESS_cost | SS % | SS bess % | SC % | SC bess % |
|--------------|--------------|-----------|-----------|-----------|------------|------------|-----------|-----------|----------|-------------|------------|-------------|------------|------------|-----------|-------|-----------|-------|-----------|
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 25.311 | 0.001 | 25.464 | 0.686191 | 25.299 | 0 | 0 | 0.15 | 0.23 | 732.44 | 758.39 | 43.43 | 43.53 | 58.73 | 58.96 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 31.34 | 0.001 | 31.643 | 1.213262 | 31.339 | 0 | 0 | 0.15 | 0.32 | 732.44 | 758.39 | 43.43 | 43.61 | 58.73 | 59.06 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 36.647 | 0.001 | 37.094 | 1.740332 | 36.636 | 0 | 0 | 0.15 | 0.4 | 732.44 | 758.39 | 43.43 | 43.69 | 58.73 | 59.17 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 39.289 | 0.001 | 39.812 | 1.95116 | 39.286 | 0 | 0 | 0.15 | 0.44 | 732.44 | 758.39 | 43.43 | 43.72 | 58.73 | 59.21 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 43.904 | 0.001 | 44.562 | 2.372816 | 43.891 | 0 | 0 | 0.15 | 0.51 | 732.44 | 758.39 | 43.43 | 43.78 | 58.73 | 59.29 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 46.536 | 0.001 | 47.274 | 2.583645 | 46.522 | 0 | 0 | 0.15 | 0.55 | 732.44 | 758.39 | 43.43 | 43.81 | 58.73 | 59.33 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 47.852 | 0.001 | 48.629 | 2.689059 | 47.836 | 0 | 0 | 0.15 | 0.57 | 732.44 | 758.39 | 43.43 | 43.83 | 58.73 | 59.35 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 54.423 | 0.001 | 55.401 | 3.110715 | 54.419 | 0 | 0 | 0.15 | 0.67 | 732.44 | 758.39 | 43.43 | 43.97 | 58.73 | 59.44 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 66.891 | 0.001 | 68.257 | 3.848613 | 66.88 | 0 | 0 | 0.15 | 0.86 | 732.44 | 758.39 | 43.43 | 44.26 | 58.73 | 59.83 |

Figure 67: Excel sheet of Main Code data result obtained by varying purchase price

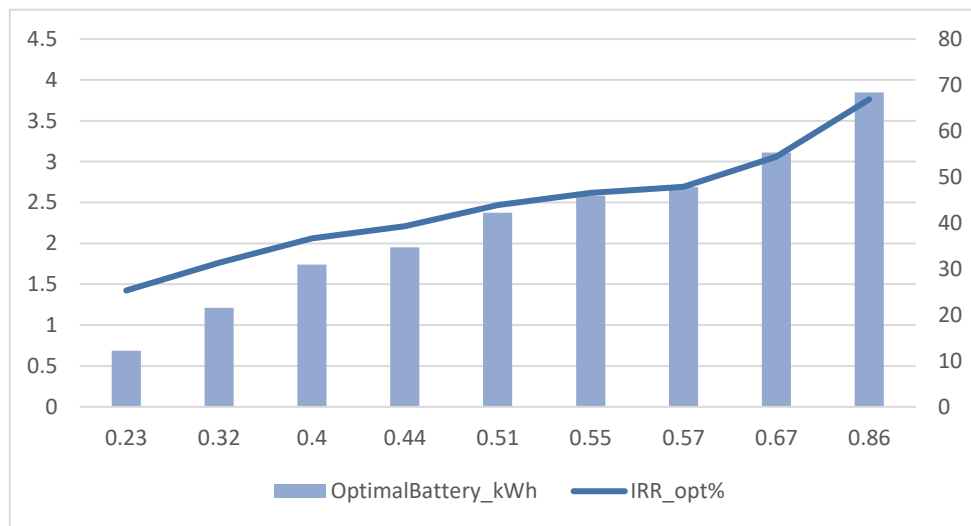


Figure 68: Excel chart obtained by varying purchase price while keeping constant selling price

The second test related to the second analysis is obtained, this time, by varying the energy selling price in a range from 0.05 €/kWh to 0.51 €/kWh while keeping the energy purchase price constant at the default value:

| lat/lon | tilt/azimut | Year_prod | Year_cons | PVsize_KW | IRR_onlyPV | BestBatter | IRR_best% | OptimalBa | IRR_opt% | TrialBatter | IRR_trial% | electr_sell | electr_pur | PV_cost_ei | BESS_cost | SS % | SS bess % | SC % | SC bess % |
|--------------|--------------|-----------|-----------|-----------|------------|------------|-----------|-----------|----------|-------------|------------|-------------|------------|------------|-----------|-------|-----------|-------|-----------|
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 35.982 | 0.001 | 36.943 | 5.32441 | 35.98 | 0 | 0 | 0.05 | 0.51 | 732.44 | 758.39 | 43.43 | 44.84 | 58.73 | 60.56 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 39.15 | 0.001 | 39.989 | 3.848613 | 39.141 | 0 | 0 | 0.09 | 0.51 | 732.44 | 758.39 | 43.43 | 44.26 | 58.73 | 59.83 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 42.319 | 0.001 | 43.037 | 2.794473 | 42.306 | 0 | 0 | 0.13 | 0.51 | 732.44 | 758.39 | 43.43 | 43.85 | 58.73 | 59.37 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 43.904 | 0.001 | 44.562 | 2.372816 | 43.891 | 0 | 0 | 0.15 | 0.51 | 732.44 | 758.39 | 43.43 | 43.78 | 58.73 | 59.29 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 45.488 | 0.001 | 46.088 | 2.003867 | 45.478 | 0 | 0 | 0.17 | 0.51 | 732.44 | 758.39 | 43.43 | 43.73 | 58.73 | 59.22 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 47.074 | 0.001 | 47.614 | 1.687625 | 47.063 | 0 | 0 | 0.19 | 0.51 | 732.44 | 758.39 | 43.43 | 43.68 | 58.73 | 59.16 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 48.659 | 0.001 | 49.141 | 1.42409 | 48.644 | 0 | 0 | 0.21 | 0.51 | 732.44 | 758.39 | 43.43 | 43.64 | 58.73 | 59.1 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 53.417 | 0.001 | 53.724 | 0.738898 | 53.416 | 0 | 0 | 0.27 | 0.51 | 732.44 | 758.39 | 43.43 | 43.54 | 58.73 | 58.97 |
| (51.034, 3.) | (40.0, 180.) | 114164.7 | 145157.7 | 107.945 | 72.473 | 0.001 | 72.088 | 0.053707 | 72.051 | 0 | 0 | 0.51 | 0.51 | 732.44 | 758.39 | 43.43 | 43.44 | 58.73 | 58.76 |

Figure 69: Excel sheet of Main Code data result obtained by varying selling price

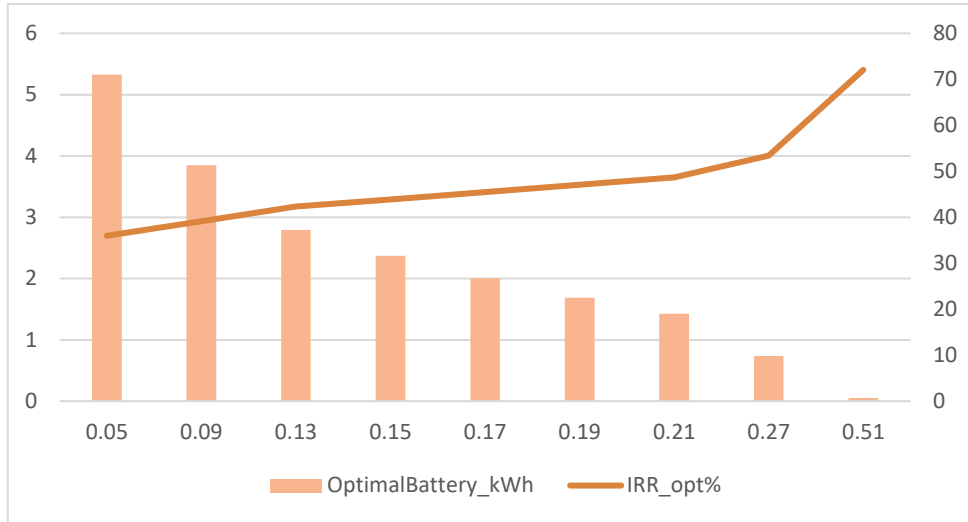


Figure 70: Excel chart obtained by varying selling price while keeping constant purchase price

The third test related to the second analysis is obtained, this time, by varying both the selling price and the purchase price of energy in their intervals of interest both in an increasing way:

| lat/lon | tilt/azimut | Year_prod | Year_cons | PVsize_KW | IRR_onlyPV | BestBatter | IRR_best% | OptimalBa | IRR_opt% | TrialBatter | IRR_trial% | electr_sell | electr_puri | PV_cost_ei | BESS_cost | SS % | SS bess % | SC % | SC bess % |
|---------------|---------------|-----------|-----------|-----------|------------|------------|-----------|-----------|----------|-------------|------------|-------------|-------------|------------|-----------|-------|-----------|-------|-----------|
| (51.034, 3.0) | (40.0, 180.0) | 114164.7 | 145157.7 | 107.945 | 17.126 | 0.001 | 17.671 | 3.795906 | 17.125 | 0 | 0 | 0.05 | 0.23 | 732.44 | 758.39 | 43.43 | 44.24 | 58.73 | 59.8 |
| (51.034, 3.0) | (40.0, 180.0) | 114164.7 | 145157.7 | 107.945 | 26.004 | 0.001 | 26.442 | 2.267402 | 25.996 | 0 | 0 | 0.1 | 0.3 | 732.44 | 758.39 | 43.43 | 43.77 | 58.73 | 59.27 |
| (51.034, 3.0) | (40.0, 180.0) | 114164.7 | 145157.7 | 107.945 | 34.661 | 0.001 | 35.053 | 1.529504 | 34.658 | 0 | 0 | 0.15 | 0.37 | 732.44 | 758.39 | 43.43 | 43.66 | 58.73 | 59.13 |
| (51.034, 3.0) | (40.0, 180.0) | 114164.7 | 145157.7 | 107.945 | 43.255 | 0.001 | 43.628 | 1.160555 | 43.251 | 0 | 0 | 0.2 | 0.44 | 732.44 | 758.39 | 43.43 | 43.6 | 58.73 | 59.05 |
| (51.034, 3.0) | (40.0, 180.0) | 114164.7 | 145157.7 | 107.945 | 51.831 | 0.001 | 52.196 | 0.949727 | 51.822 | 0 | 0 | 0.25 | 0.51 | 732.44 | 758.39 | 43.43 | 43.57 | 58.73 | 59.01 |
| (51.034, 3.0) | (40.0, 180.0) | 114164.7 | 145157.7 | 107.945 | 60.4 | 0.001 | 60.762 | 0.791605 | 60.395 | 0 | 0 | 0.3 | 0.58 | 732.44 | 758.39 | 43.43 | 43.55 | 58.73 | 58.98 |
| (51.034, 3.0) | (40.0, 180.0) | 114164.7 | 145157.7 | 107.945 | 68.967 | 0.001 | 69.329 | 0.686191 | 68.962 | 0 | 0 | 0.35 | 0.65 | 732.44 | 758.39 | 43.43 | 43.53 | 58.73 | 58.96 |
| (51.034, 3.0) | (40.0, 180.0) | 114164.7 | 145157.7 | 107.945 | 77.533 | 0.001 | 77.897 | 0.633484 | 77.512 | 0 | 0 | 0.4 | 0.72 | 732.44 | 758.39 | 43.43 | 43.53 | 58.73 | 58.95 |
| (51.034, 3.0) | (40.0, 180.0) | 114164.7 | 145157.7 | 107.945 | 86.099 | 0.001 | 86.465 | 0.580777 | 86.07 | 0 | 0 | 0.45 | 0.79 | 732.44 | 758.39 | 43.43 | 43.52 | 58.73 | 58.94 |
| (51.034, 3.0) | (40.0, 180.0) | 114164.7 | 145157.7 | 107.945 | 95.46 | 0.001 | 95.8 | 0.475363 | 95.437 | 0 | 0 | 0.51 | 0.86 | 732.44 | 758.39 | 43.43 | 43.5 | 58.73 | 58.92 |

Figure 71: Excel sheet of Main Code data result obtained by varying both selling and purchasing price in an increasing way

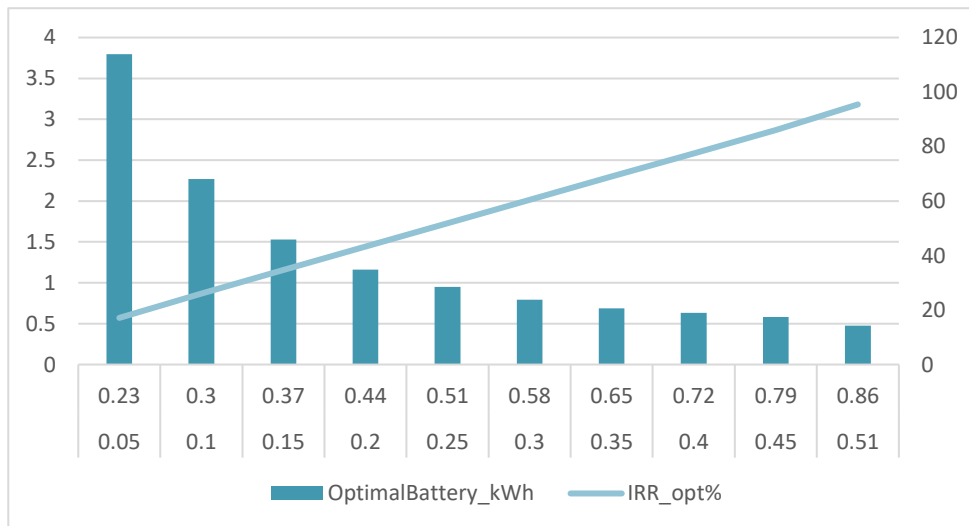


Figure 72: Excel chart obtained by varying both selling and purchasing price in an increasing way

The fourth test related to the second analysis is obtained by varying both the selling price and purchase price of energy in their intervals of interest, but this time in opposite ways to each other:

| lat/lon | tilt/azimut | Year_prod | Year_cons | PVsize | KW | IRR | onlyPI | BestBatter | IRR_best% | OptimalBa | IRR_opt% | TrialBatter | IRR_trial% | electr_sell | electr_puri | PV_cost_ei | BESS_cost | SS % | SS bess % | SC % | SC bess % |
|---------------|---------------|-----------|-----------|---------|--------|-------|--------|------------|-----------|-----------|----------|-------------|------------|-------------|-------------|------------|-----------|-------|-----------|-------|-----------|
| (51.034, 3.4) | (40.0, 180.0) | 114164.7 | 145157.7 | 107.945 | 28.029 | 0.001 | 29.043 | 5.482531 | 28.026 | 0 | 0 | 0 | 0 | 0.51 | 0.23 | 732.44 | 758.39 | 43.43 | 44.87 | 58.73 | 60.59 |
| (51.034, 3.4) | (40.0, 180.0) | 114164.7 | 145157.7 | 107.945 | 53.449 | 0.001 | 53.237 | 0.053707 | 53.206 | 0 | 0 | 0 | 0 | 0.45 | 0.3 | 732.44 | 758.39 | 43.43 | 43.44 | 58.73 | 58.76 |
| (51.034, 3.4) | (40.0, 180.0) | 114164.7 | 145157.7 | 107.945 | 54.521 | 0.001 | 54.165 | 0.053707 | 54.136 | 0 | 0 | 0 | 0 | 0.4 | 0.37 | 732.44 | 758.39 | 43.43 | 43.44 | 58.73 | 58.76 |
| (51.034, 3.4) | (40.0, 180.0) | 114164.7 | 145157.7 | 107.945 | 55.16 | 0.001 | 55.091 | 0.053707 | 55.065 | 0 | 0 | 0 | 0 | 0.35 | 0.44 | 732.44 | 758.39 | 43.43 | 43.44 | 58.73 | 58.76 |
| (51.034, 3.4) | (40.0, 180.0) | 114164.7 | 145157.7 | 107.945 | 55.797 | 0.001 | 56.017 | 0.52807 | 55.779 | 0 | 0 | 0 | 0 | 0.3 | 0.51 | 732.44 | 758.39 | 43.43 | 43.51 | 58.73 | 58.93 |
| (51.034, 3.4) | (40.0, 180.0) | 114164.7 | 145157.7 | 107.945 | 56.434 | 0.001 | 56.941 | 1.265969 | 56.425 | 0 | 0 | 0 | 0 | 0.25 | 0.58 | 732.44 | 758.39 | 43.43 | 43.62 | 58.73 | 59.07 |
| (51.034, 3.4) | (40.0, 180.0) | 114164.7 | 145157.7 | 107.945 | 57.07 | 0.001 | 57.864 | 2.214695 | 57.054 | 0 | 0 | 0 | 0 | 0.2 | 0.65 | 732.44 | 758.39 | 43.43 | 43.76 | 58.73 | 59.26 |
| (51.034, 3.4) | (40.0, 180.0) | 114164.7 | 145157.7 | 107.945 | 57.705 | 0.001 | 58.785 | 3.321543 | 57.7 | 0 | 0 | 0 | 0 | 0.15 | 0.72 | 732.44 | 758.39 | 43.43 | 44.05 | 58.73 | 59.55 |
| (51.034, 3.4) | (40.0, 180.0) | 114164.7 | 145157.7 | 107.945 | 58.34 | 0.001 | 59.706 | 4.744633 | 58.329 | 0 | 0 | 0 | 0 | 0.1 | 0.79 | 732.44 | 758.39 | 43.43 | 44.62 | 58.73 | 60.3 |
| (51.034, 3.4) | (40.0, 180.0) | 114164.7 | 145157.7 | 107.945 | 58.973 | 0.001 | 60.625 | 6.483965 | 58.96 | 0 | 0 | 0 | 0 | 0.05 | 0.86 | 732.44 | 758.39 | 43.43 | 45.05 | 58.73 | 60.79 |

Figure 73: Excel sheet of Main Code data result obtained by varying both selling and purchasing price in a mutual opposite way

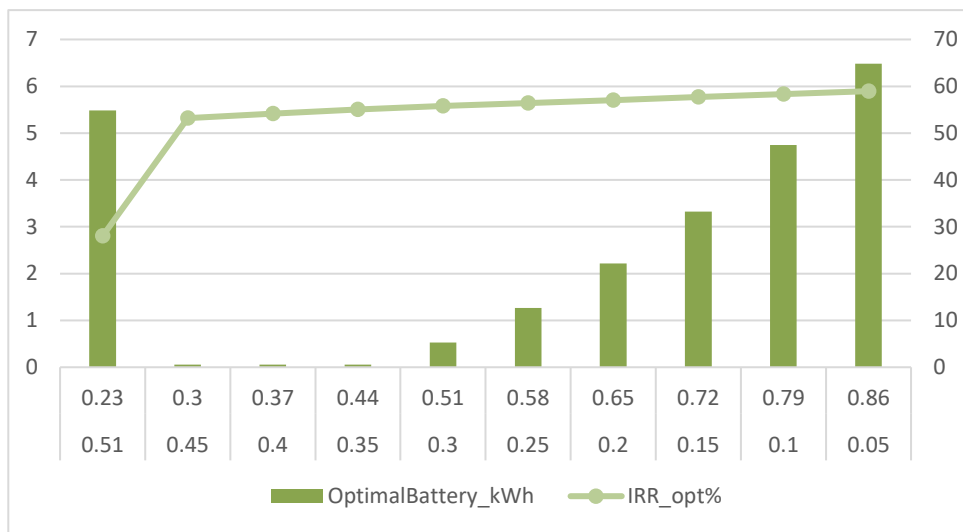


Figure 74: Excel chart obtained by varying both selling and purchasing price in a mutual opposite way

These tests related to the second analysis show that an increase in the energy selling price is related to a decrease in the optimal battery size. On the other hand, when the purchase price of energy goes up, the IRR of the PV plant+optimal battery size investment increases.

The goal ,for any investor, of being able to get the highest possible economic return from an investment while minimizing the cost of that investment, translates, in our case of interest, into trying to get the IRR of the investment related to PV plant+ optimal battery capacity as high as possible while reducing the size of the optimal battery capacity.

However, the need to reduce the size of the optimal battery capacity, thereby reducing the expense,must take into account the improvement in self-consumption and self-sufficiency intended to be achieved by installing a battery storage system. An increase in battery size, in fact, leads to an increase in the energy performance of the building.This increase in size, however, must still remain within the limits of a more profitable investment than PV plant alone.

The goal then is to have a high optimal battery capacity size to increase energy performance but still have a high IRR to ensure the profitability of the investment. Having a low BESS investment cost and a high PV plant investment cost results in an increase in the optimal battery size and also the high PV plant investment cost leads to a decrease in IRR. As already noted, having a low selling price and a high purchase price leads to an increase in IRR and the size of the optimal battery capacity required. In order to have both an IRR and a size of the required optimal battery capacity as high as possible, it is necessary to have both the investment cost of PV plant and the investment cost of BESS with an economic gap as small as possible between them. The

opposite goes for the energy price. This is far from illogical since if it costs more to buy the energy than to sell it then it makes sense to produce it with a PV plant. If, however, the PV plant's investment cost is really high and selling energy to the grid at a low price does not give a valid economic return it is better to use a high battery capacity to store and consume that energy. This is the more possible the lower the investment cost is and by doing so there would be an increase in energy performance by improving self-consumption and self-sufficiency. By no longer buying this energy from the grid at a high price a high IRR of the investment is guaranteed.

CONCLUSIONS

The output results of the final code show how it provides a useful tool that can optimize the energy performance and economic viability of BESS in the construction of photovoltaic plants. The results of the analysis conducted by varying the sale and purchase prices, as well as the investment costs of BESS and PV, lead to the conclusion that achieving the highest possible economic return from the investment while minimizing its cost requires a balance between the size of the optimal battery capacity and the IRR of the investment. Increasing the optimal battery size leads to improved energy performance, but it must still remain within the limits of a more profitable investment than PV plant alone. To achieve this balance, it is necessary to have a high investment cost of PV plant and a low investment cost of BESS with a wide economic gap between them. Additionally, if the investment costs of the PV plant is high, it is better to use a high battery capacity to store and consume energy, resulting in increased energy performance and in the highest possible IRR of the investment. Overall, the findings emphasize the importance of carefully considering the trade-offs between investment costs, battery size, and IRR in achieving the highest possible economic return while improving energy performance.

APPENDIX 1

MainCode.py

```
###
#####
##### OPTIMIZING BESS FOR CONSUMPTION 5 OFFICE BUILDING FLOORS UPGRADE ESTATE #####
#####

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.metrics import mean_absolute_percentage_error, mean_absolute_error
import matplotlib.dates as mdates
import numpy_financial as npf
from numpy_financial import npv, irr, pv
from sklearn.metrics import mean_absolute_percentage_error, mean_absolute_error
import pickle

#-----
#----- FINANCIAL ANALYSIS -----
#-----

#ricorda:investimento iniziale negativo all'inizio e i risparmi positivi dopo

def payback(cashflow):
    investment, cashflow = cashflow[0], cashflow[1:]
    if investment < 0 : investment = -investment
    return payback_of_investment(investment, cashflow)

def payback_of_investment(investment, cashflows):
    total, years, cumulative = 0.0, 0, []
    if not cashflows or (sum(cashflows) < investment):
        raise Exception("insufficient cashflows")
    for cashflow in cashflows:
        total += cashflow
    if total < investment:
        years += 1
    cumulative.append(total)
    A = years
    B = investment - cumulative[years-1]

    C = cumulative[years] - cumulative[years-1]
    return A + (B/C)

#DISCOUNTED PAYBACK PERIOD
#https://sushanthukeri.wordpress.com/2017/03/29/discounted-payback-periods/

#-----
# START CODE
#-----

#####
# ASK INPUT VALUES
#####

import keyboard
import sys

def get_input(prompt, default_value):
    value = default_value
    while True:
        try:
            value = float(input(prompt) or default_value)
            break # exit loop if valid input is provided
        except keyboard.is_pressed('Esc'):
            sys.exit()
        except ValueError:
            print("Invalid input. Please enter a valid number.")
            if keyboard.is_pressed('Esc'):
                sys.exit()
    return value

ask_modify_all_inputs=input('Do you want to manually edit all the inputs?: [Y] or [Enter to modify only main data] ')

if ask_modify_all_inputs.lower()=='y':
    insert_loadxlsx_file=input('insert LOAD file between brackets': [namefile.xlsx or Enter to continue with default]:') or 'datiloadmodelUppest.xlsx'
    latitude=get_input("latitude : [00.00]", 51.034)
    longitude=get_input("longitude : [00.00]", 3.695)

#####
#####
# per il momento metti di default 40,180 ma ricorda che se se vuoi poter calcolare una loro approssimazione,
# anzichè mettere la tua di default, metti: ],0) a entrambi
tilt=get_input("tilt : [panel flat tilt=0 - panel standing on its edge tilt=90, selected 40]", 40)
```

```

azimuth=get_input("azimuth : [ N=0, E=90, S=180, W=270.selected 180]", 180)
#####
pv_size_wp=get_input("PV size : [Wp]", 0)
TrialBatt_kWh = get_input("trial battery capacity : [kWh]", 0)
energy_cost_vendita_input = get_input("electricity selling price : [0.15 euro/kWh]", 0.15)
energy_cost_acquisto_input = get_input("electricity purchase price : [0.51 euro/kWh]", 0.51)
pv_installation_cost = get_input("PV installation cost : [732.44 euro/kWp]", 732.44)
OeM_cost = get_input("operation and maintenance cost : [euro/kWp/year]", 9.4)
year_life_pvplant_input = int(get_input("solar panels lifespan : [year]", 25))
nominal_discount_rate = get_input("nominal discount rate : [0.000]", 0.05)
expected_inflation_rate = get_input("expected electricity price inflation rate : [0.000]", 0.00)
real_discount_rate_input = round(((nominal_discount_rate-expected_inflation_rate)/(1+expected_inflation_rate)), 3)
BESS_cost_input = get_input("battery energy storage system cost : [758.39 euro/kWh]", 758.39)
inverter_pv_cost = 0.12 * pv_installation_cost
inverter_bpv_cost = 0.10 * (pv_installation_cost + BESS_cost_input)
annual_module_degradation_input = get_input("yearly panels degradation rate : [0.000]", 0.005)
year_first_replacement_battery_input = int(get_input("year of first battery replacement : [year]", 10))
year_second_replacement_battery_input = int(get_input("year of second battery replacement : [year]", 20))
year_replacement_inverters_input = int(get_input("year of inverters replacement : [year]", 15))
SOC_max_input = get_input("maximum State Of Charge : [0.000]", 1)
SOC_min_input = get_input("minimum State Of Charge : [0.000]", 0.2)
SOC0_input = get_input("initial State Of Charge : [0.000]", 0.9)
else:
insert_loadxlsx_file=input('insert LOAD file between brackets': [namefile.xlsx or Enter to continue with default:]') or 'datiloadmodelUpEst.xlsx'
latitude=get_input("latitude : [00.00]", 51.034)
longitude=get_input("longitude : [00.00]", 3.695)
#####
# per il momento metti di default 40,180 ma ricorda che se se vuoi poter calcolare una loro approssimazione,
# anziché mettere la tua di default, metti: ]',0) a entrambi
tilt=get_input("tilt : [panel flat tilt=0 - panel standing on its edge tilt=90, selected 40]", 40)
azimuth=get_input("azimuth : [ N=0, E=90, S=180, W=270.selected 180]", 180)
#####
pv_size_wp=float(get_input("PV size : [110000 Wp]", 0))
TrialBatt_kWh = get_input("trial battery capacity : [kWh]", 0)
energy_cost_vendita_input = get_input("electricity selling price : [0.15 euro/kWh]", 0.15)
energy_cost_acquisto_input = get_input("electricity purchase price : [0.51 euro/kWh]", 0.51)
pv_installation_cost = get_input("PV installation cost : [732.44 euro/kWp]", 732.44) #732.44euro/kWp (782$/kWp in BELGIUM) IRENA Renewable power generation costs in
2021,pag.89
OeM_cost=9.4 # 10 $/kWp/y IRENA Renewable power generation costs in 2021,pag.95,EUROPE
year_life_pvplant_input= 25 #years
nominal_discount_rate= 0.05 #0.05 renewable-energy-discount-rate-survey-results-2018->france and germany have 5%
expected_inflation_rate=0.00
real_discount_rate_input=round(((nominal_discount_rate-expected_inflation_rate)/(1+expected_inflation_rate)),3) #HOMER
BESS_cost_input = get_input("battery energy storage system cost : [758.39 euro/kWh]", 758.39) #758.39 euro/kWh pag.954 IRENA(media tra tutti 806.8 $/kWh)---conversione
1$=0.94euro al 15 febbraio 2023

inverter_pv_cost=0.12*pv_installation_cost
inverter_bpv_cost=0.10*(pv_installation_cost+BESS_cost_input)
annual_module_degradation_input=0.005 #annuo Techno-economic analysis of #photovoltaic battery systems and the influence #of different consumer load profiles
year_first_replacement_battery_input=10 #Optimal planning of solar photovoltaic and battery storage systems for grid-connected residential sector: Review, challenges and new
perspectives
year_second_replacement_battery_input=20
year_replacement_inverters_input=15
SOC_max_input=1
SOC_min_input = 0.2
SOC0_input= 0.9

#####
# EVALUATE PRODUCTION AND IMPORT CONSUMPTION
#####
import function_script_pvmodel
hourly_PV_synth_prod_UpEst,hourly_building_cons_UpEst,yearly_production,yearly_consumption,pv_size=function_script_pvmodel.Simple_PVWatts_array_power(pv_size_wp,latitude,lon
hourly_building_cons_UpEst=function_script_pvmodel.Simple_PVWatts_array_power(pv_size_wp,latitude,longitude,slope,azimuth,insert_loadxlsx_file)
yearly_production_kWh=(yearly_production)/1000
yearly_consumption_kWh=(yearly_consumption)/1000

print('\nyearly production:',round(((yearly_production)/1000),2),'kWh')
print('yearly consumption:',round(((yearly_consumption)/1000),2),'kWh')

pv_size=round((pv_size/1000),3) #kWp
print('PV_SIZE:',pv_size,'kWp')
pv_investment_input = pv_installation_cost * pv_size
inverters_tot_cost_input = inverter_pv_cost * pv_size
inverters_tot_cost_bpv_input = inverter_bpv_cost * pv_size
OeM_tot_cost_input = OeM_cost * pv_size

av_mism_hour_upest_1=pd.DataFrame(hourly_PV_synth_prod_UpEst['PV_prod_synthetic']-hourly_building_cons_UpEst['P_tot_build_mod'])

Em_input=av_mism_hour_upest_1['h_av_mismatch'].values.tolist()

```



```

#####
# EVALUATE SOC & ECONOMICAL ANALISYS
#####

def
bestbattery(Bess_tot_size,energy_cost_vendita,energy_cost_acquisto,year_life_pvplant,real_discount_rate,BESS_cost,annual_module_degradation,year_first_replacement_battery,
year_second_replacement_battery,year_replacement_inverters,pv_investment,inverters_tot_cost,inverters_tot_cost_bpv,0eM_tot_cost,SOC_max,SOC_min,SOC0,Em,av_mism_hour_upest_1):

Eb_tot0=SOC0*Bess_tot_size*1000 #[Wh] battery size
Eb_tot=Bess_tot_size*1000
SOC=[SOC0]
Eb_new=[Eb_tot0]
Eg=[0]
check_batt_en_supplied=[0]

for ehm in Em:
SOC_in=SOC[-1]
Eb=(SOC_in*Eb_tot)+ ehm
if ehm>=0:

if (Eb/Eb_tot)< SOC_max:
Eb_new.append(Eb)
SOC.append(Eb/Eb_tot)
Eg.append(0)
check_batt_en_supplied.append(Eb_new[-1]-Eb_new[-2])
else:
x_ehm=(SOC_max-SOC_in)*Eb_tot
Eb_new.append(SOC_max*Eb_tot)
SOC.append(SOC_max)
Eg.append(ehm-x_ehm)
check_batt_en_supplied.append(Eb_new[-1]-Eb_new[-2])

else:

if (Eb/Eb_tot) > SOC_min:
Eb_new.append(Eb)
SOC.append(Eb/Eb_tot)
Eg.append(0)
check_batt_en_supplied.append(Eb_new[-1]-Eb_new[-2])
else:
x_ehm=(SOC_min-SOC_in)*Eb_tot
Eb_new.append(SOC_min*Eb_tot)
SOC.append(SOC_min)

Eg.append(ehm-x_ehm)
check_batt_en_supplied.append(Eb_new[-1]-Eb_new[-2])
#Else:
#x_ehm=(SOC_min-SOC_in)*Eb_tot
#Eb_new.append(SOC_min*Eb_tot)
#SOC.append(SOC_min)
#Eg.append(ehm-x_ehm)
#check_batt_en_supplied.append(Eb_new[-1]-Eb_new[-2])
# add a fake final value to the lists Eg and check_batt_en_supplied
# because they didn't have an initial value so they have 1 lass value
# and it cannot be possible to join them into a dataframe-> different lenght
Em=pd.DataFrame(Em,columns=['h_av_mismatch'])
new_row = pd.DataFrame({'h_av_mismatch':0},index =[0])
# simply concatenate both dataframes
Em = pd.concat([new_row, Em]).reset_index(drop = True)
Em.head(5)
#Creates data frame
SOC_100=[]
for i in SOC:
SOC_100.append(round((i * 100),1))

df2=Em.copy()
hour_time=pd.date_range(start='2021-07-01 00:00 ',end='2022-06-30 23:59',periods=8761)
df2.set_index(hour_time,inplace=True)

data_dict1 = {"W mism": df2['h_av_mismatch'], "supplied_check":check_batt_en_supplied, "SOC%":SOC_100 , "battery_charge":Eb_new, "gain":Eg}

SOC_df= pd.DataFrame(data_dict1)
SOC_df.set_index(hour_time,inplace=True)
#Plots
#add an initial zero value in order to print it
plot_prod=hourly_pv_synth_prod_UpEst.copy()
new_row_pv = pd.DataFrame({'PV_prod_synthetic':0},index =[0])
plot_prod= pd.concat([new_row_pv, plot_prod]).reset_index(drop = True)
plot_prod.set_index(hour_time,inplace=True)
#plot_prod.head(5)

plot_cons=hourly_building_cons_UpEst.copy()
new_row_cons = pd.DataFrame({'P_tot_build_mod':0},index =[0])
plot_cons= pd.concat([new_row_cons, plot_cons]).reset_index(drop = True)
plot_cons.set_index(hour_time,inplace=True)
#plot_cons.head(5)

new_df_tot=pd.concat([plot_prod['PV_prod_synthetic'],plot_cons['P_tot_build_mod'],SOC_df['W
mism'],SOC_df['gain'],SOC_df['supplied_check'],SOC_df['SOC%'],SOC_df['battery_charge']],axis=1)

```

```

#monthly mismatch
#anche qui il primo mese è luglio

m_mes1_UpEst=av_mism_hour_upest_1.iloc[0:731,:]
m_mes2_UpEst=av_mism_hour_upest_1.iloc[730:1461,:]
m_mes3_UpEst=av_mism_hour_upest_1.iloc[1460:2191,:]
m_mes4_UpEst=av_mism_hour_upest_1.iloc[2190:2921,:]
m_mes5_UpEst=av_mism_hour_upest_1.iloc[2920:3651,:]
m_mes6_UpEst=av_mism_hour_upest_1.iloc[3650:4381,:]
m_mes7_UpEst=av_mism_hour_upest_1.iloc[4380:5111,:]
m_mes8_UpEst=av_mism_hour_upest_1.iloc[5110:5841,:]
m_mes9_UpEst=av_mism_hour_upest_1.iloc[5840:6571,:]
m_mes10_UpEst=av_mism_hour_upest_1.iloc[6570:7301,:]
m_mes11_UpEst=av_mism_hour_upest_1.iloc[7300:8031,:]
m_mes12_UpEst=av_mism_hour_upest_1.iloc[8030:8761,:]

meslistmisl=(m_mes1_UpEst,m_mes2_UpEst,m_mes3_UpEst,m_mes4_UpEst,m_mes5_UpEst,m_mes6_UpEst,m_mes7_UpEst,m_mes8_UpEst,m_mes9_UpEst,m_mes10_UpEst,m_mes11_UpEst,m_mes12_UpEst)
#print(meslistmisl)

#----sum POSITIVE MISMATCH during each day [Wh]-----
upest_pos_av_mism_d_11=[]

for ciaschmesmisl in meslistmisl:
    mism_month_ar1=ciaschmesmisl.to_numpy()
    upest_positive_av_mism_d1=mism_month_ar1[mism_month_ar1 >= 0]
    upest_positive_av_mism_d1_SUM=round(upest_positive_av_mism_d1.sum(),2)
    upest_pos_av_mism_d_11.append(upest_positive_av_mism_d1_SUM)
    upest_pos_av_mism_d0=pd.DataFrame(upest_pos_av_mism_d_11,columns=['sum pos mism month'])
    #print('-SUM of each positive mismatch month kWh:\n',round((upest_pos_av_mism_d0/1000),2))
    #print('-DESCRIBE in kWh:\n',round((upest_pos_av_mism_d/1000).describe(),2))

#----sum POSITIVE MISMATCH during each day YEAR [Wh]-
#####
#dopo aver preso la media annua per ogni ora 00->23 (hour_mism_year_upest) -----
#-----> prendo solo le ore con mism positivo e le sommo
#####
yearmismpos=av_mism_hour_upest_1[av_mism_hour_upest_1>= 0]
yearmismposum=round(yearmismpos.sum(),2)
yearmismposum2=round(upest_pos_av_mism_d0.sum(),2)
#print(yearmismposum,yearmismposum2)

#----sum NEGATIVE MISMATCH during each day [Wh]-----
upest_neg_av_mism_d_12=[]
for sembupest_neg2 in meslistmisl:

    upest_negative_av_mism_days2=mism_month_ar2[mism_month_ar2 < 0]
    upest_negative_av_mism_days2_SUM=round(upest_negative_av_mism_days2.sum(),2)
    upest_neg_av_mism_d_12.append(upest_negative_av_mism_days2_SUM)

    #print('-SUM of each negative mismatch month kWh:\n',round((upest_neg_av_mism_d00/1000),2))
    #print('-DESCRIBE in kWh:\n',round((upest_neg_av_mism_d/1000).describe(),2))

#----sum NEGATIVE MISMATCH during each day YEAR [Wh]-
#####
#dopo aver preso la media annua per ogni ora 00->23 (hour_mism_year_upest) -----
#-----> prendo solo le ore con mism negativo e le sommo
#####
yearmismneg=av_mism_hour_upest_1[av_mism_hour_upest_1< 0]
yearmismnegsum=round(yearmismneg.sum(),2)
yearmismposum3=round(upest_neg_av_mism_d00.sum(),2)
#print('total',yearmismnegsum,'real',yearmismposum3)

c=-93387498.33/-93545829.66
upest_neg_av_mism_d00=upest_neg_av_mism_d00*c
#print('modif',upest_neg_av_mism_d00.sum())

month_en_purchase=pd.DataFrame(round((upest_neg_av_mism_d00/1000),2))
month_en_purchase.columns=['purchase(neg mism)']

month_en_injection=pd.DataFrame(round((upest_pos_av_mism_d0/1000),2))
month_en_injection.columns=['injection(pos mism)']
date_month1=pd.date_range(start='2019-07-01',periods=12,freq='M')
month_en_injection.set_index(date_month1,inplace=True)

#dataframe with monthly prod/cons/pos-mism/neg-mism e come ultima riga la somma annua dei rispettivi
total_month_scheme=pd.concat([month_en_injection['injection(pos mism)'],month_en_purchase['purchase(neg mism)']],axis=1)
new_row_tot=pd.DataFrame({'monthly prod [kWh]':115089.368336,'monthly cons [kWh]':145450.0,'injection(pos mism)':63006.93987,'purchase(neg mism)':-93387.49833},index=[0])
total_month_scheme= pd.concat([total_month_scheme,new_row_tot]).reset_index(drop = True)
date_month_tot=pd.date_range(start='2019-07-01',periods=13,freq='M')
total_month_scheme.set_index(date_month_tot,inplace=True)
year_cons_kWh=round(((hourly_building_cons_UpEst['P_tot_build_mod'].sum())/1000),2)
year_pos_mism_kWh=round((total_month_scheme['injection(pos mism)'].iloc[-1]),0)
year_neg_mism_kWh=round((total_month_scheme['purchase(neg mism)'].iloc[-1]),0)

#without PV plant

```

```

year_energy_bill=round((year_cons_kWh*energy_cost_acquisto),2)

#with PV plant
#positive income
year_en_pos_income=year_pos_mism_kWh*energy_cost_vendita
#negative income
year_en_neg_income=year_neg_mism_kWh*energy_cost_acquisto

#degradation affects production thus positive income
pos_inc_list = [year_en_pos_income]
for i in range(year_life_pvplant):
new_posinc = pos_inc_list[-1] * (1 - annual_module_degradation)
pos_inc_list.append(new_posinc)

neg_inc_list = [year_en_neg_income]*year_life_pvplant
final_bill_list = [abs(x + y) for x, y in zip(pos_inc_list, neg_inc_list)]
savings_listas=[ year_energy_bill-x for x in final_bill_list]

#ECONOMICAL ANALISYS PV ONLY
cashflows_onlyPV=[-pv_investment]
for irisp_pv in savings_listas:
cashflows_onlyPV.append(irisp_pv - OeM_tot_cost)

cashflows_onlyPV[year_replacement_inverters]=cashflows_onlyPV[year_replacement_inverters]-(inverters_tot_cost)

ec_an=[round(npf.npv(real_discount_rate,cashflows_onlyPV),2)]
ec_an.append(round((npf.irr(cashflows_onlyPV))*100,3)) #%
ec_an.append(round(payloadback(cashflows_onlyPV),2))
#print('ECONOMICAL ANALISYS onlyPV',ec_an)

#with BESS+PV
#with BESS+PV
bess_gain=new_df_tot['gain'] *Wh

#positive income
gain_pos=bess_gain[bess_gain>=0]
gain_pos_year_kWh=((gain_pos.sum())/1000)
year_gain_pos_income=gain_pos_year_kWh*energy_cost_vendita

#negative income
gain_neg=bess_gain[bess_gain<0]
gain_neg_year_kWh=((gain_neg.sum())/1000)
year_gain_neg_income=gain_neg_year_kWh*energy_cost_acquisto

#degradation affects production thus positive income
pos_inc_list_bpv = [year_gain_pos_income]
for i in range(year_life_pvplant):
new_posinc_bpv = pos_inc_list_bpv[-1] * (1 - annual_module_degradation)
pos_inc_list_bpv.append(new_posinc_bpv)

neg_inc_list_bpv = [year_gain_neg_income] * year_life_pvplant
final_bill_list_bpv = [abs(x + y) for x, y in zip(pos_inc_list_bpv, neg_inc_list_bpv)]
savings_listas_bpv = [year_energy_bill - x for x in final_bill_list_bpv]

#ECONOMICAL ANALISYS
investimeto_bpv=pv_investment+(BESS_cost*(Bess_tot_size))
cashflows_bpv=[-investimeto_bpv]
for i_bpv in savings_listas_bpv:
cashflows_bpv.append(i_bpv - OeM_tot_cost)

cashflows_bpv[year_replacement_inverters]=cashflows_bpv[year_replacement_inverters]-(OeM_tot_cost)
cashflows_bpv[year_first_replacement_battery]=cashflows_bpv[year_first_replacement_battery]-(BESS_cost*(Bess_tot_size))

if year_second_replacement_battery==0:
cashflows_bpv[year_second_replacement_battery]=cashflows_bpv[year_second_replacement_battery]-0
else:
cashflows_bpv[year_second_replacement_battery]=cashflows_bpv[year_second_replacement_battery]-(BESS_cost*(Bess_tot_size))

ec_an_bpv=[(npf.npv(real_discount_rate,cashflows_bpv))]
ec_an_bpv.append(round((npf.irr(cashflows_bpv))*100,3)) #%
ec_an_bpv.append(payloadback(cashflows_bpv))
#print('ECONOMICAL ANALISYS PV+BESS',ec_an_bpv)
economical_analisis=np.array([ec_an,ec_an_bpv])
return economical_analisis[1,1],economical_analisis[0,1],economical_analisis,SOC_df

#####
# BESS OPTIMIZATION
#####

# define the objective function for optimization IRR_bpv

```

```

def objective_func_IRR_bpv(x):
    return
bestbattery(x,energy_cost_vendita_input,energy_cost_acquisto_input,year_life_pvplant_input,real_discount_rate_input,BESS_cost_input,annual_module_degradation_input,year_first_r
year_second_replacement_battery_input,year_replacement_inverters_input,pv_investment_input,inverters_tot_cost_input,inverters_tot_cost_bpv_input,OeM_tot_cost_input,SOC_max_inpu

# define the objective function for optimization IRR_pv
def objective_func_IRR_pv(x):
    return
bestbattery(x,energy_cost_vendita_input,energy_cost_acquisto_input,year_life_pvplant_input,real_discount_rate_input,BESS_cost_input,annual_module_degradation_input,year_first_r
year_second_replacement_battery_input,year_replacement_inverters_input,pv_investment_input,inverters_tot_cost_input,inverters_tot_cost_bpv_input,OeM_tot_cost_input,SOC_max_inpu

# define the objective function economical analisys
def objective_func_economical_analisys(x):
    return
bestbattery(x,energy_cost_vendita_input,energy_cost_acquisto_input,year_life_pvplant_input,real_discount_rate_input,BESS_cost_input,annual_module_degradation_input,year_first_r
year_second_replacement_battery_input,year_replacement_inverters_input,pv_investment_input,inverters_tot_cost_input,inverters_tot_cost_bpv_input,OeM_tot_cost_input,SOC_max_inpu

import numpy as np
from scipy.optimize import minimize

# Set up multiple starting points
start_points=[]
for sp in range(1,int(round(pv_size)),10):
    start_points.append(sp)
start_points[0]=0.001

# Find battery capacity that maximizes IRR with constraint and multiple starting points
max_irr = 0
best_capacity_kWh = None #[kWh]
for initial_guess in start_points:
    res = minimize(lambda x: -objective_func_IRR_bpv(x[0]), x0=np.array([initial_guess]),bounds=[(0.001, None)],options={'gtol': 1e-4, 'disp': True})

if -res.fun > max_irr:
    max_irr = -res.fun
best_capacity_kWh = res.x[0]

BestCap_second_result =
bestbattery(best_capacity_kWh,energy_cost_vendita_input,energy_cost_acquisto_input,year_life_pvplant_input,real_discount_rate_input,BESS_cost_input,annual_module_degradation_in
year_second_replacement_battery_input,year_replacement_inverters_input,pv_investment_input,inverters_tot_cost_input,inverters_tot_cost_bpv_input,OeM_tot_cost_input,SOC_max_inpu
[2]
BestCap_irr_bpv = BestCap_second_result[1,1]

BestCap_npv_pv= BestCap_second_result[0,0]
BestCap_irr_pv= BestCap_second_result[0,1]
BestCap_pbt_pv= BestCap_second_result[0,2]

    t('\nPVonly NPV:',BestCap_npv_pv,'euro')
print('\nPVonly IRR:',BestCap_irr_pv,'%')
print('\nPVonly PBT:',BestCap_pbt_pv,'years')

def find_min_battery_capacity(battery_capacity_range,capacity_maximize_IRR_bpv, IRR_bess_PV_func, IRR_onlyPV_func, tolerance=0.0001):
# battery_capacity_range: intervallo di ricerca di battery_capacity
# IRR_bess_PV_func: funzione che calcola IRR_bess+PV dato battery_capacity
# IRR_onlyPV_func: funzione che calcola IRR_onlyPV dato battery_capacity
# tolerance: precisione richiesta

# applica la ricerca binaria all'intervallo di ricerca
lower, upper = battery_capacity_range
while upper - lower > tolerance:
    mid = (lower + upper) / 2
    IRR_bess_PV = IRR_bess_PV_func(mid)
    IRR_onlyPV = IRR_onlyPV_func(mid)
    if capacity_maximize_IRR_bpv < mid:
    if IRR_bess_PV < IRR_onlyPV:
        upper = mid
    else:
        lower = mid
    else:
    if IRR_bess_PV < IRR_onlyPV:
        lower = mid
    else:
        upper = mid

# restituisci il valore di battery_capacity che soddisfa la condizione
return upper

battery_cap_bound_range = [0.001, pv_size]
min_battery_capacity = find_min_battery_capacity(battery_cap_bound_range,best_capacity_kWh,objective_func_IRR_bpv,objective_func_IRR_pv,tolerance=0.1)
print('\nMinimum profitable battery capacity: ',round(min_battery_capacity,3),'KWh')
NPV_bpv_optimal=objective_func_economical_analisys(min_battery_capacity)[1,0]
IRR_bpv_optimal=objective_func_IRR_bpv(min_battery_capacity)
PBT_bpv_optimal=objective_func_economical_analisys(min_battery_capacity)[1,2]

```

```

print('NPV_bpv_optimal:',NPV_bpv_optimal,'euro')
print('IRR_bpv_optimal:',IRR_bpv_optimal,'%')
print('PBT_bpv_optimal:',PBT_bpv_optimal,'years')

if best_capacity_kWh is not None:
if min_battery_capacity < 1 and best_capacity_kWh ==1 :
best_capacity_kWh=0.001
print('\nBattery capacity that maximizes IRR:', best_capacity_kWh, 'kWh')
    BestCap_irr_bpv, '%'')
else:
print('\nBattery capacity that maximizes IRR:', round(best_capacity_kWh, 3), 'kWh')
print('max_IRR:', BestCap_irr_bpv, '%')
else:
best_capacity_kWh=0.001
print('Error: best_capacity_kWh is not defined')

import PERF_analisis

if TrialBatt_kWh >=0.001 :
print('\nTrial battery capacity:',TrialBatt_kWh,'kWh')
TrialBatt_second_result =
bestbattery(TrialBatt_kWh,energy_cost_vendita_input,energy_cost_acquisto_input,year_life_pvplant_input,real_discount_rate_input,BESS_cost_input,annual_module_degradation_input,
year_second_replacement_battery_input,year_replacement_inverters_input,pv_investment_input,inverters_tot_cost_input,inverters_tot_cost_bpv_input,Oem_tot_cost_input,SOC_max_inpu

TrialBatt_irr_bpv = TrialBatt_second_result[1,1]

TrialBatt_npv_bpv = TrialBatt_second_result[1,0]
TrialBatt_pbt_bpv= TrialBatt_second_result[1,2]
print('\nTrialBatt_NPV:', TrialBatt_npv_bpv, 'euro')
print('TrialBatt_IRR:', TrialBatt_irr_bpv, '%')
print('TrialBatt_PBT:', TrialBatt_pbt_bpv, 'years')

print('\n---TRIAL BATTERY CAPACITY CHART---')
SOC_df_TrialBat =
bestbattery(TrialBatt_kWh,energy_cost_vendita_input,energy_cost_acquisto_input,year_life_pvplant_input,real_discount_rate_input,BESS_cost_input,annual_module_degradation_input,
year_second_replacement_battery_input,year_replacement_inverters_input,pv_investment_input,inverters_tot_cost_input,inverters_tot_cost_bpv_input,Oem_tot_cost_input,SOC_max_inpu
[-1])
figure,fifty=PERF_analisis.perf_analisis(hourly_PV_synth_prod_UpEst,hourly_building_cons_UpEst,SOC_df_TrialBat)
print(figure)
print(fifty)

else:
TrialBatt_irr_bpv=0
TrialBatt_npv_bpv = 0
TrialBatt_pbt_bpv= 0

print('\n---MINIMUM PROFITABLE BATTERY CAPACITY CHART---')
SOC_df_optCap =
bestbattery(min_battery_capacity,energy_cost_vendita_input,energy_cost_acquisto_input,year_life_pvplant_input,real_discount_rate_input,BESS_cost_input,annual_module_degradation
year_second_replacement_battery_input,year_replacement_inverters_input,pv_investment_input,inverters_tot_cost_input,inverters_tot_cost_bpv_input,Oem_tot_cost_input,SOC_max_inpu

figure,fifty=PERF_analisis.perf_analisis(hourly_PV_synth_prod_UpEst,hourly_building_cons_UpEst,SOC_df_optCap)
print(figure)
print(fifty)

#####
##### CREATING EXCEL FILE WITH RESULT #####
#####

dataf_res = pd.DataFrame(columns=
['_lat/lon','tilt/azimuth','Year_prod_kWh','Year_cons_KWh','PVsize_KW','IRR_onlyPV%','NPV_onlyPV','PBTtime_onlyPV','BestBattery_kWh','IRR_best%','OptimalBattery_kWh','IRR_opt%',
%','SS bess %','SC %','SC bess %'])#

# Aggiunta di nuovi risultati al DataFrame

flat_list = [item for sublist in fifty_list for item in sublist]

```

APPENDIX 2

function_script_pvmodel.py

```
###
##### THIS IS A FUNCTION THAT TAKES A TMY.CSV FILE AND RETURN THE PV HOURLY POWER #####

:#####
##### Simple PVWatts array power #####

# Here we will demonstrate one of the more basic PV models implemented by pvlib.
# The PVWatts module model requires only two array parameters -
# the array size (nameplate capacity) and the array's efficiency change with cell temperature.
✓ Typical array sizes range from a few kW for residential arrays to hundreds of MW for large
# utility-scale systems. The cell temperature response parameter,
# often called the module's temperature coefficient, determines the efficiency loss for a temperature increase.
# For example, a module that produces 100W at test conditions but only 95W when the cell temperature is
# increased by 10 degrees has a temperature coefficient of -0.5%/°C. Typical temperature coefficients
# range from -0.5%/°C to -0.2%/°C.

import os # for getting environment variables
import pathlib # for finding the example dataset
import pvlib
import pandas as pd # for data wrangling
import matplotlib.pyplot as plt # for visualization
import numpy as np
from pvlib import location
from pvlib import irradiance
import pvlib.iotools
import math

def
Simple_PVWatts_array_power(pv_size_Wp,latitude_def,longitude_def,slope_def,azimuth_def,excel_file_office_building):

#-----consumption data-----
Dati_LPG_office_Build_UpEst = pd.read_excel(excel_file_office_building)
Dati_LPG_office_Build_UpEst.columns=
['N_occ_tot_build', 'N_occ_floor1', 'N_occ_floor2', 'N_occ_floor3', 'N_occ_floor4', 'N_occ_floor5', 'P_real', 'P_tot_build_mo
date_h_y=pd.date_range(start='2019-01-01 00:00',end='2019-12-31 23:59:59',periods=8760)
Dati_LPG_office_Build_UpEst.set_index(date_h_y,inplace=True)
Tot_Year_building_consumption_UpEst=round(Dati_LPG_office_Build_UpEst['P_tot_build_mod'].sum())

hourly_building_cons_UpEst=pd.DataFrame((round(Dati_LPG_office_Build_UpEst['P_tot_build_mod'],2)))

size_step1=yarly_building_consumption/365 #[Wh/day] Dividing the total consumption by 365 days
14) #[Wh/day] Adding 14% losses (PVGIS)
size_step3=size_step2/0.175 #[Wh/day] Dividing by average panel efficiency (17,5%)
pv_size_approx=size_step3/24 #[Wp] Dividing by 24h we obtain the size
return pv_size_approx

pv_estimated_size=pv_est_size(Tot_Year_building_consumption_UpEst) #Wp

if pv_size_Wp == 0:
PV_SIZE=pv_estimated_size
else:
PV_SIZE=pv_size_Wp
```

```

#the output of pvlib.iotools.get_pvgis_tmy is a tuple containing two elements: a dictionary with the weather data
and a string with metadata.
#Since you want to convert the values of the dictionary to a NumPy array, you need to extract the dictionary from
the tuple first.
# Dictionary mapping PVGIS names to pvlib names

# G(h)'ghi', poa_global float Global irradiance on inclined plane (W/m^2)
# Gb(n), 'dni' poa_direct float Beam (direct) irradiance on inclined plane (W/m^2)
# Gd(h) 'dhi', poa_sky_diffuse float Diffuse irradiance on inclined plane (W/m^2)"""
#'RH': 'relative_humidity'
# T2m, temp_air float Air temperature at 2 m (degrees Celsius)
# WS10m, wind_speed float Wind speed at 10 m (m/s)
#'SP': 'pressure',
#'WD10m': 'wind_direction'

# Obtain the weather data from the tuple returned by get_pvgis_tmy as a DataFrame
tmy_data = pvlib.iotools.get_pvgis_tmy(latitude_def,
longitude_def, map_variables=False, url='https://re.jrc.ec.europa.eu/api/')[0]

#time index resample to obtain a hourly year values, for ex. jan 2019-jan 2020
TmY_df_2019=tmy_data.copy()
newfakeindex=pd.date_range(start='2019-01-01 00:00', end='2019-12-31 23:00', freq='H')
TmY_df_2019.set_index(newfakeindex,inplace=True)
tz = 'GMT'

# Create location object to store lat, lon, timezone
site = location.Location(latitude_def, longitude_def, tz=tz)

def get_irradiance( tilt, azimuth):

times =tmy_data.index

POA_irradiance = pvlib.irradiance.get_total_irradiance(
surface_tilt=tilt,
surface_azimuth=azimuth,
dni=tmy_data['Gb(n)'],
ghi=tmy_data['G(h)'],
dhi=tmy_data['Gd(h)'],
solar_zenith=solar_position['apparent_zenith'],
solar_azimuth=solar_position['azimuth'])

return POA_irradiance['poa_global']

# Get irradiance data for summer and winter solstice, assuming 40 degree tilt
# and a south-est facing array (generally south is 180° but on PVGIS is 0 so optimum is
# slope 40, azimuth -2 (sud-est) so --> azimuth 178)
optimal_tilt=latitude_def
azimuth_range = np.arange(0, 360, 10) # degrees
# Calculate the solar irradiance on the panel for each azimuth angle
irradiance = []
for azimuth_opt in azimuth_range:
total_irradiance_opt = get_irradiance( optimal_tilt, azimuth_opt)
irradiance.append(sum(total_irradiance_opt))

# Find the optimal azimuth that maximizes the POA irradiance
optimal_azimuth = azimuth_range[np.argmax(irradiance)]

if slope_def == 0:
tilt1 = optimal_tilt

```

```

else:
    tilt1 = slope_def

if azimuth_def == 0:
    azimuth1 = optimal_azimuth
else:
    azimuth1 = azimuth_def

print(f"The optimal azimuth for a fixed tilt of {tilt1} degrees is {azimuth1} degrees.")
total_irradiance = get_irradiance(tilt1, azimuth1)
total_irradiance=pd.DataFrame(total_irradiance)
total_irradiance.columns=['POA']

#####
##### irradiance dal 2005 al 2020 (TMY) ma disposti ###
##### su base di tempo annua 2019-2020 #####
total_irradiance_2019=total_irradiance.copy()
total_irradiance_2019.set_index(newfakeindex,inplace=True)

#Here we will use the thermal model from the Sandia Array Performance Model (SAPM)
# to estimate cell temperature from ambient conditions
parameters = pvlib.temperature.TEMPERATURE_MODEL_PARAMETERS['sapm']['open_rack_glass_polymer']
cell_temperature = pvlib.temperature.sapm_cell(total_irradiance_2019['POA'],
TmY_df_2019['T2m'],
TmY_df_2019['WS10m'],
**parameters)

#If we wanted to use other parameters, here are the options

#all_parameters = pvlib.temperature.TEMPERATURE_MODEL_PARAMETERS['sapm']
#list(all_parameters.keys())
#['open_rack_glass_glass',
# 'close_mount_glass_glass',
# 'open_rack_glass_polymer',
# 'insulated_back_glass_polymer']

gamma_pdc = -0.004 # divide by 100 to go from %/°C to 1/°C
nameplate = PV_SIZE

array_power = pvlib.pvsystem.pvwatts_dc(total_irradiance_2019['POA'], cell_temperature, nameplate, gamma_pdc)

#AC/DC
#PVWatts has a simplified inverter model.
# Use pvlib.inverter.pvwatts(pdc, pdc0) to return the AC output given DC output,
# pdc, and the DC limit, pdc0 which is the AC rated power over the nominal inverter efficiency.

#Recall we assumed a 110kW array, so we'll continue the hypothetical case and assume an
# AC size of 91.6 kW, a DC/AC ratio of 1.2. The default PVWatts nominal inverter efficiency
# is 0.96 which we use to get pdc0.
ac_siz=PV_SIZE/1.2
pdc0 = ac_siz/0.96 # W
ac = pvlib.inverter.pvwatts(array_power, pdc0)
PV_prod_ac_2019=pd.DataFrame(ac, columns=['PV_prod_ac_model'])

# Load and process production data
#-----production data-----

PV_hourly_synth=PV_prod_ac_2019
PV_hourly_synth.columns=['PV_prod_synthetic']
date_hour_radiationpv=pd.date_range(start='2019-01-01 00:00',end='2019-12-31 23:00',periods=8760)
PV_hourly_synth.set_index(date_hour_radiationpv,inplace=True)

```


APPENDIX 3

PERF_analisis.py

```
###
##### EVALUATE SS & SC #####
#####
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.metrics import mean_absolute_percentage_error,mean_absolute_error
import matplotlib.dates as mdates
import numpy_financial as npf
from numpy_financial import npv, irr,pv
from sklearn.metrics import mean_absolute_percentage_error,mean_absolute_error
import pickle
from sklearn.metrics import mean_absolute_percentage_error,mean_absolute_error

def perf_analisis(hourly_PV_synth_prod_UpEst,hourly_building_cons_UpEst,SOC_df):
##### DAILY production #####
#####

pv_split=np.array_split((hourly_PV_synth_prod_UpEst['PV_prod_synthetic'].to_numpy()), 365)
#-----sum PV production during each day [Wh]
pv_d=[]
for p in pv_split:
pv=pd.DataFrame(p)
pv_sum=pv.sum()
pv_d.append(pv_sum)

##### produzione fotovoltaica giornaliera per un anno #####
pv_sum_d=pd.DataFrame(pv_d)

pv_sum_d.columns=['PV_tot_days']
#print(pv_sum_d.sum()) #check -> should be 115069 kWh

##### DAILY consumption #####
#####

cons_split=np.array_split((hourly_building_cons_UpEst['P_tot_build_mod'].to_numpy()), 365)
#-----sum consumption during each day [Wh]
cons_d=[]
for co in cons_split:
cns=pd.DataFrame(co)
cns_sum=cns.sum()
cons_d.append(cns_sum)

##### consumo energia giornaliero per un anno #####
cons_sum_d=pd.DataFrame(cons_d)
cons_sum_d.columns=['cons_tot_days']
#print(cons_sum_d.sum()) #check -> should be 145450 kWh

##### DAILY mismatch #####
#####

av_mism_orario=pd.DataFrame(hourly_PV_synth_prod_UpEst['PV_prod_synthetic']-
hourly_building_cons_UpEst['P_tot_build_mod'])
av_mism_orario.columns=['h_av_mismatch']
average_mism_days_splt=np.array_split((av_mism_orario['h_av_mismatch'].to_numpy()),365)

#-----sum mismatch during each day [Wh]
avv_mism_d=[]
```

```

for pav in average_mism_days_splt:
peav=pd.DataFrame(pav)
peav_sum=peav.sum()
avv_mism_d=avv_mism_d+ [peav_sum]

##### mismatch giornaliero per un anno #####
average_mism_daily=pd.DataFrame(avv_mism_d)
average_mism_daily.columns=['h_av_mismatch']

#####
##### WITH BESS #####
#####

##### GAIN POSITIVI [Wh]
tot_houly_gain_list=SOC_df[['gain']].to_numpy()

pos_hour_gain0=[]
for eachposgain in tot_houly_gain_list :
if eachposgain > 0 :
pos_hour_gain0.append(eachposgain)

else :
pos_hour_gain0.append(0)

pos_hour_gain0_df=pd.DataFrame(pos_hour_gain0,columns=['pos_hour_gain'])
pos_hour_gain0_daily=np.array_split((pos_hour_gain0_df[['pos_hour_gain']].to_numpy()), 365)

posgdy=[]
for poshgn in pos_hour_gain0_daily:
phmismgain=pd.DataFrame(poshgn)
phmismgain_sum=float(phmismgain.sum())
posgdy.append(phmismgain_sum)

pos_gain_day=pd.DataFrame(posgdy)
pos_gain_day.columns=['pos_gain_day'] #[Wh]

##### GAIN NEGATIVI [Wh]

neg_hour_gain0=[]
for eachneggain in tot_houly_gain_list :
if eachneggain < 0 :
neg_hour_gain0.append(eachneggain)
else :
neg_hour_gain0.append(0)

neg_hour_gain0_df=pd.DataFrame(neg_hour_gain0,columns=['neg_hour_gain'])
neg_hour_gain0_daily=np.array_split((neg_hour_gain0_df[['neg_hour_gain']].to_numpy()), 365)

neggdy=[]
for neghgn in neg_hour_gain0_daily:
nhmismgain=pd.DataFrame(neghgn)

```

```

nhmismgain_sum=float(nhmismgain.sum())
neggdy.append(nhmismgain_sum)

neg_gain_day=pd.DataFrame(neggdy)
neg_gain_day.columns=['neg_gain_day'] #[Wh]

#### sottraggo alla produzione fotovoltaica il positive gain totale giornaliero -> area Elpc per SC #####
PV_minus_posgain=(pv_sum_d['PV_tot_days']-pos_gain_day['pos_gain_day'])
ElpcSC_day_bess=pd.DataFrame(PV_minus_posgain,columns=['ElpcSC_day_bess'])
cons_minus_neggain=(cons_sum_d['cons_tot_days']+neg_gain_day['neg_gain_day'])
ElpcSS_day_bess=pd.DataFrame(cons_minus_neggain,columns=['ElpcSS_day_bess'])

Elpc_day_bess=pd.concat([ElpcSC_day_bess['ElpcSC_day_bess'],ElpcSS_day_bess['ElpcSS_day_bess']],axis=1)
date_d=pd.date_range(start='2019-07-01 00:00',end='2020-06-30 23:59:59',periods=365)
Elpc_day_bess.set_index(date_d,inplace=True)

MAE_Elpc_day_bess=round((mean_absolute_error(Elpc_day_bess['ElpcSC_day_bess'], Elpc_day_bess['ElpcSS_day_bess'],
multioutput='uniform_average')),2)
MAPE_Elpc_day_bess=mean_absolute_percentage_error(Elpc_day_bess['ElpcSC_day_bess'],
Elpc_day_bess['ElpcSS_day_bess'], multioutput='uniform_average')
print('MAPE_Elpc_day_bess:',round((MAPE_Elpc_day_bess*100),2),'%')
#print('MAE:',MAE_Elpc_day_bess,' ->[W] error')

##### NOTA BENE #####
# COULD HAPPEND THAT: the value when some element of the y_true is zero is arbitrarily high because of the division
by epsilon
#y_true = [1., 0., 2.4, 7.] ; y_pred = [1.2, 0.1, 2.4, 8.]; mean_absolute_percentage_error(y_true,
y_pred)=112589990684262.48

if MAPE_Elpc_day_bess > 0.12:
lenght_dataframe=len(Elpc_day_bess)
grouped = np.array_split(Elpc_day_bess.index , lenght_dataframe)
MAE_multioutput=[]
MAPE_multioutput=[]
for w in grouped:
mae=round((mean_absolute_error(Elpc_day_bess['ElpcSC_day_bess'].loc[w], Elpc_day_bess['ElpcSS_day_bess'].loc[w],
multioutput='uniform_average')),1)
mape=round((mean_absolute_percentage_error(Elpc_day_bess['ElpcSC_day_bess'].loc[w],
Elpc_day_bess['ElpcSS_day_bess'].loc[w], multioutput='uniform_average')),1)
MAE_multioutput.append(mae)
MAPE_multioutput.append(mape)

Mape_without_ytrue_near_00=[]
for eachmape0 in MAPE_multioutput:
if eachmape0 >1: #scegliamo valori sopra 100% di errore ESCLUSO in modo tale che se un'errore è davvero 100% è
giusto che venga inserito
Mape_without_ytrue_near_00.append(0) #we put 0 % di errore
else:
Mape_without_ytrue_near_00.append(eachmape0)
# adesso vediamo qual è il vero errore massimo al di sotto del' errore tetto massimo scelto da noi 100.1%
# e lo usiamo come errore tetto massimo per calcolare il MAPE
Max_mape=max(Mape_without_ytrue_near_00)
# after checking the chart of MAPE weekly there are the weeks:15,21,28,31,32,36,51 in which MAPE arbitrary high
# because y_true=B_SOC['battery_soc'] is zero and dividing
Mape_without_ytrue_near_00=[]

```

```

#####
##### SS & SC #####
#####

if MAPE_Elpc_day_bess <= 0.5 or Mape_TOT_without_ytrue_near_0 <= 0.5:
print('MAPE IS LESS THAN 50%: both SS and SC evaluated with the same [ElpcSC_day_bess] ')

SS=pd.DataFrame(((Elpc_day['Elpc_daily'] /cons_sum_d['cons_tot_days'])*100),columns=['SS %'])
SS_bess=pd.DataFrame(((ElpcSC_day_bess['ElpcSC_day_bess']/cons_sum_d['cons_tot_days'])*100),columns=['SS bess %'])
SC=pd.DataFrame(((Elpc_day['Elpc_daily'] /pv_sum_d['PV_tot_days'])*100),columns=['SC %'])
SC_bess=pd.DataFrame(((ElpcSC_day_bess['ElpcSC_day_bess']/pv_sum_d['PV_tot_days'])*100),columns=['SC bess %'])

# creazione del nuovo DataFrame con i valori del 50%
concatenated_df = pd.concat([SS, SS_bess,SC,SC_bess], axis=0)
describe = concatenated_df.describe()
fifty_percent_df =round(describe.loc[['50%']],2)

#plot chart
fig, ax = plt.subplots()

ax.scatter(SC, SS, s=100, c='gold', edgecolor='k', alpha=0.6, label='No action')
ax.scatter(SC_bess, SS_bess, s=100, c='red', edgecolor='k', alpha=0.6, label='with BESS')
ax.set_title('Energy Matching Chart')
ax.set_xlabel('Self-consumption [%]')
ax.set_ylabel('Self-sufficiency [%]')
ax.set_ylim([0, 100])
ax.set_xlim([0, 100])
ax.legend()
x_1 = np.linspace(-10, 200, 100)
y_1 = x_1 * 1
ax.plot(x_1, y_1, 'r', label='Net Energy Zero Buildings')
ax.legend(loc='lower left')
plt.show()

else:
print('ATTENTION! MAPE IS GREATER THAN 50%: SS evaluated with [ElpcSS_day_bess] and SC with [ElpcSC_day_bess] ')

SS=pd.DataFrame(((Elpc_day['Elpc_daily'] /cons_sum_d['cons_tot_days'])*100),columns=['SS %'])
SS_bess=pd.DataFrame(((ElpcSC_day_bess['ElpcSC_day_bess']/cons_sum_d['cons_tot_days'])*100),columns=['SS bess %'])
SC=pd.DataFrame(((Elpc_day['Elpc_daily'] /pv_sum_d['PV_tot_days'])*100),columns=['SC %'])
SC_bess=pd.DataFrame(((ElpcSC_day_bess['ElpcSC_day_bess']/pv_sum_d['PV_tot_days'])*100),columns=['SC bess %'])

# creazione del nuovo DataFrame con i valori del 50%
concatenated_df = pd.concat([SS, SS_bess,SC,SC_bess], axis=0)
describe = concatenated_df.describe()
fifty_percent_df = round(describe.loc[['50%']],2)

#plot chart
fig, ax = plt.subplots()
ax.scatter(SC, SS, s=100, c='gold', edgecolor='k', alpha=0.6, label='No action')
ax.scatter(SC_bess, SS_bess, s=100, c='red', edgecolor='k', alpha=0.6, label='with BESS')
ax.set_title('Energy Matching Chart')

ax.set_xlabel('Self-consumption [%]')
ax.set_ylabel('Self-sufficiency [%]')
ax.set_ylim([0, 100])
ax.set_xlim([0, 100])
ax.legend()
x_1 = np.linspace(-10, 200, 100)
y_1 = x_1 * 1
ax.plot(x_1, y_1, 'r', label='Net Energy Zero Buildings')
ax.legend(loc='lower left')
plt.show()

return fig,fifty_percent_df
# %%

```

REFERENCES

- [1] C. L. Montenegro Cardona, "Reducing the energy mismatch between demand and supply in the Gemini south building," 2019. [Online]. Available: <http://repository.tudelft.nl/>.
- [2] G. R. Chandra Mouli, P. Bauer, and M. Zeman, "System design for a solar powered electric vehicle charging station for workplaces," *Appl Energy*, vol. 168, pp. 434–443, Apr. 2016, doi: 10.1016/j.apenergy.2016.01.110.
- [3] H. Lund, A. Marszal, and P. Heiselberg, "Zero energy buildings and mismatch compensation factors," *Energy Build*, vol. 43, no. 7, pp. 1646–1654, Jul. 2011, doi: 10.1016/j.enbuild.2011.03.006.
- [4] V. R. M. Höfte, "Energy-flat housing Towards continuous balance in the residential energy system."
- [5] I. Sartori, A. Napolitano, and K. Voss, "Net zero energy buildings: A consistent definition framework," *Energy Build*, vol. 48, pp. 220–232, May 2012, doi: 10.1016/j.enbuild.2012.01.032.
- [6] M. Ala-Juusela, T. Crosbie, and M. Hukkalainen, "Defining and operationalising the concept of an energy positive neighbourhood," *Energy Convers Manag*, vol. 125, pp. 133–140, Oct. 2016, doi: 10.1016/j.enconman.2016.05.052.
- [7] IEEE, "IEEE Recommended Practice for Sizing Lead-Acid Batteries for Photovoltaic (PV) Systems," 2000.
- [8] R. Luthander, A. M. Nilsson, J. Widén, and M. Åberg, "Graphical analysis of photovoltaic generation and load matching in buildings: A novel way of studying self-consumption and self-sufficiency," *Appl Energy*, vol. 250, pp. 748–759, Sep. 2019, doi: 10.1016/j.apenergy.2019.05.058.
- [9] A. Ciocia, A. Amato, P. Di Leo, G. Malgaroli, G. Sbriglia, and F. Spertino, "Photovoltaic-Battery Systems Design to Improve the Self-Sufficiency of Telecommunication Towers," in *2022 International Symposium on Power Electronics, Electrical Drives, Automation and Motion, SPEEDAM 2022*, 2022, pp. 383–388. doi: 10.1109/SPEEDAM53979.2022.9842208.
- [10] R. Luthander, J. Widén, D. Nilsson, and J. Palm, "Photovoltaic self-consumption in buildings: A review," *Applied Energy*, vol. 142. Elsevier Ltd, pp. 80–94, Mar. 05, 2015. doi: 10.1016/j.apenergy.2014.12.028.
- [11] "0," Manila, Philippines, Dec. 2018. doi: 10.22617/TCS189791-2.
- [12] T. Kaschub, P. Jochem, and W. Fichtner, "Solar energy storage in German households: profitability, load changes and flexibility," *Energy Policy*, vol. 98, pp. 520–532, Nov. 2016, doi: 10.1016/j.enpol.2016.09.017.
- [13] KU Leuven, "D4.15.1. KU Leuven Living Lab Monitoring Report D4.15.1. Report on the KU Leuven Living Lab."
- [14] ECO, "ECO ESS product catalogue 2022."
- [15] FHaas, "building_stock", Accessed: Mar. 13, 2023. [Online]. Available: <https://gitlab.com/hotmaps/building-stock/-/tree/master/data>
- [16] R. Monografico, "Benchmark di consumo energetico degli edifici per uffici in Italia," 2019.
- [17] S. Birchall BSRIA Ian Wallis, B. David Churcher, B. Simon Pezzutto, and E. Roberto Fedrizzi, "Project Title: Development of Systemic Packages for Deep Energy Renovation of Residential and Tertiary Buildings including Envelope and Systems Project Acronym: iNSPiRe Deliverable Title: D2.1a Survey

on the energy needs and architectural features of the EU building stock Dissemination Level: PU Lead beneficiary: BSRIA,” 2014. [Online]. Available: www.inspirefp7.eu

- [18] H. Kazmi, I. Munné-Collado, F. Mehmood, T. Abbas Syed, and J. Driesen, “Towards data-driven energy communities: a review of open-source datasets, models and tools.” [Online]. Available: <https://www.tvindkraft.dk/en/>
- [19] D. A. Broden, K. Paridari, and L. Nordstrom, “Matlab applications to generate synthetic electricity load profiles of office buildings and detached houses,” in *2017 IEEE Innovative Smart Grid Technologies - Asia: Smart Grid for Smart Community, ISGT-Asia 2017*, Jun. 2018, pp. 1–6. doi: 10.1109/ISGT-Asia.2017.8378371.
- [20] A. P. Dobos, “PVWatts Version 5 Manual,” 2014. [Online]. Available: <http://developer.nrel.gov>
- [21] World Weather Online, “Ghent Annual Weather Averages.” <https://www.worldweatheronline.com/ghent-weather-averages/be.aspx> (accessed Mar. 14, 2023).
- [22] SANDIA PVPMC, “Partial List of References that cite PVLIB.” https://pvpmc.sandia.gov/applications/pv_lib-toolbox/ (accessed Mar. 14, 2023).
- [23] C. W. H. and M. A. M. William F. Holmgren, “ ‘pvlib python: a python package for modeling solar energy systems.’ *Journal of Open Source Software*, 3(29), 884, (2018). <https://doi.org/10.21105/joss.00884>.”
- [24] SANDIA PVPMC, “PV Performance Modeling Collaborative or PVPMC, <https://pvpmc.sandia.gov/>.”.
- [25] PyData 2021, “Tutorial 3-Modeling an Array Power Contents.” [Online]. Available: <https://raw.githubusercontent.com/PVSC-Python-Tutorials/pyData->
- [26] Pedregosa et al., “Scikit-learn: Machine Learning in Python, *JMLR* 12, pp. 2825-2830, 2011.”.
- [27] VREG, “energy market dashboards, <https://www.vreg.be/nl/energiemarkt-cijfers>.”
- [28] NREL, “Lifetime of PV Panels”, Accessed: Mar. 14, 2023. [Online]. Available: <https://www.nrel.gov/state-local-tribal/blog/posts/stat-faqs-part2-lifetime-of-pv-panels.html>
- [29] I. Renewable Energy Agency, *Renewable power generation costs in 2021*. 2022. [Online]. Available: www.irena.org
- [30] I. trusted solar panel experts Energyd, “Solar Net Present Value (NPV) Online Calculator.” <http://energyd.ie/solar-npv-calculator/> (accessed Mar. 14, 2023).
- [31] G. U. Thornton LLP, “Renewable energy discount rate survey results-2018,” 2019. [Online]. Available: <https://www.grantthornton.co.uk/insights/renewable-energy-discount-rate-survey/>
- [32] Pauli Virtanen et al., “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17(3), 261-272.”.
- [33] J. J. Montañó Moreno, A. Palmer Pol, A. Sesé Abad, and B. Cajal Blasco, “El índice R-MAPE como medida resistente del ajuste en la previsión,” *Psicothema*, vol. 25, no. 4, pp. 500–506, 2013, doi: 10.7334/psicothema2013.23.