UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA CIVILE, EDILE E AMBIENTALE

Department Of Civil, Environmental and Architectural Engineering

Master's Degree in Mathematical Engineering

TESI DI LAUREA

A Stochastic Model for Optimal Investment in Renewable Energy Communities

Relatore:
Chiar.mo PROF. Tiziano Vargiolu

Laureando: Lorenzo Portaluri
2058087

ANNO ACCADEMICO 2022-2023

The political problem of mankind is to combine three things:
Economic Efficiency, Social Justice, and Individual Liberty

John Maynard Keynes

Speech delivered at the Manchester Reform Club,
February 9, 1926;
First published in The Nation and Athenæum,
February 20, 1926

Contents

1 Abstract **7**

2 Motivation **9**

3 Introduction **13**
3.1 The environmental issue 13
3.2 Renewable Energy Communities (RECs) 18
3.3 The growing interest in REC and the linked challenges 19
3.4 Framework 21
3.5 General setting 22

4 The mathematical model **23**
4.1 The leader-follower model 23
4.2 Geometric Brownian Motion 27
4.2.1 Introduction 27
4.2.2 Positive and negative aspects 30
4.3 Ornstein-Uhlenbeck Process 31
4.3.1 Introduction 31
4.3.2 Positive and negative aspects 33
4.4 Model choices 34

5 Processes and Model implementation **37**
5.1 Why `Python` 37
5.2 Stochastic Processes 38
5.2.1 Geometric Brownian Motion in `Python` 38
5.2.2 Ornstein-Uhlenbeck process in `Python` 40
5.3 Numerical Integration 44
5.3.1 `scipy.integrate.quad` 44
5.4 Optimization 45

5.4.1 `'SLSQP'` 47
5.5 Code 48
5.5.1 Geometric Brownian Motion 49
5.5.2 Ornstein-Uhlenbeck 50
5.5.3 Integration 53
5.5.4 Optimization 56
5.6 Output 56

6 Further research 61
6.1 Lévy Process 61
6.1.1 Introduction 61
6.1.2 Positive and negative aspects 63
6.2 Lévy process implementation 65
6.2.1 Stochastic process 65
6.2.2 Lévy processes in `Python` 66
6.2.3 Code 68
6.2.4 Output 70
6.3 Seasonality 70
6.3.1 Fourier series 71
6.3.2 Fourier series implementation 73
6.3.3 Code 74
6.3.4 Output 76
6.3.5 Time trend functions 78

7 Results and Conclusion 81

Bibliography 85

Chapter 1

Abstract

The attention on Renewable Energy Communities (REC) is fastly growing after the European Union (EU) has introduced a dedicated regulation in 2018. RECs can be composed by citizens, small- and medium-sized companies, and local administrations with the purpose of self-production and self-consumption of energy from renewable sources.

This thesis presents a stochastic model for optimizing investment in Renewable Energy Communities (RECs), taking as a starting point the paper [1]. The model focuses on a particular type of REC composed of a ”representative” household and a biogas producer, where the potential demand of the community is given by the household's demand, while both members produce renewable energy. The biogas producer invests in technology to convert biogas into electricity and sell it in the electricity market at the spot price, whereas the biogas that is not transformed into energy can be sold on the market at the gas spot price. The household invests in photovoltaic panels to reduce the energy purchased from the market in order to cover its own power demand. Moreover, investing in a renewable energy plant provides the household with the revenues of selling the excess of energy not used for self-consumption. The relevant advantage of entering into a REC for both players is that their joint self-consumption is rewarded with a governmental incentive, which must be fairly shared. The challenge of optimal investment with fair share of incentives is addressed by setting the problem as a leader-follower game, where the leader decides how to

share the incentive. The model provides insights into how RECs can effectively balance investment in renewable energy technologies with fair distribution of incentives, promoting sustainable energy production and consumption.

Chapter 2

Motivation

My Master's thesis will focus on developing a stochastic model for optimal investment in Renewable Energy Communities. This is a topic that I am deeply passionate about, for several reasons.

First and foremost, I am deeply concerned about the environmental issues facing our planet. Global warming and climate change are serious threats to our environment, and the need to reduce carbon emissions and develop sustainable energy sources has never been more pressing.

Renewable energy is a promising solution that can help address these challenges, and renewable energy communities have emerged as an innovative solution to promote the use of renewable energy sources and achieve sustainable development.

Secondly, I have a strong academic background in applied mathematics and a passion for using my skills and knowledge to make a positive contribution to society. I believe that developing a stochastic model for optimal investment in Renewable Energy Communities provides an excellent opportunity to combine my interests in mathematics and environmental sustainability.

Furthermore, I believe that this thesis provides an opportunity to make a meaningful contribution to a developing field. Renewable energy investment is a complex and challenging area, with several uncertainties and risks that must be taken into account when making investment decisions.

By developing a stochastic model for optimal investment, I hope to provide a valuable tool for investors to make informed investment decisions and guide them towards profitable and sustainable investments in Renewable Energy Communities.

Another crucial element that stood out to me was the great impact of reading Fred Espen Benth, Jurate Saltyte Benth, and Steen Koekebakker's work [4] on stochastic modelling of electricity and related markets.

Their research is a milestone in the field of electricity markets and stochastic modelling, providing a comprehensive and in-depth analysis of the complex dynamics that shape these markets. Their work sheds light on the intricate interplay of various market participants, including producers, consumers, regulators, and investors, and how their actions and decisions affect the electricity market.

Their research highlights the importance of stochastic modelling in understanding the volatility and uncertainty inherent in electricity markets. By developing sophisticated mathematical models that account for random fluctuations and hardly foreseeable events, their work demonstrates the potential for improved decision-making in these markets.

As I delved deeper into their research, I was struck by their approach to stochastic modelling, drawing on insights from mathematics, economics, and physics to develop a more comprehensive understanding of electricity related markets. Their work has inspired me to explore the potential of stochastic modelling in developing more robust and sustainable investment strategies for Renewable Energy Communities.

Overall, my Master's thesis represents a convergence of my interests in applied mathematics and environmental sustainability, with the significant contribution of my reading of [4].

Through this research, I hope to make a real contribution to the field of renewable energy investment and help advance sustainable solutions for our planet's energy needs.

Chapter 3

Introduction

3.1 The environmental issue

Environmental issues are a complex and multifaceted problem that affects every aspect of our lives. These problems are mostly caused by human activities, such as the burning of fossil fuels, deforestation, and the use of toxic chemicals, and they have a deep impact on the health of our planet and its inhabitants [5].

One of the most pressing environmental issues is climate change. Climate change refers to the long-term changes in temperature, precipitation, and other weather patterns that occur as a result of the increased levels of greenhouse gases in the atmosphere [20]. Greenhouse gases, such as carbon dioxide and methane, trap heat from the sun, causing the Earth's temperature to rise [11]. This increase in temperature is leading to a wide range of consequences, including melting polar ice caps, rising sea levels, and more frequent and severe weather events, such as hurricanes, droughts and floods. These consequences are already having a significant impact on the environment and human society and, if left unaddressed, they will only get worse in the future.

Another major environmental issue is the loss of biodiversity. Biodiversity refers to the variety of life on Earth, including the variety of species, ecosystems, and genetic diversity. The loss of biodiversity is a result of human activities, such as deforestation, overfishing and the use of toxic chemicals, which are destroying habitats and threatening the survival of many species. This loss of biodiversity has a number of consequences, including the disruption of food chains, the loss of important ecosystem services, and the extinction of species that are critical to the health of the planet. In addition, the loss of biodiversity makes our planet more vulnerable to other environmental problems, such as climate change, and disease outbreaks.

Air and water pollution are also major environmental issues. Air pollution is caused by the emission of harmful chemicals and particulates into the atmosphere, while water pollution is caused by the release of toxic chemicals and waste into rivers, lakes, and oceans. Both forms of pollution have serious impacts on human health and the environment. For example, air pollution is linked to respiratory problems, heart disease, and cancer, while water pollution can harm wildlife, disrupt food chains, and make water unsafe for human consumption.

Deforestation is another significant environmental issue. Deforestation is the destruction of forests and woodland areas, usually for the purpose of agriculture or urban development. This destruction of forests has a number of negative impacts, including the loss of habitats for wildlife, the release of carbon dioxide into the atmosphere, and the destruction of important ecosystems, such as tropical rainforests. Deforestation also contributes to soil degradation, which can lead to decreased crop yields, increased food insecurity, and other environmental problems.

Soil degradation is another environmental issue that has a significant impact on the health of our planet. Soil degradation is caused by factors such as overuse, deforestation, and the use of toxic chemicals, and it results in reduced soil fertility, increased erosion, and decreased water-holding capacity. This degradation of soil can lead to decreased crop yields, increased food insecurity, and other environmental problems. In addition, soil degradation contributes to desertification, which is the process by which fertile land becomes a desert, and is a major threat to food security in many regions of the world.

In conclusion, environmental issues are a complex and pressing problem that requires immediate and sustained action from individuals, communities and governments around the world. By addressing these issues, we can help to ensure a healthier, more sustainable future for ourselves and future generations. This requires a global effort, including the reduction of greenhouse gas emissions, the preservation of biodiversity, and the adoption of sustainable practices in our daily lives. Additionally, it requires investment in new technologies.

One way of facing some of the above challenges and issues is trying to develop a new way of producing the extremely significant quantity of energy which is required in almost every human activity.

The shift towards sustainable energy production has the potential to significantly mitigate environmental problems. Here are a few ways this transition can have a positive impact.

Reduction of greenhouse gas emissions: the most significant benefit of sustainable energy production is the reduction of greenhouse gas emissions [8]. Fossil fuels such as coal, oil, and natural gas are the primary sources of greenhouse gas emissions and are responsible for the majority of the global warming. On the other hand, renewable energy sources such as wind, solar, geothermal, and hydropower emit minimal or no greenhouse gases, thus reducing their impact on the environment.

Decreased air pollution: fossil fuels also contribute to air pollution, which can have significant negative impacts on human health and the environment. The shift to sustainable energy production can help reduce air pollution, leading to healthier air quality and improved public health.

Protection of natural habitats: the production and extraction of fossil fuels often involve the destruction of sensitive natural habitats such as wetlands, forests, and wildlife habitats. Renewable energy sources can be produced in ways that have minimal impact on the environment, preserving these habitats for future generations.

Conservation of water resources: water is a scarce resource and many conventional energy production methods require significant amounts of it. Sustainable energy production methods, such as solar and wind power, use much less water and help to conserve this vital resource.

Economic benefits: sustainable energy production not only helps to address environmental problems, but it also has economic benefits. The growth of renewable energy industries is creating jobs, spurring economic development, and contributing to energy security.

Renewable energy sources, however, can sometimes have also a controversial impact on the environment. So, before diving into the analysis it necessary to ponder also these points.

The installation of solar or photovoltaic panels, for example, can require the clearing of large areas of land, which can have an impact on biodiversity and ecosystem function. This is particularly true in areas with high conservation value, such as forests, wetlands, or grasslands. In addition, the construction of access roads, transmission lines and other infrastructure necessary for the installation of solar or photovoltaic panels can also contribute to deforestation and habitat fragmentation.

To address these concerns, it's important to carefully consider the location of solar installations and to prioritize the use of previously disturbed lands, such as brownfields or abandoned industrial sites, for solar development. Additionally, the implementation of wildlife-friendly designs and management practices, such as the use of native plant species or reduced mowing, can help to mitigate the impacts of solar installations on biodiversity.

Overall, while solar energy is a clean and renewable energy source, it is important to carefully consider its potential impacts on the environment, including deforestation and loss of biodiversity. By implementing sustainable practices and prioritizing the use of previously disturbed lands, we can ensure that solar energy contributes to a more sustainable and resilient energy system.

Another controversial example is the use of wind produced energy. One of the main concerns is its impact on wildlife, particularly birds and bats, which can collide with the turbines or suffer habitat disruption. The visual impact of wind turbines is also a concern for some communities, who argue that the installation of turbines can spoil the beauty of natural landscapes and affect tourism and property values. Additionally, noise pollution generated by wind turbines during their operation can be audible up to several kilometers away, potentially causing sleep disturbance and other health effects for nearby residents.

Furthermore, the construction of wind turbines and their associated infrastructure can lead to land use changes and the fragmentation of wildlife habitats, which can have implications for biodiversity and long-term sustainability. Lastly, the intermittent nature of wind energy production requires energy storage and transmission systems, which can also have environmental impacts, such as habitat destruction and land use changes.

Summing up, while wind energy has the potential to provide a sustainable alternative to fossil fuels, it is important to consider and address its potential environmental impacts in order to promote its responsible development. By balancing the benefits of wind energy with its potential negative impacts, we can work towards a cleaner and more sustainable energy future.

In conclusion, a sustainable way of producing energy has a number of positive effects on environmental problems, including reducing greenhouse gas emissions, decreasing air pollution, protecting natural habitats, conserving water resources, and providing economic benefits. We must note, however, that there are some critical aspects which must be carefully managed.

One way of producing energy in a sustainable way, which also takes into account the most controversial aspects, is the development of Renewable Energy Communities. These communities, that we are about to study, are usually brought to life in a local and small scale setting. This allows to avoid the most negative consequences which can originate from the exploitation of some renewable energy sources, by having all the necessary attention as seen above.

3.2 Renewable Energy Communities (RECs)

Renewable Energy Communities (RECs) [6] are a recent concept in the world of sustainable energy and environmentalism. A REC is a group of individuals, organizations or companies that come together with a shared goal of transitioning to a more sustainable and environmentally friendly energy system. The main objective of a REC is to generate energy through renewable sources, such as solar, wind, and hydro power, and to distribute it among its members, while reducing dependence on traditional non-renewable sources of energy.

One of the key advantages of a REC is the ability for individuals to generate their own energy, without having to rely on the traditional energy grid. This is achieved through the installation of renewable energy systems, such as photovoltaic panels, wind turbines, and hydro-generators, which are owned and operated by the members of the community. The generated energy is then distributed among the members, and any excess energy can be sold back to the traditional energy grid.

Another important aspect of a REC is the creation of a more sustainable energy system. Renewable energy sources are inherently more sustainable and environmentally friendly than traditional sources, such as coal, oil, and gas. They emit significantly less greenhouse gases, which are the primary cause of global warming, and they are not depleted, unlike non-renewable sources, which are finite and will eventually run out.

In addition to the environmental benefits, RECs also offer economic benefits to their members. By generating their own energy, members can significantly reduce their energy bills, while also creating new job opportunities in the areas of renewable energy installation, maintenance and management.

To create a REC, a group of individuals, organizations or companies must first come together and decide on the specific goals and objectives for the community. They must then work together to identify potential locations for the installation of renewable energy systems and determine the best type of renewable energy systems for their particular community. The next step is to secure financing for the project, either through loans, grants, or investment from members or outside sources.

Once the renewable energy systems are installed, the community must establish a governance structure to manage the community and ensure the fair distribution of energy among its members. This includes the creation of a governing board, the establishment of policies, and procedures for the management of the community and the development of a billing and metering system to accurately measure energy usage and production.

In conclusion, Renewable Energy Communities are an innovative and effective way for individuals, organizations and companies to transition to a more sustainable and environmentally friendly energy system. By generating their own energy and reducing their dependence on the traditional energy grid, RECs provide a multitude of benefits, including reduced energy costs, new job opportunities, and a more sustainable energy system. With the growing global demand for renewable energy, it is likely that the concept of Renewable Energy Communities will continue to grow and evolve in the years to come.

3.3 The growing interest in REC and the linked challenges

The interest in Renewable Energy Communities (REC) has rapidly increased since the European Union introduced a specific regulation in 2018 [7]. These communities can consist of citizens, small to medium-sized businesses, and local authorities with the goal of producing and consuming energy from renewable sources. The

Italian government has also recently implemented an incentive tariff for RECs, with a special feature that rewards self-consumption. This refers to the demand of the community being satisfied by self-generated energy.

However, there are two key challenges that need to be addressed. Firstly, the optimal investment in new technologies is crucial and secondly, a fair division of the incentive among community members is essential. These two challenges are interdependent, as an unfair division of the incentives could negatively impact the profitability and discourage members from joining a REC.

To address these challenges, we consider a particular type of REC composed of a "representative" household and a biogas producer. The household's demand serves as the potential demand of the community, while both members generate renewable energy. This type of REC is common in both rural and urban areas.

In this scenario, the biogas producer invests in new technology to convert biogas into electricity for sale in the electricity market at the spot price, while the household invests in photovoltaic panels to reduce their dependence on the energy market. Both parties benefit from the joint self-consumption that is rewarded by a governmental incentive, which must be shared fairly.

We set this problem as a leader-follower problem, with the leader determining how to share the incentive and the followers determining their own optimal investment strategy. We can assume that the leader is an authority, such as a local administration or the block of flats administrator (this is, for example, common in Italy) and the members of the community are, instead, followers. In our analysis we will set the optimization problem to avoid the issue of share division of incentives focusing on optimal investment.

3.4 Framework

Climate change is a global problem that requires the attention of all segments of society to find solutions. One of the solutions that has the potential to reduce CO_2 emissions is for consumers to become self-producers and self-consumers of electricity generated from Renewable Energy Sources (RES).

In line with this, the European Union (EU) has introduced two recent directives to allow private households, small businesses and local public authorities to join local Renewable Energy Communities and share the generation of their renewable energy plants. This can lead to a more efficient national energy system and reduce grid congestion [6].

However, the implementation of these EU directives across member states is happening at different speeds. One of the first examples is Italy, which has introduced a sophisticated "virtual" consumption framework. Under this framework, the members of a REC do not directly consume the energy generated by their plants, but instead, all the energy must be offered to the market.

The central authority acts as a natural broker to guarantee a REC a selling price calculated as an average of the gross price of electricity. To satisfy their energy needs, the members must continue to buy energy from their retailers. However, they are compensated through the sale of energy generated by the REC's power plants and receive an incentive for the energy they virtually self-consumed.

The implementation of this incentive framework poses a non-trivial problem for the potential members of a REC to find a fair sharing rule for the incentive. This becomes even more complicated with larger groups and different consumption profiles. The success of a REC depends on many factors, including the ability of its members to find a fair sharing rule for the self-consumption incentive.

3.5 General setting

The EU regulation defines an energy community as a group of members who are both producers and consumers of energy, referred to as "prosumers". According to Italian law, there are two types of energy communities that are eligible for public incentives: Renewables Energy Communities (RECs) and Collective Self-consumption Groups (CSGs).

A REC is a legal entity composed of diverse members, including individual households, small- and medium-sized enterprises, and local or regional authorities, that aims to provide members with self-generated and self-consumed renewable energy. On the other hand, a CSG is a group of at least two individuals located in the same building or block of flats who collectively consume renewable energy. Both types of RECs can be combined with other entities, both private and public, that produce energy from renewable sources.

As per the Italian law, members of a REC do not own the energy they generate. They pay for all the energy they use, including the self-generated portion, at the rate fixed by their retailer. However, they will receive an incentive tariff from the Gestore dei Servizi Energetici (GSE) for their share of self-generated energy and can also sell any unused energy to the market. Members are free to leave the REC at any time.

In a typical scenario, a biogas producer and a household can form a REC, which can be seen in both rural and urban areas where a biogas plant is established to reduce the impact of organic waste or animal effluents.

Chapter 4

The mathematical model

4.1 The leader-follower model

In this chapter, we describe the mathematical model and problem formulation of a simplified version of a Renewable Energy Community.

Consider a complete filtered probability space $(\Omega, \mathcal{F}, \mathcal{F}_t, \mathbb{P})$, where a four dimensional Brownian motion (W_c, W_v, W_p, W) is defined, satisfying the usual conditions.

$(W_t)_{t\geq 0}$ is a one-dimensional Brownian motion correlated with W_c with correlation factor $\rho_c := \mathrm{Corr}(W, W_c)$. In fact, the four Brownian motions W, W_c, W_v, and W_p can be correlated, but only the correlation ρ_c might be relevant.

The following is the starting point for this thesis, which aims to re-implement the leader-follower model with more realistic stochastic processes. As mentioned above the starting point is taken from [1].

We consider two possible members of an energy community: a biogas producer and a household. On one hand, the biogas producer has a total gas production capacity K_g, which is sold in the market at the gas spot price $(P^p(t))_{t\geq 0}$, which evolves according to a Geometric Brownian Motion:

$$P^p(s) = pe^{\mu_p s + \sigma_p W_p(s)} \tag{4.1}$$

where $\mu_p \in \mathbb{R}$ and $\sigma_p > 0$. The biogas producer has the option to install new technologies, up to a maximum power of θ_b, to transform the gas into power and sell it in the electricity market at the spot sale electricity price $(X^{x_v}(t))_{t\geq 0}$, which is assumed to evolve according to a Geometric Brownian Motion:

$$X_v^{x_v}(s) = x_v e^{\mu_v s + \sigma_v W_v(s)} \tag{4.2}$$

where $\mu_v \in \mathbb{R}$ and $\sigma_v > 0$.

On the other hand, the household buys energy in the electricity market to meet its demand at the purchase electricity price $(X_c^{x_c}(t))_{t\geq 0}$, which also follows a Geometric Brownian Motion:

$$X_c^{x_c}(s) = x_c e^{\mu_c s + \sigma_c W_c(s)} \tag{4.3}$$

where $\mu_c \in \mathbb{R}$ and $\sigma_c > 0$. The household can install new photovoltaic panels, up to a maximum power of θ_h, to produce power and reduce its costs.

We consider an energy community that can be formed by these two members. The demand of the energy community is given by the household power demand $(D^d(t))_{t\geq 0}$, which follows a Geometric Brownian Motion:

$$D^d(s) = de^{\mu_d s + \sigma_d W(s)} \tag{4.4}$$

where $\mu_d \in \mathbb{R}$ and $\sigma_d > 0$.

Both members contribute to the total power produced by the community. The household provides power y_h by installing photovoltaic panels, while the biogas producer contributes power y_b by installing turbines to transform the gas into energy. Hence, the energy shared by the community can be expressed as

$$\min\{D^d(t), y_h + y_b\} \tag{4.5}$$

As previously stated, the government offers an incentive, denoted by Z, to groups that form an energy community. The incentive rewards the energy shared by the community that is produced by new installations of renewable energy sources, such as photovoltaic panels or turbines to transform biogas into power. The entire community is given the incentive and it is up to a coordinator to determine how to share it. We will assume that this coordinator will share the incentive using the Nash bargaining solution of the sharing problem, that is, the coordinator will search for a $\beta \in (0,1)$ that satisfies:

$$\beta \in \arg\max_{\beta \in (0,1)} \left(J_h(\beta) - d_h\right)\left(J_b(\beta) - d_b\right) \tag{4.6}$$

where $J_h(\beta)$ and $J_b(\beta)$ are the profits of the household and the biogas producer, respectively, if they enter into the community, while d_h and d_b are known as the disagreement points and correspond to the profits of the household and the biogas producer if they do not agree to enter into the energy community, that is, d_h and d_b are their profits without the incentive. The parameter β is the part of the incentive that the household receives, while $1-\beta$ is the part that the biogas producer receives.

Let us now define the profits J_h and J_b for the household and the biogas producer, assuming that the household receives a part $\beta \in (0,1)$ of the incentive, while the biogas producer receives $1-\beta$.

The biogas profit functional is written as:

$$\begin{aligned} J_b(x_v, x_c, p, d, y_b, y_h, \beta) &:= \mathbb{E}\left[\int_0^\infty e^{-rs} X_v^{x_v}(s) y_b ds\right] \\ &+ \mathbb{E}\left[\int_0^\infty e^{-rs}(1-\beta) Z \min\{D^d(s), y_b + y_h\} ds\right] \\ &+ \mathbb{E}\left[\int_0^\infty e^{-rs} P^p(s)(bK_g - y_b) ds\right] - c_b y_b \end{aligned} \tag{4.7}$$

where c_b is the installation cost of a turbine that produces 1 MW, $b > 0$ is a conversion factor expressing the cubic meters of gas needed to produce 1 MW of power, and $r > 0$ is a discount factor.

The profit of the household is written as:

$$\begin{aligned} J_h(x_v, x_c, p, d, y_b, y_h, \beta) &:= \mathbb{E}\left[\int_0^\infty e^{-rs} X_v^{x_v}(s) y_h ds\right] \\ &- \mathbb{E}\left[\int_0^\infty e^{-rs} X_c^{x_c}(s) D^d(s) ds\right] \\ &+ \mathbb{E}\left[\int_0^\infty e^{-rs} Z\beta \min\{D^d(s), y_h + y_b\} ds\right] - c_h y_h \end{aligned} \tag{4.8}$$

where c_h is the installation cost of photovoltaic panels to produce 1 MW.

In the above discussion the main process are modeled as Geometric Brownian Motions in equations (4.1),(4.2),(4.3),(4.4). A possibility to improve the realism of the model is to switch to Ornstein-Uhlenbeck processes. So, before going further, let us analyse the features of the two kinds of stochastic processes just mentioned.

4.2 Geometric Brownian Motion

4.2.1 Introduction

The Geometric Brownian Motion (GBM) [19] is a stochastic process used in finance to describe the evolution of asset prices over time. It is based on the assumption that the relative change in the price of an asset follows a random walk, where each change is proportional to the current price and a log-normal random variable. In mathematical terms, GBM can be expressed as:

$$dS_t = \mu S_t dt + \sigma S_t dW_t \tag{4.9}$$

where S_t is the price of the asset at time t, μ is the drift, which represents the expected rate of return of the asset, σ is the volatility, which describes the dispersion of the returns, and W is a Wiener process.

We can also write the exponential form of the GBM. In order to do that we need to apply Itô's lemma, which relates the differential of a function of a stochastic process to the partial derivatives of the function with respect to the process and its time.

Consider the function $f(S_t, t) = f(S_t) = \ln(S_t)$, where ln denotes the natural logarithm. Applying Itô's lemma, we get:

$$df(S_t) = \frac{\partial f(S_t)}{\partial x} dS_t + \frac{1}{2}\frac{\partial^2 f(S_t)}{\partial x^2}(dS_t)^2 \tag{4.10}$$

Since $S_t = e^{f(S_t)}$, we can rewrite dS_t in terms of $df(S_t)$ as:

$$dS_t = e^{f(S_t)} df(S_t) = S_t df(S_t) \tag{4.11}$$

Substituting this into the original GBM equation, we get:

$$d(\ln S_t) = (\mu - \frac{1}{2}\sigma^2)dt + \sigma dW_t \tag{4.12}$$

Integrating both sides from 0 to t, we get:

$$\ln \frac{S_t}{S_0} = (\mu - \frac{1}{2}\sigma^2)t + \sigma W_t \tag{4.13}$$

where S_0 is the initial value of the process and W is a standard Brownian motion.

Exponentiating both sides, we get the exponential form of GBM:

$$S_t = S_0 e^{(\mu - \frac{1}{2}\sigma^2)t + \sigma W_t} \tag{4.14}$$

This equation shows that the price of the asset follows a log-normal distribution with a drift term of μ and a volatility term of σ. The drift term represents the expected rate of return of the asset, while the volatility term describes the uncertainty or risk associated with the returns.

The mean and variance of a Geometric Brownian Motion at time t are given by [21]:

$$\mathbb{E}[S_t] = S_0 e^{\mu t} \tag{4.15}$$

$$\text{Var}[S_t] = S_0^2 e^{2\mu t} \left(e^{\sigma^2 t} - 1 \right), \tag{4.16}$$

where S_0 is the initial value of the process.

GBM is widely used in finance to model the behavior of stock prices, commodities, and other financial assets. The model is attractive because of its simplicity and ability to capture the essential features of asset price movements.

One important application of GBM is the Black-Scholes option pricing model. The Black-Scholes model uses GBM to describe the evolution of the underlying asset price, and it provides a method to price European options. European options are options that can only be exercised at the expiration date, and they are the most common type of options traded in financial markets. The Black-Scholes formula is a well-known mathematical formula that provides the price of a European option based on the underlying asset price, the option strike price, the time to expiration, the risk-free interest rate, and the volatility of the underlying asset.

GBM can also be used in portfolio optimization, where the goal is to determine the optimal portfolio weights that maximize the expected return for a given level of risk. In this context, GBM is used to model the expected returns of the assets in the portfolio and the volatility is used as a measure of risk.

Another application of GBM is in risk management, where the model is used to estimate the Value at Risk (VaR) of a portfolio. VaR is a measure of the potential loss of a portfolio over a specified time horizon and confidence level. By using GBM to model the evolution of asset prices, it is possible to estimate the VaR of a portfolio and take appropriate risk management measures.

So, Geometric Brownian Motion is a widely used mathematical model in finance, and it provides a simple and intuitive way to describe the evolution of asset prices over time. Despite its limitations, GBM has proven to be a useful tool for option pricing, portfolio optimization, and risk management.

4.2.2 Positive and negative aspects

Geometric Brownian Motion (GBM) is a widely used mathematical model in finance that describes the evolution of asset prices over time. While GBM has several advantages, it also has some limitations that should be considered when using it. In this subsection, we will discuss in detail the pros and cons of GBM.

Pros of GBM:

- Simplicity: one of the main advantages of GBM is its simplicity. The model assumes that asset prices follow a random walk and can be described by a simple mathematical equation, making it easy to understand and implement.
- Ability to capture essential features of asset price movements: GBM is able to capture the essential features of asset price movements, including the expected rate of return, volatility, and the random nature of asset price changes.
- Widely used in finance: GBM is widely used in finance and has been applied to a variety of financial problems, including option pricing, portfolio optimization, and risk management. This widespread use makes it easier to find information and resources related to GBM.
- Basis for Black-Scholes model: GBM is the basis for the Black-Scholes option pricing model, which is one of the most widely used models in finance for pricing European options.

Cons of GBM:

- Limited realism: one of the main limitations of GBM is its limited realism. The model assumes that asset price changes are normally distributed and follow a random walk, which may be not always the case in the real world.
- Assumes constant volatility: GBM assumes that the volatility of asset prices is constant, which may not be true in practice. Volatility can change over time and can have different levels for different assets.

- Not suitable for modeling extreme events: GBM is not well suited for modeling extreme events, such as crashes or sudden changes in market conditions, as the model assumes that asset prices follow a continuous distribution.
- Does not account for market frictions: GBM does not account for various market frictions, such as transaction costs, taxes and market impact, which can have a significant impact on asset prices.

In conclusion, Geometric Brownian Motion is a widely used mathematical model in finance that has several advantages, including its simplicity, ability to capture the essential features of asset price movements and widespread use in finance.

However, it also has several limitations, including its limited realism, assumption of constant volatility, lack of consideration for market frictions, and unsuitability for modeling extreme events. When using GBM, it is important to understand both its advantages and limitations and to consider the specific context in which it will be applied.

4.3 Ornstein-Uhlenbeck Process

4.3.1 Introduction

The Ornstein-Uhlenbeck (OU) [4] process is a mathematical model used in finance and economics to describe the evolution of a mean-reverting time series. The OU process is a type of stochastic process that is characterized by a mean-reverting behavior, where the values of the time series tend to move towards a long-term average value over time.

The mathematical representation of the OU process can be written as:

$$dX_t = -\lambda(X_t - \mu)dt + \sigma dW_t \tag{4.17}$$

where X_t is the value of the time series at time t, μ is the long-term average value, λ is the rate of mean reversion, σ is the volatility, and W is a Wiener process.

The mean of an Ornstein-Uhlenbeck process is given by [21]:

$$\mathbb{E}[X_t] = e^{-\lambda t}X_0 + \mu(1 - e^{-\lambda t}) \tag{4.18}$$

where X_0 is the initial value of the process.

The covariance of an Ornstein-Uhlenbeck process $Cov(X_s, X_t)$ is given by [21]:

$$Cov(X_s, X_t) = \frac{\sigma^2}{2\lambda}(e^{-\lambda|t-s|} - e^{-\lambda(t+s)}) \tag{4.19}$$

The OU process is commonly used in finance to model the evolution of interest rates, foreign exchange rates and other financial variables that exhibit mean-reverting behavior. For example, in the case of interest rates, the OU process can be used to model the movements of short-term interest rates around a long-term average value.

One of the main advantages of the OU process is its ability to capture the mean-reverting behavior of time series, which is often observed in financial markets. The OU process also provides a simple and intuitive way to model the volatility and the rate of mean reversion, which are important parameters in many financial applications.

However, the OU process assumes that the mean-reverting behavior is constant over time, which may be not always the case in practice.

In conclusion, the Ornstein-Uhlenbeck process is a useful mathematical model used in finance and economics to describe the evolution of mean-reverting time series. The model provides a simple and intuitive way to model the mean-reverting behavior, volatility, and rate of mean reversion of time series.

4.3.2 Positive and negative aspects

The Ornstein-Uhlenbeck (OU) process is a widely used mathematical model in finance and economics for modeling mean-reverting time series. The OU process has several advantages and limitations, which should be considered when using it. In this subsection, we will discuss in further detail the pros and cons of the OU process.

Pros of the OU process:

- Ability to capture mean-reverting behavior: one of the main advantages of the OU process is its ability to capture the mean-reverting behavior of time series, which is often observed in financial markets.
- Simple and intuitive modeling: the OU process provides a simple and intuitive way to model the mean-reverting behavior, volatility, and rate of mean reversion of time series.
- Widely used in finance and economics: the OU process is widely used in finance and economics, making it easier to find information and resources related to its use and implementation.

Cons of the OU process:

- Limited realism: the OU process may not perfectly capture the real-world behavior of mean-reverting time series. In addition, the model assumes that the mean-reverting behavior is constant over time, which may not always be the case in practice.

- Limited applicability: the OU process is only applicable to mean-reverting time series and may not be suitable for modeling other types of time series.
- May require a more demanding estimation of parameters: the OU process requires the estimation of more parameters than the GBM, such as the long-term average value, rate of mean reversion, and volatility, which may be challenging in some cases.
- Like GBM, it is not suitable for modeling extreme events: the OU is not well suited for modeling extreme events, such as crashes or sudden changes in market conditions, as the model assumes that asset prices follow continuous simple paths.
- Like GBM, it does not account for market frictions: the OU does not account for various market frictions, such as transaction costs, taxes, and market impact, which can have a significant impact on asset prices.

In conclusion, the Ornstein-Uhlenbeck process is a widely used mathematical model in finance and economics for modeling mean-reverting time series. The model has several advantages, including its ability to capture the mean-reverting behavior of time series and its simple and intuitive modeling approach.

However, it also has several limitations, including its limited realism, limited applicability and the need to estimate more parameters. When using the OU process, it is important to understand both its advantages and limitations and to consider the specific context in which it will be applied.

4.4 Model choices

In this thesis, we face the challenge of modeling the uncertain dynamics of energy prices and demand in a community setting. Specifically, we need to choose a stochastic process that could accurately capture the key features of the dynamics, such as mean reversion.

Traditionally, Geometric Brownian Motion (GBM) has been the go-to stochastic process for modeling asset prices, including energy prices. However, as seen above, GBM has some limitations, such as assuming constant volatility and lacking the ability to capture mean reversion and jumps in the price dynamics.

In order to overcome some of these limitations, we try the stochastic process explored above: the Ornstein-Uhlenbeck (OU) process.

The OU process captures the mean-reverting behavior of energy prices, which is an important feature of energy markets. The use of the OU process allows to develop a stochastic model for optimal investment in Renewable Energy Communities that better reflects the real-world dynamics of energy prices. The model can be used to determine the optimal investment decisions for a Renewable Energy Community, taking into account the uncertain and volatile nature of energy prices and demand.

In conclusion, the decision to use the Ornstein-Uhlenbeck process instead of Geometric Brownian Motion is a critical one that allows to develop a more accurate and realistic stochastic model for Renewable Energy Communities. The use of this alternative stochastic processes highlights the importance of carefully choosing the appropriate model for the problem at hand, taking into account the specific features of the system being modeled.

Chapter 5

Processes and Model implementation

To implement the stochastic model and conduct an optimization, we use `Python` as programming language. Let us point out the main reasons for this choice and the main tools of the language employed.

5.1 Why `Python`

`Python` [13] is a versatile programming language that has gained significant popularity in the scientific computing community. Here are some reasons why one might choose to use `Python`:

- Open-source: `Python` is an open-source language, meaning that it is free to use and distribute. This makes it accessible to a wide range of users, including students and researchers.
- Large ecosystem: `Python` has a large and active community that has developed many useful libraries and tools for scientific computing. These libraries include `NumPy`, `SciPy`, `Pandas`, and `Matplotlib` among others. They provide tools for data manipulation, scientific computing, and data visualization.
- Easy to learn: `Python` has a simple and intuitive syntax that is easy to learn, even for non-programmers. This makes it a popular choice for teaching and research.

- Interactivity: Python provides an interactive environment through tools like Jupyter notebooks, which allow users to combine code, text and visualizations in a single document. This makes it easy to explore data and communicate results.
- Versatility: Python can be used for a wide range of applications, from web development to machine learning. This means that users can leverage their Python skills across many different domains.

For a thesis on stochastic differential equations, Python is an excellent choice of programming language for the following reasons:

- Numerical computing: Python has powerful numerical computing capabilities, including libraries like NumPy and SciPy, which are well-suited to dealing with stochastic differential equations.
- Visualization: Matplotlib and other visualization tools in Python make it easy to generate plots and visualizations of the results of stochastic processes simulations.
- Code readability: Python's clean and intuitive syntax makes it easy to write and understand code, which can be important when working with complex mathematical concepts like stochastic differential equations.

5.2 Stochastic Processes

To simulate the stochastic processes in Python we employ the following tools.

5.2.1 Geometric Brownian Motion in Python

The role played by Python libraries in this code is significant. The NumPy [12] library is used to create an array of zeros and to generate random numbers from a normal distribution. The mathematical operations used to simulate the GBM are also performed using

`NumPy`. Specifically, the `np.exp()` function is used to calculate the exponential of a number, and the `np.sqrt()` function is used to calculate the square root of a number.

Here's how the code works:

The `for` loop then iterates over the specified number of time steps. In each iteration, the code updates the value of the process using the Geometric Brownian Motion equation.

Specifically, the code implements the Euler-Maruyama method to simulate a drifted Brownian Motion and then exponentiates it to obtain the exact solution of a Geometric Brownian Motion. The Euler-Maruyama method is an extension of the Euler method, which is a basic numerical approximation scheme for ordinary differential equations. The key difference is the inclusion of the stochastic term in the update step, allowing for the simulation of stochastic processes such as Geometric Brownian Motion.

The Euler-Maruyama method discretizes a drifted Brownian Motion, which is exponentiated to achieve a numerical simulation of the exact solution of the stochastic differential equation of the Geometric Brownian Motion (4.9). The discretized version, as implemented in the code, can be represented mathematically as follows:

$$X_{i+1} = X_i \exp\left((\mu - \frac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t} Z_i\right) \tag{5.1}$$

where:

- μ is the drift or long-term growth rate,
- σ is the volatility,
- X_{i+1} is the process value at the next time step,
- X_i is the process value at the current time step (initialized to x_0 at the beginning),

- Δt is the time step size,
- Z_i is a random number drawn from a standard normal distribution.

The `mu - 0.5 * sigma ** 2` term represents the drift component of the process. The `sigma * np.sqrt(dt) * np.random.normal(0,1)` term represents the random component of the process, which is generated using the `np.random.normal()` [14] function from `NumPy`. Finally, the code returns the simulated values of the GBM process as an array.

So, with the Euler-Maruyama method, the update for each time step is given by:

```
x[i] = x[i-1] * np.exp((mu - 0.5 * sigma ** 2) * dt +
sigma * np.sqrt(dt) * np.random.normal(0,1))
```

where: `x[i]` represents the value of the process at the current time step i; `x[i-1]` is the value of the process at the previous time step; `mu` is the drift or long-term growth rate; `sigma` is the volatility; `dt` is the time step size; `np.random.normal(0,1)` generates a random number from a standard normal distribution (mean = 0, standard deviation = 1). It represents the random component of the process.

In summary, the `NumPy` library is used extensively in this code to simulate a Geometric Brownian Motion process, which is a stochastic process commonly used in finance to model stock prices and other financial instruments.

5.2.2 Ornstein-Uhlenbeck process in `Python`

In the context of simulating the Ornstein-Uhlenbeck (OU) process in `Python`, `NumPy`, `random`, and `normal` play the following specific roles:

`NumPy` [12] is used in the OU process to create arrays of random numbers. In particular, `NumPy`'s zeros function is used to create an array of zeros to store the simulated values of the OU process. `NumPy`'s `sqrt` function is also used to calculate the square root of the time step, which is needed to simulate the stochastic term in the OU process.

The `random` [14] module is used in the OU process to generate random numbers that are normally distributed. In particular, the `normal` function from the `random` module is used to generate normally distributed random numbers with a mean of zero and a standard deviation of one. These random numbers are used to simulate the stochastic term in the OU process.

To simulate the OU process, we start with an initial value and use an algorithm that updates the value of the process at each time step using the OU equation. In the OU equation, the stochastic term is modeled by a normally distributed random variable. Specifically, the code implements the Euler-Maruyama method to simulate the Ornstein-Uhlenbeck process.

The Euler-Maruyama method discretizes the SDE (4.17) to simulate the process numerically. The discretized version, as implemented in the code, can be represented mathematically as follows:

$$X_{i+1} = X_i + \lambda(\mu - X_i)\Delta t + \sigma\sqrt{\Delta t} Z_i \tag{5.2}$$

where:

- X_{i+1} is the process value at the next time step,
- X_i is the process value at the current time step (initialized to x_0 at the beginning),
- λ is the rate of mean reversion,
- μ is the long-term mean,
- Δt is the time step size,

- Z_i is a random number drawn from a standard normal distribution.

The implementation involves a deterministic drift term (`_lambda * (mu - x[i-1]) * dt`) and a stochastic term (`sigma * np.sqrt(dt) * np.random.normal(0,1)`). The `normal` function from `NumPy` is used to generate this random variable at each time step, which is then used to update the value of the process. The zeros function from `NumPy` is used to initialize an array of zeros to store the simulated values of the process. Together with the normal function, these tools from `NumPy` and `random` enable to simulate the behavior of the OU process over time.

So, with the Euler-Maruyama method, the update step for each time step is given by:

```
x[i] = x[i-1] + _lambda * (mu - x[i-1]) * dt + sigma
* np.sqrt(dt) * np.random.normal(0,1)
```

where `x[i]` represents the value of the process at time step i; `x[i-1]` represents the value of the process at the previous time step `i-1`; `_lambda` is the rate of mean reversion; `mu` is the long-term mean of the process; `dt` is the time step; `sigma` is the volatility of the process; `np.sqrt(dt) * np.random.normal(0,1)` represents the stochastic term, which is a random number drawn from a normal distribution with mean 0 and standard deviation 1, scaled by the square root of the time step.

To allow consistency of measures employed, we build all the stochastic processes assuming hourly time-steps.

An alternative to the above strategy could be to use the exact solution to the OU SDE:

$$\begin{aligned} X_{t_{i+1}} = X_{t_i} e^{-\lambda(t_{i+1}-t_i)} &+ \mu\lambda \int_{t_i}^{t_{i+1}} e^{-\lambda(s-t_i)} ds \\ &+ \sigma \int_{t_i}^{t_{i+1}} e^{-\lambda(s-t_i)} dW_s \end{aligned} \tag{5.3}$$

which contains a Wiener integral and can be written explicitly as:

$$X_{t_{i+1}} = X_{t_i} e^{-\lambda\Delta t} + \mu(1 - e^{-\lambda\Delta t}) + \sigma\sqrt{\frac{1 - e^{-2\lambda\Delta t}}{2\lambda}}\sqrt{\Delta t} Z_i \tag{5.4}$$

where:

- $X_{t_{i+1}}$ is the value of the process at time t_{i+1},
- X_{t_i} is the initial value of the process at t_i,
- λ is the rate of mean reversion,
- μ is the long-term mean,
- Δt is the time step size,
- σ is the volatility,
- Z_i is a random number drawn from a standard normal distribution.

This explicit formula provides the exact solution of the OU process at any given time t, given its initial value X_0 and the model parameters λ, μ, and σ. It eliminates the need for numerical approximation methods and allows direct computation of the process values at different time points.

5.3 Numerical Integration

To compute the integrals which are present in the code it is necessary to rely on numerical integration. Specifically we use `scipy.integrate.quad`.

5.3.1 `scipy.integrate.quad`

`scipy.integrate.quad` [18] uses a combination of Gaussian quadrature and adaptive subdivision to approximate the definite integral of a given function over a specified interval.

Gaussian quadrature is a method for approximating integrals by evaluating the integrand at a set of carefully chosen points, called the quadrature points, and weighting the results by a set of corresponding weights. The goal of Gaussian quadrature is to choose the quadrature points and weights in such a way that the resulting approximation is as accurate as possible for polynomials up to a certain degree.

In `scipy.integrate.quad`, the quadrature points and weights are determined by the Legendre-Gauss quadrature rule, which is a specific type of Gaussian quadrature that is optimized for the interval $[-1, 1]$. To use this rule for a general interval $[a, b]$, `scipy.integrate.quad` first transforms the interval to the interval $[-1, 1]$ using a change of variables, and then applies the Legendre-Gauss quadrature rule to the transformed integrand.

The adaptive subdivision aspect of `scipy.integrate.quad` comes into play when the algorithm encounters regions of the integrand that are particularly difficult to approximate accurately using Gaussian quadrature. In these cases, `scipy.integrate.quad` subdivides the integration interval into smaller subintervals and applies the quadrature rule to each subinterval. This adaptive subdivision process continues until the estimated error in the approximation falls below the specified tolerance level.

In summary, `scipy.integrate.quad` uses a combination of

Gaussian quadrature and adaptive subdivision to approximate the definite integral of a given function over a specified interval. It does this by first transforming the integration interval to the interval $[-1, 1]$ using a change of variables, and then applying the Legendre-Gauss quadrature rule to the transformed integrand. If the estimated error in the approximation is too high, `scipy.integrate.quad` subdivides the integration interval into smaller subintervals and repeats the process until the estimated error falls below the specified tolerance level.

So, to integrate to infinity, it is necessary to set as upper bound a very large finite number, according to the specific kind of problem that we are facing. In the implementation of this model, however, we decide to integrate over a reasonable life horizon of the equipment - and not over an infinite time - estimated around 20 years.

5.4 Optimization

To find the optimal investment of the two members of the REC, avoiding the problem of fair share of incentives, we define as objective function the sum of the profits of the household and of the biogas producer

$$f(y_b, y_h) = J_b(x_v, x_c, p, d, y_b, y_h, \beta) + J_h(x_v, x_c, p, d, y_b, y_h, \beta) \quad (5.5)$$

which is, as it is clear from the above formula, a function of two variables. These two variables are, respectively, the biogas producer y_b and the "representative" household y_h investments. Defining this objective function for the optimization we get a cancellation of the terms involving β, allowing us to working only on optimal investment.

This optimization problem corresponds, according to the following lemma from [2], to finding a Pareto optimal solution.

Lemma 1. *If $\hat{I} \in argmax\, J(I)$, then $\hat{I}$ is Pareto optimal.*

Proof. Suppose $\hat{I} \in \operatorname{argmax} J(\bar{I})$ and assume $\hat{I}$ is not Pareto optimal. Then, there exists I^* such that,

$$J_i(I_i^*) \geq J_i(\hat{I}_i), \quad \forall i \in \{1, \ldots, N\} \tag{5.6}$$

where at least one inequality is strict. Then,

$$\sum_{i=1}^{N} J_i(I_i^*) > \sum_{i=1}^{N} J_i(\hat{I}_i) \tag{5.7}$$

contradicting the fact that $\hat{I}$ is maximizing.

□

Therefore, the optimal investments for the two members of the community that we are willing to find are such that we cannot improve the condition of one member without worsening the condition of the other one.

At this point, we define all the constraints mentioned in the model. Finally, we optimize the objective as a function of the two variables (y_b, y_h), given all the inputs and the constraints, using the `'SLSQP'` method.

The `'SLSQP'` optimization method is part of the `scipy.optimize` module in the `SciPy` library for `Python`. The `SciPy` [17] library is an open-source library used for scientific computing and technical computing. To use the method for optimization, it is first necessary to import the `minimize` function from the `scipy.optimize` [16] module.

5.4.1 `'SLSQP'`

The `'SLSQP'` [15] method is based on the Sequential Quadratic Programming (SQP) algorithm, which is a type of optimization algorithm that uses quadratic approximations of the objective function and constraints. The main idea of the SQP algorithm is to iteratively solve a sequence of quadratic programming subproblems that approximate the nonlinear optimization problem.

The `'SLSQP'` method starts by defining the objective function and the constraints of the optimization problem. The objective function is the function that needs to be minimized or minus the function that needs to be maximized, while the constraints are the conditions that must be satisfied by the solution. The `'SLSQP'` method requires the objective function to be twice continuously differentiable and the constraints to be continuous functions.

The `'SLSQP'` method then creates a quadratic approximation of the objective function and the constraints at the current iterate. This approximation is based on the first and second derivatives of the objective function and constraints, evaluated at the current iterate.

If the explicit functions of the derivatives are not available, the `'SLSQP'` method can use the numerical approximations computed with the finite differences method.

The `'SLSQP'` method then solves the quadratic programming subproblem using a Sequential Linear Programming (SLP) algorithm. The SLP algorithm solves a sequence of linear programming subproblems, where each subproblem is obtained by linearizing the quadratic approximation of the objective function and constraints.

If the solution of the quadratic programming subproblem satisfies the Kuhn-Tucker conditions, which are the necessary conditions for optimality in nonlinear constrained optimization, then the solution is accepted as the new iterate. Otherwise, the `'SLSQP'` method backtracks to the previous iterate and tries again with a smaller step size.

The 'SLSQP' method continues iterating until a convergence criterion is met. The convergence criterion is typically based on the magnitude of the gradient of the objective function and the constraints, or on the change in the objective function and the constraints between iterations.

One of the advantages of the 'SLSQP' method is that it is a well-established and widely used algorithm for nonlinear constrained optimization. It has been extensively tested and is known to be robust and efficient for a wide range of optimization problems. Moreover, the 'SLSQP' method can handle problems with both inequality and equality constraints, and it can handle problems with nonlinear constraints.

In conclusion, the 'SLSQP' method is a powerful and versatile optimization algorithm that can solve a wide range of nonlinear constrained optimization problems. Its effectiveness and efficiency make it a valuable tool for many applications in science and engineering.

5.5 Code

Here follows the `Python` implementation of the processes and of the mathematical model previously described. The parameters used to model the processes of the model are based on the information found in [1].

The first few lines of the code import the necessary packages: `Pandas` for data analysis, `NumPy` for scientific computing, `math` for mathematical functions, `random` for generating random numbers, and `matplotlib.pyplot` for data visualization.

The last two lines import specific functions from the `SciPy` library, namely `quad` for numerical integration and `minimize` for optimization.

```
# Master's thesis, Lorenzo Portaluri
# Import packages
import pandas as pd
import numpy as np
import math
import random
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 24})
from scipy.integrate import quad
from scipy.optimize import minimize
```

5.5.1 Geometric Brownian Motion

```
def geometric_brownian_motion(x0, mu, sigma, dt, n_timesteps):

    # x0 is the initial value of the process
    # mu is the drift or long-term growth rate
    # sigma is the volatility
    # dt is the time step
    # n_timesteps is the number of time steps to simulate for

    # Initialize an array of zeros to store the values
    # of the process
    x = np.zeros(n_timesteps)

    # Set the initial value of the process
    x[0] = x0

    # Loop over the time steps to simulate the process
    for i in range(1, n_timesteps):

        # Update the value of the process at each time step
        # using the Geometric Brownian Motion equation
        x[i] = x[i-1] * np.exp((mu - 0.5 * sigma ** 2) * dt
        + sigma * np.sqrt(dt) * np.random.normal(0,1))

    # Return the simulated values of the process
    return x

# Define the parameters for the simulation
x0 = 10
mu = 0
sigma = 0.2
dt = 0.01
n_timesteps = 100
n_simulations = 10

# Call the function to simulate the process
fig, ax = plt.subplots(figsize=(30, 15))

for i in range(n_simulations):
    x = geometric_brownian_motion(x0, mu, sigma, dt, n_timesteps)

    # Plot the simulated values
    plt.plot(x)

# Show the graph
plt.xlabel("Time step")
plt.ylabel("Value")
plt.title("Simulation of Geometric Brownian Motions")
plt.show()
```

5.5.2 Ornstein-Uhlenbeck

```
# Define a function implementing Ornstein-Uhlenbeck processes
def ornstein_uhlenbeck_process(x0, _lambda, mu, sigma, dt,
n_timesteps):

    # x0 is the initial value of the process
    # _lambda is the rate of mean reversion
    # mu is the long-term mean
    # sigma is the volatility
    # dt is the time step
    # n_timesteps is the number of time steps to simulate for

    # Initialize an array of zeros to store the values
    # of the process
    x = np.zeros(n_timesteps)

    # Set the initial value of the process
    x[0] = x0

    # Loop over the time steps to simulate the process
    for i in range(1, n_timesteps):

        # Update the value of the process at each time step
        # using the Ornstein-Uhlenbeck equation
        x[i] = x[i-1] + _lambda * (mu - x[i-1]) * dt
        + sigma * np.sqrt(dt) * np.random.normal(0,1)

    # Return the simulated values of the process
    return x
# Define the parameters for the simulation
x0 = 10
_lambda = 1
mu = 10
sigma = 0.2
dt = 0.01
n_timesteps = 100
n_simulations = 10

# Call the function to simulate the process
fig, ax = plt.subplots(figsize=(30, 15))

for i in range(n_simulations):
    x = ornstein_uhlenbeck_process(x0, _lambda, mu, sigma, dt,
    n_timesteps)

    # Plot the simulated values
    plt.plot(x)

# Show the graph
```

```
plt.xlabel("Time step")
plt.ylabel("Value")
plt.title("Simulation of Ornstein-Uhlenbeck processes")
plt.show()

# Define the parameters for the processes of the model
_lambda = 1
dt = 0.01
n_timesteps = 45000 # = 9 hours * 250 days * 20 years

# Spot price at which biogas is sold by the producer
# on the market
# Let us assume it goes as an OU process
P = ornstein_uhlenbeck_process(74.7, _lambda, 74.7, 0.84, dt,
n_timesteps)

# Biogas producer total gas production capacity K_g
K_g = 18.9394

# Spot price at which electricity produced
# from biogas is sold by the producer on the market
# Let us assume it goes as an OU process
X_v = ornstein_uhlenbeck_process(56.7, _lambda, 56.7, 0.09, dt,
n_timesteps)

# Biogas producer maximum power to transform gas
# into power theta_b
theta_b = 0.2

# Spot price at which the household buys energy in
# the electricity market in order to satisfy its demand
# Let us assume it goes as an OU process
X_c = ornstein_uhlenbeck_process(65, _lambda, 65, 0.001, dt,
n_timesteps)

# Household maximum allowed power to install new
# photovoltaic panels theta_h
theta_h = 0.32

# The demand of the energy community is given
# by the household power demand
# Let us assume it goes as an OU process
D = ornstein_uhlenbeck_process(0.3, _lambda, 0.3, 0.04,
dt, n_timesteps)
```

5.5.3 Integration

```
# Additionl parameters relevant for the model

# Suppose that the household receives a part beta in (0, 1)
# of the incentive, while the biogas producer receives 1 - beta
beta = 0.5 # Let us assume a half-half split

# Interest rate (hourly translated)
r = 3e-6 # From the paper

# Incentives
Z = 110

# The biogas profit functional is then written as
def J_b(y_b, y_h):
    my_J_b = 0
    b = 0.1056
    c_b = 900000

    my_integral_1 = []
    my_integral_2 = []
    my_integral_3 = []

    for i in range(0, 100):
    # Argumets of the integrals for each timestep
        for s in range(0, n_timesteps):

            def arg_1(s, y_b):
                return math.exp(-r*s) * X_v[int(s)] * y_b
            def arg_2(s, y_b, y_h):
                return math.exp(-r*s) * (1-beta) * Z *
                min(D[int(s)], y_b+y_h)

            def arg_3(s, y_b):
                return math.exp(-r*s) * P[int(s)] *
                (b*K_g - y_b)

        integral_1, error_1 = quad(arg_1, 0, n_timesteps,
        args=(y_b), epsabs=1e-3, epsrel=1e-3, limit = 10000)

        integral_2, error_2 = quad(arg_2, 0, n_timesteps,
        args=(y_b, y_h), epsabs=1e-3, epsrel=1e-3, limit = 10000)

        integral_3, error_3 = quad(arg_3, 0, n_timesteps,
        args=(y_b), epsabs=1e-3, epsrel=1e-3, limit = 10000)

        my_integral_1.append(integral_1)
        my_integral_2.append(integral_2)
        my_integral_3.append(integral_3)

    # Do a number of simulations of integrals to have
    # something meaningful
    comp_1 = np.mean(my_integral_1)
    comp_2 = np.mean(my_integral_2)
    comp_3 = np.mean(my_integral_3)

    # Final computation
    my_J_b = comp_1 + comp_2 + comp_3 - c_b*y_b

    return my_J_b

# The profit of the household is written as
def J_h(y_b, y_h):
    my_J_h = 0
    c_h = 2500000

    my_integral_1 = []
    my_integral_2 = []
    my_integral_3 = []
```

```
for i in range(0, 100):
# Argumets of the integrals for each timestep
    for s in range(0, n_timesteps):

        def arg_1(s, y_h):
            return math.exp(-r*s) * X_v[int(s)] * y_h

        def arg_2(s):
            return math.exp(-r*s) * X_c[int(s)] * D[int(s)]

        def arg_3(s, y_b, y_h):
            return math.exp(-r*s) * Z * beta *
            min(D[int(s)], y_b+y_h)

    integral_1, error_1 = quad(arg_1, 0, n_timesteps,
    args=(y_h), epsabs=1e-3, epsrel=1e-3, limit = 10000)

    integral_2, error_2 = quad(arg_2, 0, n_timesteps,
    epsabs=1e-3, epsrel=1e-3, limit = 10000)

    integral_3, error_3 = quad(arg_3, 0, n_timesteps,
    args=(y_b, y_h), epsabs=1e-3, epsrel=1e-3, limit = 10000)

    my_integral_1.append(integral_1)
    my_integral_2.append(integral_2)
    my_integral_3.append(integral_3)

# Do a number of simulations of integrals to have
# something meaningful
comp_1 = np.mean(my_integral_1)
comp_2 = np.mean(my_integral_2)
comp_3 = np.mean(my_integral_3)

# Final computation
my_J_h = comp_1 - comp_2 + comp_3 - c_h*y_h

return my_J_h
```

5.5.4 Optimization

```
# Define the objective function,
# to maximize a function minimize minus the function
def obj_function(y):

    return (-(J_b(y[0],y[1]) + J_h(y[0],y[1])))

# Define the constraints functions
def ggg(y):
    return [theta_b, theta_h] - y
def hhh(y):
    return y

# Define the constraints
cons = {'type': 'ineq', 'fun': ggg,
       'type': 'ineq', 'fun': hhh}

# Set initial guess
x_0 = [0.20,0.32]

# Optimize, using 'SLSQP'
res0 = minimize(obj_function, x_0, constraints=cons,
method='SLSQP', tol=1e-6)
display(res0)
```

5.6 Output

Executing the above `Python` code we obtain the following output in Figures 1, 2 and 3:

Figure 1

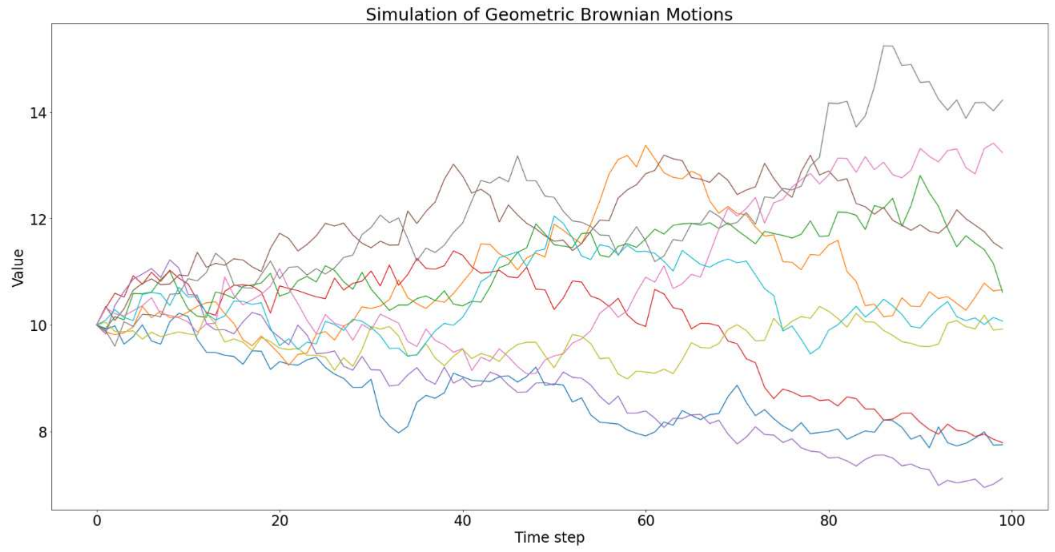


Figure 2

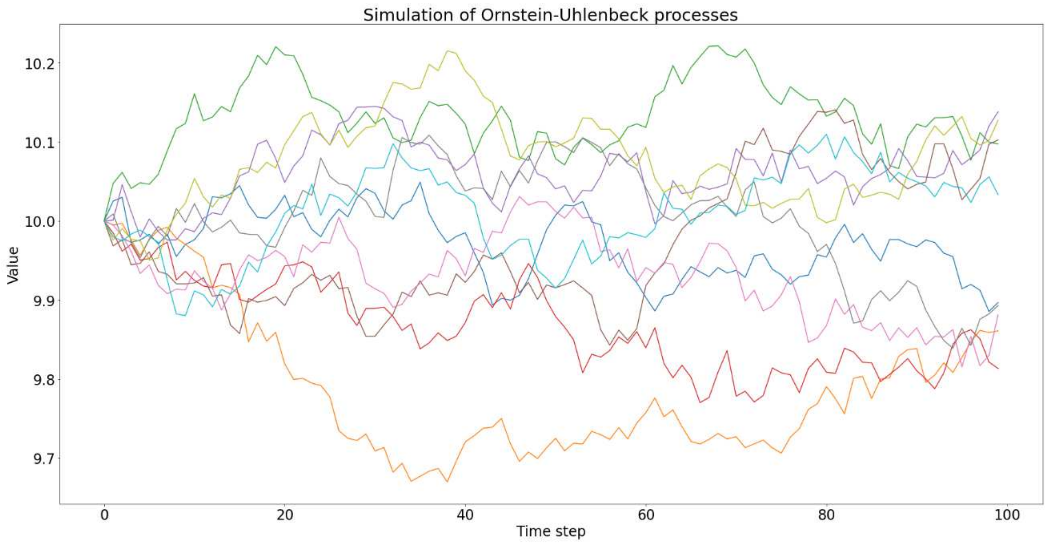


Figure 3

```
     fun: -6675388.01987983
     jac: array([1656687.6875,  113453.625 ])
 message: 'Optimization terminated successfully'
    nfev: 10
     nit: 6
    njev: 2
  status: 0
 success: True
       x: array([0.08688509, 0.31212891])
```

Figures 1 and 2 show generic simulations of Geometric Brownian Motions and Ornstein-Uhlenbeck processes. We can notice in Figure 1 how Geometric Brownian Motions tend to diverge from the initial value. Ornstein-Uhlenbeck processes in Figure 2, instead, revert to the long-term mean, which we set equal to the initial value of the processes.

Moreover, we see in Figure 3 that the optimization procedure was completed successfully. In the specific case proposed, the parameters of the model are set as follows, starting from [1]:

- The conversion factor b expressing the cubic meters of gas needed to produce 1 MW of power is set $b = 0.1056\,\mathrm{m}^3/\mathrm{MW}$;
- The discount factor r is set $r = 3 \times 10^{-6}\,1/\mathrm{h}$;
- The installation cost of photovoltaic panels c_h required to produce 1 MW of solar power is set $c_h = 2500000$ €/MW;
- The installation cost of a turbine c_b that produces 1 MW of power is set $c_b = 900000$ €/MW;
- The biogas producer's total gas production capacity K_g is set $K_g = 18.9394\ \mathrm{m}^3$;
- The biogas producer's maximum power transformation capacity θ_b is set $\theta_b = 0.2$ MW;
- The household's maximum power transformation capacity θ_h is set $\theta_h = 0.32$

- The incentives received by the energy community Z is set $Z = 110$ €/MW.

Regarding the parameters of the stochastic processes, instead, in [1] they were estimated for a GBM-based model. For our OU-based implementation we took the initial value of the processes, setting it as long-term mean too, and the estimated values of volatility $\sigma_v = 0.09$, $\sigma_p = 0.84$, $\sigma_v = 0.04$, $\sigma_v = 0.01$. Finally, we set the rate of mean reversion $\lambda = 1$.

In the end, we obtained a suggested optimal investment of 312.13 kW for the "representative" household and 86.90 kW for the biogas producer.

Chapter 6

Further research

A possible example for further research is to explore other stochastic processes such as Lévy processes. Let us see the main features of Lévy processes.

6.1 Lévy Process

6.1.1 Introduction

A Lévy process [9] is a stochastic process with stationary and independent increments, named after the French mathematician Paul Lévy. It is a continuous-time process that takes values in the real line or in a higher dimensional Euclidean space.

Mathematically, a Lévy process can be defined as a stochastic process $\{X_t; t \geq 0\}$, with the following properties:

- $X_0 = 0$ almost surely.
- The process has stationary and independent increments, meaning that for any $s < t$, the random variable $X_t - X_s$ is independent of the sigma-algebra generated by the process up to time s, and its distribution depends only on the time difference $t - s$.
- The process is stochastically continuous, which means that for any $\epsilon > 0$ and any $s < t$, the probability of $|X_t - X_s| > \epsilon$ goes

to zero as $(t - s) \to 0$.

These properties imply that a Lévy process may have jumps, and the sizes and timings of these jumps are determined by the underlying Lévy measure. The Lévy measure is a measure on the real line or on the Euclidean space that describes the distribution of the jumps of the process.

Lévy processes have several important applications in finance, physics, and other areas of science. Their infinite divisibility property, which means that any finite segment of a Lévy process can be decomposed into independent segments, makes it a useful tool for modeling complex systems with random behavior. The dynamics of a Lévy process can be described by the Lévy-Khintchine formula, which expresses the characteristic function of the process as a function of the drift, the diffusion coefficient, and the Lévy measure.

The Lévy-Khintchine formula is a mathematical expression that describes the characteristic function of a Lévy process. The characteristic function is a complex-valued function that completely determines the distribution of a stochastic process. In the case of a Lévy process, the characteristic function describes the probability distribution of the process at any time.

The Lévy-Khintchine formula expresses the characteristic function of a Lévy process as follows:

$$\mathbb{E}[e^{i\xi X_t}] = e^{t\psi(\xi)} \tag{6.1}$$

where $\psi(\xi)$ is the characteristic exponent of the Lévy process, which is given by:

$$\psi(\xi) = i\gamma\xi - \frac{1}{2}\sigma^2\xi^2 + \int_{\mathbb{R}\setminus\{0\}} (e^{i\xi x} - 1 - i\xi x \mathbf{1}_{|x|<1})\nu(dx) \tag{6.2}$$

Here, γ is the drift coefficient, σ is the diffusion coefficient, and $\nu(dx)$ is the Lévy measure, which describes the distribution of the jumps of the process.

The Lévy-Khintchine formula is a powerful tool for analyzing the properties of Lévy processes. It allows us to derive various statistical properties of the process, such as its mean, variance, and higher moments. Moreover, it can be used to simulate Lévy processes numerically, which is important for applications in finance and other areas.

Overall, the Lévy-Khintchine formula is a fundamental result in the theory of Lévy processes, and it plays a key role in their analysis and application.

6.1.2 Positive and negative aspects

Lévy processes are a powerful tool for modeling a wide range of phenomena in science, finance, and engineering. However, like any modeling approach, they have pros and cons, which must be carefully considered when using it to analyze data or make predictions.

Pros:

- Flexibility: one of the key advantages of Lévy processes is the flexibility. The process can be customized to fit a wide range of data sets and phenomena, making it a versatile tool for modeling complex systems.
- Jumps: Lévy processes are especially useful for modeling systems that exhibit sudden jumps or discontinuous changes. This feature is particularly important in finance, where asset prices can experience large fluctuations in short periods of time.
- Multiscale: Lévy processes are inherently multiscale, meaning that they can capture behavior over different time scales. This is important in many applications, including financial model-

ing, where it is necessary to capture both short-term and long-term trends.

- Heavy tails: Lévy processes often have heavy-tailed distributions, which means that extreme events are more likely to occur than in other types of stochastic processes. This is important in many fields, including finance, where it is necessary to model rare events such as market crashes.

Cons:

- Complex: a Lévy process is a complex mathematical model that can be difficult to understand and apply correctly. Its complexity means that it requires careful calibration and interpretation, which can be time-consuming and error-prone.
- Parameters: every Lévy process has two parameters and a measure that must be carefully chosen in order to accurately model a system. These parameters can be difficult to estimate, particularly when the data is noisy or incomplete.
- Computational demands: The simulation of Lévy processes can be computationally demanding, particularly when modeling large data sets or complex systems. This can make it impractical for certain applications.
- Limited applicability: while Lévy processes are a powerful tool for modeling certain types of systems, they may not be suitable for all applications. For example, they may not be appropriate for systems that exhibit smooth, continuous behavior, such as Brownian motion.
- Does not account for market frictions: also this kind of process, as the two seen before, does not account for various market frictions, such as transaction costs, taxes, and market impact, which can have a significant impact on asset prices.

In conclusion, Lévy processes are a powerful tool for modeling complex systems that exhibit sudden jumps or discontinuous changes. However, they requires careful calibration and interpretation, and may not be suitable for all applications. As with any modeling approach, it is important to carefully consider the pros and cons of Lévy processes before applying them to a particular problem.

6.2 Lévy process implementation

6.2.1 Stochastic process

As for Geometric Brownian Motion and Ornstein-Uhlenbeck processes, we implement Lévy processes in `Python`. The kind of Lévy process we are simulating is built with a linear combination of Brownian increments and Poisson increments, to allow for the presence of jumps. To allow for both jumps upward and downward, the function in the code is built with a difference of two Poisson increments with equal intensities, which can result in both a positive and negative outcome.

The Lévy process implemented in the code can be represented mathematically as:

$$X(n) = X(0) + \sum_{i=1}^{n} (\beta \Delta t + \sigma \Delta W_i + \Delta N_i), \tag{6.3}$$

where:

- $X(n)$ is the Lévy process at step n;
- $X(0)$ is the initial value of the process;
- β is the drift coefficient;
- σ is the diffusion coefficient;
- Δt is the time increment;
- ΔW_i is the increment of the standard Brownian motion, generated as $\Delta W_i \sim N(0, \sqrt{\Delta t})$;

- ΔN_i is the increment of the Poisson process, generated as $\Delta N_i \sim \text{Poisson}(\alpha \Delta t) - \text{Poisson}(\alpha \Delta t)$, where α is the Poisson process intensity.

The Lévy process is obtained by cumulatively summing the increments starting from $X(0)$, where the sum includes the drift term ($\beta \Delta t$), the diffusion term ($\sigma \Delta W_i$), and the jump term (ΔN_i).

6.2.2 Lévy processes in `Python`

The `Python` libraries and tools used are essential for simulating a Lévy process with the given parameters and initial value.

`NumPy` [12] is used to generate random normal and Poisson distributed numbers, which are used as increments for the Brownian Motion and Poisson processes, respectively.

The `numpy.ndarray` data type is used to store the simulated process values. From `random` [14] the `random.normal()` and `random.poisson()` functions in `NumPy` are used to generate random normal and Poisson distributed numbers, respectively.

The `concatenate()` function in `NumPy` is used to join together the initial value of the process with the cumulatively summed increments to get the simulated process values.

All of these tools are crucial for simulating a Lévy process in `Python` and allow for efficient and accurate numerical simulations.

To simulate a Lévy process, we start with an initial value and use the algorithm that generates random increments based on a Poisson process and a Brownian motion. The simulated values of the process are then calculated by cumulatively summing the increments starting from the initial value of the process using `np.cumsum`. Finally, the simulated values are stored in a `NumPy` array, which is returned as the output of the function.

The function generates a stochastic process that has both Brownian motion and Poisson increments, which are the building blocks of a Lévy process. Therefore, the function simulates a Lévy process.

The function assumes that the increments of the Poisson process have a constant intensity `alpha*dt`. To allow both jumps upward and downward, the function is built with a difference of Poisson increments, which can result in both a positive and negative outcome.

Brownian motion increment:

```
dW = np.random.normal(0, np.sqrt(dt), n_steps)
```

The code generates a random array called `dW`, representing the increment of a standard Brownian motion process. It uses the `NumPy` function `np.random.normal(0, np.sqrt(dt), n_steps)`. Each element of `dW` is a random number drawn from a normal distribution with mean 0 and standard deviation equal to the square root of the time increment `dt`. The array `dW` captures the random fluctuations associated with the diffusion term.

Poisson increment:

```
dN = np.random.poisson(alpha*dt, n_steps) -
np.random.poisson(alpha*dt, n_steps)
```

The code generates another random array called `dN`, representing the increment of a Poisson process. It uses the difference between two arrays: `np.random.poisson(alpha * dt, n_steps) - np.random.poisson(alpha * dt, n_steps)`. Each element of `dN` represents the difference between two random numbers drawn from a Poisson distribution with mean equal to the product of the Poisson process intensity alpha and the time increment `dt`. By subtracting two arrays of Poisson random numbers, we obtain an array `dN` with a mean of 0, reflecting the stochastic jumps in the process.

The overall formula to calculate the process values is as follows:

```
X = x0 + np.concatenate(([0], np.cumsum(beta*dt +
sigma*dW + dN)))
```

The code initializes an array called `X` to store the simulated values of the Lévy process. It starts with the initial value `x0` and adds the cumulative sum of the increments, resulting in the array `X`. The cumulative sum is obtained using `np.cumsum(beta * dt + sigma * dW + dN)`. Here, `beta * dt` represents the deterministic drift term scaled by the time increment. `sigma * dW` represents the diffusion term, where sigma is the diffusion coefficient and `dW` is the Brownian motion increment. `dN` represents the Poisson increment capturing the jumps in the process. The cumulative sum is calculated using `np.cumsum()`, which sums the elements of the increments array cumulatively. Finally, the initial value `x0` is added at the beginning of the cumulative sum array using `np.concatenate(([0], ...))`.

6.2.3 Code

Here follows the `Python` implementation of a Lévy processes simulation

```
# Define a function implementing Lévy processes
def levy_process(alpha, beta, sigma, n_steps, dt, x0):

    # alpha is the Poisson process intensity
    # beta is the drift coefficient
    # sigma is the diffusion coefficient
    # n_steps is the number of time steps to simulate
    # dt is the time increment
    # x0 is the initial value of the process (default is 0)


    # Generate the increments of the process
    dW = np.random.normal(0, np.sqrt(dt), n_steps)  # Brownian
    # motion increment

    dN = np.random.poisson(alpha*dt, n_steps) -
    np.random.poisson(alpha*dt, n_steps)  # Poisson increment

    # Calculate the process values by cumulatively summing the
    # increments starting from X0
    X = x0 + np.concatenate(([0], np.cumsum(beta*dt
    + sigma*dW + dN)))

    return X

# Define the parameters for the simulation
n_steps = 100 # number of time steps
dt = 0.01  # time increment
alpha = 1  # Poisson process intensity
beta = 0  # drift coefficient
sigma = 1  # diffusion coefficient
x0 = 10  # initial value

# Simulate the process 10 times with starting point
# x0 and plot the results
fig, ax = plt.subplots(figsize=(30, 15))

for i in range(10):
    X = levy_process(alpha, beta, sigma, n_steps, dt, x0)
    plt.plot(X)

# Set the plot labels and title
plt.xlabel('Time step')
plt.ylabel('Value')
plt.title('Simulation of Lévy Processes')

# Show the plot
plt.show()
```

6.2.4 Output

Executing the above `Python` code we obtain the following output in Figure 4:

Figure 4

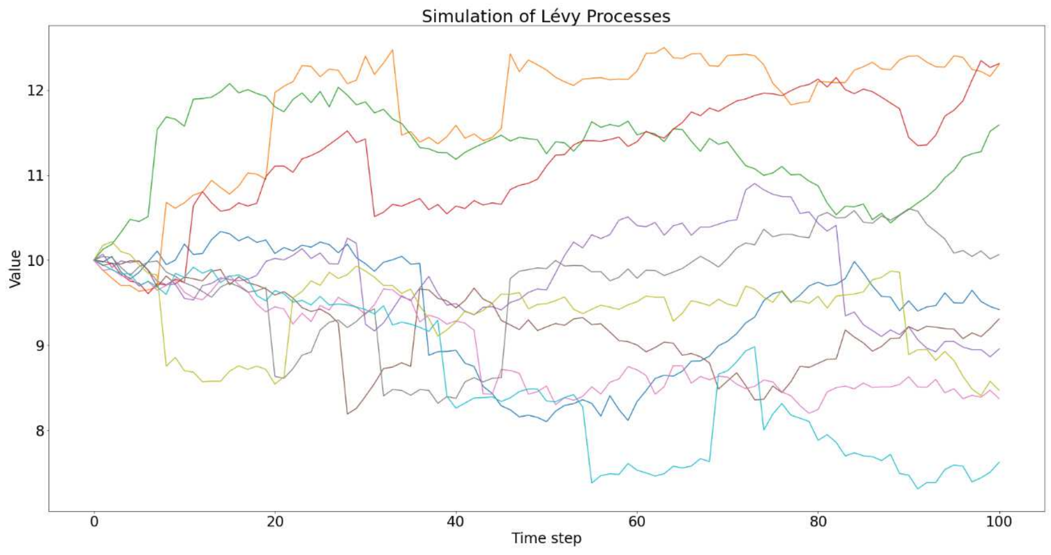


Figure 4 represents a generic simulation of Lévy processes. We can notice how these processes show the presence of random jumps both upwards and downwards.

6.3 Seasonality

Another major improvement to the model could be to add seasonality functions. Seasonality is a well-known phenomenon in various fields, including economics, energy markets, and agriculture. Seasonal patterns can have a significant impact on prices, production, and demand for goods and services, and therefore it is important to incorporate them into many models.

In the context of financial modeling, a seasonality function can be added as a deterministic element to the stochastic process to capture these patterns. This involves modifying the model to include a periodic component that captures the fluctuations in the data due to seasonal effects.

By incorporating seasonality into the model, it becomes more realistic and accurate, and can provide more reliable forecasts for future trends. For instance, seasonality is a crucial factor in energy markets, affecting both the production and consumption of various energy sources.

For example, the demand for natural gas tends to increase during the winter months due to heating needs, while the demand for electricity tends to increase during the summer months due to air conditioning needs. In addition, the production of certain types of energy, such as solar and wind power, can be influenced by seasonal factors such as weather patterns and daylight hours.

Overall, adding a seasonality function to the model is a significant improvement that can enhance the accuracy and usefulness of the model. It enables the model to better capture the dynamics of the underlying processes and provides more reliable forecasts that account for the impact of seasonal patterns. Therefore, it is an crucial component to consider when modeling various real-world phenomena that exhibit seasonality.

There are several deterministic functions that can be combined with stochastic processes to model seasonality in energy prices, production and demand. We can employ, for example, Fourier series as suggested in the paper [10].

6.3.1 Fourier series

Energy prices, production, and demand often exhibit seasonal patterns that repeat themselves over time. These seasonal patterns can be modeled using deterministic functions represented as Fourier series. Fourier series can be used to decompose a periodic function into a sum of sine and cosine waves, with each wave having a different frequency and amplitude. By including Fourier series in a stochastic model, the seasonal patterns in energy prices, production, and demand can be modeled and incorporated into a stochastic model that also captures the random fluctuations.

The Fourier series [3] representation of a periodic function $f(x)$ with period T is given by:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos(n\omega x) + b_n \sin(n\omega x)) \tag{6.4}$$

where a_0, a_n, and b_n are Fourier coefficients, $\omega = 2\pi/T$ is the angular frequency, and x is time. The first term $a_0/2$ is the average value of the function over one period,

$$a_0 = \frac{2}{T} \int_0^T f(x) dx \tag{6.5}$$

and the summation represents the contribution of each harmonic frequency to the periodic function. Each harmonic frequency is a multiple of the fundamental frequency ω, which is the reciprocal of $T/(2\pi)$. Finally, the coefficients a_n, and b_n are given by the following integrals:

$$a_n = \frac{2}{T} \int_0^T f(x) \cos\left(\frac{2\pi n x}{T}\right) dx \tag{6.6}$$

$$b_n = \frac{2}{T} \int_0^T f(x) \sin\left(\frac{2\pi n x}{T}\right) dx \tag{6.7}$$

The higher the number of harmonics included in the Fourier series, the better the approximation of the periodic function. However, it is important to note that Fourier series assumes that the seasonal pattern is completely deterministic, which may not always be the case in practice. Therefore, it is common to combine Fourier series with stochastic processes to account for the stochastic component of seasonality.

In summary, a Fourier series can be a useful deterministic function for capturing seasonality in energy prices, production, and demand. By combining Fourier series with stochastic processes, a more accurate and robust model can be developed for representing values of energy prices, production, and demand.

6.3.2 Fourier series implementation

Let us show an example of Fourier series implementation. We will show the approximation of a square-wave function with an increasing number of harmonics.

The code uses several Python libraries:

`NumPy` is used to generate an array of x-values with a given range and resolution, and to perform mathematical operations on these arrays.

`matplotlib.pyplot` is used to plot the square wave and its Fourier series approximation.

The code defines a `squareWave()` function that generates a square wave with a given period and phase shift. It also defines a `fourierSeries()` function that computes the partial sum of the Fourier series up to a given number of harmonics. The `bn()` and `wn()` functions are used to compute the coefficients and frequencies of the Fourier series.

The code then creates a list of harmonics and a list of colors for the plot. It generates a subplot for each number of harmonics in `harmonics`, plots the true square wave and its Fourier series representation, and sets the title of the subplot. Finally, it shows the plot using `plt.show()`.

The code uses only the coefficient b_n to compute the Fourier series approximation of the square wave. The coefficients a_0 and a_n are not needed in this case because the function being approximated, a square wave, is an odd function, meaning that its even Fourier coefficients are all zero (including a_0), and its odd Fourier coefficients are given by the b_n coefficients. Therefore, only the b_n coefficients are defined in the code.

6.3.3 Code

Here follows the `Python` implementation of Fourier series representations of a periodic function:

```
# Setup
x = np.linspace(-20, 20, 10000) # Define the x-axis range
# and resolution
T = 10 # Define the period of the square wave

# Define a function to generate a square wave
def squareWave(x, T, phi=0):
    return 2 * (np.sin(2 * np.pi * x / T + 4*phi) > 0) - 1

# Define the bn coefficients of the Fourier series
def bn(n):
    if n % 2 != 0:
        return 4 / (np.pi * n)
    else:
        return 0

# Define the wn frequencies of the Fourier series
def wn(n):
    return 2 * np.pi * n / T

# Define a function to compute the partial sum
# of the Fourier series up to n_max
def fourierSeries(n_max, x):
    a0 = 0 # The DC component of the Fourier series
    partialSums = a0

    for n in range(1, n_max):
        partialSums += bn(n) * np.sin(wn(n) * x)
    return partialSums

# Plot the Fourier series approximation of
# the square wave with different numbers of harmonics
harmonics_list = [2, 5, 10, 15, 20]
colors = ['lightblue', 'red', 'green', 'orange', 'pink']
fig, axs = plt.subplots(len(harmonics_list), figsize=(30,
15*len(harmonics_list)))

for i, harmonics in enumerate(harmonics_list):

    y = squareWave(x, T, np.pi/2) # Generate the true square
    # wave with a phase shift of pi/2

    f = fourierSeries(harmonics, x) # Compute the Fourier
    # series approximation up to n_max=armonics

    axs[i].plot(x, y, color='blue', label='Signal') # Plot the
    # true square wave
    axs[i].plot(x, f, color=colors[i], label='Fourier series
    approximation') # Plot the Fourier series approximation
    axs[i].set_title(f"Fourier Series approximation with
    {harmonics} harmonics") # Set the title of the subplot
    axs[i].legend() # Add a legend to the subplot

plt.show() # Show the plot
```

6.3.4 Output

Executing the above `Python` code we obtain the following output in Figures 5, 6, 7, 8 and 9:

Figure 5

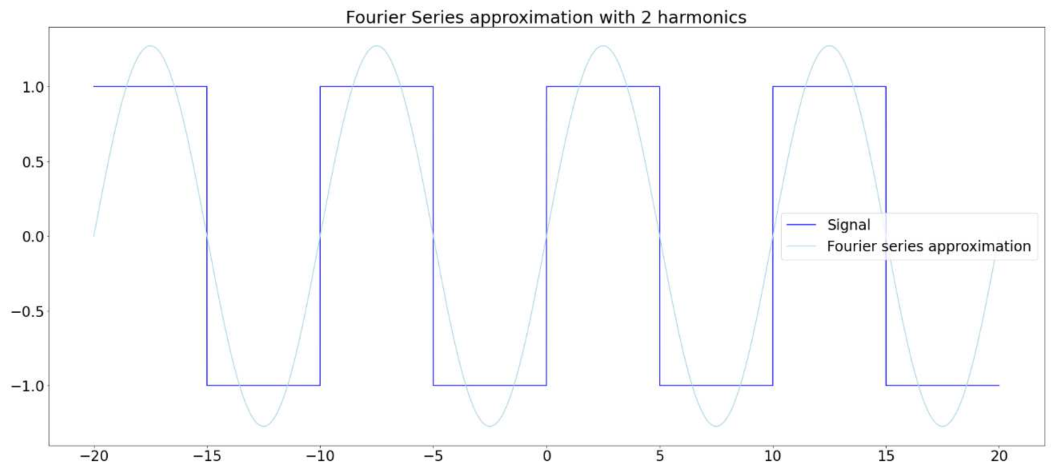


Figure 6

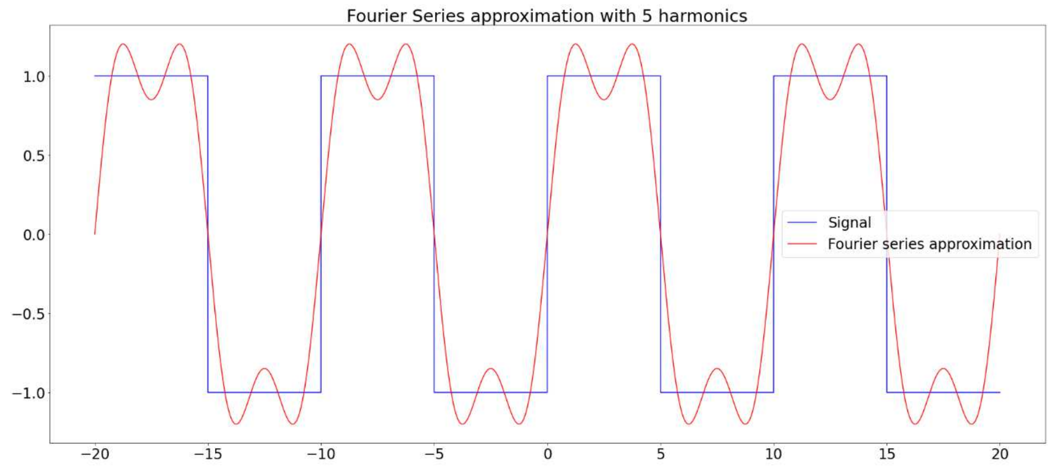


Figure 7

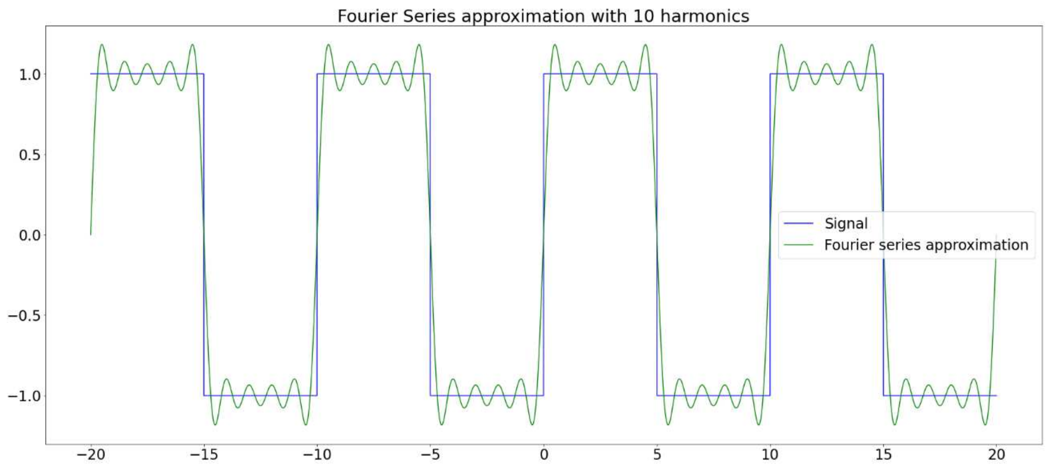


Figure 8

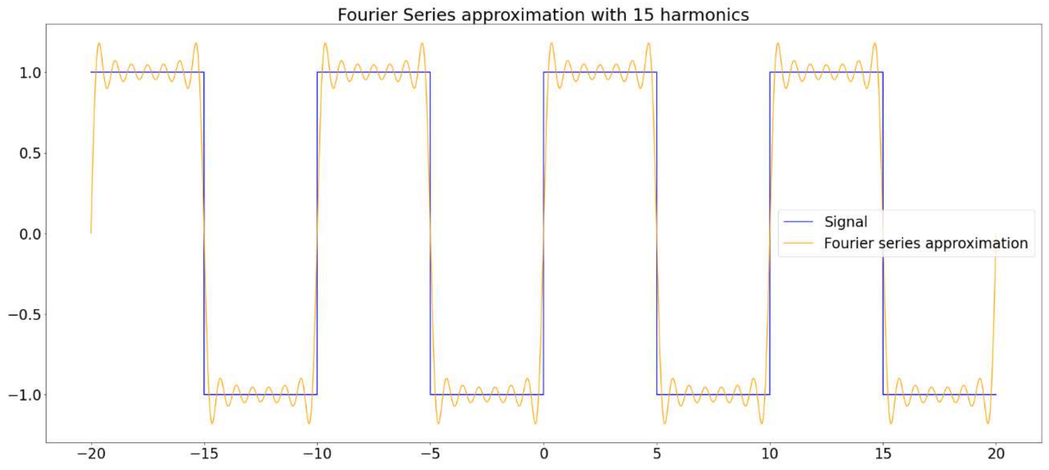
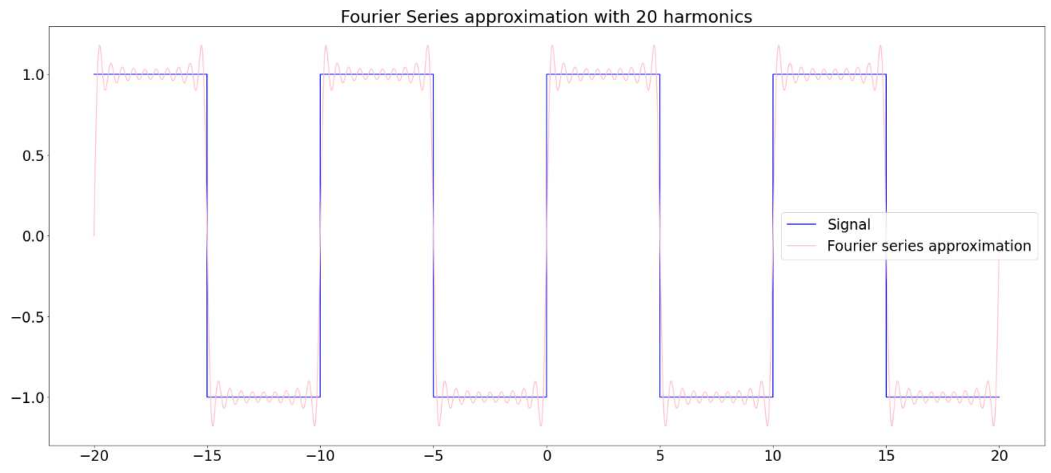


Figure 9



Figures 5, 6, 7, 8 and 9 represent the Fourier Series representations of a square wave function with an increasing number of harmonics.

6.3.5 Time trend functions

While Fourier series has been a widely used technique for capturing seasonality in time series data, it is by no means the only option. In fact, another method available involves the use of time trend functions.

Time trend functions are deterministic functions that allow to capture seasonal patterns in time series data. Unlike Fourier series, which decomposes a signal into a sum of sine and cosine waves of varying frequencies, time trend functions represent the pattern of the data over time as a simple linear function. These functions are typically used to model the trend of the data over time, but they can also be used to capture seasonal patterns.

The advantage of using time trend functions to capture seasonality is that they are relatively simple to implement and do not require as much computation as Fourier series. Additionally, time trend functions allow to capture the trend of the data over time, which can be useful in identifying long-term patterns in the data.

One popular method of implementing time trend functions is through the use of seasonal dummy variables. These variables are binary indicators that take a value of 1 during a specific season and 0 during all other seasons. By including seasonal dummy variables we can capture the seasonal patterns in the and estimate the effect of each season on the outcome of interest.

Overall, while Fourier series remains a popular method for capturing seasonality in time series data, the use of time trend functions provides an alternative approach that can be just as effective in capturing seasonal patterns. With the added benefit of being relatively simple to implement and providing insight into long-term trends, time trend functions should not be overlooked when attempting to model seasonal patterns.

Chapter 7

Results and Conclusion

In approaching the conclusion of this discussion let us briefly restate the goal of this work, review the key points and results of the analysis, and explain why it is relevant.

This thesis has allowed an in depth analysis of a proposed stochastic model for optimal investment in Renewable Energy Communities. Specifically, it focused on the variety of stochastic processes available and commonly used to model energy-related markets and communities, the choices among the variety of processes, and the model implementation.

The model is built on a particular type of REC composed of a "representative" household and a biogas producer, where the potential demand of the community is given by the household's demand, while both members produce renewable energy. The biogas producer invests in technology to convert biogas into electricity and sell it in the electricity market at the spot price, whereas the biogas that is not transformed into energy can be sold on the market at the gas spot price. The household invests in photovoltaic panels to reduce the energy purchased from the market in order to cover its own power demand. Moreover, investing in a renewable energy plant provides the household with the revenues of selling the excess of energy not used for self-consumption. The relevant advantage of entering into a REC for both players is that their joint self-consumption is rewarded with a governmental incentive, which must be fairly shared.

The stochastic processes analyzed in this work are some of the most commonly employed for economic and financial modelling, namely Geometric Brownian Motion, Ornstein-Uhlenbeck processes, and Lévy processes.

At the beginning of this work it was possible to expand the picture and have a detailed view of the broader framework in which the model lies. This includes an introductory part on Renewable Energy Communities, their definition, the regulatory framework and possible future developments, and a discussion on the global environmental problem, considering the role that Renewable Energy Communities can play in its relieve.

The last point mentioned is given a substantial weight as it is particularly pressing in our society not only inside the borders of the scientific community but also, and with great echo, in everyday chats and in political agendas. Moreover, the environmental problem is one of the main drivers for scientific research and innovation in the field of renewable energy.

At this point, after a careful and comprehensive description of the model, we dived into a detailed analysis on Geometric Brownian Motion and Ornstein-Uhlenbeck processes, explaining the reasons for preferring the latter in our scenario.

In the next chapter of this thesis we proceeded to implement the proposed model employing Ornstein-Uhlenbeck processes. Because of its versatility, open source tools and widespread use we chose as programming language `Python`. We computed integrals thanks to a numerical integration library and conducted an optimization procedure thanks again to a numerical optimization library.

Executing our code we obtained a suggested optimal investment of 312.13 kW for the "representative" household and 86.90 kW for the biogas producer, in the specific case proposed. Of course, being our model a stochastic model, each execution can produce slightly different results.

Later in this work we designed a couple of suggestions to be considered as a starting point for further research on the model. The first one is to implement the model using Lévy processes, which are stochastic processes with the capability to show jumps. The second one is the inclusion in the stochastic model of deterministic seasonality contributions. These two elements could help to capture some of the key features of energy-related markets just mentioned, namely the presence of jumps and seasonality.

We deeply hope that this work can have a significant and positive impact on our society, playing a role in the necessary transition to a new and sustainable way of producing and consuming energy. The years and decades to come will inevitably have to face challenges and overcome obstacles in the transition process. The positive outcome of this struggle also depends on the tools we are equipped with. So, the final goal of this thesis is exactly to give a contribution in providing these tools which are necessary for a successful transition.

Bibliography

[1] A. Awerkin, P. Falbo, C. Pelizzari, and T. Vargiolu. *Optimal Investment and Fair Sharing Rules of the Incentives for Renewable Energy Communities.*

[2] A. Awerking and T. Vargiolu. *Optimal installation of renewable electricity sources: the case of Italy.* Decisions in Economics and Finance, 2021, vol. 44, issue 2, No 30, pp. 1179-1209.

[3] Belk. *Fourier Series.* URL: https://e.math.cornell.edu/people/belk/measuretheory/Fourier%20Series.pdf.

[4] Fred Espen Benth, Jurate Saltyte Benth, and Steen Koekebakker. *Stochastic Modelling of Electicity and Related Markets.* World Scientific, 2008.

[5] Euroean Environmental Agency. *Environmental impact of energy.* URL: https://www.eea.europa.eu/help/glossary/eea-glossary/environmental-impact-of-energy.

[6] European Commission. *Energy communities.* URL: https://energy.ec.europa.eu/topics/markets-and-consumers/energy-communities_en.

[7] European Commission. *Renewable energy directive.* URL: https://energy.ec.europa.eu/topics/renewable-energy/renewable-energy-directive-targets-and-rules/renewable-energy-directive_en.

[8] International Energy Agency. *Emissions savings.* URL: https://www.iea.org/reports/multiple-benefits-of-energy-efficiency/emissions-savings.

[9] Steven P. Lalley. *Lévy Processes, Stable Processes, and Subordinators.* University of Chicago, Department of Statistics, USA. URL: https://galton.uchicago.edu/ lalley/Courses/385/LevyProcesses.pdf.

[10] Manuel Moreno, Alfonso Novales, and Federico Platania. *Long-term swings and seasonality in energy markets.* European Journal of Operational Research, 2019, 279, pp.1011 - 1023. URL: https://www.sciencedirect.com/science/article/pii/S0377221719304722.

[11] NASA. *What Is the Greenhouse Effect?* URL: https://climatekids.nasa.gov/greenhouse-effect/.

[12] NumPy. *NumPy Documentation.* URL: https://numpy.org/doc/.

[13] Python. *Python 3.11.2 documentation.* URL: https://docs.python.org/3/.

[14] Python. *random - Generate pseudo-random numbers.* URL: https://docs.python.org/3/library/random.html.

[15] SciPy. *minimize(method='SLSQP').* URL: https://docs.scipy.org/doc/scipy/reference/optimize.minimize-slsqp.html.

[16] SciPy. *Optimization (scipy.optimize).* URL: https://docs.scipy.org/doc/scipy/tutorial/optimize.html.

[17] SciPy. *SciPy documentation.* URL: https://docs.scipy.org/doc/scipy/.

[18] SciPy. *scipy.integrate.quad.* URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.quad.html.

[19] Steven Shreve. *Stochastic calculus for finance 2: Continuous-time models.* Springer, 2004.

[20] United Nations. *What Is Climate Change?* URL: https://www.un.org/en/climatechange/what-is-climate-change.

[21] Tiziano Vargiolu. *Course in Stochastic Differential Equations with Numerics at the University of Padova.*