Pablo Borao Fortún

Josep Fanals Batllori

SEMI-DECENTRALIZED BLOCKCHAIN-BASED OPTIMAL FRAMEWORK FOR DISTRIBUTION POWER SYSTEMS

Team Project

Electric Equipment

Master's degree in Energy Engineering

May 18, 2021

Contents

1 Introduction

During the last decades, power systems have evolved towards decentralized schemes with a larger contribution of renewables. This transformation has caused a reduction in the generation from traditional power plants, which in turn implies a decrease in the associated carbon footprint. However, this does not come freely. There are many challenges to face, such as controlling the upcoming bi-directional power flow, empowering prosumers, managing huge volumes of data, among others.

This work attempts to focus on two imperative issues. One of them has to do with the physical transmission of electricity, namely the power flow problem. As far as we are aware, the current iterative methods are not scalable, in the sense that the required computation time grows with the dimensions of the system. One dimension corresponds to the number of buses, while a second one could be related to the discretized time frames, another to the ranges of powers of both loads and generators, etc. Magnifying one dimension by a certain ratio causes the total computational effort to increase by that factor at least. This ever-growing difficulty is especially acute when solving the optimal power flow, as input data may take a wide range of values. To combat such complications, we implement the Proper Generalized Decomposition (PGD) methodology. In short, it allows to compute the solution to the multi-dimensional power flow much more efficiently, and thus, more insight can be gained from it in the same amount of time.

The second issue considered in this project has to do with enhancing prosumers' participation and placing more power on their hands as well. We propose a flow of information based on smart contracts in order to make the transition towards a semi-decentralized network. Prosumers' participation in generating and consuming power will be stimulated so as to reach a near-optimal state, previously computed with the PGD tool. Despite the attractiveness found in that idea, the transaction costs (the so-called gas fees) linked to the Ethereum blockchain are likely to be exceedingly high. Therefore, we deploy the smart contracts on a level 2 layer, such as Matic. This project allows transactions to become several orders of magnitude cheaper than what they typically are. The code will still be programmed in Solidity and it will establish communication with the Python code that solves the optimal operating point of the system.

This work is structured as follows: Chapter 2 describes the current state of the art related to the methodology to solve the optimal power flow as well as to blockchain and in particular to smart contracts; Chapter 3 presents the challenges addressed that are detailed further on; Chapters 4 and 5 constitute the main work of the project, as they focus on the smart contracts and the PGD respectively; Chapter 6 shows the practical implementation of the covered tools; Chapter 7 discusses the economical and business-related aspects, together with a potential integration into EDP's system; finally, Chapter 8 articulates the conclusions and suggestions for future work.

2 State of the art

This chapter covers the current status of the topics related to the project, i.e., blockchain and computational methods to solve the power flow problem. It presents a solid explanation regarding the history involved in both subjects with the goal of revealing challenges, which will are laid out in Chapter 3.

2.1 Blockchain

2.1.1 Technical aspects

Blockchain arrived in the world by the anonymous hand of Satoshi Nakamoto in its famous 2008 white paper [1]. It was and has been the underlying technology behind Bitcoin, a system meant to allow peer-to-peer cash transactions in a decentralized fashion. In a sense, blockchain is about concatenating data. Data are grouped in blocks and they are in turn linked to the previous sequence of blocks, thus forming a chain, as Figure 1 displays.

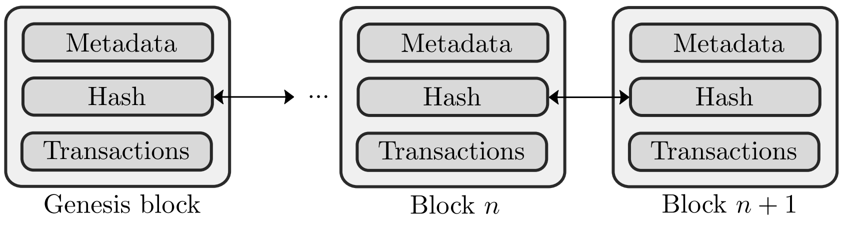

FIGURE 1. Blockchain schematic representation. Own elaboration, adapted from [2], [3].

Here metadata refers to the timestamp and the nonce. The former is an indicator of the date of creation associated to such block, whereas the latter stands for ”number only used once”. Adding a block to the chain is no easy task, as there is a lot of computational power spent in solving a cryptographic puzzle whose solution becomes the aforementioned nonce. Whoever solves first the problem is given a certain amount of Bitcoin. People, or rather, machines who compete in solving such problems are called miners.

Technically speaking this required mechanism to ensure the addition of blocks is commonly referred to as proof of work. It is based on computing the nonce, which results in a given hash obtained by means of a hash function, in this case, the SHA-256 [1], [4]. Simply put a hash is a string of characters with a fixed length responsible for identifying a certain amount of data [5]. In the Bitcoin network, an accepted hash has to respect some requisits. The difficulty of meeting them is adjusted so as to every 10 minutes approximately a new block joins the chain. Modifying past data on the blockchain only becomes feasible if 51% (or more precisely, the majority) of the computational power is owned by a single entity. This is why Bitcoin and in general blockchain has been thought of as a secure distributed system aimed at avoiding the double-spending problem [6] and immune against Byzantine faults [7], a common issue in distributed computer systems.

While Bitcoin was conceived to allow monetary transactions based on the blockchain concept, the truth is that this idea has a much wider application range. This is why many other blockchain projects have emerged during the last decade. Behind Bitcoin, the most relevant platform in terms of market capitalization is Ethereum [8]. Ethereum focuses on expanding the blockchain's usage by allowing the creation and deployment of applications, which not only permits trasferring money but also content, property, etc. All of such possible operations are still carried out in a decentralized manner and are practically impossible to corrupt. This is commonly called *code as law*, which stands by the fact that regulations are enforced via code, independently on human intervention [9].

The smart contracts vocable was coined back in 1994 by Nick Szabo as a way to encapsulate contractual terms into code [10], [11]. Ethereum is precisely a framework to develop smart contracts written in Solidity. Although it has become the most popular platform to do so, at the point of writing this the highly-anticipated Ethereum 2.0 has still not arrived. Contrarily to the traditional Ethereum network, Ethereum 2.0 is meant to allow substantially more transactions per second and especially at a lower cost [12]. In the technical argot, this is mentioned as reducing the gas fees. Since they are extremely expensive, we have decided to build smart contracts on the Matic mainnet, a layer 2 built on top of the Ethereum network [13]. It is visually depicted in Figure 2.

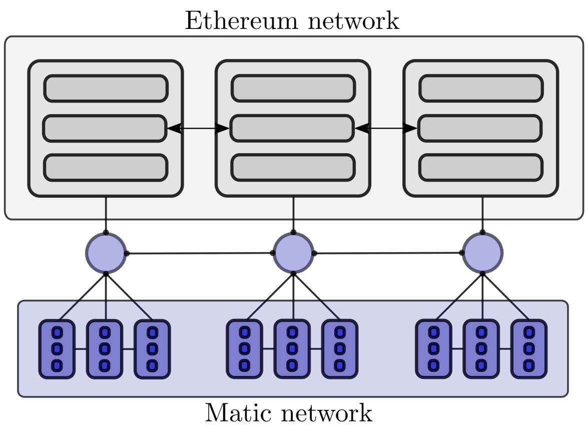

FIGURE 2. Scheme of the Matic layer in conjunction with the Ethereum newtork. Own elaboration, adapted from [13].

The Matic network is connected to the Ethereum network through the so-called Plasma Checkpoint Nodes, which act as a bridge between both networks. This way multiple blocks in the Matic network are embedded in a single Ethereum block. Contrarily to Bitcoin and Ethereum, Matic is based on the proof of stake consensus algorithm [13]. Miners - who are called validators in the proof of stake scheme - do not compete against each other; instead, the network selects a semi random node to validate the transaction [14], [15]. By following this approach, the Matic network becomes a fast low-cost scalable Ethereum extension.

2.1.2 Energy-related applications

Focusing the attention on the energy sector, the German Energy Agency states that the utilization of blockchain can improve the efficiency of the current energy companies' operations, as well as the peer-to-peer energy trading [16]. This is expected to find its applicability in the energy markets and propel smart grids, among others. The creation of smart contracts is meant to enhance prosumers' participation in the energy market, favoring a decentralized or at least semi-decentralized electric system which influences both generation and consumption. The ideas are still immature, as some projects have been proposed but lack practical implementation [17], [18]. Regulatory, economical, and technological issues are some of the current difficulties.

Overall we are still early in the adoption curve of blockchain in the context of energy, as it is visually depicted in Figure 3. Blockchain is expected to be first implemented as a tool to facilitate the integration of the electric vehicle [19], [20] and then proceed to constitute a fundamental technology to allow peer-to-peer energy trading in microgrids [17], [21].

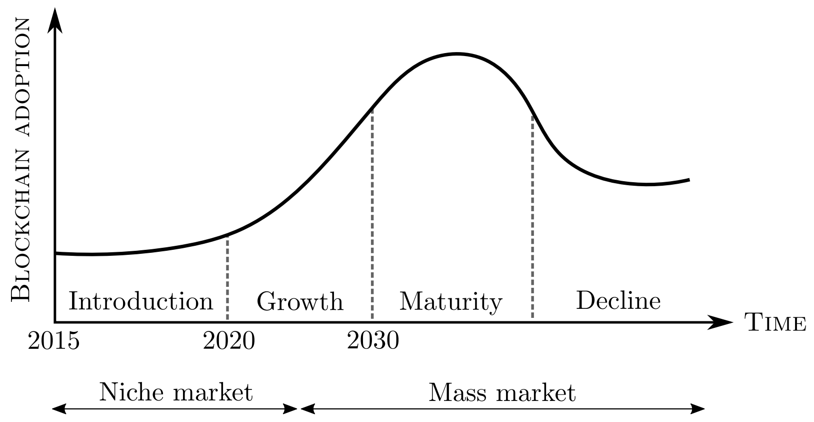

FIGURE 3. Representation of a hypothetical adoption curve of blockchain in the energy sector. Own elaboration, adapted from [22].

In fact, the literature mostly focuses on peer-to-peer energy trading. For instance, [23] propose a market design and transaction models to integrate blockchain at the distribution level; [24] present a framework to optimally trade energy in smart grids, considering both the day-ahead and the real-time market; [25] assess the impact of coalition formation on microgrids that make use of blockchain to reduce the greenhouse gas emissions; [26] exemplify the implementation of blockchain on Irish microgrids. Generally speaking, such projects have been carried out during the last lustrum, so in all likelihood, we are in the first stages of growth, if not earlier.

While peer-to-peer energy trading can be regarded as an application with significant affinity due to its decentralized nature and its incorruptibly-centered approach, we believe there are more urgent matters with lesser complexity. One would be to construct design rules with the goal of permitting transactions between prosumers and the DSO, instead of a purely peer-to-peer approach. This explains the semi-decentralized component of this work.

2.2 Optimal power flow methods

The optimal power flow (OPF) is defined as the determination of the most convenient operating point of a given system, all this while satisfying the constraints [27], [28]. Sometimes the objective functions could be related to minimizing the total generation cost, while in other situations it could focus on minimizing the active power losses, among other possibilities. In any case, the OPF constitutes an active research field that has evolved drastically since its inception back in 1962 by Carpentier [29]. Surveys regarding the classification of algorithms to solve the OPF problem often consider two segments: one has to do with the conventional approach, whereas a more novel borrows concepts such as artificial intelligence [28], [30]. These two categories are also called deterministic and non-deterministic, respectively.

Deterministic methods rely on relatively well-known mathematical tools that have formed the basis of classical programming. Some of them are the Linear Programming method (LP), which linearizes the objective function; the Nonlinear Programming method (NLP), capable of working with nonlinear objective functions and/or nonlinear constraints; the Quadratic Programming (QP) method, which stands as a variation of NLP; Newton's method, based on the Lagrangian and the Hessian matrix; and the Interior Point (IP) method, which was a highly-regarded method two decades ago [31], [32]. Due to the nonlinear and nonconvex behavior of the OPF, such deterministic methods are not always acceptable.

A new trend that has emerged is related to the usage of non-deterministic techniques. Instead of sticking to the equations that define the power flow, they employ heuristics. This is supposed to facilitate the obtention of an optimal solution for both large and small systems, converge rapidly, and overall be more robust than conventional approaches [28]. Most of these methods are inspired by natural phenomena, such as the Genetic Algorithm (GA), the Particle Swarm Optimization (PSO), the Graw Wolf Optimizer (GWO), the Differential Evolution (DE), etc. [33], [34].

The Proper Generalized Decomposition (PGD) that we introduce is a sort of hybrid method, a mix between the two categories. For the most part, it makes use of the power flow equations, but instead of solving the power flow problem each time, it decomposes the variables to speed the calculation process [35]. It is usually said to combat the curse of dimensionality: a situation where the complexity of the problem grows with an increase in the dimensions. Although it was not first conceptualized as a tool to solve the OPT problem, its characteristics make it suitable for doing so, especially in time-varying scenarios [36]. In this project, we dedicate a particular chapter to present such methodology.

3 Challenge identification

Concerning the state of the art, we have identified three ideas that remained either uncovered or that offer the possibility of being extended:

- The deployment of smart contracts with low transaction costs (also referred to as gas fees on the Ethereum network) would induce the participation of prosumers in a blockchain-based electric system.
- The utilization of an efficient methodology to solve the time series power flow. This is particularly imperative in the case of real-time markets.
- The combination of the two aforementioned ideas in a single program that combines the best of both worlds.

In more detail, smart contracts deployed on the Ethereum network imply highly variable transaction costs. For example, the fees can more than triple in a weekly time frame [37]. This would not be a problem if these transaction costs were insignificant in comparison to the cost of electricity. However, at the time of writing this, a single transaction is charged at around 100 gwei, which in monetary terms translates into 4 € approximately [38]. This exorbitant value would be unacceptable in a real-time market, not only because of the price fluctuations but because the transaction cost could exceed the price of the traded electricity. Matic, on the other hand, becomes much more suitable for real-time energy trading. The average time to form a block is close to 2 seconds (contrarily to 13 s for Ethereum), and the associated transaction cost is around 0.0001 € [39], [40].

Regarding the power flow, traditional and current methods solve it at every time step. It does not matter *a priori* if a load just changes slightly its value; to ensure the obtention of a satisfactory result, the power flow has to be computed again and again. From the outside, this seems a waste of computational effort because the structure of the system remains the same. Therefore, the question to formulate is: could we retain the general properties of the system to solve the system more efficiently? In a nutshell, the PGD formulation answers this issue. It generates a fast and precise enough solution to the power flow problem from where the optimal situation can be easily derived.

Finally, both the smart contracts and the PGD are implementable separately. Nevertheless, we consider that the most favorable outcomes lie in the intersection, i.e., smart contracts can benefit from an efficient calculation of the power flow problem and vice versa. As the Greek philosopher Aristotle once stated, "the whole is greater than the sum of its parts". We stand by this, and so our mission is to now show why this is the case.

4 Smart contracts framework

Electricity markets have revolved around the concept of a pool, an environment where buyers submit bids and sellers submit offers [41]. This is the typical structure of the day-ahead market, where most of the energy is traded. The increase in renewables justifies a transition towards a fast operating scheme since generation becomes rather unpredictable. While a near real-time intraday market tends to minimize the impact of this issue [42], in all these structures there is no place for empowering prosumers. One of the many fields of application of smart contracts is precisely in the energy sector; they could very likely motivate the active participation of prosumers.

For this reason, this chapter depicts the methodologies to integrate smart contracts with the electricity market known up to date. Then, the chosen option is explained in greater detail from a structural perspective but also regarding technical aspects related to the deployment of these smart contracts.

4.1 Smart contracts for energy transactions

The decentralized nature of blockchain has inspired the proposal of peer-to-peer (P2P) electricity trading schemes. As shown in Figure 4 participants trade electricity bilaterally according to their needs without a trusted third party. Aspects such as the amount of energy, the time frames, the pricing of the offer/bid have to be specified in the smart contract, to name a few. Figure 4 compares both options.

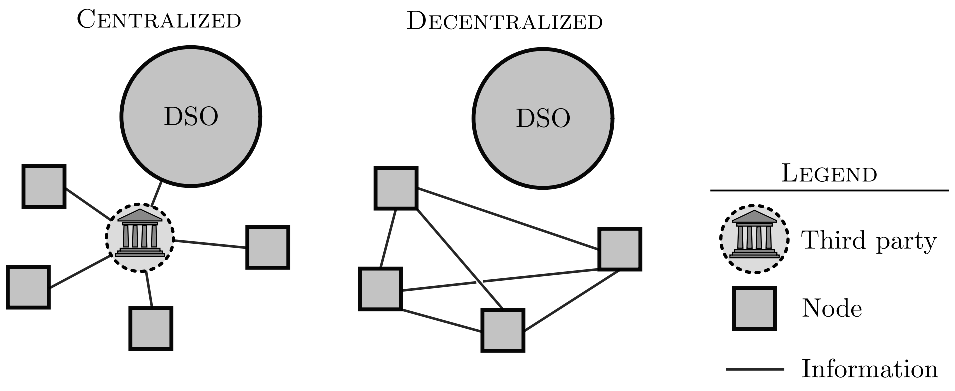

FIGURE 4. Schematic comparison between a centralized system and a decentralized peer-to-peer structure

On the contrary, the centralized scheme relies on an entity that agglutinates information, imposes a price for electricity as well as terms and conditions. This is in essence the retailer. Despite the fact that users (or nodes) can choose amongst many retailers in a de-regulated environment, they are still restricted to agree on prices. This aspect complicates the integration of renewables at the household level since prosumers inject power at a predefined price without any stimulus to support the grid. Therefore, it seems logical to think that the electricity grid will physically change to accommodate renewables and at the same time there will be an underlying layer managing the information.

The grid of the future is expected to be decentralized. For instance, the Spanish TSO Red Eléctrica conceptualizes a modern power system where distributed generation will be combined with the typical transmission and distribution systems; in addition, this complex network should be sustained by a data hub that employs blockchain technologies [43]. In that regard, blockchain is seen as a fundamental pillar for smart grids [44]–[46].

Some companies have started to implement blockchain into local electricity markets based on the P2P concept. Such is the case of PowerLedger, a precursor in the application of blockchain to empower prosumers, especially focused on the integration of photovoltaic generation. Even though it has its own ERC20 token, the deployment takes place in the Ethereum network [47], so the transactions are not precisely cheap. Lightency and Electrify are other examples of companies devoted to blockchain for P2P energy trading, although their solutions seem to be in the early stages [48], [49].

In short, current P2P decentralized schemes have two main issues. One is the fact that decentralized systems completely remove the dependency on utilities. The lack of interaction between a node and the grid becomes fruitless when trying to enhance communications in a smart grid environment. The second issue has to do with the deployment of the contracts on the Ethereum network. If electricity trading has to evolve towards a continuous market, paying several euros for a single transaction is unacceptable.

4.2 Chosen framework

As depicted in Figure 4, the information flows between peers or nodes in the network but does not interact directly with grid operators. Yet, transmission system operators (TSO) and distribution system operators (DSO) ought to interact in some shape or form with the bilateral trades of electricity. There are physical constraints such as ampacity limits and voltage safe-operating margins that can not be neglected. Although renewable distributed generation is still not predominant in today's system, a decision framework to determine the optimal generation of renewables could be of great interest to reduce the losses and make the prices more competitive. This justifies why we believe semi-decentralized systems have a bright future ahead of them.

The idea of the chosen semi-centralized framework is shown in Figure 5. In essence, the DSO behaves like another node in the blockchain network.

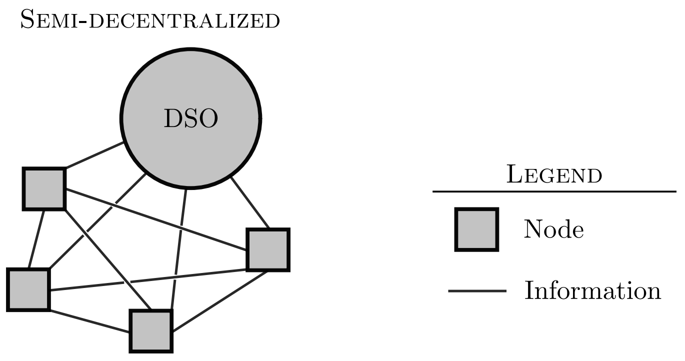

Figure 5. Schematic representation of the proposed semi-decentralized approach

A semi-centralized framework means that transactions will still take place between nodes (or prosumers) without a trusted third party, but there will be communication between the nodes and the DSO to motivate prosumers to operate close to the optimal point. This does not mean prosumers lose freedom of choice. They may still find it convenient to consume or generate a certain amount of power without caring about the status of the grid. However, if their act following the grid requirements, lower prices will have to be paid and they will obtain more revenue for generating energy.

On the other hand, the deployment of the contracts takes place on Matic's Mainnet. Matic is a layer 2 blockchain network. It is built on top of Ethereum, which means that every once in a while, some of its blocks that group multiple transactions are embedded into a block from the Ethereum network. Table 1 shows a comparison of the most notable aspects of each network[1].

TABLE 1. Comparison of attributes between Ethereum and Matic. Technical data about both networks was extracted from [38]–[40].

Characteristic	**Ethereum**	**Matic**
Consensus mechanism	Proof of work	Proof of stake
Average block time (s)	13.1	2.1
Scalable	Yes	Yes
Cost per transaction ($)	3.40	0.0001
Transactions per day	$1.43 \cdot 10^6$	$1.53 \cdot 10^6$
Total transactions	$1113 \cdot 10^6$	$31 \cdot 10^6$
Market capitalization (M$)	$342 \cdot 10^3$	$4.12 \cdot 10^3$

Both options are scalable compared to Bitcoin, in the sense that they have the capability of handling a large volume of transactions. Despite Matic being a much more immature network, two characteristics make it stand above: its average block time and the cost per transaction. A lower average block time means that Matic is more suitable to react fast to new setpoints. The cost per transaction is the main issue with Ethereum; even though the Berlin hard fork, aimed at reducing gas fees, has been implemented at the time of writing this [50], Matic's costs are still several orders of magnitude lower.

4.3 Algorithm

A generic network is formed by nodes, which make proposals depending on their needs, capabilities, and preferences. They specify a feasible range of powers, and with each power, there is an associated price. This is the amount of money they are willing to pay or earn for their consumption/generation. A smart contract is built considering all these data. Then, the DSO looks for the optimal solution and determines the incentives it is prone to give to each node. Finally, the nodes choose which offer they shall pick. Figure 6 visually shows the high-level algorithm particularized for a node i.

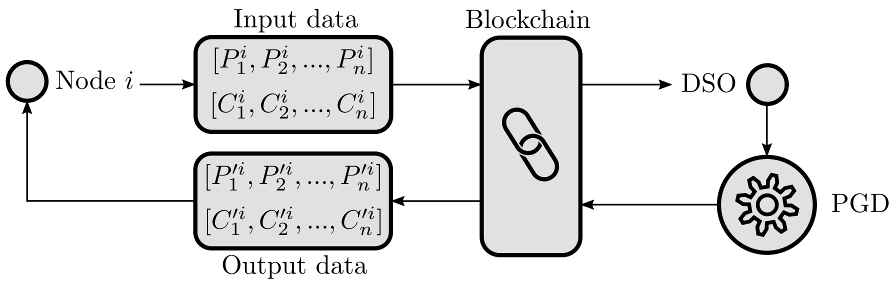
Input data
Blockchain
Node i
$[P_1^i, P_2^i, ..., P_n^i]$
$[C_1^i, C_2^i, ..., C_n^i]$
DSO
$[P_1'^i, P_2'^i, ..., P_n'^i]$
$[C_1'^i, C_2'^i, ..., C_n'^i]$
PGD
Output data

FIGURE 6. High-level scheme to trade electricity

A given node specifies a discretized range of powers with the associated range of prices. This input object depends on the decisions of the node. It can choose to offer a wide range of powers if it can provide flexibility, or on the contrary, it may be interested in just consuming a fixed amount of power, for instance. The data are inserted into a smart contract with the DSO. With that information, the DSO runs the PGD to look for an optimal solution.

[1]Numerical values are highly variable across time. Data gathered here were representative for the 3rd of May of 2021.

Notice that the optimality of an operation point is an arbitrary concept. The company may prioritize low power losses, voltages close to the nominal value, or it could also try to maximize power from renewable sources. The decision is entirely up to the DSO, and it does not have to be static, that is, it could change across time. In any case, the DSO calls the smart contract and then writes on it a modified range of powers with also modified prices. The new ranges of power ought to be constrained to the limits of the initial powers, if not more. Finally, the i node picks a power $P_k'^i$ with the associated cost $C_k'^i$. Again, the node is free to select among all possibilities. It will depend on its preferences. One option would be to choose the cheapest alternative.

Here a node can represent either a single user, a neighborhood, a community, etc. There are no restrictions in this case, although nodes could be capable of offering more flexibility if they were aggregated. This general formulation of the problem is likely to empower them to vary powers in order to participate in demand response schemes and motivate the generation from renewable sources.

5 Proper Generalized Decomposition

The Proper Generalized Decomposition (PGD) is a technique to efficiently solve a multidimensional problem. It relies on progressive enrichments performed not on each individual case of the problem, but on the structure of the problem as a whole [36], [51]. The outcome of the PGD becomes a potentially well-refined solution for various scenarios that have not been computed directly. Indeed, this comes with an associated saving of time.

Backed by its mathematical proof, this chapter presents the notation, the definitions of the objects, and more importantly, a practical algorithm overview.

5.1 Derivation

As defined in [52], the power flow problem is mainly concerned with four magnitudes: voltages denoted by $\boldsymbol{V}$, currents represented by $\boldsymbol{I}$, complex powers given by $\boldsymbol{S}$ and the bus admittance matrix $\boldsymbol{Y}$. The usage of bold variables indicates that these are multidimensional objects.

The first novelty of the PGD has to do with their tensorial representation. This arises from the crossing of multiple vectors, each describing a dimension of the form

$$\boldsymbol{Z}(\boldsymbol{x}, \boldsymbol{t}, \boldsymbol{p_1}, \boldsymbol{p_2}, ..., \boldsymbol{p_D}) = \sum_{i=1}^{n} X(\boldsymbol{x}) \otimes T(\boldsymbol{t}) \otimes P_1(\boldsymbol{p_1}) \otimes P_2(\boldsymbol{p_2}) \otimes ... \otimes P_D(\boldsymbol{p_D}) \ , \tag{1}$$

where $\boldsymbol{Z}$ represents either $\boldsymbol{V}$, $\boldsymbol{I}$ or $\boldsymbol{S}$. Note the dependence on position ($\boldsymbol{x}$), time ($\boldsymbol{t}$) and the parameters meant to change ($\boldsymbol{p_1}$ to $\boldsymbol{p_D}$). In the power flow problem, $\boldsymbol{x}$ stands for the indices attributed to the buses while the parameters can be for instance variations in power in several buses. Changes in impedance could also be parametrized, although it is not clear how the PGD should be adapted to it. Notice also that the final tensor is the result of the sum of multiple tensors with the same dimensions. One tensor could represent the stationary power we predict, while another one could be the variations we introduce, for instance.

Recall that by definition solving the power flow means solving

$$\boldsymbol{Y}\boldsymbol{V} = \frac{\boldsymbol{S^*}}{\boldsymbol{V^*}} \ , \tag{2}$$

for $\boldsymbol{V}$. Considering the presence of a slack bus, where the voltage is already known and its current injection to the rest of the buses is symbolized by $\boldsymbol{I_0}$, Equation 2 becomes

$$\boldsymbol{Y}\boldsymbol{V} = \boldsymbol{I_0} + \frac{\boldsymbol{S^*}}{\boldsymbol{V^*}} \ . \tag{3}$$

which translates into the following in compact for

$$\boldsymbol{V} = \boldsymbol{Y^{-1}}(\boldsymbol{I_0} + \boldsymbol{S^{*}} \oslash \boldsymbol{V}) \ , \tag{4}$$

where $\oslash$ denotes the Hadamard division, that is, the element-wise division. As described in [52], the power flow problem can be solved with the so-called alternating search directions (ASD) method. This has the advantage of combining quite adequately with the PGD methodology due to its speed, but mainly, because of the linear relationship between voltages and currents during one of the two steps in which the PGD is divided.

The two steps we are referring to are first of all the non-linear connection between voltages and currents

$$\boldsymbol{I} = \boldsymbol{S^*} \oslash \boldsymbol{V^{*[\gamma]}} \ . \tag{5}$$

where γ indicates the outer loop iteration number. Here the variables are in fact tensors. Note that the traditional way to solve Equation 5 would be to compute all the cases, one by one. However, this is not beneficial and it is not the point of the PGD. In fact, the PGD circumvents this problem.

The second step to perform is described by

$$\boldsymbol{V}^{[\gamma+1]} = \boldsymbol{Y}^{-1}(\boldsymbol{I} + \boldsymbol{I_0}) \ . \tag{6}$$

The procedure consists of first computing $\boldsymbol{I}$ from Equation 5 using the PGD method. Once this is done, we proceed to update $\boldsymbol{V}$. In the end, it follows the same structure as $\boldsymbol{I}$ because they are related by a linear transformation, which saves computation time.

The complexity of solving this problem lies in Equation 5. The second step given by Equation 6 is direct. Recall that the tensor $\boldsymbol{Z}$ shown in Equation 1 represents in a general form $\boldsymbol{S}$, $\boldsymbol{V}$ and $\boldsymbol{I}$. Hence, we define $\boldsymbol{u^T V^* I} = \boldsymbol{u^T S^*}$, where the superscript T stands for the transposed operation. The redefinition of $\boldsymbol{I}$ is

$$\boldsymbol{I} = \sum_{i=1}^{n} \boldsymbol{i_1} \otimes \boldsymbol{i_2} \otimes ... \otimes \boldsymbol{i_D} + \boldsymbol{R_1} \otimes ... \otimes \boldsymbol{R_D} \ . \tag{7}$$

The so-called test field follows

$$\boldsymbol{u} = \boldsymbol{R_1^*} \otimes \boldsymbol{R_2} \otimes ... \otimes \boldsymbol{R_D} + ... + \boldsymbol{R_1} \otimes \boldsymbol{R_2} \otimes ... \otimes \boldsymbol{R_D^*} \ . \tag{8}$$

This implies the following, in extended form:

$$\begin{aligned} &\sum_{i=1}^{N_v}\sum_{j=1}^{n} \boldsymbol{R_1^{T*} V_1^{i*} i_1^j} \times ... \times \boldsymbol{R_D^T V_D^{i*} i_D^j} + ... + \sum_{i=1}^{N_v}\sum_{j=1}^{n} \boldsymbol{R_1^T V_1^{i*} i_1^j} \times ... \times \boldsymbol{R_D^{T*} V_D^{i*} i_D^j} \\ &\quad + \sum_{i=1}^{N_v} \boldsymbol{R_1^{T*} V_1^{i*} R_1} \times ... \times \boldsymbol{R_D^T V_D^{i*} R_D} + ... + \sum_{i=1}^{N_v} \boldsymbol{R_1^T V_1^{i*} R_1} \times ... \times \boldsymbol{R_D^{T*} V_D^{i*} R_D} = \\ &\quad \sum_{i=1}^{N_s} \left(\boldsymbol{R_1^{T*} S_1^{i*}} \times ... \times \boldsymbol{R_D^T S_D^{i*}} + ... + \boldsymbol{R_1^T S_1^{i*}} \times ... \times \boldsymbol{R_D^{T*} S_D^{i*}} \right) \ . \end{aligned} \tag{9}$$

Which can be compacted by defining

$$\sum_{i=1}^{N_c} \boldsymbol{C_1^i} \otimes ... \otimes \boldsymbol{C_D^i} = \sum_{i=1}^{N_s} \boldsymbol{S_1^{i*}} \otimes ... \otimes \boldsymbol{S_D^{i*}} - \sum_{i=1}^{N_v}\sum_{j=1}^{n} \boldsymbol{V_1^{i*} i_1^j} \otimes ... \otimes \boldsymbol{V_D^{i*} i_D^j}, \tag{10}$$

where $N_c = N_s + N_v n$. This way, the problem described by Equation 9 takes the form of

$$\begin{aligned} &\sum_{i=1}^{N_v} \boldsymbol{R_1^{T*} V_1^{i*} R_1} \times ... \times \boldsymbol{R_D^T V_D^{i*} R_D} + ... + \sum_{i=1}^{N_v} \boldsymbol{R_1^T V_1^{i*} R_1} \times ... \times \boldsymbol{R_D^{T*} V_D^{i*} R_D} = \\ &\quad \left(\boldsymbol{R_1^{T*}} \otimes ... \otimes \boldsymbol{R_D^T} + ... + \boldsymbol{R_1^T} \otimes ... \otimes \boldsymbol{R_D^{T*}} \right) \sum_{i=1}^{N_c} \boldsymbol{C_1^i} \otimes ... \otimes \boldsymbol{C_D^i} \ . \end{aligned} \tag{11}$$

At this stage, a greedy algorithm is implemented [53]. This technique serves the purpose to find the multiple $\boldsymbol{R_i^{[\Gamma]}}$ for $i = 1, ..., D$, where Γ represents the iteration number of this inner loop. This way, there are three loops. The outer one is the ASD as such, that is, the algorithm responsible for solving the power flow problem. The intermediate one deals with finding the residues at every superposition of the $\boldsymbol{I}$ tensor. To do so, the last and most inner loop is the one that goes over the several Γ indices to iterate and find convenient residues. This can be summarized by:

- Outer loop: iterate on γ to solve the power flow as such.

- Intermediate loop: iterate on i to find the superposition of terms of the $\boldsymbol{I}$ tensor.
- Inner loop: iterate on Γ to find the residues $\boldsymbol{R_k^\Gamma} \forall k \in [1, ..., D]$.

Finally, the residues of the form $\boldsymbol{R_k^\Gamma}$ are computed with

$$\left(\sum_{i=1}^{N_v} \boldsymbol{V_k^{i*}} \prod_{j \neq k}^{D} \boldsymbol{R_j^T V_j^{i*} R_j} \right) \boldsymbol{R_k} = \sum_{i=1}^{N_c} \boldsymbol{C_k^i} \prod_{j \neq k}^{D} \boldsymbol{R_j^T C_j^i} \ . \tag{12}$$

Once this is done for all residuals k, an inner iteration Γ has finished. Note that the superscript Γ has been avoided in Equation 12 to alleviate the notation. But the point is that $\boldsymbol{R_k}$ is computed from the previously known residues, which are also going to be updated. In essence, this replicates the Gauss-Seidel method. It is recommended to iterate several times in this inner loop to refine the solution. Usually, around 10 to 20 times is more than enough. The same applies to the intermediate loop, where the number of iterations can change at each step according to the necessities. That is, the parameter M is arbitrary and can change from time to time [52]. Once $\boldsymbol{I}$ is fully built, the voltages are updated and the iteration procedure continues until a satisfactory solution has been generated.

5.2 Application

The steps involved in the PGD are schematically shown in Algorithm 1 for a generic case with arbitrary parameters.

Algorithm 1 Pseudocode for the PGD combined with the ASD

```
1: for $\gamma = 1$ to $N_\gamma$ do
2:     Calculate power side of the problem with PGD: $\boldsymbol{I} = \boldsymbol{S^*} \oslash \boldsymbol{V}^{*[\gamma]}$
3:     for $i = 1$ to $n$ do
4:         Define $\boldsymbol{I} = \sum_{i=1}^{n-1} \boldsymbol{P_1^{(i)}} \otimes \boldsymbol{P_2^{(i)}} \otimes ... \otimes \boldsymbol{P_D^{(i)}} + \boldsymbol{P_1^{(n)}} \otimes \boldsymbol{P_2^{(n)}} \otimes ... \otimes \boldsymbol{P_D^{(n)}}$
5:         for $\Gamma = 1$ to $N_\Gamma$ do
6:             Compute $\boldsymbol{P_1^{(n)[\Gamma+1]}}$ with $\boldsymbol{P_2^{(n)[\Gamma]}}$,...,$\boldsymbol{P_D^{(n)[\Gamma]}}$.
7:             Compute $\boldsymbol{P_2^{(n)[\Gamma+1]}}$ with $\boldsymbol{P_1^{(n)[\Gamma+1]}}$ and $\boldsymbol{P_3^{(n)[\Gamma]}}$,...,$\boldsymbol{P_D^{(n)[\Gamma]}}$.
8:             $\vdots$
9:             Compute $\boldsymbol{P_D^{(n)[\Gamma+1]}}$ with $\boldsymbol{P_1^{(n)[\Gamma+1]}}$,...,$\boldsymbol{P_{D-1}^{(n)[\Gamma+1]}}$.
10:        end for
11:    end for
12:    Calculate admittances side of the problem directly: $\boldsymbol{V}^{[\gamma+1]} = \boldsymbol{Y}^{-1}(\boldsymbol{I} + \boldsymbol{I_0})$.
13: end for
```

Solving the power flow for each variation in the parameters with the classical Newton-Raphson is without a doubt more straightforward than following the procedure summarized in Algorithm 1. The price to pay for this simplicity is in the time required, which in computational terms is usually called complexity. The Newton-Raphson involves at least one inversion of the Jacobian matrix, and usually between three to five in order to obtain results with appropriate precision. It is a well-known fact that the complexity of a $n \times n$ inverse matrix increases exponentially with its size [54], and in principle, it is $\mathcal{O}(n^3)$. Therefore, having to compute the inverse so many times in power flow problems where parameters change slightly becomes an arduous task.

On the contrary, the PGD becomes a much more efficient methodology. There are no inverse matrices involved (apart from the impedance matrix, which can be precomputed) and it is able to decouple the dimensions of the problem. Take for instance a system where three parameters p_1, p_2 and p_3 vary in ranges of n_1, n_2 and n_3 values. If the total number of non-slack buses in the system is n_b, the amount of calculations N_u turns out to be

$$N_u \propto M(n_b + n_1 + n_2 + n_3) \ , \tag{13}$$

where M is the so-called number of modes and corresponds to the number of iterations performed in the intermediate loop. Instead, a Newton-Raphson-based algorithm would require to compute N_r unknowns, where

$$N_r \propto n_b n_1 n_2 n_3 \ . \tag{14}$$

When the dimensions of the system tend to large values, the PGD becomes considerably faster than the Newton-Raphson. For example, [36] reported solving in a matter of minutes a time-series power flow which took a couple of hours with the Newton-Raphson method.

Leaving aside the mathematical formulation, we introduce a basic example to illustrate the dimensionally of the power flow problem in a renewable penetration scenario. Figure 7 shows how the total power P changes according to the demand of power $P_d(t)$ and the power generated from hypothetical windmills, denoted by $P_g(v_w)$. Thus, the variables involved are the time t and the wind speed v_w. The total power has been defined as $P = P_d(t) - P_g(v_w)$, which can be understood as a net balance of the demand, and for convenience it is expressed as an adimensional quantity.

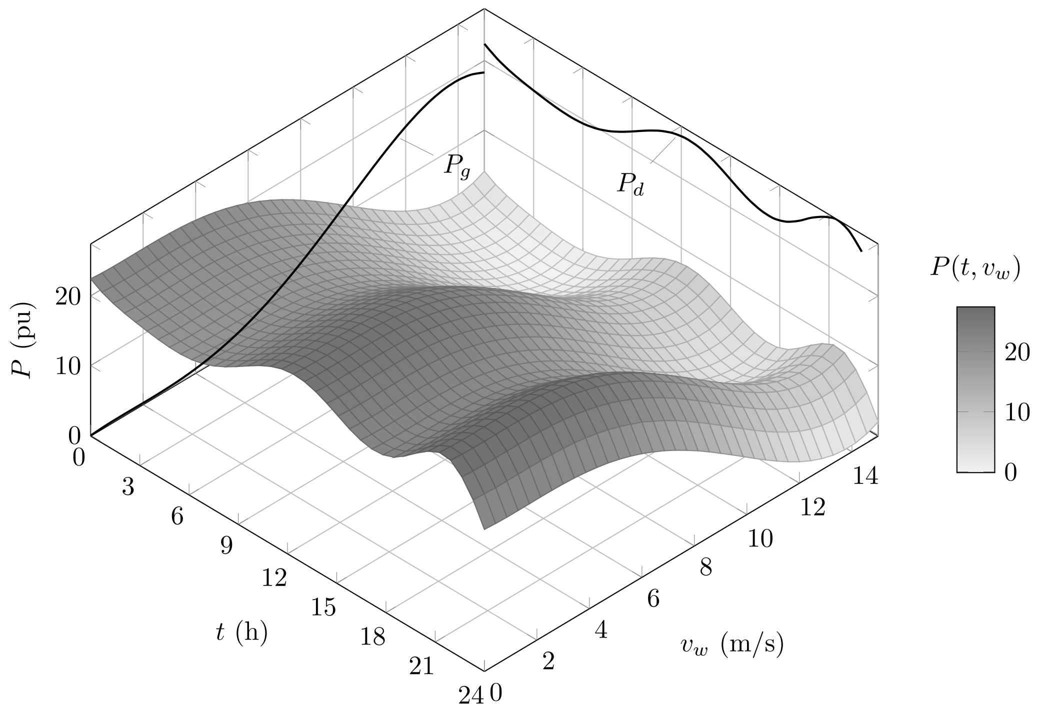

FIGURE 7. Evolution of the net power depending on the time of consumption and on the wind speed. Data for the demand curve are extracted from Red Eléctrica de España [55].

Considering the combinations of power is key if all the grid states have to be assessed. On the one hand the demand can vary in its typical form, with peaks around mid-day and the evening, and a valley at night. The power generated by a wind power plant is assumed to first follow a cubic evolution and then stabilizes around the nominal power. No cut-out speed has been represented for convenience purposes. In any case, wind can blow at any speed independently of the time of the day. Thus, if all possible operating states have to be studied, one needs to solve the system for each combination.

There are only two parameters in the situation considered in Figure 7, that is, time t and wind speed v_w. It is not hard to imagine how an electric system may have more than two parameters. For instance, solar irradiance could also be taken into account, and in addition to that, various places may have different natural resources at the same point in time. So in that regard, conventional power flow strategies would have to compute thousands and thousands of cases. This is of course a non-scalable way to attack the problem. Because of that, we have introduced the PGD as a way to efficiently solve the multidimensional power flow.

6 Case study

We have selected a partially made-up distribution system, located in the region of Baixo Alentejo, to further test the implementation of both the PGD and the smart contracts. Figure 8 displays the geographical position of the 13 *concelhos* forming Baixo Alentejo along with the interconnections.

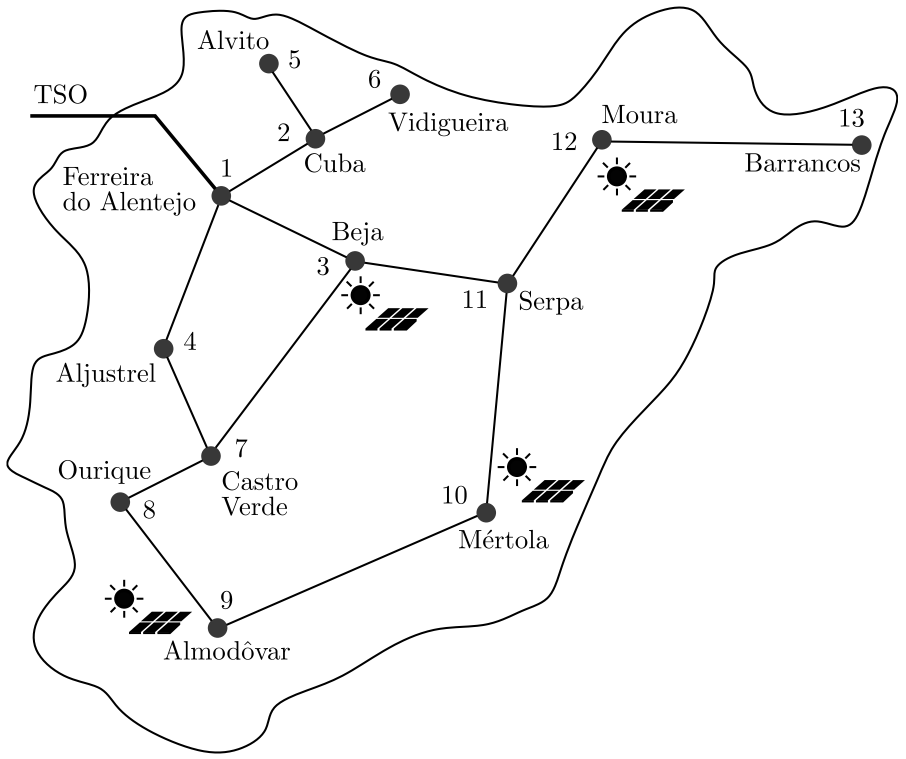

FIGURE 8. Schematic representation of the made up distribution system in Baixo Alentejo. Own elaboration. Data of the map extracted from [56].

We assume the presence of four distributed photovoltaic (PV) power plants. According to ENTSO-E, the transmission system is connected to Ferreira do Alentejo [57]. This will act as the slack bus of the distribution system, while the rest of the buses are load buses (also called PQ buses). Their demand is proportional to the population, whose data is extracted from [58]. Table 2 presents the powers' information.

TABLE 2. Power data for the demand and the PV generation

Bus	Town	Peak demand (MW)	PV installed (MW)
1	Ferreira do Alentejo	5.45	-
2	Cuba	3.22	-
3	Beja	23.66	30.00
4	Aljustrel	6.11	-
5	Alvito	1.65	-
6	Vidigueira	3.92	-
7	Castro Verde	4.80	-
8	Ourique	3.55	-
9	Almodôvar	4.92	9.81
10	Mértola	4.80	8.85
11	Serpa	10.31	-
12	Moura	10.01	13.44
13	Barrancos	1.21	-

To work with non-null reactive power demands, a power factor $\cos\phi = 0.9$ has been assumed. No current limitations are imposed to keep the problem simple enough, although the PGD is flexible enough to consider them. On the other hand, the presence of PV power plants does not imply the exclusion of PV panels installed at the residential level. However, as they are supposed to make up a tiny percentage of the total, they can be modeled as variations in demand and not purely generation sources.

The installed PV capacity has been chosen to surpass the local peak demand. While at current times it may be hard to imagine, if renewables have to cover a large chunk of the electricity mix, this may not be far from the truth. The second reason regarding this decision is a didactic one. If PV generation were to be too small, all of its power would be consumed by the load in the bus where they are both connected. Contrarily, if PV power takes large values, decisions have to be made to prioritize power coming from Ferreiro do Alentejo (our slack bus) or from the installed PV panels.

This way, the complex power is mathematically speaking decomposed as

$$\boldsymbol{S}(\boldsymbol{x}, \boldsymbol{t}, \boldsymbol{p_f}) = X(\boldsymbol{x}) \otimes T(\boldsymbol{t}) \otimes S_d + X(\boldsymbol{x}) \otimes T(\boldsymbol{t}) \otimes P_g(\boldsymbol{p_f}) \ , \tag{15}$$

where X represents the position dimension, T stands for the time dimension, S_d is the known power consumption and P_g corresponds to the active power generation from renewables. Note that P_g depends on $\boldsymbol{p_f} \in [0, 1]$. This is an array of scalars whose job is to account for variations in the PV generation.

The demand and the PV power generation over time have profiles like the ones depicted in Figure 9. They have been normalized to per-unit values. The PGD considers 3 dimensions. In this case, as shown in Equation 15, they are space, time, and variation in renewable powers. We have assumed demand is constant, although a fourth dimension could be included to parametrize it. PV generation can take any value under the bell. While each PV generation plant could be decoupled from every other, this would imply the usage of a total of 6 dimensions. To present a simple enough and didactic problem, we have assumed all PV generation sources are scaled by the same factor.

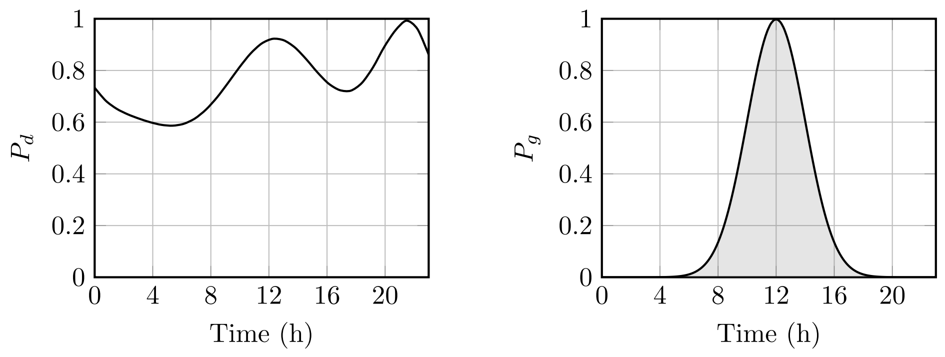

FIGURE 9. Power consumed P_d and power generated from PV P_g to exemplify the profiles depending on time

Once the PGD has computed the multiple power flow cases, some decision rules are necessary to discern which one becomes the preferable scenario. The range of possibilities is near infinite. However, this study focuses on minimizing the losses, or in other words, on maximizing the efficiency, which we will denote by η. From an economical standpoint this may be the most sound option.

6.1 Results

The main results of the power flow simulation with the PGD are shown in Table 3, with time frames of one hour. The range of possible PV values has been discretized in 20 uniform intervals.

TABLE 3. Optimal results from the PGD at each time step

Time (h)	**PV generation (%)**	η **(%)**	**Losses (kW)**	**Optimal** p_f
0	0.00	98.50	870.86	1.00
1	0.00	98.62	731.25	1.00
2	0.00	98.69	658.46	1.00
3	0.00	98.74	612.50	1.00
4	0.00	98.77	577.39	1.00
5	0.29	98.80	555.12	1.00
6	1.49	98.81	550.77	1.00
7	5.90	98.80	552.88	1.00
8	18.90	98.83	519.63	1.00
9	52.97	98.90	414.38	1.00
10	111.25	98.88	338.45	0.90
11	102.24	98.79	418.48	0.65
12	102.04	98.74	457.33	0.60
13	110.17	98.75	434.79	0.70
14	106.84	98.80	400.29	0.95
15	45.55	98.76	546.01	1.00
16	16.36	98.65	688.36	1.00
17	5.01	98.59	763.22	1.00
18	1.19	98.52	849.41	1.00
19	0.21	98.37	1025.05	1.00
20	0.02	98.18	1300.46	1.00
21	0.00	98.01	1560.85	1.00
22	0.00	98.00	1563.25	1.00
23	0.00	98.26	1174.10	1.00

The results indicate one of the main challenges PV generation has to face, which is that its maximum available power does not coincide in time with the peak demand. During sun peak hours, there is an excess of power from PV. Most of it is consumed by at the distribution grid, and a tiny part flow towards the transmission grid. This has been found to be the optimal choice to minimize losses, and not to inject the maximum available power. Notice that the losses decrease considerably around 12 o'clock. The power consumption is near the peak value during this time, but the injection of distributed power allows to operate with only a 25% of the losses that occur at night.

Opting to solve the OPF problem instead of setting the renewable sources at its maximum possible production has its benefits. For instance, at 12 o'clock the efficiency would have been 97.61% instead of 98.74%. Thus, the PGD has automatically determined that PV should not work at its full power during peak hours in order to minimize the losses. Another benefit has to do with the computational effort. Running on an AMD Ryzen 5 3600 6-Core at 3.60 GHz, the mean total time to solve all cases has been 1899 ms. Not only the PGD is a fast methodology, but it is also a scalable approach to fight against the curse of dimensionality. For instance, computing 10 times more cases has resulted in a mean total computational time of 6347 ms, which compared to the aforementioned 1899 ms, supposes just an increase by a factor of 3.34. This suggests that the algorithm is extremely well suited for parametric power flows where the operator decides to study diverse cases with variations in the input parameters.

Regarding the smart contracts deployed in the blockchain, each participant allowing the possibility of changing its power output between a range of values writes such values in a contract. Figure 10 depicts the deployed smart contracts, exemplified at 12 o'clock.

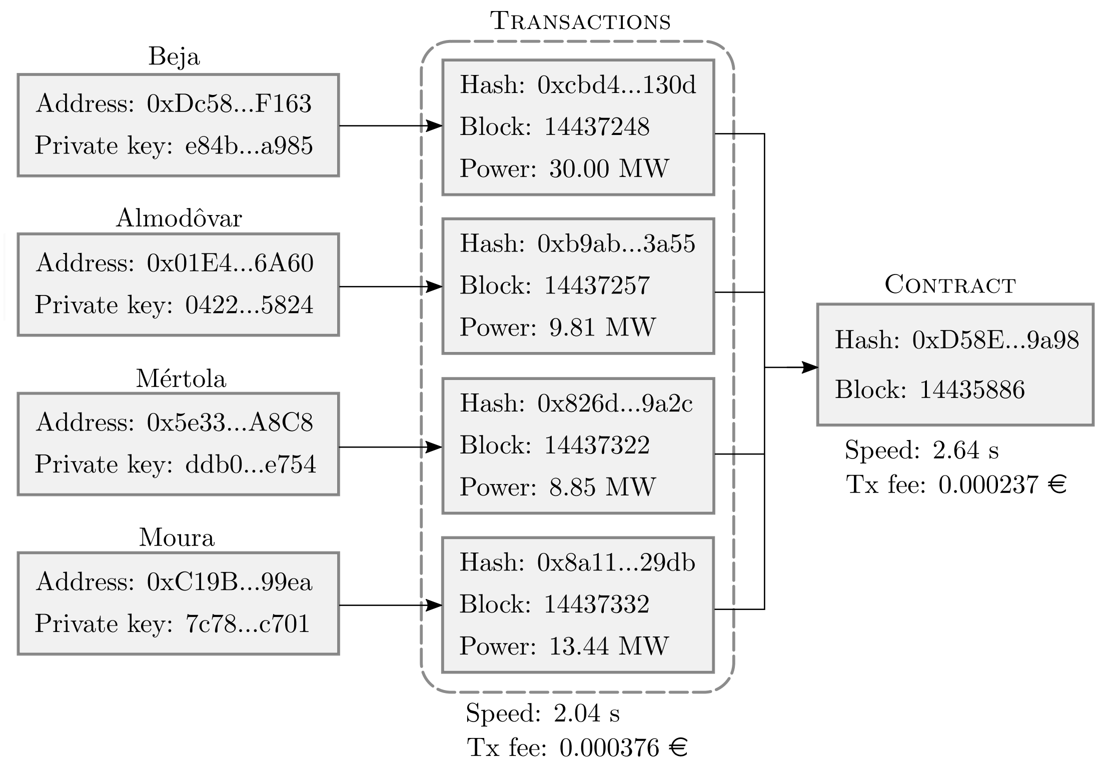

Figure 10. High-level representation of the flow of information regarding smart contracts for the test case at 12 o'clock

Each node stands for a *concelho* with installed PV. In the blockchain, they are identified by their public adress. There is also a private key that gives them access to transact, although as the name suggests, this is invisible to other nodes. These nodes are responsible for calling the contract. In this particular situation, they specify a maximum power, which is kept in the blockchain forever. This can be thought of as a transaction between the node and the contract as such. Therefore, each one of them has a identifier (hash) and belongs to a certain block. Prices have been set at 0, since otherwise this power would remain unused.

The transaction fees are extremely low, at least when compared to Ethereum. A total of 0.000376 € was paid for the four transactions. The creation of the smart contract, which is in fact a previous step, required a payment of 0.000237 €. These low fees combined with the fast timings (in the order of 2 seconds), make Matic a viable alternative as a blockchain network to build a system w[illegible] power is negotiated among nodes. A near-real time operation may also be feasible, although [illegible]s is not the goal of the present project.

One of the advantages of working on the blockchain consists of its transperancy. In other words, it becomes possible to retrieve all the information that is sent from one side to another. To achieve so, the functions in the Solidity code are accompanied with the `public` label. Even if smart contracts deployed on the Ethereum network are written in Solidity, they can be called from Python with the help of the `web3` package, which in simple terms, allows the user to operate in the blockchain with code fully written in Python. All together this facilitates the interoperatibility with the PGD program.

The PGD is responsible for calculating the optimal powers in accordance with the predefined objective. The DSO, which is the agent who computes such data, calls the contract to rewrite these powers. Figure 11 shows schematically the data involved for the same case, i.e, at 12 o'clock.

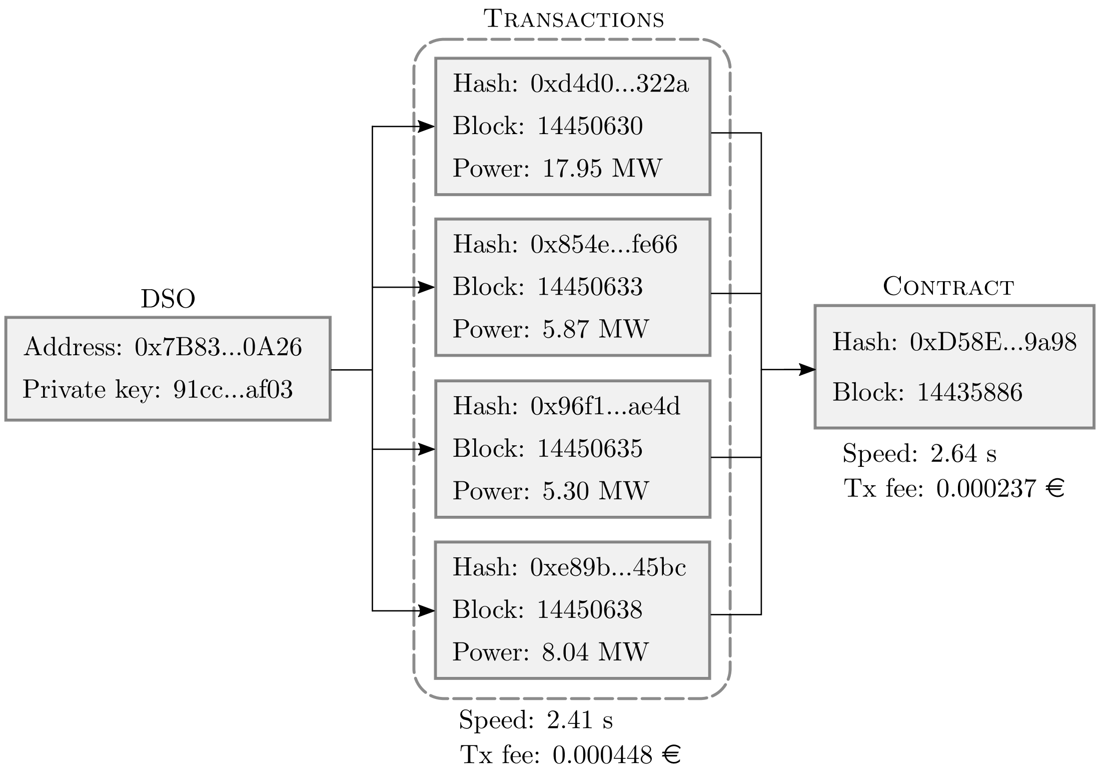

Figure 11. High-level representation of the flow of information regarding smart contracts for the test case at 12 o'clock after the PGD computation. Costs have been omitted for simplicity.

The powers could be grouped under a single array and therefore only one transaction would have been necessary. But in any case, the main takeaway is the same as before. Information can be secured in the blockchain at a low price and fast speeds. At this point, the contract has been completed, in the sense that now the particular PV power plants know their setpoints.

7 Business assessment

Compared to most engineering projects, the present work is profiled as a distinct, highly conceptual idea. Investment costs are practically null. There is no equipment required other than a computer or cluster to run the PGD. In essence, rather than an installation, the project details a potentially beneficial service to customers, prosumers, generators, and the DSO. A Business model Canvas is depicted in Figure 12 to put into perspective the business side of the project.

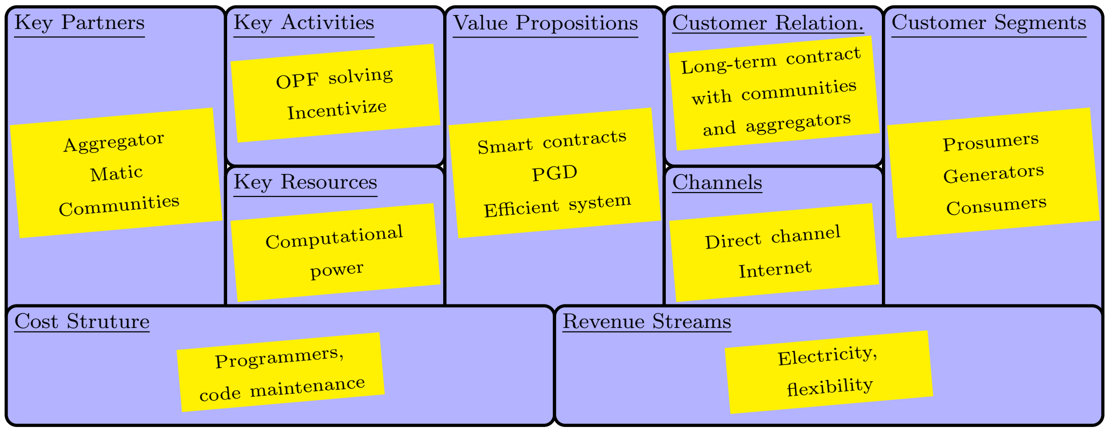

FIGURE 12. Business model Canvas for the combination of the PGD and smart contracts

The project propositions revolve around the idea of obtaining the optimal operating point and embedding this information in the blockchain, with a special focus on the efficiency and computational effort required. What makes this project stand apart is not only the combination of both ideas but also the scalability it offers. If the number of buses in the system grows, the PGD would still be able to solve the power flow relatively fast, at least much faster than traditional methods. Besides, even if the amount of transactions increases dramatically, the overall absolute cost would remain low, and the speed of the transaction would be kept at around 2 seconds.

The DSO has to establish partnerships with communities, rather than individual household customers. It is not viable to solve an OPF at the individual level, as the distribution grid becomes complex to model in detail. Thus, in order to reduce the simplicity, it is convenient to aggregate users in communities or *concelhos*. In a future electricity market, the aggregator might become the stakeholder responsible for doing precisely this. Therefore, long-term commercial relationships have to be established between the DSO and the communities or aggregators.

Generally speaking, the customer segment involves all users connected to the grid. The PGD can handle theoretically any number of dimensions, which means that there could be one dimension attributed to every bus. This would be responsible for scaling the powers, for instance. Consequently, in practical terms, it would not make any difference if the user is a consumer, generator, or prosumer. In this context where the DSO acts simultaneously as a retailer, both power and flexibility are exchanged with the user. The DSO then rewards users in a way that incentivizes the common good. Both parties are expected to benefit from it. Users may be able to pay less for their electricity, generate revenue from selling it, and be compensated for the offered flexibility. The DSO could reduce the total losses of the system, and minimize the power exchanged with the TSO.

Regarding the strengths, weaknesses, opportunities and threats of the project, they are illustrated in Figure 13 in the form of a SWOT analysis.

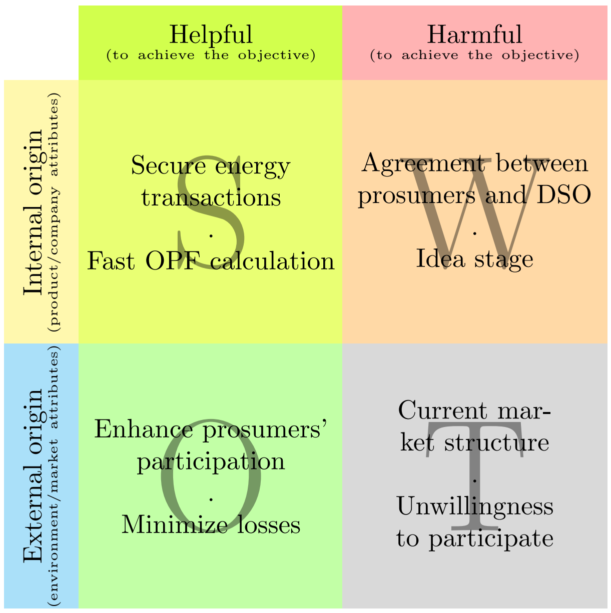
Helpful
(to achieve the objective)
Harmful
(to achieve the objective)
Internal origin
(product/company attributes)
External origin
(environment/market attributes)
S
Secure energy transactions
Fast OPF calculation
W
Agreement between prosumers and DSO
Idea stage
O
Enhance prosumers' participation
Minimize losses
T
Current market structure
Unwillingness to participate

FIGURE 13. SWOT for combination of the PGD and smart contracts

The difficulty of implementing the project is found in the necessity of establishing solid agreements with users. Users have to identify their participation as valuable, not only for their economical gains but also to sustain an optimal grid operation. In addition to that, the current market structure is likely to complicate the implementation of the project. Since users have contracted a particular retailer, communication should be established between the DSO and the retailer, and subsequently between the retailer and the users. This overcomplicates the approach selected in the project, where the retailer is dismissed, or rather, integrated into the DSO. Changes in regulation could promote the evolution of the electricity market towards the vision depicted in this project.

Regarding the economic part, a reduction of just 0.1% in the losses of the system (smaller by an order of magnitude than the amount found in the case study) is likely to be achieved. This represents a quite significant saving of energy if the numbers are scaled up. For instance, Portugal had an energy generation of 56129 GWh and an energy consumption of 48035 GWh back in 2018 [59]. This way, a 0.1% improvement in the overall efficiency of the system would imply a saving of 65.51 GWh. Assuming a price of 57 €/MWh, which corresponds to the wholesale electricity market price of Iberia in 2018, the total saving represents 3.73 M€ [60].

There are only two costs associated with the project. One has to do with the transactions and is represented by the so-called gas fees. Although extremely small, these could be paid between the DSO and the users. The second cost is the engineering team. It would be in charge of ensuring the correct real-time operation of the PGD and setting the platform to interact with the users.

8 Conclusions and future work

Let every action aim solely at the common good

Marcus Aurelius

8.1 Conclusions

The present project has unveiled a framework of operation that combines scalable mechanisms for both computing the power flow and allowing transactions between nodes in the grid. This approach is likely to establish the necessary bases to move from the current power system towards a decentralized grid where nodes in the grid are motivated to actively participate at improving the operating state of the system and are rewarded for doing so.

The cooperation of users is first of all achieved by deploying smart contracts on the blockchain, which allow users to securely trade energy. A critical decision regarding the chosen approach has to do with opting for the Matic Mainnet, a network that allows transacting by paying fees which become several orders of magnitude than in case they were deployed directly on Ethereum. The code has been written in Solidity and called from Python to offer more flexibility.

Regarding the calculation of the optimal power flow, the PGD methodology has been mathematically derived as a tool to solve multidimensional power flow problems involving not only position and time dimensions but also covering variations in the parameters of the system. In the particular scenario that has been analyzed in the case study section, the PV power generation could vary between a wide range of values. The PGD obtained the optimal power so that losses are minimized. We believe the PGD is a transformative idea that can offer operators a deeper understanding of the system in shorter time frames.

Even though the project has many difficulties to overcome, it presents an original idea that could become a reality. We are aware of the potential and the future of blockchain technology, and also the need to run fast power flow calculations, which the PGD handles remarkably well. Combining this with almost negligible operating costs, we believe that this work is just the start of something bigger.

8.2 Future work

It can be considered that the project has laid out a fundamental methodology to fight against some of the most troubling aspects related to the future of power systems. However, there are several required steps to transition from an idea to actual implementation. Some of them are listed below:

- Smart contracts should integrate the information coming from smart meters in order to assess if nodes have met the requirements specified in the transaction.

- Solve the power flow with more dimensions, some of which considering variations in demand to study the viability of demand-response mechanisms.

- Develop an accessible mobile application for communities to be aware of the energy they are trading.

Acknowledgements

The authors would like to express their gratitude to Santiago Peñate Vera, from Red Eléctrica de España, for his valuable help at parallelizing the PGD Python code.

References

[1] S. Nakamoto, "Re: Bitcoin p2p e-cash paper," *The Cryptography Mailing List*, 2008.

[2] M. Nofer, P. Gomber, O. Hinz, and D. Schiereck, "Blockchain," *Business & Information Systems Engineering*, vol. 59, no. 3, pp. 183–187, 2017.

[3] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *International Journal of Web and Grid Services*, vol. 14, no. 4, pp. 352–375, 2018.

[4] S. S. Gupta, "Blockchain," *IBM Onlone (http://www. IBM. COM)*, 2017.

[5] M. Di Pierro, "What is the blockchain?" *Computing in Science & Engineering*, vol. 19, no. 5, pp. 92–95, 2017.

[6] L. S. Sankar, M. Sindhu, and M. Sethumadhavan, "Survey of consensus protocols on blockchain applications," in *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*, IEEE, 2017, pp. 1–5.

[7] A. Miller and J. J. LaViola Jr, "Anonymous byzantine consensus from moderately-hard puzzles: A model for bitcoin," *University of Central Florida Tech. Report CS-TR-14-01 (accessed 5 June 2019) https://socrates1024. s3. amazonaws. com/consensus. pdf*, 2014.

[8] *Today's cryptocurrency prices by market cap*, https://coinmarketcap.com/, Accessed: 2021-04-19.

[9] L. Lessig, "Code is law," *Harvard magazine*, vol. 1, no. 2000, 2000.

[10] N. Szabo, "Smart contracts," *Unpublished manuscript*, 1994.

[11] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *Ieee Access*, vol. 4, pp. 2292–2303, 2016.

[12] *The eth2 upgrades. upgrading ethereum to radical new heights.* https://ethereum.org/en/eth2/, Accessed: 2021-04-19.

[13] *Matic. scalable and instant blockchain transactions*, https://matic.network/, Accessed: 2021-04-19.

[14] F. Saleh, "Blockchain without waste: Proof-of-stake," *The Review of financial studies*, vol. 34, no. 3, pp. 1156–1190, 2021.

[15] P. Gaži, A. Kiayias, and D. Zindros, "Proof-of-stake sidechains," in *2019 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2019, pp. 139–156.

[16] C. Burger, A. Kuhlmann, P. Richard, and J. Weinmann, "Blockchain in the energy transition. a survey among decision-makers in the german energy industry," *DENA German Energy Agency*, vol. 60, 2016.

[17] K. Khan *et al.*, “Blockchain-based peer-to-peer energy trading using iot devices,” 2019.

[18] I. El-Sayed, K. Khan, X. Dominguez, and P. Arboleya, “A real pilot-platform implementation for blockchain-based peer-to-peer energy trading,” in *2020 IEEE Power & Energy Society General Meeting (PESGM)*, IEEE, 2020, pp. 1–5.

[19] V. Brilliantova and T. W. Thurner, “Blockchain and the future of energy,” *Technology in Society*, vol. 57, pp. 38–45, 2019.

[20] J. Kang, R. Yu, X. Huang, S. Maharjan, Y. Zhang, and E. Hossain, “Enabling localized peer-to-peer electricity trading among plug-in hybrid electric vehicles using consortium blockchains,” *IEEE Transactions on Industrial Informatics*, vol. 13, no. 6, pp. 3154–3164, 2017.

[21] M. Pipattanasomporn, M. Kuzlu, and S. Rahman, “A blockchain-based platform for exchange of solar energy: Laboratory-scale implementation,” in *2018 International Conference and Utility Exhibition on Green Energy for Sustainable Development (ICUE)*, IEEE, 2018, pp. 1–9.

[22] A. Faugeras, “State of the art report smart contract and blockchain,” Interreg, North-West Europe D2Grids, Tech. Rep., 2019.

[23] D. Vangulick, B. Cornélusse, and D. Ernst, “Blockchain for peer-to-peer energy exchanges: Design and recommendations,” in *2018 Power Systems Computation Conference (PSCC)*, IEEE, 2018, pp. 1–7.

[24] S. Wang, A. F. Taha, J. Wang, K. Kvaternik, and A. Hahn, “Energy crowdsourcing and peer-to-peer energy trading in blockchain-enabled smart grids,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 8, pp. 1612–1623, 2019.

[25] S. Thakur and J. G. Breslin, “Peer to peer energy trade among microgrids using blockchain based distributed coalition formation method,” *Technology and Economics of Smart Grids and Sustainable Energy*, vol. 3, no. 1, pp. 1–17, 2018.

[26] P. Verma, B. O’Regan, B. Hayes, S. Thakur, and J. G. Breslin, “Enerport: Irish blockchain project for peer-to-peer energy trading,” *Energy Informatics*, vol. 1, no. 1, pp. 1–9, 2018.

[27] *Electricity and power. the optimal power flow (opf).* https://neos-guide.org/content/optimal-power-flow, Accessed: 2021-04-22.

[28] M. Ebeed, S. Kamel, and F. Jurado, “Optimal power flow using recent optimization techniques,” in *Classical and recent aspects of power system optimization*, Elsevier, 2018, pp. 157–183.

[29] J. Carpentier, “Contribution a l’etude du dispatching economique,” *Bulletin de la Societe Francaise des Electriciens*, vol. 3, no. 1, pp. 431–447, 1962.

[30] S. Frank, I. Steponavice, and S. Rebennack, “Optimal power flow: A bibliographic survey ii,” *Energy systems*, vol. 3, no. 3, pp. 259–289, 2012.

[31] J. A. Momoh and J. Zhu, “Improved interior point method for opf problems,” *IEEE Transactions on Power Systems*, vol. 14, no. 3, pp. 1114–1120, 1999.

[32] Z. Qiu, G. Deconinck, and R. Belmans, “A literature survey of optimal power flow problems in the electricity market context,” in *2009 IEEE/PES Power Systems Conference and Exposition*, IEEE, 2009, pp. 1–6.

[33] A. K. Khamees, N. Badra, and A. Y. Abdelaziz, “Optimal power flow methods: A comprehensive survey,” *International Electrical Engineering Journal (IEEJ)*, vol. 7, no. 4, pp. 2228–2239, 2016.

[34] M. R. AlRashidi and M. E. El-Hawary, “A survey of particle swarm optimization applications in electric power systems,” *IEEE transactions on evolutionary computation*, vol. 13, no. 4, pp. 913–918, 2008.

[35] F. Chinesta, R. Keunings, and A. Leygue, *The proper generalized decomposition for advanced numerical simulations: a primer.* Springer Science & Business Media, 2013.

[36] R. G. Blanco, “Efficient solvers for power flow equations: Parametric solutions with accuracy control assessment,” Ph.D. dissertation, Universitat Politècnica de Catalunya (UPC), 2017.

[37] *Ethereum price. ethereum gas charts.* https://ethereumprice.org/gas/, Accessed: 2021-04-22.

[38] *Etherscan. ethereum gas tracker.* https://etherscan.io/gastracker, Accessed: 2021-04-22.

[39] *Ycharts. ethereum average block time.* https://ycharts.com/indicators/ethereum_average_block_time, Accessed: 2021-04-22.

[40] *Matic mainnet explorer.* https://explorer-mainnet.maticvigil.com/, Accessed: 2021-04-22.

[41] E. Onaiwu, “How does bilateral trading differ from electricity pooling,” *Universisty of Dundee*, 2009.

[42] J. P. Chaves-Ávila and C. Fernandes, “The spanish intraday market design: A successful solution to balance renewable generation?” *Renewable Energy*, vol. 74, pp. 422–432, 2015.

[43] *Hacer posible la transición energética. red eléctrica y la integración de renovables*, https://www.ree.es/sites/default/files/11_PUBLICACIONES/Documentos/Transicion_Energetica.pdf, Accessed: 2021-04-30.

[44] A. Aderibole, A. Aljarwan, M. H. U. Rehman, H. H. Zeineldin, T. Mezher, K. Salah, E. Damiani, and D. Svetinovic, “Blockchain technology for smart grids: Decentralized nist conceptual model,” *IEEE Access*, vol. 8, pp. 43 177–43 190, 2020.

[45] T. Alladi, V. Chamola, J. J. Rodrigues, and S. A. Kozlov, “Blockchain in smart grids: A review on different use cases,” *Sensors*, vol. 19, no. 22, p. 4862, 2019.

[46] A. S. Musleh, G. Yao, and S. Muyeen, “Blockchain applications in smart grid–review and frameworks,” *Ieee Access*, vol. 7, pp. 86 746–86 757, 2019.

[47] *Power ledger. the power behind new energy*, https://www.powerledger.io/, Accessed: 2021-04-30.

[48] *Lightency. the sustainable african energy platform*, https://lightency.io/, Accessed: 2021-04-30.

[49] *Electrify. transactive, energy, sustainable future*, https://electrify.asia/, Accessed: 2021-04-30.

[50] *Ethereum berlin upgrade announcement*, https://blog.ethereum.org/2021/03/08/ethereum-berlin-upgrade-announcement/, Accessed: 2021-05-02.

[51] F. Chinesta, A. Ammar, and E. Cueto, “Recent advances and new challenges in the use of the proper generalized decomposition for solving multidimensional models,” *Archives of Computational methods in Engineering*, vol. 17, no. 4, pp. 327–350, 2010.

[52] R. García-Blanco, P. Díez, D. Borzacchiello, and F. Chinesta, “A reduced order modeling approach for optimal allocation of distributed generation in power distribution systems,” in *2016 IEEE International Energy Conference (ENERGYCON)*, IEEE, 2016, pp. 1–6.

[53] P. E. Black, “Greedy algorithm,” *Dictionary of Algorithms and Data Structures*, vol. 2, p. 62, 2005.

[54] Y. He, Y. Song, G. Du, and D. Zhang, “Research of matrix inversion acceleration method,” in *2009 International Conference on Computational Intelligence and Software Engineering*, IEEE, 2009, pp. 1–4.

[55] *Península - seguimiento de la demanda de energía eléctrica*, https://demanda.ree.es/visiona/peninsula/demanda/total, Accessed: 2021-04-25.

[56] *Caraterização do baixo alentejo*, https://cimbal.pt/pt/menu/598/caraterizacao-do-baixo-alentejo.aspx, Accessed: 2021-05-08.

[57] ENTSO-E, *Entso-e transmission system map*, https://www.entsoe.eu/data/map/, Accessed: 2021-05-09.

[58] Wikipedia, *Baixo alentejo (intermunicipal community)*, https://en.wikipedia.org/wiki/Baixo_Alentejo_(intermunicipal_community), Accessed: 2021-05-09.

[59] Expansión, *Portugal - consumo de electricidad*, https://datosmacro.expansion.com/energia-y-medio-ambiente/electricidad-consumo/portugal, Accessed: 2021-05-14.

[60] EDP, *Annual report, 2018.* https://www.edp.com/sites/default/files/rc_2018_en_compress.pdf, Accessed: 2021-05-14.

A Technical data

Table 4. Peak power demand and installed PV capacity in per unit values

Bus	$\mathbf{P_d}$ **(pu)**	$\mathbf{Q_d}$ **(pu)**	$\mathbf{P_g}$ **(pu)**
1	-0.0545	-0.0264	0.0000
2	-0.0322	-0.0156	0.0000
3	-0.2366	-0.1145	0.3000
4	-0.0611	-0.0296	0.0000
5	0.0165	0.0079	0.0000
6	-0.0392	-0.0189	0.0000
7	-0.048	-0.0232	0.0000
8	-0.0355	-0.0172	0.0000
9	-0.0492	-0.0238	0.0981
10	-0.048	-0.0232	0.0885
11	-0.1031	-0.0499	0.0000
12	-0.1001	-0.0484	0.1344
13	-0.0121	-0.0059	0.0000

Table 5. Parameters of the lines

Bus i	**Bus j**	**R (pu)**	$\mathbf{X_L}$ **(pu)**
1	2	0.015	0.035
1	3	0.020	0.040
1	4	0.030	0.050
2	5	0.010	0.020
2	6	0.010	0.020
3	7	0.060	0.080
3	11	0.023	0.047
4	7	0.012	0.028
5	8	0.010	0.025
8	9	0.030	0.050
9	10	0.070	0.110
10	11	0.060	0.090
11	12	0.020	0.040
12	13	0.024	0.051

B Code

```
    // Pablo Borao, Josep Fanals
pragma solidity >=0.7.0 <0.8.0;

contract EnergyProposal{
    struct Proposal{
        uint32 ID;
        address seller;
        uint32 energy;
        uint32 price;
        uint timeProposed;
    }

    mapping(uint32 => Proposal) public proposals;
    uint32 public proposalID;

    function addOffer(uint32 energy, uint32 price) public{
        Proposal storage newProposal = proposals[proposalID];
        newProposal.ID = proposalID;
        newProposal.seller = msg.sender;
        newProposal.energy = energy;
        newProposal.price = price;
        newProposal.timeProposed = block.timestamp;
        proposalID += 1;
    }

    function get() public view returns (uint) {
        return 0;
    }
}
```

Listing 1. Smart contract code written in Solidity

```
        # Pablo Borao, Josep Fanals

from web3 import Web3, HTTPProvider
from web3.middleware import geth_poa_middleware
import time
web3 = Web3(Web3.HTTPProvider('https://rpc-mainnet.matic.network'))
print(web3.isConnected())

web3.middleware_onion.inject(geth_poa_middleware, layer=0)  # avoid errors

# load smart contract
abi_full = [{"inputs":[{"internalType":"uint32","name":"energy","type":"uint32"},{"internalType":"uint32","name":"price","type":"uint32"}],"name":"addOffer","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[],"name":"proposalID","outputs":[{"internalType":"uint32","name":"","type":"uint32"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"uint32","name":"","type":"uint32"}],"name":"proposals","outputs":[{"internalType":"uint32","name":"ID","type":"uint32"},{"internalType":"address","name":"seller","type":"address"},{"internalType":"uint32","name":"energy","type":"uint32"},{"internalType":"uint32","name":"price","type":"uint32"},{"internalType":"uint256","name":"timeProposed","type":"uint256"}],"stateMutability":"view","type":"function"}]
contract_x = web3.eth.contract(abi = abi_full)

# set accounts
account_1 = "0xDc587838956cC1642c73EfeB03C4BE9247a7F163"
account_2 = "0x01E42BEAa16c42ee7d9314e79Ac59d29D2866A60"
account_3 = "0x5e335D154A1515bcE3b237bBCDDca1E1398FA8C8"
account_4 = "0xC19Bf8141a295d356d9667bac06a1d3D259e99ea"
account_DSO = "0x7B83a155F88aC066Ac89e3d34ee9966Dd5710A26"
private_key_1 = "e84b..."
private_key_2 = "0422..."
private_key_3 = "ddb0..."
private_key_4 = "7c78..."
private_key_DSO = "91cc..."

# choose account to transact
account = account_DSO
private_key = private_key_DSO

```

```
# call the smart contract
nonce = web3.eth.getTransactionCount(account)
tx1 = {'nonce': nonce, 'to':'0xD58E1ED59876e742Fe56C40b59Cc3942c02B9a98', 'gas':
    2000000, 'gasPrice': web3.toWei('5', 'gwei')}
contracte_txn = contract_x.functions.addOffer(8043, 0).buildTransaction(tx1)
signed_contr = web3.eth.account.signTransaction(contracte_txn, private_key)
tx_hash_c = web3.eth.sendRawTransaction(signed_contr.rawTransaction)
```

Listing 2. Python code to call the smart contract

```
    # Pablo Borao, Josep Fanals

import time
import pandas as pd
import numpy as np
import sys
from math import *
# from mpmath import mp
# mp.dps = 100

pd.options.display.precision = 2
pd.set_option('display.precision', 2)

def progress_bar(i, n, size):  # show the percentage
    percent = float(i) / float(n)
    sys.stdout.write("\r"
                     + str(int(i)).rjust(3, '0')
                     + "/"
                     + str(int(n)).rjust(3, '0')
                     + ' ['
                     + '='*ceil(percent*size)
                     + ' '*floor((1-percent)*size)
                     + ']')


def read_grid_data(fname):
    """
    Read the grid data
    :param fname: name of the excel file
    :return: n_buses, Qmax, Qmin, Y, Yinv, V_mod, P_pq, Q_pq, P_pv, IO_pq, n_pv, n_pq
    """
    df_b = pd.read_excel(fname, sheet_name="buses")
    df_l = pd.read_excel(fname, sheet_name="lines")

    # BEGIN INITIALIZATION OF DATA
    n_b = 0
    n_pq = 0
    n_pv = 0
    pq = []
    pv = []
    pq0 = []  # store pq buses indices relative to its own
    pv0 = []  # store pv buses indices relative to its own
    d_pq = {}  # dict of pq
    d_pv = {}  # dict of pv
    for i in range(len(df_b)):
        if df_b.iloc[i, 4] == "slack":  # index 0 is reserved for the slack bus
            pass

        elif df_b.iloc[i, 4] == "PQ":
            pq0.append(n_pq)
            d_pq[df_b.iloc[i, 0]] = n_pq
            n_b += 1
            n_pq += 1
            pq.append(df_b.iloc[i, 0] - 1)

        elif df_b.iloc[i, 4] == "PV":
            pv0.append(n_pv)
            d_pv[df_b.iloc[i, 0]] = n_pv
            n_b += 1
            n_pv += 1
            pv.append(df_b.iloc[i, 0] - 1)

    n_l = len(df_l)  # number of lines

    V0 = df_b.iloc[0, 3]  # the slack is always positioned in the first row
    I0_pq = np.zeros(n_pq, dtype=complex)
    I0_pv = np.zeros(n_pv, dtype=complex)
    Y = np.zeros((n_b, n_b), dtype=complex)  # I will build it with block matrices
    Y11 = np.zeros((n_pq, n_pq), dtype=complex)  # pq pq
    Y12 = np.zeros((n_pq, n_pv), dtype=complex)  # pq pv
    Y21 = np.zeros((n_pv, n_pq), dtype=complex)  # pv pq
    Y22 = np.zeros((n_pv, n_pv), dtype=complex)  # pv pv

    for i in range(n_l):
        Ys = 1 / (df_l.iloc[i, 2] + 1j * df_l.iloc[i, 3])  # series element
        Ysh = df_l.iloc[i, 4] + 1j * df_l.iloc[i, 5]  # shunt element
        t = df_l.iloc[i, 6] * np.cos(df_l.iloc[i, 7]) + 1j * df_l.iloc[i, 6] * np.sin(
     df_l.iloc[i, 7])  # tap as a complex number

        a = df_l.iloc[i, 0]
        b = df_l.iloc[i, 1]

        if a == 0:
            if b - 1 in pq:
                I0_pq[d_pq[b]] += V0 * Ys / t
                Y11[d_pq[b], d_pq[b]] += Ys + Ysh
            if b - 1 in pv:
                I0_pv[d_pv[b]] += V0 * Ys / t
                Y22[d_pv[b], d_pv[b]] += Ys + Ysh

        elif b == 0:
            if a - 1 in pq:
                I0_pq[d_pq[a]] += V0 * Ys / np.conj(t)
                Y11[d_pq[a], d_pq[a]] += (Ys + Ysh) / (t * np.conj(t))
            if a - 1 in pv:
                I0_pv[d_pv[a]] += V0 * Ys / np.conj(t)
                Y22[d_pv[a], d_pv[a]] += (Ys + Ysh) / (t * np.conj(t))

        else:
            if a - 1 in pq and b - 1 in pq:
                Y11[d_pq[a], d_pq[a]] += (Ys + Ysh) / (t * np.conj(t))
                Y11[d_pq[b], d_pq[b]] += Ys + Ysh
                Y11[d_pq[a], d_pq[b]] += - Ys / np.conj(t)
                Y11[d_pq[b], d_pq[a]] += - Ys / t

            if a - 1 in pq and b - 1 in pv:
                Y11[d_pq[a], d_pq[a]] += (Ys + Ysh) / (t * np.conj(t))
                Y22[d_pv[b], d_pv[b]] += Ys + Ysh
                Y12[d_pq[a], d_pv[b]] += - Ys / np.conj(t)
                Y21[d_pv[b], d_pq[a]] += - Ys / t

            if a - 1 in pv and b - 1 in pq:
                Y22[d_pv[a], d_pv[a]] += (Ys + Ysh) / (t * np.conj(t))
                Y11[d_pq[b], d_pq[b]] += Ys + Ysh
                Y21[d_pv[a], d_pq[b]] += - Ys / np.conj(t)
                Y12[d_pq[b], d_pv[a]] += - Ys / t

            if a - 1 in pv and b - 1 in pv:
                Y22[d_pv[a], d_pv[a]] += (Ys + Ysh) / (t * np.conj(t))
                Y22[d_pv[b], d_pv[b]] += Ys + Ysh
                Y22[d_pv[a], d_pv[b]] += - Ys / np.conj(t)
                Y22[d_pv[b], d_pv[a]] += - Ys / t

    # add shunts connected directly to the bus
    for i in range(len(df_b)):
        a = df_b.iloc[i, 0]
        if a - 1 in pq:
            Y11[d_pq[a], d_pq[a]] += df_b.iloc[i, 5] + 1j * df_b.iloc[i, 6]

        elif a - 1 in pv:
            Y22[d_pv[a], d_pv[a]] += df_b.iloc[i, 5] + 1j * df_b.iloc[i, 6]

    Y = np.block([[Y11, Y12], [Y21, Y22]])
    Yinv = np.linalg.inv(Y)

    V_mod = np.zeros(n_pv, dtype=float)
    P_pq = np.zeros(n_pq, dtype=float)
    P_pv = np.zeros(n_pv, dtype=float)
    Q_pq = np.zeros(n_pq, dtype=float)
    for i in range(len(df_b)):
        if df_b.iloc[i, 4] == "PV":
            V_mod[d_pv[df_b.iloc[i, 0]]] = df_b.iloc[i, 3]
            P_pv[d_pv[df_b.iloc[i, 0]]] = df_b.iloc[i, 1]
        elif df_b.iloc[i, 4] == "PQ":
            Q_pq[d_pq[df_b.iloc[i, 0]]] = df_b.iloc[i, 2]
            P_pq[d_pq[df_b.iloc[i, 0]]] = df_b.iloc[i, 1]

    n_buses = np.shape(Y)[0]  # number of buses

    return n_buses, Y, Yinv, V_mod, P_pq, Q_pq, P_pv, I0_pq, n_pv, n_pq


def init_apparent_powers_decomposition(n_buses, n_scale, n_time, P_pq, Q_pq):
    """

    :param n_buses:
    :param n_scale:
    :param n_time:
    :param P_pq:
    :param Q_pq:
    :return:
    """
    SSk = np.empty((n_buses, 2), dtype=complex)
    SSp = np.empty((n_time, 2), dtype=complex)
    SSq = np.empty((n_scale, 2), dtype=complex)

    SKk0 = P_pq + Q_pq * 1j  # original load
    SPp0 = np.ones(n_time)  # the original loads do not change with time
    SQq0 = np.ones(n_scale)  # the original loads are not scaled

    SSk[:, 0] = np.conj(SKk0)
    SSp[:, 0] = np.conj(SPp0)
    SSq[:, 0] = np.conj(SQq0)

    SKk1 = - 0.2 * np.random.rand(n_buses)  # load/generator of random active power,
     change this to try different cases
    SPp1 = np.random.rand(n_time)
    SQq1 = np.random.rand(n_scale)

    # SKk1 = - np.array([0.1, 0.2, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.2])  # change
     this if the dimensions change
    # SPp1 = np.array([0.0, 0.2, 0.3, 0.7, 0.7, 0.3, 0.2, 0.1])
    # SQq1 = np.array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])

    SSk[:, 0] = np.conj(SKk1)
    SSp[:, 0] = np.conj(SPp1)
    SSq[:, 0] = np.conj(SQq1)

    return SSk, SSp, SSq


def init_voltages_decomposition(n_mm, n_buses, n_scale, n_time):
    """

    :param n_buses:
    :param n_scale:
    :param n_time:
    :return:
    """
    # DECOMPOSITION OF VOLTAGES
    VVk = np.zeros((n_buses, n_mm + 1), dtype=complex)
    VVp = np.zeros((n_time, n_mm + 1), dtype=complex)
    VVq = np.zeros((n_scale, n_mm + 1), dtype=complex)

    Kkv = np.ones(n_buses, dtype=complex)  # amplitude vector
    Ppv = np.ones(n_time)  # position vector
    Qqv = np.ones(n_scale)  # scaling vector

    VVk[:, 0] = np.conj(Kkv)
    VVp[:, 0] = np.conj(Ppv)
    VVq[:, 0] = np.conj(Qqv)

    return VVk, VVp, VVq


def init_currents_decomposition(n_gg, n_mm, n_buses, n_scale, n_time):
    """

    :return:
    """
    # DECOMPOSITION OF CURRENTS
    # IIk = np.zeros((n_buses, n_gg * n_mm), dtype=complex)
    # IIp = np.zeros((n_time, n_gg * n_mm), dtype=complex)
    # IIq = np.zeros((n_scale, n_gg * n_mm), dtype=complex)

    IIk = np.zeros((n_buses, n_mm + 1), dtype=complex)
    IIp = np.zeros((n_time, n_mm + 1), dtype=complex)
    IIq = np.zeros((n_scale, n_mm + 1), dtype=complex)
    return IIk, IIp, IIq


def fun_C(SSk, SSp, SSq, VVk, VVp, VVq, IIk, IIp, IIq, n_i_coeff, n_v_coeff, n_bus,
    n_scale, n_time):
    """

    :param SSk:
    :param SSp:
    :param SSq:
    :param VVk:
    :param VVp:
    :param VVq:
    :param IIk:
    :param IIp:
    :param IIq:
    :param n: number of coefficients fo far
    :return:
    """
    # Ns = SSk.shape[0]
    # Nc = Ns + n_v_coeff * n_i_coeff

    # CCk = np.empty((Nc, n_bus), dtype=complex)
    # CCp = np.empty((Nc, n_time), dtype=complex)
    # CCq = np.empty((Nc, n_scale), dtype=complex)

    # initialize with the S* decomposed variables
    # CCk[:2, :] = SSk
    # CCp[:2, :] = SSp
    # CCq[:2, :] = SSq
    # idx = 2
    # for ii in range(n_v_coeff):
    #     for jj in range(n_i_coeff):
    #         CCk[idx, :] = - VVk[ii, :] * IIk[jj, :]
    #         CCp[idx, :] = - VVp[ii, :] * IIp[jj, :]
    #         CCq[idx, :] = - VVq[ii, :] * IIq[jj, :]
    #         idx += 1

    Nc = n_v_coeff * n_i_coeff + 2

    CCk = np.empty((n_bus, Nc), dtype=complex)
    CCp = np.empty((n_time, Nc), dtype=complex)
    CCq = np.empty((n_scale, Nc), dtype=complex)

    CCk[:, :2] = SSk
    CCp[:, :2] = SSp
    CCq[:, :2] = SSq
    idx = 2
    for ii in range(n_v_coeff):
        for jj in range(n_i_coeff):
            CCk[:, idx] = - VVk[:, ii] * IIk[:, jj]
            CCp[:, idx] = - VVp[:, ii] * IIp[:, jj]
            CCq[:, idx] = - VVq[:, ii] * IIq[:, jj]
            idx += 1

    return CCk, CCp, CCq, Nc, n_v_coeff, n_i_coeff


def build_map(MMk, MMp, MMq, n_mm):
    """
    Build 3-D mapping from decomposed variables
    :param MMk:
    :param MMp:
    :param MMq:
    :return:
    """
    MM_map = np.multiply.outer(np.multiply.outer(MMp[:, 0], MMk[:, 0]), MMq[:, 0])
    n = len(MMk)
    for i in range(1, n_mm + 1):
        print(i)
        # the tri-dimensional representation I am looking for
        MM_map += np.multiply.outer(np.multiply.outer(MMp[:, i], MMk[:, i]), MMq[:, i])
        progress_bar(i+1, n, 50)

    return MM_map


def save_mapV(MM_map, fname, n_time):
    """

    :param MM_map: 3D tensor for voltages
    :param fname:
    :return:
    """
    writer = pd.ExcelWriter(fname)
    for i in range(n_time):
        V_map_df = pd.DataFrame(np.conj(MM_map[:][i][:]))
        V_map_df.to_excel(writer, sheet_name=str(i))
    writer.save()


def save_mapI(MM_map, fname, n_time):
    """

    :param MM_map: 3D tensor for currents
    :param fname:
    :return:
    """
    writer = pd.ExcelWriter(fname)
    for i in range(n_time):
        V_map_df = pd.DataFrame(MM_map[:][i][:])
        V_map_df.to_excel(writer, sheet_name=str(i))
    writer.save()


def save_mapS(MM_map, fname, n_time):
    """

    :param MM_map: 3D tensor for powers
    :param fname:
    :return:
    """
    writer = pd.ExcelWriter(fname)
    for i in range(n_time):
        V_map_df = pd.DataFrame(np.conj(MM_map[:][i][:]))
        V_map_df.to_excel(writer, sheet_name=str(i))
    writer.save()



def pgd(fname, n_gg=20, n_mm=20, n_kk=20, n_scale=12, n_time=8):
    """

    :param fname: data file name
    :param n_gg: outer iterations
    :param n_mm: intermediate iterations
```

```python
    :param n_kk: inner iterations
    :param n_scale: number of discretized points, arbitrary
    :param n_time: number of discretized time periods, arbitrary
    :return:
    """

    n_buses, Y, Yinv, V_mod, P_pq, Q_pq, P_pv, I0_pq, n_pv, n_pq = read_grid_data(fname)

    SSk, SSp, SSq = init_apparent_powers_decomposition(n_buses, n_scale, n_time, P_pq, Q_pq)
    # VVk, VVp, VVq = init_voltages_decomposition(n_mm, n_buses, n_scale, n_time)
    # IIk, IIp, IIq = init_currents_decomposition(n_gg, n_mm, n_buses, n_scale, n_time)

    n_max = n_gg * n_mm * n_kk
    iter_count = 0
    idx_i = 0
    idx_v = 1
    for gg in range(n_gg):  # outer loop: iterate on gamma to solve the power flow as such
        VVk, VVp, VVq = init_voltages_decomposition(n_mm, n_buses, n_scale, n_time)
        IIk, IIp, IIq = init_currents_decomposition(n_gg, n_mm, n_buses, n_scale, n_time)

        idx_i = 0
        idx_v = 1

        for mm in range(n_mm):  # intermediate loop: iterate on i to find the superposition of terms of the I tensor.
            # define the new C
            CCk, CCp, CCq, Nc, Nv, n = fun_C(SSk, SSp, SSq, VVk, VVp, VVq, IIk, IIp, IIq, idx_i, idx_v, n_buses, n_scale, n_time)
            # CCk, CCp, CCq, Nc, Nv, n = fun_C2(SSk, SSp, SSq, VVk, VVp, VVq, IIk, IIp, IIq)

            # initialize the residues we have to find
            # IIk1 = (np.random.rand(n_buses) - np.random.rand(n_buses)) * 1  # could also try to set IIk1 = VVk1
            IIk1 = (np.random.rand(n_buses) - np.random.rand(n_buses)) * (n_mm - mm) ** 2 / n_mm ** 2
            IIp1 = (np.random.rand(n_time) - np.random.rand(n_time)) * 1
            IIq1 = (np.random.rand(n_scale) - np.random.rand(n_scale)) * 1

            for kk in range(n_kk):  # inner loop: iterate on k to find the residues.

                # compute IIk1 (residues on Ik)
                RHSk = np.zeros(n_buses, dtype=complex)
                for ii in range(Nc):
                    prodRK = np.dot(IIp1, CCp[:, ii]) * np.dot(IIq1, CCq[:, ii])
                    RHSk += prodRK * CCk[:, ii]

                LHSk = np.zeros(n_buses, dtype=complex)
                for ii in range(Nv):
                    prodLK = np.dot(IIp1, VVp[:, ii] * IIp1) * np.dot(IIq1, VVq[:, ii] * IIq1)
                    LHSk += prodLK * VVk[:, ii]

                # IIk1 = RHSk / LHSk
                IIk1 = RHSk / (LHSk + 1e-8)

                # compute IIp1 (residues on Ip)
                RHSp = np.zeros(n_time, dtype=complex)
                for ii in range(Nc):
                    prodRP = np.dot(IIk1, CCk[:, ii]) * np.dot(IIq1, CCq[:, ii])
                    RHSp += prodRP * CCp[:, ii]

                LHSp = np.zeros(n_time, dtype=complex)
                for ii in range(Nv):
                    prodLP = np.dot(IIk1, VVk[:, ii] * IIk1) * np.dot(IIq1, VVq[:, ii] * IIq1)
                    LHSp += prodLP * VVp[:, ii]

                # IIp1 = RHSp / LHSp
                IIp1 = RHSp / (LHSp + 1e-8)

                # compute IIq1 (residues on Iq)
                RHSq = np.zeros(n_scale, dtype=complex)
                for ii in range(Nc):
                    prodRQ = np.dot(IIk1, CCk[:, ii]) * np.dot(IIp1, CCp[:, ii])
                    RHSq += prodRQ * CCq[:, ii]

                LHSq = np.zeros(n_scale, dtype=complex)
                for ii in range(Nv):
                    prodLQ = np.dot(IIk1, VVk[:, ii] * IIk1) * np.dot(IIp1, VVp[:, ii] *
         IIp1)
                    LHSq += prodLQ * VVq[:, ii]

                # IIq1 = RHSq / LHSq
                IIq1 = RHSq / (LHSq + 1e-8)

                progress_bar(iter_count, n_max, 50)  # display the inner operations
                iter_count += 1

            IIk[:, idx_i] = IIk1
            IIp[:, idx_i] = IIp1
            IIq[:, idx_i] = IIq1
            idx_i += 1

        for ii in range(n_mm):
            VVk[:, ii] = np.conj(np.dot(Yinv, IIk[:, ii]))
            VVp[:, ii] = np.conj(IIp[:, ii])
            VVq[:, ii] = np.conj(IIq[:, ii])

        # try to add I0 this way:
        VVk[:, n_mm] = np.conj(np.dot(Yinv, I0_pq))
        VVp[:, n_mm] = np.ones(n_time)
        VVq[:, n_mm] = np.ones(n_scale)
        idx_v = n_mm + 1

    # VVk: size (n_mm + 1, nbus)
    # VVp: size (n_mm + 1, nbus)
    # VVq: size (n_mm + 1, n_scale)
    v_map = build_map(VVk, VVp, VVq, n_mm)

    # SSk: size (2, nbus)
    # SSp: size (2, nbus)
    # SSq: size (2, n_scale)
    s_map = build_map(SSk, SSp, SSq, 1)
    # s_map = build_map(SSk, SSp, SSq, n_mm)

    # IIk: size (n_gg * n_mm, nbus)
    # IIp: size (n_gg * n_mm, nbus)
    # IIq: size (n_gg * n_mm, n_scale)
    i_map = build_map(IIk, IIp, IIq, n_mm)

    # the size of the maps is nbus, ntime, n_scale

    vec_error = checking(Y, v_map, s_map, I0_pq, n_buses, n_time, n_scale)

    return v_map, s_map, i_map, vec_error


def checking(Y, V_map, S_map, I0_pq, n_buses, n_time, n_scale):
    """

    :param Y: data file name
    :param V_map: outer iterations
    :param S_map: intermediate iterations
    :param I0_pq: inner iterations
    :param n_buses: number of buses
    :param n_scale: number of discretized points, arbitrary
    :param n_time: number of discretized time periods, arbitrary
    :return: maximum current error
    """

    I_mis = []
    for n_sheet in range(n_time):
        for n_col in range(n_scale):
            YV_prod = np.dot(Y, np.conj(V_map[n_sheet,:,n_col]))
            # YV_prod = np.dot(Y, V_map[n_sheet,:,n_col])
            I22 = YV_prod - I0_pq
            I11 = []
            for kk in range(n_buses):
                I11.append(S_map[n_sheet,kk,n_col] / V_map[n_sheet,kk,n_col])
                # I11.append(np.conj(S_map[n_sheet,kk,n_col]) / np.conj(V_map[n_sheet,kk
    ,n_col]))
            I_mis.append(abs(max(np.abs(I11 - I22))))
    return I_mis


v_map_, s_map_, i_map_, vec_error_ = pgd('data_10PQ_mesh_r.xlsx', n_gg=25, n_mm=25, n_kk
    =25, n_scale=10, n_time=80)

for i in range(len(vec_error_)):
    print(vec_error_[i])

n_time = 8
save_mapV(v_map_, 'Map_V2.xlsx', n_time)
save_mapI(i_map_, 'Map_I2.xlsx', n_time)
save_mapS(s_map_, 'Map_S2.xlsx', n_time)
```

Listing 3. Proper Generalized Decomposition code in Python