UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Grado en Ingeniería Informática

Implementación de Modelos de Inteligencia Artificial para Optimizar el Autoconsumo Energético y Minimizar Vertidos a la Red

Implementation of Artificial Intelligence Models for Energy Self-Consumption Optimization and Grid Waste Reduction

Realizado por
Juan Manuel García Delgado

Tutorizado por
José del Campo Ávila
Rafael Morales Bueno

Departamento
Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, junio de 2024

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA INFORMÁTICA

Implementación de Modelos de Inteligencia Artificial para Optimizar el Autoconsumo Energético y Minimizar Vertidos a la Red

Implementation of Artificial Intelligence Models for Energy Self-Consumption Optimization and Grid Waste Reduction

Realizado por
Juan Manuel García Delgado

Tutorizado por
Jose del Campo Ávila
Rafael Morales Bueno

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, JUNIO DE 2024

Fecha defensa: Julio de 2024

Resumen

El presente documento constituye la memoria del Trabajo de Fin de Grado en Ingeniería Informática, en la mención de computación. Su título es: *Implementación de Modelos de Inteligencia Artificial para Optimizar el Autoconsumo Energético y Minimizar Vertidos a la Red.*

El propósito fundamental de este proyecto consiste en el desarrollo e implementación de modelos de optimización destinados a calcular coeficientes que contribuyan a la minimización del vertido de energía a la red eléctrica. Algunas comunidades de vecinos comparten placas solares, de forma que la energía generada a lo largo del día se reparte con unos coeficientes constantes entre los vecinos dueños de dichas placas solares, es decir, los coeficientes de reparto son iguales para cada vecino a lo largo del tiempo, sin importar el uso personal de la energía de cada vecino. A partir del Real Decreto 244/2019, se permite a las distribuidoras el uso de coeficientes diferentes y dinámicos, con el objetivo de minimizar el vertido de energía a la red, y evitar así su posterior compra.

Se ha conseguido configurar exitosamente un conocido método de optimización inicializándolo con un algoritmo de diseño propio que proporciona rápidamente una solución preliminar muy cercana al óptimo. Este enfoque permite un arranque eficiente sin comprometer significativamente los tiempos de computación, mientras que el método continúa refinando esta solución hacia una mayor optimización.

Palabras clave: Optimización, SLSQP, Autoconsumo eléctrico

Abstract

This document constitutes the thesis for the Bachelor's Degree in Computer Engineering, with a specialization in computing. Its title is: *Implementation of Artificial Intelligence Models to Optimize Self-Consumption and Minimize Energy Injection to the Grid.*

The primary purpose of this project is the development and implementation of optimization models aimed at calculating coefficients that contribute to minimizing the injection of energy into the electrical grid. Some neighborhood communities share solar panels, so the energy generated throughout the day is distributed among the panel-owning neighbors using constant coefficients. That is, the distribution coefficients are the same for each neighbor over time, regardless of individual energy usage. According to Royal Decree 244/2019, distributors are allowed to use different and dynamic coefficients to minimize energy injection into the grid and thus avoid subsequent energy purchases.

A well-known optimization method has been successfully configured by initializing it with a proprietary design algorithm that quickly provides a preliminary solution very close to the optimum. This approach allows for an efficient start without significantly compromising computing times, while the method continues to refine this solution towards greater optimization.

Keywords: Optimization, SLSQP, Electrical Self-Consumption

Índice

1 Introducción

1.1. Motivación

Conforme al Real Decreto 244/2019, se definen las condiciones regulatorias para el autoconsumo fotovoltaico. Los aspectos más relevantes modificados por la nueva legislación incluyen:

1. La energía producida a partir de instalaciones de autoconsumo queda completamente libre de impuestos. Con esto se elimina el conocido como *impuesto al sol*.

2. Se simplifican los trámites administrativos y técnicos.

3. Se elimina el límite de potencia. Antes, únicamente se podía instalar una potencia fotovoltaica igual o inferior a la contratada.

4. Se reconoce el derecho de autoconsumo colectivo.

Este Trabajo de Fin de Grado se inspira principalmente en el cuarto punto mencionado. Este marco permite la creación de mecanismos que faciliten la implementación de coeficientes de reparto dinámicos.

La utilidad de estos coeficientes dinámicos se manifiesta en que la energía excedente vertida a la red se adquiere a un costo significativamente inferior al precio de compra. Por ende, si los residentes de una comunidad comparten su energía, reduciendo al mínimo la cantidad vertida a la red, se traduce en un ahorro económico sustancial. Esto ocurre porque, bajo ciertas condiciones, un residente podría estar utilizando la energía suministrada por otros, a cambio de proporcionar acceso a su propia energía en otro momento.

1.2. Objetivos

El propósito fundamental de este proyecto consiste en el desarrollo e implementación de modelos de optimización destinados a calcular coeficientes que contribuyan a la minimización del vertido de energía a la red eléctrica. Este objetivo general se descompone en varios objetivos específicos:

1. Obtención de una muestra pequeña con usos energéticos dispares.
2. Diseño y aplicación de modelos de optimización para ajustar los coeficientes de forma dinámica, minimizando el vertido a la red.
3. Aplicación de técnicas de análisis de datos para obtener muestras más representativas pero con perfiles dispares.
4. Estudio de la evolución del modelo dependiendo de cómo aumenta la muestra con perfiles dispares.
5. Estudio de la evolución el modelo usando una muestra de perfiles más similares.
6. Incorporación de variables adicionales, como el coste variable de la energía por hora, en el modelo de optimización.

1.3. Metodología de trabajo

Como metodología adaptada en este trabajo se ha utilizado CRISP-DM (Cross-Industry Standard Process for Data Mining). CRISP-DM es un método ampliamente utilizado en la industria de la minería de datos. Como metodología, incluye descripciones de las fases normales de un proyecto y sus relaciones. Además, ofrece un ciclo vital de minería de datos.[6]

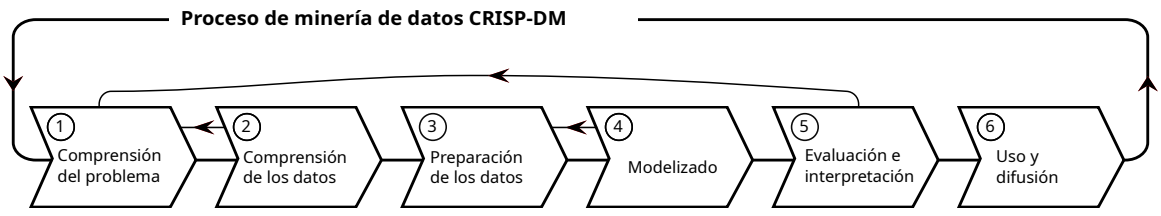


Figura 1: Ciclo de vida de CRISP-DM.

El ciclo vital consiste en 6 fases, y las flechas indican las dependencias principales, como se puede observar en la Figura 1. La secuencia de las fases no es estricta. A menudo, la mayoría de los proyectos avanzan y retroceden entre fases si es necesario.

Las fases son las siguientes:

1. **Comprensión del problema**: También conocida como comprensión del negocio, esta fase se centra en comprender el problema al que queremos encontrar una solución y los objetivos del proyecto a nivel empresarial. Se establecen los requisitos y se determina cómo los resultados del proyecto pueden afectar y cumplir con los objetivos comerciales.

2. **Comprensión de los datos**: Esta fase implica la recopilación, análisis y evaluación de la calidad de los datos relevantes para el proyecto. Es crucial entender la naturaleza y la calidad de los datos disponibles antes de proceder con cualquier análisis.

3. **Preprocesado de los datos**: En esta fase se efectúan tareas como la limpieza de datos, el manejo de valores faltantes, la transformación de datos y la selección de características, con el fin de preparar los datos para el análisis y modelado posterior.

4. **Modelizado**: Durante esta etapa se desarrollan y calibran modelos predictivos o descriptivos empleando los datos preparados, utilizando técnicas de aprendizaje automático y minería de datos, entre otras.

5. **Evaluación e interpretación**: Una vez construidos los modelos, se evalúan para determinar su efectividad y precisión, utilizando métricas adecuadas que reflejen los objetivos del proyecto.

6. **Despliegue**: Esta fase final implica la implementación práctica de los resultados analíticos. Esto puede incluir la integración de modelos en sistemas informáticos, la presentación de resultados a las partes interesadas y el desarrollo de estrategias para la utilización continua de los hallazgos obtenidos.

La fase de despliegue no se abordará en este proyecto por razones específicas al contexto del mismo. Asimismo, la fase de comprensión del problema se ha tratado previamente en la introducción, en la sección de Motivación [6].

1.4. Estructura del documento

El documento se estructura en seis capítulos detallados a continuación:

1. El primer capítulo sirve como introducción y establece la motivación, los objetivos del estudio, y una comprensión profunda del problema abordado. Este capítulo se alinea con la Fase 1 de la metodología CRISP-DM mencionada en el anteproyecto, denominada "Comprensión del Problema".

2. El segundo capítulo ofrece una breve revisión de la teoría matemática que subyace a los algoritmos implementados para alcanzar los objetivos propuestos. Se discuten conceptos fundamentales como problema de optimización, región factible, y solución óptima, junto con una descripción de los métodos empleados.

3. El tercer capítulo explica los datos utilizados, examinando su relevancia y relación con el problema en cuestión. Aquí, los datos son procesados y ajustados para cumplir con las exigencias del proyecto. Este capítulo abarca las Fases 2 y 3 de CRISP-DM.

4. El cuarto capítulo se dedica al modelado. Se ajustan modelos existentes al problema específico del estudio y se desarrolla un método personalizado que ofrece aproximaciones eficaces en tiempos reducidos.

5. El quinto capítulo se enfoca en la evaluación de los modelos según la metodología CRISP-DM. Se realiza una comparativa de los modelos desarrollados, analizando su rendimiento y estabilidad, entre otros aspectos.

6. El sexto y último capítulo actúa como conclusión del documento, donde se resumen los hallazgos de la evaluación y se proponen líneas futuras de investigación para la mejora de los modelos y algoritmos utilizados.

1.5. Tecnologías utilizadas

El desarrollo de este proyecto se ha realizado íntegramente en el lenguaje de programación Python. Específicamente, las bibliotecas utilizadas han sido:

- Pandas: Utilizada para el manejo y análisis de datos.

- Os: Empleada para la gestión de directorios dentro del sistema operativo.
- Scipy: Concretamente, se ha aprovechado el módulo Optimize para optimización matemática.
- Numpy: Destinada a realizar operaciones matemáticas básicas y avanzadas.
- Papermill: Aplicada para la automatización y parametrización de notebooks de Jupyter, facilitando la ejecución de pruebas en servidores remotos.

El código desarrollado se ha organizado principalmente en archivos Jupyter Notebook, los cuales permiten la integración de código Python, texto en formato Markdown y expresiones matemáticas en LaTeX, proporcionando un entorno de trabajo dinámico y versátil.

Además, se ha utilizado Git junto con GitHub como sistemas de control de versiones, lo que ha permitido un manejo eficiente y sistemático de las revisiones del código. El desarrollo se ha efectuado en un sistema operativo Linux, utilizando el protocolo SSH para la ejecución de pruebas extensivas en una máquina de alto rendimiento suministrada por el grupo de Investigación y Aplicaciones en Inteligencia Artificial de la Universidad de Málaga. Los resultados obtenidos han sido transmitidos a través de la API de Telegram para su posterior procesamiento en un entorno local.

2

Métodos de Optimización

La optimización es una herramienta poderosa para minimizar costes o maximizar beneficios, al mismo tiempo que se satisfacen ciertos requisitos y restricciones. Para obtener resultados de optimización que puedan ajustarse bien a la producción del mundo real, es deseable utilizar modelos rigurosos. Los desafíos surgen particularmente de la naturaleza altamente no lineal, no convexa y a menudo mal condicionada de los problemas reales a gran escala, lo cual incrementa la dificultad y el costo computacional de resolverlos. Por esta razón, se han desarrollado múltiples técnicas avanzadas para abordar estos problemas.

2.1. Problemas de Optimización

Un problema de optimización generalmente se escribe como sigue:

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} f(x) \\ &\text{s.a.} \quad h(x) = 0, \end{aligned}$$

donde $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, $h : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ son funciones continuas y diferenciables, y x un vector de n dimensiones. f es llamada la función objetivo y $h(x) = 0$ son restricciones de igualdad.

Aunque existen restricciones tanto de igualdad como de desigualdad, por conveniencia y claridad en este documento se tratarán como restricciones de igualdad. Es relevante mencionar que la teoría de optimización proporciona métodos para convertir restricciones de desigualdad en equivalentes de igualdad mediante el uso de variables de holgura.

Algunos ejemplos de problemas de optimización pueden ser: minimizar costes mientras mantiene un nivel de producción específico, maximizar ventas, o hasta la diseñar una planificación logística que minimice el tiempo total de transporte de mercancías.

Antes de entrar en materia, es importante conocer tres definiciones relevantes.

Definición 1. *Se define la **región factible** de un problema como un conjunto de puntos de $\mathbb{R}^n$ que verifican las restricciones del problema, es decir, la región factible es un subconjunto K de $\mathbb{R}^n$ dado por:*

$$K = \{x \in \mathbb{R}^n \mid h(x) = 0\}$$

Definición 2. *Se dice que $x \in \mathbb{R}^n$ es una **solución posible**, o abreviadamente **SP**, del problema si es un vector que cumple las restricciones, es decir, $h(x) = 0$.*

Definición 3. *Se dice que $x \in \mathbb{R}^n$ es **solución posible óptima**, o abreviadamente **SPO**, del problema si es un vector que cumple las restricciones, es decir, $h(x) = 0$, y además, $f(x) \leq f(y)$, para todo $y \in K$, siendo K la región de factibilidad.*

La idea es pensar que, en un espacio $\mathbb{R}^n$ se delimita una región por las restricciones. Esa región es la que hemos definido anteriormente como región factible. Aquí, todos los puntos serán soluciones posibles, y existirán uno o infinitos puntos que sean soluciones posibles óptimas. Nótese que la región de factible puede ser vacía, y en este caso, no existirá solución posible al problema. Se tratará de un problema imposible.

2.2. Métodos de Optimización

Existen numerosos métodos de optimización que varían según la cantidad y el tipo de información disponible sobre la función objetivo. La diferenciabilidad de la función objetivo es un criterio crucial, dividiendo los algoritmos en dos categorías principales como se puede observar en la Figura 2: aquellos que utilizan el gradiente y los que no.

1. **Algoritmos que usan información del gradiante**:

   a) **Algoritmos Bracketing**: Utilizados para funciones unidimensionales con un óptimo conocido dentro de un intervalo definido, como la *búsqueda de Fibonacci* y el *método de bisección.*

   b) **Algoritmos de descenso local**: Aplicables a funciones con un único óptimo global y varias variables de entrada, optimizando a través de métodos como la *búsqueda de líneas.*

c) **Algoritmos de primer orden**: Los algoritmos de optimización de primer orden implican explícitamente el uso de la primera derivada (gradiente) para elegir la dirección en la que moverse en el espacio de búsqueda. El procedimiento implica primero calcular el gradiente de la función y luego seguir el gradiente en la dirección opuesta (por ejemplo, cuesta abajo hasta el mínimo para problemas de minimización) utilizando un tamaño de paso (también llamado tasa de aprendizaje). Estos algoritmos se basan todos en el de *descenso del gradiente*, y el resto generalmente son modificaciones de este, como el *método de Adagrad*.

d) **Algoritmos de segundo orden**: Los algoritmos de optimización de segundo orden usan la segunda derivada (Hessiana) para encontrar la mejor dirección en el espacio de búsqueda. Son adecuados para funciones donde se puede calcular o aproximar la matriz Hessiana. Entre los ejemplos se incluyen el *Método de Newton* para funciones de una variable y los *Métodos Cuasi-Newton* como *BFGS*, *SQP* o *SLSQP* para funciones de más de una variable.

2. **Algoritmos que no usan la información del gradiente**: Las funciones objetivo no diferenciables presentan un desafío para los algoritmos de optimización, ya que no se puede calcular su derivada, lo que es fundamental para muchos métodos rápidos y eficientes. Podemos dividir este grupo en:

   a) **Algoritmos directos**: Los algoritmos de optimización directa son para funciones objetivo donde no se pueden calcular derivadas. Son deterministas y asumen que la función objetivo tiene un único óptimo global. Utilizan métodos de búsqueda directa, a menudo llamados "búsqueda de patrones", que pueden navegar por el espacio de búsqueda utilizando formas geométricas o decisiones. Entre los ejemplos se incluyen el *Método de Powell*, el *Método de Hooke-Jeeves* y la *Búsqueda Simplex de Nelder-Mead*.

   b) **Algoritmos estocásticos**: Los algoritmos estocásticos son para funciones objetivo donde no se pueden calcular derivadas y hacen uso de aleatoriedad en el procedimiento de búsqueda. A diferencia de los métodos de búsqueda directa deterministas, los algoritmos estocásticos implican muestreo adicional de la función objetivo, pero pueden manejar problemas con óptimos locales engañosos. Entre los ejemplos

se incluyen la *Estrategia Evolutiva* o el *Método de Entropía Cruzada*.

c) **Algoritmos de población**: Los algoritmos de población mantienen una población de soluciones candidatas para muestrear, explorar y encontrar un óptimo. Son adecuados para problemas con evaluaciones de función ruidosas y múltiples óptimos globales, donde encontrar una solución adecuada es difícil con otros métodos. Entre estos métodos se pueden destacar los *algoritmos genéticos* o la optimización por *enjambre de partículas* [4] [5].

Los algoritmos que utilizan información de gradiente inician desde un punto y avanzan hacia la solución óptima iterativamente. Una práctica común es aproximar la derivada para emplear estos algoritmos, como se verá en el método *SLSQP* de la librería Scipy de Python, que será central en este trabajo.

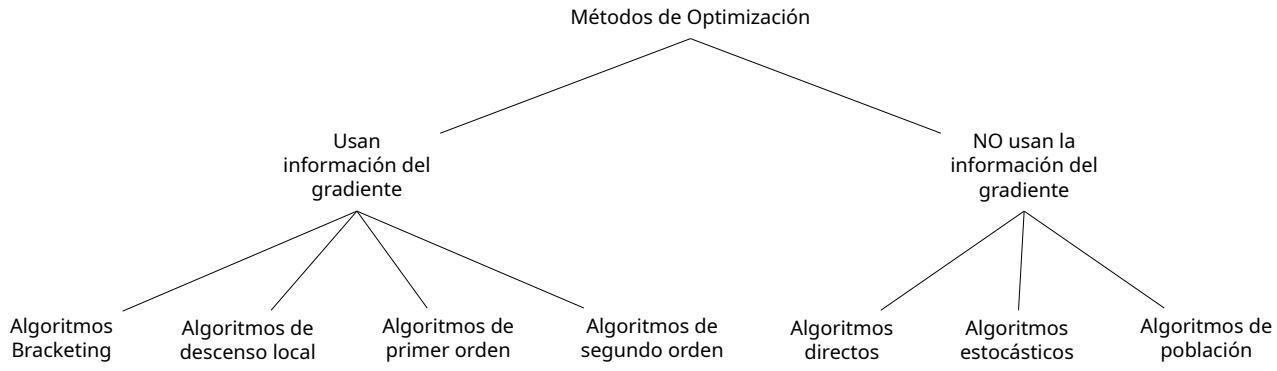


Figura 2: Árbol con los métodos de optimización

2.3. Algoritmos que usan información del gradiante

Cuando una función f es suficientemente diferenciable, es posible calcular sus derivadas de primer y segundo orden en cualquier punto. Estas derivadas nos proporcionan dos herramientas matemáticas clave: el **vector gradiente** y la **matriz Hessiana**. El vector gradiente se compone de las derivadas parciales de f y señala la dirección de mayor incremento de la función. Por otro lado, la matriz Hessiana, formada por las segundas derivadas parciales de f, indica la curvatura de la función en cada dirección.

Estos elementos son importantes, ya que en un punto óptimo —un mínimo local en nuestro

estudio— el vector gradiente debe ser un vector nulo. Además, para confirmar que este punto es un mínimo, la matriz Hessiana debe ser definida positiva, es decir, todas sus autovalores deben ser positivos, lo que asegura que la función se curva hacia arriba en todas las direcciones alrededor de dicho punto [8].

A continuación, definiremos:

- $w_0, w_1, \ldots, w_i, \ldots$ representan la secuencia de puntos generados durante las iteraciones de los métodos de optimización.
- $f_i = f(w_i)$ indica el valor de la función objetivo en el i-ésimo paso de la iteración.
- $g_i = \nabla f(w_i)$ denota el valor del gradiente de la función objetivo en el i-ésimo paso de la iteración, reflejando la dirección y magnitud del cambio más rápido de f en w_i.
- $H_i = \nabla^2 f(w_i)$ corresponde al valor de la matriz Hessiana en el i-ésimo paso de la iteración.

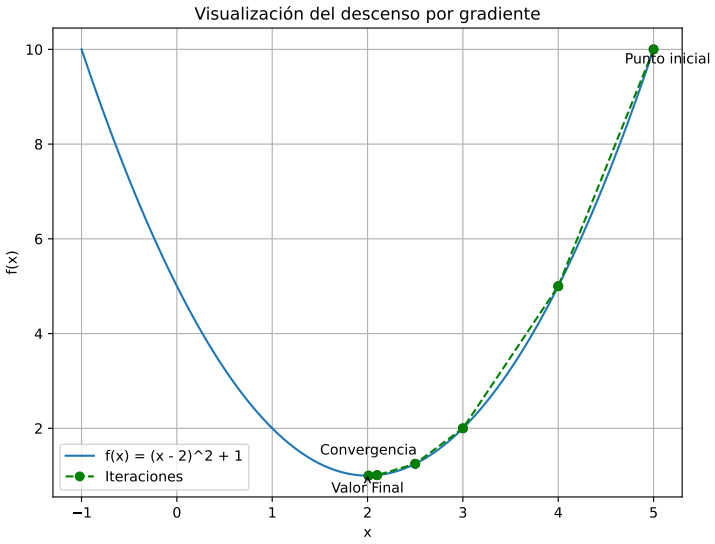


Figura 3: Descenso de gradiante tras 4 iteraciones.

2.3.1. Descenso del Gradiente

El *descenso del gradiente* es uno de los algoritmos más conocidos cuando se comienza con la optimización de problemas. Este método de *primer orden* utiliza únicamente el vector gradiente

para su operación. Para calcular el punto siguiente w_{i+1} a partir de w_i, el método avanza en la dirección opuesta al gradiente, es decir:

$$w_{i+1} = w_i - \alpha_i g_i$$

donde α_i representa el *tamaño del paso*, que puede ser un valor fijo o determinarse de manera adaptativa a través de técnicas de búsqueda de línea para optimizar el avance en cada iteración. Aunque el uso de un valor constante para α_i simplifica el algoritmo, emplear un proceso de ajuste dinámico para el tamaño del paso generalmente proporciona mejores resultados.

A pesar de su simplicidad, el descenso del gradiente tiene limitaciones significativas. En particular, para funciones con valles largos y estrechos, el método puede requerir muchas iteraciones para converger. Esto se debe a que la dirección de mayor descenso local —aunque efectivamente reduce el valor de la función objetivo— no necesariamente garantiza la convergencia más eficiente hacia el mínimo global. En la Figura 3 se muestra un descenso del gradiente con paso variable [8].

2.3.2. Método de Newton

El *método de Newton*, también conocido como *método de Newton-Raphson*, es un algoritmo de optimización de *segundo orden* que utiliza la matriz Hessiana para encontrar puntos críticos de una función objetivo. Este método busca optimizar la función haciendo uso de las derivadas segundas para encontrar direcciones de descenso más eficientes. Se fundamenta en el *desarrollo de Taylor de orden 2* de la función, f, proporcionando una aproximación alrededor de un punto inicial w_0:

$$f(w) = f_0 + g_0(w - w_0) + \frac{1}{2}(w - w_0)^T H_0(w - w_0)$$

Considerando que el gradiente g de f debe ser nulo en un mínimo, derivamos la siguiente ecuación:

$$g(w) = g_0 + H_0(w - w_0) = 0$$

El método de Newton actualiza los parámetros partiendo de un punto inicial w_0, y genera una sucesión de puntos conforme a la siguiente expresión:

$$w_{i+1} = w_i - H_i^{-1} g_i$$

Este método podría conducir a encontrar un máximo si el Hessiano no es definido positivo, situación en la cual no se garantiza una reducción del error en cada iteración. Para evitar esto, la actualización de parámetros a menudo se modifica incluyendo un parámetro de escala v, el cual puede ser fijado o determinado mediante un proceso de optimización unidimensional:

$$w_{i+1} = w_i - (H_i^{-1} g_i)v$$

Generalmente, el método de Newton requiere menos iteraciones que métodos de primer orden, como el Descenso del Gradiente, para encontrar un mínimo local debido a su uso de información de curvatura. Sin embargo, calcular el Hessiano y su inversa es computacionalmente costoso, limitando su aplicabilidad en problemas de gran escala [8] [3].

2.3.3. Método de Cuasi-Newton

Como hemos visto, el método de Newton es muy costoso desde el punto de vista computacional, ya que requiere muchas operaciones para calcular el Hessiano y su inversa. Los métodos alternativos que se han construido para resolver este problema se conocen genéricamente como *métodos Cuasi-Newton*. En estos métodos, en lugar de calcular el Hessiano directamente y después su inversa, se construye una aproximación de la inversa en cada iteración utilizando solo información sobre las primeras derivadas de la función objetivo. Si llamamos a esta aproximación G_i (en la iteración i), la fórmula de quasi-Newton se puede expresar como:

$$w_{i+1} = w_i - (G_i g_i)v$$

Los dos métodos más utilizados para la aproximación de la inversa de la Hessiana son la fórmula de *Davidon-Fletcher-Powell (DFP)* y la fórmula de *Broyden-Fletcher-Goldfarb-Shanno (BFGS)*.

Este es el método por defecto para usar en la mayoría de los casos, ya que es más rápido que los anteriores y no conlleva el inconveniente del cálculo del Hessiano y su inversa [8] [3].

2.3.4. Método de mínimos cuadrados

El método de mínimos cuadrados, también conocido como *algoritmo de Levenberg-Marquardt*, ha sido diseñado para trabajar específicamente con funciones objetivo que se expresan como

suma de funciones cuadráticas. Este método es efectivo sin necesidad de calcular la matriz Hessiana exacta, utilizando en su lugar el vector gradiente y la matriz Jacobiana.

Supongamos que la función objetivo se puede expresar como:

$$f(w) = \sum_{i=0}^{m} e_i^2(w)$$

Donde m es el número de observaciones en el conjunto de datos, y $e_i(w)$ es el sumando i-ésimo de la función objetivo.

La matriz Jacobiana de la función de error se calcula como la formada por las derivadas de los errores con respecto a los parámetros, es decir:

$$J_{ij}(w) = \frac{\partial e_i}{\partial w_j}, \quad \text{donde } i = 1, \ldots, m \text{ y } j = 1, \ldots, n$$

$$J_f = \begin{pmatrix} \frac{\partial e_1}{\partial w_1} & \frac{\partial e_1}{\partial w_2} & \cdots & \frac{\partial e_1}{\partial w_n} \\ \frac{\partial e_2}{\partial w_1} & \frac{\partial e_2}{\partial w_2} & \cdots & \frac{\partial e_2}{\partial w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_m}{\partial w_1} & \frac{\partial e_m}{\partial w_2} & \cdots & \frac{\partial e_m}{\partial w_n} \end{pmatrix}$$

El vector gradiente de la función de error se puede calcular ahora como:

$$\nabla f = 2J^T e$$

Finalmente, el Hessiano se puede aproximar por:

$$H \approx 2J^T J + \lambda I$$

(donde λ es el factor de amortiguamiento que asegura la positividad del Hessiano, e I es la matriz de identidad).

A partir de las igualdades anteriores se define el proceso de mejora de parámetros con el *algoritmo de Levenberg-Marquardt*:

$$w_{i+1} = w_i - (J^T J + \lambda I)^{-1}(2J^T e)$$

Cuando $\lambda = 0$ se obtiene el método de Newton utilizando el Hessiano aproximado. Si λ es muy grande, se convierte en el algoritmo de Descenso del Gradiente con una tasa de entrenamiento pequeña. Por ello, λ se inicializa para que sea grande, de modo que las primeras actualizaciones sean pequeños pasos en la dirección de descenso del gradiente. Si alguna

iteración genera un fallo, entonces λ se incrementa por algún factor, si no, a medida que disminuye el error, λ disminuye, de manera que el algoritmo de Levenberg-Marquardt se aproxima al método de Newton. Este proceso normalmente acelera la convergencia al mínimo [8].

2.3.5. Algoritmo SLSQP básico

Ahora ya estamos en condiciones de tener una intuición de cómo funciona el algoritmo *Sequential Least Squares Programming* (*SLSQP*). Se trata de un tipo de método secuencial de mínimos cuadrados, es decir, el problema lo transforma en un problema de mínimos cuadrados usando un método de aproximación. Gracias a esto da una aproximación de la Hessiana y usa una función de prueba L1 en el algoritmo para una mejor elección de la longitud de paso [7] [11].

2.4. Algoritmos que no usan la información del gradiante. Algoritmos genéticos

Los *algoritmos genéticos* son técnicas de optimización inspiradas en la teoría evolutiva de Darwin, que operan simulando la selección natural. Se utilizan para resolver problemas complejos en diversas áreas, como la ingeniería, la inteligencia artificial y las finanzas. Estos algoritmos representan posibles soluciones a un problema mediante estructuras conocidas como individuos o cromosomas, que forman una población inicial. A partir de esta población, los algoritmos aplican operadores como la selección, el cruce y la mutación para generar nuevas soluciones [15].

Un algoritmo genético puede entenderse como un proceso iterativo que explora un espacio de soluciones S mediante operadores probabilísticos. La población inicial es una muestra aleatoria de individuos, cada uno representando un vector $x_i \in S$ con una serie de características o *genes*. La función de aptitud, $f : S \to \mathbb{R}$, evalúa la calidad de cada individuo con respecto al problema a resolver.

El proceso de selección elige individuos con probabilidad proporcional a sus puntuaciones en $f(x_i)$, emulando la supervivencia de los más aptos. Los operadores de cruce y mutación modifican los vectores x_i para crear nuevos individuos que, combinando características de los más aptos y explorando variaciones aleatorias, permiten recorrer el espacio de soluciones

con la finalidad de maximizar (o minimizar) la función de aptitud. Así, el algoritmo busca la solución óptima moviéndose hacia mejores regiones del espacio de búsqueda de manera estocástica, siguiendo los principios evolutivos [1].

3

Obtención, Comprensión y Preprocesado de los datos

En esta sección tratamos las fase 2 y 3 de CRISP-DM, la comprensión de los datos y su preparación. Se explicarán los datos en bruto, el tratamiento de dichos datos y su orientación para el objetivo del proyecto.

3.1. Comprensión de los datos

Disponemos de los datos de consumo eléctrico de usuarios de Irlanda correspondiente al año 2009. Estos datos han sido proporcionados por *The Research Perspective Ltd* [2]. Dichos datos están sin tratar y pueden incluir errores. En este capítulo nos encargaremos de adaptarlos a nuestras necesidades, limpiándolos (desechando los posibles datos anómalos) y tomando muestras adecuadas a nuestro objetivo.

Dichos datos se encuentran en 6 archivos de texto, llamados File1.txt, File2.txt, ..., File6.txt. El formato de cada fila es de 3 columnas correspondiente a:

- Id del usuario.
- Código de 5 dígitos:
  - Día del año: Dígito del 1 al 3. Si el número excede 365, indica que el día corresponde al año siguiente. Por ejemplo, 366 indicaría el 1 de enero de 2010, del año siguiente.

- Código horario: Dígito del 4 al 5. Estos dígitos van del 1 al 48, para cada intervalo de media hora del día. Por ejemplo, 01, representa la hora 00:30.

• Consumo eléctrico en kWh en cada intervalo de media hora.

Ejemplo: $\underbrace{1392}_{\text{Id del usuario}}$ $\underbrace{19503}_{\substack{\text{Dia 195 del año, intervalo horario 03:}\\ \text{- Día: 14/07/2019}\\ \text{- Hora: 01:30}}}$ $\underbrace{0.157}_{\text{Consumo eléctrico en kWh}}$

3.2. Limpieza de datos

Una vez tenemos los datos en cada archivo de texto, tenemos que limpiarlos y adaptarlos a nuestras necesidades.

Por un lado, observamos que para algunas filas, no se dispone de la energía consumida. En estos casos, los valores de consumo energético aparecen como NaN. En estos casos, por algún motivo no se consiguió registrar el consumo energético. Es por ello, que lo más inteligente es considerar el consumo del intervalo de la hora anterior.

Luego, se presenta otra anomalía en algunos datos. Para algunos días, en el código horario aparece un número que excede el 48. En estos casos los datos son desechados.

Esta limpieza de datos es realizada en el archivo de python `energy_data_processing.py` que se facilita. Se incluyen las funciones necesarias para esta limpieza de datos.

3.3. Transformación de los datos

Los datos ya limpios se adaptan a un formato que nos interesa para tratarlos. En primer lugar, haciendo las transformaciones pertinentes, creamos un archivo cuya primera columna es el *Id* del usuario, la segunda columna corresponde al día, y las siguientes 48 columnas son los consumos cada media hora. Como el precio de la energía se paga por hora, nos interesa tener los consumos cada hora, así que simplemente sumamos las columnas dos a dos. Así, nos queda una tabla de consumos base que es la que usaremos en el resto del proyecto. Un ejemplo del resultado con 3 usuarios, del día 0 se muestra en el Cuadro 1.

Ahora bien, tenemos tantos días como son incluidos en los datos en bruto. Nuestro objetivo es estimar coeficientes en un mes, ya que el reparto de coeficientes que permite la distribuidora se hace a nivel mensual. Entonces, agruparemos los usuarios en nuevas filas, de forma que en

Cuadro 1: Ejemplo de tabla con 3 usuarios

UserId	Día	00	01	...	23
Usuario 1	0	Consumo U1 H00	Consumo U1 H01	...	Consumo U1 H23
Usuario 2	0	Consumo U2 H00	Consumo U2 H01	...	Consumo U2 H23
Usuario 3	0	Consumo U3 H00	Consumo U3 H01	...	Consumo U3 H23

la fila del Usuario 1 se muestren de la columna 0 a la 23 los consumos del primer día, de la 24 a la 47 los del segundo día y así sucesivamente. Estas transformaciones nos terminan generando una nueva tabla con un total de $24\text{h} \cdot 30$ días $= 720$ columnas, y tantas filas como usuarios existan en los datos en bruto. Un ejemplo del resultado con 3 usuarios, del día 0 se muestra en el Cuadro 2.

Cuadro 2: Ejemplo de tabla con 3 usuarios

UserId	00	01	...	719
Usuario 1	Consumo U1 H00	Consumo U1 H01	...	Consumo U1 H719
Usuario 2	Consumo U2 H00	Consumo U2 H01	...	Consumo U2 H719
Usuario 3	Consumo U3 H00	Consumo U3 H01	...	Consumo U3 H719

Una vez tenemos esta tabla, ya estamos en condiciones de avanzar a la siguiente fase. Sin adelantar acontecimientos, nuestro objetivo será obtener una matriz con las mismas dimensiones, que corresponderá a los coeficientes de los usuarios por horas que debemos minimizar. Esta matriz será la matriz de consumo, y siempre tendrá tantas filas como usuarios estemos tratando. Nótese que en la columna 24 estaremos en la hora 0 del día 2. La matriz anterior, sin las columnas de usuarios y horas, será la que llamaremos matriz de consumo, y haremos referencia en lo que sigue como E.

3.4. Datos de generación y precio de la energía

Los datos de generación de energía han sido obtenidos de la web de la Comisión Europea, *Photovoltaic Geographical Information System*, para una instalación de 10kWh [10]. Se encuentran en bruto en Wh, así que dividimos entre 1000 para transformarlo en kWh. Disponemos

de los datos de un mes, es decir, 720 horas. Así que se pueden entender como una matriz de la forma del cuadro 3:

Cuadro 3: Matriz generación de energía: $total\_e$

00	01	02	...	719
generación H00	generación H01	generación H02	...	generación H719

De forma similar, como se muestra en el cuadro 4 podemos construir la matriz de precio. Ya disponemos de la energía en euros por kWh, así que no necesitamos ninguna transformación:

Cuadro 4: Matriz precio de energía: $energy\_prices$

00	01	02	...	719
precio H00	precio H01	precio H02	...	precio H719

En lo que sigue, $total\_e$ será la matriz de generación y $energy\_prices$ la matriz de precios. Además, también se hará referencia a $nUser$ como el número de usuarios que están siendo tratados, es decir, el número de filas que tiene la matriz E. El número de columnas de E tiene que coincidir con el número de columnas de $total\_e$ y $energy\_prices$. Lo llamaremos, a partir de ahora, como $nCol$.

4

Modelado. Minimización de vertido a la red

El propósito de esta sección radica en minimizar el vertido de energía a la red eléctrica de un conjunto de usuarios que comparten la generación del sistema de placas solares. Correspondiente a la fase 4 de CRISP-DM, en esta sección usaremos algunos métodos de aproximación ya existentes, y se desarrollará un algoritmo que permite dar una aproximación rápida a la solución óptima.

Partimos de una matriz E de un total de 720 columnas y tantas filas como usuarios estén implicados en el reparto. El caso base es que, cada usuario tenga siempre el mismo coeficiente en cada hora. Este coeficiente no es más que el porcentaje de energía que le corresponde a un usuario en una determinada hora.

En este caso, ocurre que, cuando un usuario no consume energía, por ejemplo, porque dados sus hábitos siempre está fuera de casa, la energía correspondiente es vertida a la red, nadie la usa. Como ya explicamos en la introducción, la idea es que otro usuario use esa energía. Es decir, los coeficientes de los usuarios pasen a ser dinámicos. Este funcionamiento se ilustra en el Ejemplo 1:

Ejemplo 1. *Consideremos la siguiente matriz E de consumo, y la generación del sistema de placas solares de 3 horas con 1 kWh.*

0	1	2
1	1	1

Cuadro 5: Matriz $total\_e$ en kWh

	0	1	2
Usuario 0	1	0.5	0
Usuario 1	0	0.5	1

Cuadro 6: Matriz E de consumo en kWh

	0	1	2
Usuario 0	0.5	0.5	0.5
Usuario 1	0.5	0.5	0.5

Cuadro 7: Matriz c de coeficientes no dinámicos

	0	1	2
Usuario 0	1	0.5	0
Usuario 1	0	0.5	1

Cuadro 8: Matriz c de coeficientes dinámicos

Las tablas anteriores son un ejemplo reducido de las tablas explicadas en la sección anterior. Las matrices E y c tienen 3 columnas y 2 filas, es decir, 3 horas y 2 usuarios. La matriz total_e tiene 3 columnas (horas), que debe coincidir con el número de columnas del resto de matrices. Esta matriz, como ya se observó, siempre tiene una única fila. La energía que vierte cada usuario es su coeficiente multiplicado por la generación menos su consumo, en caso de ser positivo. En caso de ser negativo no hay vertido, simplemente consume más de lo que las placas pueden generan. Esta fórmula, que se detallará mas abajo es la siguiente:

$$Min \sum_{j=1}^{nCol} \sum_{i=1}^{nUser} \text{máx}(0, c_{ij} \cdot total\_e_j - E_{ij})$$

donde c es la matriz de coeficientes y E la matriz de consumo.
Evaluando la expresión anterior con la matriz de coeficientes no dinámicos observamos que las horas 0 y 2 se está vertiendo energía, en total 1 kWh. Sin embargo, usando la matriz de coeficientes dinámicos, no se vierte energía. Además, cada usuario consume la mitad de la generación y la suma de los coeficientes en cada hora es 1.

Ya hemos hecho nuestro primer ejemplo de optimización. El objetivo de esta sección es ver cómo obtenemos la matriz de coeficientes dinámicos, verificando ciertas restricciones. Todo el texto de esta sección se irá acompañando de varios ejemplos, para facilitar la comprensión de lo que se expone, pero teniendo en cuenta que el caso real y práctico es de un mes completo.

Además, es importante el concepto de *coeficiente del sistema* o *participación del sistema*. El coeficiente del sistema que tiene un usuario hace referencia al porcentaje de energía que le corresponde a cada usuario. Estos coeficientes pueden ser equitativos (como en el Ejemplo 1) o no, pero lo que es claro es que la suma de estos coeficientes del sistema ha de ser 1. Si llamamos sys_coef a estos coeficientes, se debe verificar:

$$\sum_{i=1}^{nUser} sys\_coef_i = 1$$

En la sección se hará referencia a estos coeficientes. Sin importar si son equitativos o no, siempre se asume que se verifica la condición anterior.

4.1. Casos para evitar vertido

Disponemos de los datos de producción energéticos por horas, y de los consumos por usuarios. Para simplificar los datos haremos una serie de modificaciones:

1. Si la producción en una hora es nula, eliminamos dicha hora (o columna) de la matriz de consumo E y de la $total\_e$. Estos casos son habituales, pues por la noche las placas solares no generan electricidad.

2. Si la producción en una hora es mayor o igual que la suma de la energía que consumen los usuarios. En este caso, es inevitable el vertido a la red. Sin embargo, es conveniente tratar los datos de producción como si en esa hora se hubiera producido el total del consumo de los usuarios. Es decir, si

   $$total\_e_j \geq \sum_{i=1}^{nUser} E_{ij},$$

   para algún $j \in nCol$, entonces se reasigna el valor de $total\_e_j$ para igualarlo al consumo total en esa hora:

   $$total\_e_j = \sum_{i=1}^{nUser} E_{ij}.$$

Gracias a cómo hemos definido los datos, el vertido se puede minimizar a cero en un caso ideal, pues los casos de posible vertido que son el punto 2 anterior se han redefinido para que el vertido se pueda minimizar a 0. El punto 1 es útil para simplificar el problema. El Ejemplo 2 muestra estos cambios:

Ejemplo 2. *Ejemplo de producción y consumo para 4 horas. A continuación, en el cuadro 9 y 10 se muestran los datos originales.*

Cuadro 9: Tabla de consumo con 3 usuarios y 4 horas

	00	01	02	03
Usuario 0	0.23	0.5	1.0	2.1
Usuario 1	0.45	0.4	1.2	2.2
Usuario 2	2.43	3.1	1.3	1.6

Cuadro 10: Tabla de producción en 5 horas

00	01	02	03
0	0	4	2.3

Las tablas resultantes tras el tratamiento de los datos son las mostradas en el cuadro 11 y 12:

Cuadro 11: Tabla de consumo con 3 usuarios reducido a 2 horas

	02	03
Usuario 1	1.0	2.1
Usuario 2	1.2	2.2
Usuario 3	1.3	1.6

Cuadro 12: Tabla de producción reducida a 2 horas

02	03
3.5	2.3

Se puede observar en el cuadro 12 como las primeras dos columnas con producción nula han sido eliminadas, la tercera columna se mantiene, pero se actualiza su valor de generación y la cuarta columna no sufre ningún cambio. Del cuadro 11 han sido eliminadas las mismas columnas que del cuadro 12.

Es por esto que, a pesar de que inicialmente teníamos una matriz E y $total\_e$ con 720 columnas, tras estos cambios las dimensiones de estas matrices se reducen. Sin pérdida de generalidad, continuaremos llamando a partir de ahora como $nCol$ al número de columnas reducido, junto a E y $total\_e$ como las nuevas matrices sin las columnas pertinentes.

4.2. Consumo de cada usuario

Como ya indicamos al principio de la sección, cada usuario tiene asociado un coeficiente o participación del sistema. Este coeficiente hace referencia a lo que le corresponde consumir a cada usuario. Se puede interpretar como que, de un total de energía generada mensualmente, cada usuario debe consumir su coeficiente multiplicado por esa energía. Si llamamos $consumption_i$ al consumo del usuario i, podríamos escribir como:

$$consumption_i = sys\_coef_i \cdot total\_gen \tag{1}$$

donde $tot\_gen = \sum_{j=1}^{nCol} total\_e_j$ y hace referencia a la generación total del sistema en el mes. Por recapitular, $total\_e_j$ es el array de generación diaria y tot_gen es la suma de esta generación.

De esta forma es natural pensar que una restricción del problema de optimización es que cada usuario consuma según el consumo asociado a su coeficiente de sistema. Sin embargo, puede ocurrir que a un usuario le pertenezca más consumo del total que consume en ese mes. Si no hiciéramos nada, se estaría vertiendo energía a la red, y nuestro objetivo es minimizarlo, por tanto, en estos casos se propone un reparto alternativo.

La idea es que si un usuario consume menos de lo que le corresponde, se sature el consumo de ese usuario a lo que consume ese mes, y la diferencia sea repartida de forma equitativa entre el resto de usuarios. Se hace de la misma forma si hay más de un usuario cuyo consumo está por debajo de la energía producida que le corresponde. En el caso irreal de que todos los usuarios consuman menos de lo que se consume, el vertido será inevitable.

Por ello, se puede definir el consumo del i-ésimo usuario como:

$$consumption_i = \begin{cases} \sum_{j=1}^{nCol} E_{ij} & \text{si } tot\_gen \cdot sys\_coef_i \geq \sum_{j=1}^{nCol} E_{ij}, \\ tot\_gen \cdot sys\_coef_i + \frac{dif}{nUserExc} & \text{si } tot\_gen \cdot sys\_coef_i < \sum_{j=1}^{nCol} E_{ij}. \end{cases}$$

donde $dif = \text{máx}(0, tot\_gen \cdot sys\_coef_i - \sum_{j=1}^{nCol} E_{ij})$ y $nUserExc$ es el número de usuarios que tienen consumo mayor que la energía producida que les corresponde y que, por tanto, se intenta repartir.

Observación 1. *Si ningún usuario tiene consumo menor que lo que le corresponde, entonces $dif = 0$ y la expresión anterior es equivalente a la de la ecuación 1.*

Observación 2. *Es posible que durante el proceso de redistribución del consumo, un usuario que originalmente consumía menos energía de la que le correspondía pueda terminar excediendo su asignación debido a la redistribución. Para prevenir que esto suceda, la fórmula descrita se ejecuta repetidamente en un bucle iterativo. Este bucle asegura que ningún usuario exceda la cantidad de energía que le corresponde, ajustando continuamente la asignación hasta que todas las asignaciones de consumo sean correctas y no se superen los límites establecidos.*

4.3. Problema de minimización

Ahora ya estamos en condiciones de poder definir el problema de minimización. Tenemos que minimizar el vertido a la red encontrando una matriz c con las mismas dimensiones que la matriz E. Esta matriz será la matriz de coeficientes que buscamos optimizar. Con un uso de coeficientes no dinámicos, la matriz de coeficientes c se reduce a que cada columna es igual que los coeficientes del sistema. Es decir, $c_{ij} = sys\_coef_i$ para todo $j \in nCol$.

Ejemplo 3. *Diferencia entre una tabla de coeficientes estáticos y una con coeficientes dinámicos.*

	0	1	2	3	4
Usuario 0	0.2	0.2	0.2	0.2	0.2
Usuario 1	0.3	0.3	0.3	0.3	0.3
Usuario 2	0.5	0.5	0.5	0.5	0.5

Cuadro 13: Coeficientes estáticos

	0	1	2	3	4
Usuario 0	0.1	0.2	0.2	0.5	0.8
Usuario 1	0.1	0.2	0.4	0.4	0.1
Usuario 2	0.8	0.6	0.4	0.1	0.1

Cuadro 14: Coeficientes dinámicos

Nótese que la suma de cada columna siempre debe ser 1. Y naturalmente, los coeficientes son positivos, pues están acotados en 0 y 1.

Por tanto, una primera restricción a nuestro problema será.

$$\sum_{i=1}^{nUser} c_{ij} = 1, \forall j \in nCol$$

Otra restricción es la que viene por parte del consumo. Cada usuario i debe terminar recibiendo lo calculado en el array $consumption$, es decir:

$$\sum_{j=1}^{nCol} c_{ij} \cdot total\_e_j = consumption_i, \forall i \in nUser$$

Partiendo de una matriz c no óptima, el vertido se produce cuando $c_{ij} \cdot total\_e_j > E_{ij}$. Por tanto, se puede modelar el vertido total a la red como sigue, con la que será la función objetivo del problema:

$$\text{Min} \sum_{j=1}^{nCol} \sum_{i=1}^{n} \text{máx}(0, c_{ij} \cdot total\_e_j - E_{ij})$$

De esta forma, el problema de optimización quedaría como:

$$\begin{aligned} \text{Min} \quad & \sum_{j=1}^{nCol} \sum_{i=1}^{nUser} \text{máx}(0, c_{ij} \cdot total\_e_j - E_{ij}) \\ \text{s.a.} \quad & \sum_{i=1}^{nUser} c_{ij} = 1, \quad \forall j \in nCol \\ & \sum_{j=1}^{nCol} c_{ij} \cdot total\_e_j = consumption_i, \quad \forall i \in nUser \end{aligned}$$

Observación 3. *Nótese que se ha omitido la restricción de que los coeficientes deben ser positivos. Esto se debe a que la segunda restricción tiene implícita esta condición. Dado que* $consumption_i$ *y* $total\_e_j$ *son no negativos, para todo* $i \in nUser$ *y para todo* $j \in nCol$.

Observación 4. *Las variables a calcular son la matriz* c *de dimensiones* $nUser \times nCol$.

Observación 5. *Se trata de un problema no lineal, pues la función máximo no es lineal en algunos puntos. Además, tiene un gran número de variables y restricciones. Tiene tantas variables como celdas la matriz de coeficientes, y tiene* $nCol + nUser$ *restricciones.*

Observación 6. *En el texto a menudo se hará referencia como restricciones de fila y restricciones de columna, entendiéndose siempre por todas las* $nUser$ *restricciones de fila y todas las* $nCol$ *restricciones de columna.*

4.4. Cotas superiores e inferiores

Dada la complejidad del problema, es útil obtener información adicional de la solución de este. Por cómo hemos modelado los datos, eliminado las horas de no producción y saturando la producción al consumo en caso de que se generase más de lo que se consumía, sabemos que una cota inferior del problema es 0. Es decir, se buscará siempre que el vertido este lo más cerca posible de 0, y si se alcanza 0, sabremos que se trata de una solución óptima. Esto no quiere decir que siempre sea posible minimizar el vertido a 0, ya que pueden existir combinaciones de coeficientes que no permitan esto. Pero gracias a esta cota inferior, podremos saber qué tan cerca o lejos estamos de la solución óptima.

Por otro lado, también podemos considerar una cota superior. Esta cota puede venir dada por evaluar la función objetivo con la matriz de coeficientes estáticos, que sería la opción más ingenua. De esta forma, podremos ver el vertido por defecto, y observar cómo mejora y se acerca a 0 gracias al proceso de optimización.

4.5. Optimizadores

Para abordar el problema de optimización con múltiples variables y una función objetivo no lineal evaluada en un número limitado de puntos, se evaluaron diferentes métodos de optimización, seleccionando aquellos que mejor se adaptan a las restricciones y características del problema.

Inicialmente, se descartaron los métodos que requieren información detallada del gradiente, como los algoritmos de bracketing y descenso local, debido a las dificultades inherentes de calcular derivadas precisas en funciones no lineales. Los algoritmos de primer orden, que también dependen de gradientes precisos, no eran viables dada la complejidad de la función objetivo, y el gran número de variables.

El enfoque se centró entonces en los métodos de segundo orden, en particular el algoritmo *Sequential Least Squares Programming* (*SLSQP*). Este método es particularmente adecuado para problemas no lineales, ya que incorpora un aproximador de la derivada, facilitando la optimización cuando los cálculos directos del gradiente y el hessiano son de alto coste computacional, y en algunos puntos ni siquiera se puede calcular. *SLSQP* ajusta la búsqueda de la solución óptima utilizando un modelo cuadrático basado en la aproximación del hessiano, lo

cual es ideal en contextos donde las evaluaciones de la función son limitadas y costosas.

Además, se experimentó con algoritmos genéticos, una técnica que no utiliza información del gradiente y es capaz de explorar ampliamente el espacio de soluciones. Sin embargo, los resultados obtenidos con este método no fueron satisfactorios, probablemente debido a la dificultad de configurar adecuadamente sus parámetros y su naturaleza estocástica que podría no converger eficientemente, sumado a la gran cantidad de variables que el problema tiene.

Con estos factores en consideración, *SLSQP* fue la opción más prometedora para nuestro problema específico. Su capacidad para manejar eficientemente la no linealidad y trabajar con aproximaciones de la matriz Hessiana justificó su elección sobre otras alternativas posibles, además de mostrar resultados prometedores en las primeras evaluaciones.

4.5.1. Optimizador SLSQP

Para abordar la optimización usando *SLSQP* usamos la biblioteca Scipy de python, dentro del submódulo Optimize podemos encontrar la función minimize. Esta función incluye diferentes métodos como *COBYLA*, *BFGS*... Pasando como parámetro el método *SLSQP* podemos seleccionarlo. También debemos pasar como parámetro a la función minimize nuestra función objetivo, y las restricciones [16].

Realmente no es necesario seleccionar el método *SLSQP*, ya que, cuando pasas restricciones al problema, automáticamente se selecciona este método. Esto se debe a que *SLSQP* es uno de los métodos existentes mas potentes para optimizar funciones con restricciones. Además, permite el uso de funciones no lineales gracias al solucionador de mínimos cuadrados. El resto de métodos que proporcionaba la librería no trataban problemas con restricciones [13].

El método también necesita un punto inicial para comenzar la ejecución del algoritmo. Este punto inicial lo escogeremos aleatoriamente, haciendo que al menos se verifique la restricción de las columnas. Con el paso de las iteraciones del método, este ajustará el resto de restricciones.

Si llamamos a $n = nUser \cdot nCol$, que no es más que el número de variables que tiene nuestro problema, la complejidad espacial del algoritmo es $O(n^2)$, debido a que necesita almacenar la matriz Hessiana. La complejidad temporal del algoritmo no es fácil de obtener, ya que es complejo y están involucrados diferentes métodos. La información de que complejidad tiene tampoco es abundante, pero se estima que está entre $O(n^2)$ y $O(n^3)$ [12].

Para iniciar el método necesitamos un punto, sobre el que tras cada iteración el método va a devolver una solución mejor a la óptima, al igual que hace el descenso del gradiente en la Figura 3. Esto será importante para las próximas secciones.

4.5.2. Método de aproximación

Para mejorar el proceso de optimización, se ha desarrollado un algoritmo lineal que consigue una aproximación de la optimización. La idea es conseguir una solución, que no se aleje mucho de las restricciones y que sea capaz de dar una solución cercana a la óptima. El método esta inspirado en los métodos usados para problemas de transporte. Estos problemas son similares al nuestro, con muchas variables y restricciones, aunque difieren de nuestro problema en la forma de las restricciones y de las funciones objetivos. Por este motivo, no se pudo optar por usar un método que optimice problemas de transporte, pero sí nos podemos basar en el funcionamiento de estos para crear uno personalizado.

La idea de los métodos que resuelven problemas de transporte es la dar valores a los coeficientes de la matriz c siguiendo un cierto algoritmo. Entre ellos podemos destacar los métodos *de la esquina noroeste* o el *método de Vogel*, que son los que permiten dar una solución óptima al problema de transporte [9].

En nuestro caso, vamos a dar valores a la matriz c, luego comprobaremos que tal se verifican las restricciones, y aplicaremos un suavizado para mejorar aún más la solución. Dicho algoritmo se facilita con el nombre de `coefficient_aproximation`, y usa las librerías pandas y numpy de Python.

Las entradas del algoritmo son:

- "E": Matriz que representa datos específicos de consumo por usuario.
- "total_e": Vector con la generación de energía.
- "max_iterations": Número máximo de iteraciones para la fase de corrección (por defecto 1000).
- "p": Tasa de corrección (por defecto 0.01).

El funcionamiento del algoritmo se detalla a continuación. Será acompañado de un ejemplo básico preparado y con menos columnas de los casos reales para entender el funcionamiento.

Ejemplo 4. *Supongamos una matriz E de consumo con 3 usuarios y 4 columnas y una matriz de generación. Además se usa un reparto equitativo para los coeficientes del sistema. Es decir, a cada usuario le corresponde un tercio de la energía generada:*

	0	1	2	3
Usuario 0	1	2	2	1
Usuario 1	2	1	1	2
Usuario 2	1	1	1	1

Matriz E de consumo

0	1	2	3
2	4	4	2

Matriz $total\_e$

Como a cada usuario le corresponde un tercio de la generación, el array $consumption$ *es:*

Usuario 0	4
Usuario 1	4
Usuario 2	4

Array $consumption$

Y el total generado es:

$$tot\_gen = \sum_{j=0}^{3} total\_e_j = 12$$

1. **Inicialización**:

   1.1 Determinar las dimensiones de la matriz E (número de usuarios y columnas).

   1.2 Inicializa una matriz de coeficientes c con las mismas dimensiones de E, con todas las celdas a 0, como se muestra en el Cuadro 15.

	0	1	2	3
Usuario 1	0	0	0	0
Usuario 2	0	0	0	0
Usuario 3	0	0	0	0

Cuadro 15: Matriz c inicializada a 0

2. **Asignación de coeficientes iniciales**: Iteramos sobre cada columna de la matriz E.

2.1 Para cada columna, calcula el coeficiente máximo permitido sin superar la unidad. Esto podemos hacerlo con la siguiente fórmula, dado un $j \in nCol$ y un usuario $i \in nUser$

$$max\_coef_i = min(1, \frac{E_{ij}}{total\_e_j})$$

El array max_coef contiene los coeficientes máximos permitidos para cada usuarios. El uso de la función mínimo es para evitar que un usuario tenga un coeficiente que exceda 1. Es decir, este array indica qué coeficiente podría tener el usuario i en un caso de que toda la energía se le diera a él. Cuando exceda tendrá 1, y cuando se quede por debajo de 1 tendrá el coeficiente máximo para suplir su consumo.

El llamarlo max_coef procede de que es el coeficiente máximo que un usuario puede tener. En caso de que sea superior al coeficiente calculado anteriormente, dicho usuario está vertiendo energía.

En nuestro ejemplo, max_coef es un array con los consumos máximos de cada usuario. Nótese que el uso del mínimo es porque un usuario no puede tener un coeficiente superior a 1.

Usuario 0	0.5
Usuario 1	1
Usuario 2	0.5

Matriz max_coef de la columna 0

2.2 Se calculan los consumos temporales. Esto lo hacemos con la siguiente fórmula, dado un usuario $i \in nUser$

$$temp\_consp_i = \sum_{i=j}^{nCol} total\_e_j \cdot c_{ij}$$

Al igual que en caso anterior, temp_consp representa un array y la posición i es el consumo temporal del usuario i. Al escribir c_i hacemos referencia a la matriz de coeficientes que vamos rellenando de forma iterativa, en este caso a la del Cuadro 15.

En nuestro ejemplo, inicialmente todos los coeficientes son 0, luego el consumo temporal es 0.

Usuario 0	0
Usuario 1	0
Usuario 2	0

Matriz $temp\_consp$ en la iteración 0

2.3 Ordenamos $temp\_consp$ de menor a mayor, y guardamos estos índices en el array índices. En caso de tener el mismo valor, lo ordenamos por índice. Por tanto, en nuestro ejemplo el orden es 0, 1, 2. En la implementación este orden es almacenado en un array, aunque aquí no mostraremos tablas para indicarlo.

2.4 Comenzamos a completar la columna j de la matriz de coeficientes, en el orden de los índices anteriores, y asegurándonos de que se verifica la restricción de columnas. Para ello, comenzamos rellenando la matriz c_{ij} por los valores del array índices. Si la suma de la columna es 1, terminamos. Si no es uno, continuamos rellenándola hasta que sea 1. Hay que asegurarse de no exceder 1, por tanto, en caso de que se exceda, pondremos de coeficiente el restante para completar 1.

	0	1	2	3
Usuario 0	0.5	0	0	0
Usuario 1	0.5	0	0	0
Usuario 2	0	0	0	0

Matriz c tras la iteración 0

Seguimos el orden. Primero completamos el índice mayor del usuario 0, por tanto, ponemos en su celda su coeficiente máximo. El siguiente en el orden es el usuario 1, que tiene de coeficiente máximo 1. Como 0.5 + 1 excede 1, saturamos esa celda con el máximo posible para que sume 1, es decir, 0.5. Obsérvese que la columna suma 1, verificando la restricción de columna.

2.5 Volvemos al punto 2.1.

Continuamos con las 4 iteraciones del ejemplo para ilustrar el funcionamiento:

- *Iteración 1: Orden 2, 0, 1 (temp_consp ordenado de menor a mayor).*

Usuario 0	1
Usuario 1	1
Usuario 2	0

$temp\_consp$, iteración 1

Usuario 0	0.5	0.5	0	0
Usuario 1	0.5	0.25	0	0
Usuario 2	0	0.25	0	0

Matriz c, iteración 1

Usuario 0	0.5
Usuario 1	0.25
Usuario 2	0.25

max_coef, iteración 1

Nótese que a la hora de obtener el orden, si los valores de $temp\_consp$ coinciden su seguir como criterio de ordenación el usuario que tenga índice i menor.

- *Iteración 2: Orden 2, 1, 0 ($temp\_consp$ ordenado de menor a mayor).*

Usuario 0	3
Usuario 1	2
Usuario 2	1

$temp\_consp$, iteración 2

Usuario 0	0.5	0.5	0.5	0
Usuario 1	0.5	0.25	0.25	0
Usuario 2	0	0.25	0.25	0

Matriz c, iteración 2

Usuario 0	0.5
Usuario 1	0.25
Usuario 2	0.25

max_coef, iteración 2

- *Iteración 3: Orden 2, 1, 0 ($temp\_consp$ ordenado de menor a mayor).*

Usuario 0	5
Usuario 1	3
Usuario 2	2

$temp\_consp$, iteración 3

Usuario 0	0.5	0.5	0.5	0
Usuario 1	0.5	0.25	0.25	0.5
Usuario 2	0	0.25	0.25	0.5

Matriz c, iteración 3

Usuario 0	0.5
Usuario 1	1
Usuario 2	0.5

max_coef, iteración 3

Ahora bien, los resultados de la aproximación son los siguientes: Con coeficientes constantes, es decir, $c_{ij} = 1/3$, el vertido es de 1.33 kWh. Sin embargo, usando los coeficientes calculados, el vertido es de 0 kWh. En cuanto a las restricciones, las restricciones de que la columna sume 1 son verificadas por construcción del método, pero, en las restricciones de fila obtenemos estos resultados:

	Consumo	Consumo óptimo	Error	Desvío
Usuario 1	5	4	1	25 %
Usuario 2	4	4	0	0 %
Usuario 3	3	4	-1	25 %

Cuadro 16: Restricciones de fila

Los cálculos anteriores los omitimos para facilitar la lectura, pero se obtienen evaluando los coeficientes en la función objetivo, y para los consumos de cada usuario, es la suma de la multiplicación de cada coeficiente por la generación de cada hora.

Como podemos observar, estamos lejos de verificar las restricciones de fila, es por ello que se desarrolló la siguiente fase, la fase de corrección.

3. **Fase de corrección**: Esta fase del algoritmo es crucial para asegurar que se cumplan las restricciones de fila del problema, ajustando los coeficientes para que se adhieran

tanto a las restricciones individuales de consumo de los usuarios como a las restricciones generales del sistema.

3.1 **Inicialización de Variables:**

- "exceed": Variable booleana que indica si se ha excedido alguno de los límites de los coeficientes en las iteraciones previas.
- "l": Factor de corrección inicializado en 1, ajustable en cada iteración para controlar la magnitud de la corrección aplicada.
- "error": Array que almacena el error calculado para cada usuario, indicando la desviación del consumo real del consumo deseado.

3.2 **Cálculo del error por usuario:** En cada iteración, el error se calcula como:

$$error_i = \sum_{j=1}^{nCol} (total\_e_i \cdot c_{ij}) - consumption_i$$

Este paso evalúa cuánto se aleja cada usuario de cumplir su restricción de consumo asignada.

En nuestro ejemplo, este error es:

Usuario 0	1
Usuario 1	0
Usuario 2	-1

Array de $error$

3.3 **Evaluación y ordenación de errores:** Los errores se ordenan de menor a mayor para identificar a los usuarios con el mayor y menor cumplimiento de sus restricciones de consumo.

En nuestro caso, el usuario con mayor error es 0 y el que tiene menor error es 2.

3.4 **Cálculo de la diferencia de error:** Se determina la diferencia de errores entre el usuario con el mayor exceso y el de mayor déficit, y se divide por dos para obtener el valor medio necesario para la corrección:

$$dif = \frac{error_{max} - error_{min}}{2}$$

3.5 **Ajuste del factor de corrección "l":** Si la variable "exceed" es verdadera, se incrementa "l" por una tasa de corrección predeterminada para afinar la corrección en las siguientes iteraciones.

3.6 **Verificación del umbral de diferencia:** Si la diferencia "dif" es menor que 10^{-4}, el proceso se detiene, asumiendo que la solución actual es adecuada.

3.7 **Aplicación de la corrección:** La cantidad de corrección se calcula utilizando la fórmula:

$$correction\_amount = \frac{dif}{nCol} \cdot \frac{1}{l}$$

Esta cantidad se intenta aplicar a los coeficientes de los usuarios implicados en las columnas donde sea posible hacerlo sin violar los límites de los coeficientes, que deben permanecer dentro del rango [0, 1]. La corrección se suma al usuario con el error más bajo y se resta al usuario con el error más alto en cada columna, siempre y cuando dichos ajustes no resulten en coeficientes fuera de los límites establecidos.

Si en alguna columna específica, aplicar la corrección completa llevaría a que alguno de los coeficientes exceda los límites de 0 o 1, la corrección en esa columna no se realiza. Esto asegura que todos los coeficientes permanezcan válidos y dentro de los rangos permitidos.

Debido al paso 3.5, si durante una iteración se detecta que no es posible aplicar la corrección en alguna columna debido a que se excederían los límites, la variable booleana "exceed" sea activa y el factor de corrección "l" se incrementa. Este aumento en "l" reduce la magnitud de la corrección aplicada en iteraciones futuras. Este ajuste es necesario para garantizar que las correcciones futuras sean más finas y menos propensas a exceder los límites, permitiendo un ajuste gradual.

Esta estrategia de corrección iterativa permite una convergencia progresiva hacia una solución que cumpla con todas las restricciones.

3.8 **Iteración continua:** El algoritmo retorna al paso 3.2 para aplicar nuevas correcciones si es necesario, asegurando una optimización continua hasta alcanzar el número máximo de iteraciones o la verificación de restricción de filas óptima.

Tras aplicar este método, obtenemos la matriz de coeficientes actualizada que demuestra

una adecuada verificación de las restricciones de fila, como se muestra en la tabla de la matriz c tras la corrección y en la verificación de la función objetivo, que indica un vertido cero.

En este caso, no seguiremos el ejemplo ya que la complejidad de los cálculos aumenta, sin embargo, ejecutando el programa obtenemos la siguiente matriz de coeficientes:

	0	1	2	3
Usuario 1	0.375	0.375	0.375	0.125
Usuario 2	0.5	0.25	0.25	0.5
Usuario 3	0.125	0.375	0.375	0.375

Matriz c tras la fase de corrección

Si se evalúa esta matriz en la función objetivo, se observa que el vertido es 0. Además, la solución verifica la restricción de filas como sigue:

	Consumo	Consumo óptimo	Error	Desvío
Usuario 0	4	4	0	0 %
Usuario 1	4	4	0	0 %
Usuario 2	4	4	0	0 %

Cuadro 17: Restricciones de fila

El ejemplo que ha acompañado todo el algoritmo nos muestra varias observaciones a destacar.

Observación 7. *El método verifica por construcción la restricción de columnas.*

Observación 8. *Gracias a la fase de corrección, la restricción de filas se verifica con muy poco error (cuadro 17) frente al error obtenido en el cuadro 16.*

El algoritmo puede terminar porque se alcance un resultado de precisión lo suficientemente bueno, o puede alcanzar el número máximo de iteraciones, max_iterations. Dicho valor se puede configurar dependiendo del tamaño del problema.

Observación 9. *La complejidad es $O(n)$, pues si $n = nUser \cdot nCol$, se tiene:*

1. *Inicialización de la matriz:* $O(n)$

2. *Asignación de coeficientes iniciales:* $O(n)$

3. *Fase de corrección:* $O(max\_iter)$

Sumando todas las etapas, la complejidad temporal es $O(n)$.

4.5.3. Optimizador SLSQP Inicializado

Ahora partimos de la idea de que *SLSQP* necesita un punto inicial sobre el que comienza el algoritmo. Este punto, en el caso de la sección 4.5.1 lo estamos inicializando de manera aleatoria. Ya que hemos construido un método rápido que da una buena aproximación, usaremos este mismo punto para inicializar de nuevo el método *SLSQP*. Esto es lo que llamaremos como *SLSQP Inicializado*.

Si bien no podemos garantizar en todos los casos que eligiendo un punto más cercano a la solución óptima lleguemos antes a ella, como se verá en el próxima capitulo esto prácticamente siempre ocurre. No podemos garantizarlo porque el espacio en el que se encuentran nuestras variables es un espacio con muchas dimensiones, y puede ser, que a pesar de que la solución óptima y la generada por el método anterior estén cerca, para llegar a ella haya que dar un "rodeo" porque hay pendiente positiva hacia la solución óptima.

4.6. Problema de optimización con el precio de la energía

El precio de la energía fluctúa dependiendo del día y la hora, es por ello que un reparto por energía a los usuarios puede que no sea lo más justo, ya que, si un usuario consume toda su energía a horas donde la energía es notablemente más barata, a pesar de que ese usuario consuma del sistema su energía correspondiente a final de mes, en términos de euros ahorrados, esta ahorrando mucho menos que otros usuarios que consumen a horas más caras. Es por ello

que se plantea un problema de optimización similar, incluyendo el precio de la energía:

$$\text{Min} \quad \sum_{j=1}^{nCol} \sum_{i=1}^{nUser} \text{máx}(0, c_{ij} \cdot total\_e_j \cdot energyPrice_j - E_{ij} \cdot energyPrice_j)$$

$$\text{s.a.} \quad \sum_{i=1}^{nUser} c_{ij} = 1, \forall j \in nCol$$

$$\sum_{j=1}^{nCol} c_{ij} \cdot total\_e_j = cost_i, \quad \forall i \in nUser$$

donde $cost_j$ representa el coste en euros que cada usuario debe obtener cada mes. En el problema de minimizar el vertido de energía tratábamos de evitar que se vertieran kWh a la red. En este caso, la idea es evitar "verter euros" a la red.

Para calcular $cost_j$ seguimos una fórmula similar a la de $consumption$:

$$\sum_{j=1}^{nCol} cij \cdot total\_e_j \cdot energyPrice_j = cost_i, \forall i \in nUser$$

El método de aproximación se adapta fácilmente a este caso. La única diferencia es que max_coef ahora está en función del precio, y que el consumo temporal del paso 2.2 ahora es coste temporal:

$$max\_coef_i = min(1, \frac{E_{ij}}{total\_e_j \cdot energyPrice_j})$$

$$temp\_cost_i = \sum_{i=j}^{nCol} total\_e_j \cdot c_{ij} \cdot energyPrice_j$$

5
Evaluación

Esta sección corresponde con la fase de evaluación de CRISP-DM. A continuación, explicaremos las ventajas y desventajas de los métodos propuestos anteriores, compararemos su complejidad, tiempo de ejecución, mejoras en la aproximación, comportamiento ante diferentes perfiles.

5.1. Método de Aproximación

Ahora analizaremos los resultados del método de aproximación desarrollado en la sección anterior. Este método, como ya se mencionó, es de complejidad $O(n)$, por tanto, ofrece resultados muy rápidos para cualquier número de usuarios. Para el estudio de este método hay dos factores a tener en cuenta: cómo de bien verifica las restricciones de nuestro problema y cómo de cerca se queda de dar una solución óptima.

5.1.1. Evolución con el número de usuarios

A medida que aumentamos el número de usuarios, el método tarda más en ejecutarse, pues tiene más coeficientes que optimizar. Además, el tiempo de ejecución del método viene directamente ligado al número de iteraciones que emplee la fase de corrección. Si se alcanza una configuración que verifique la tolerancia establecida el método puede acabar antes. Sin embargo, estudiamos ahora cómo es el error cometido en la restricción de filas, ya que la restricción de columnas se verifica siempre por construcción. Para ello, se ha simulado una batería de pruebas, con 20 ejemplos por usuarios, comenzando desde 4 usuarios hasta 35.

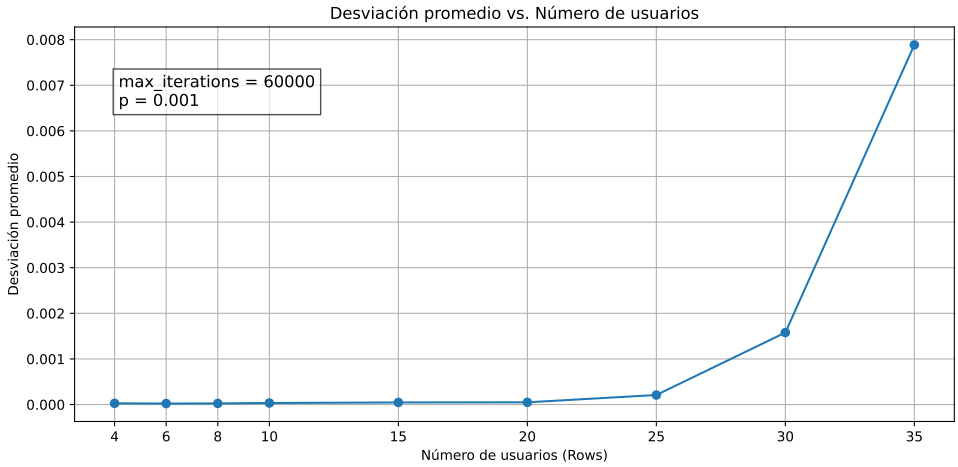


Figura 4: Evolución del error de la restricción de filas

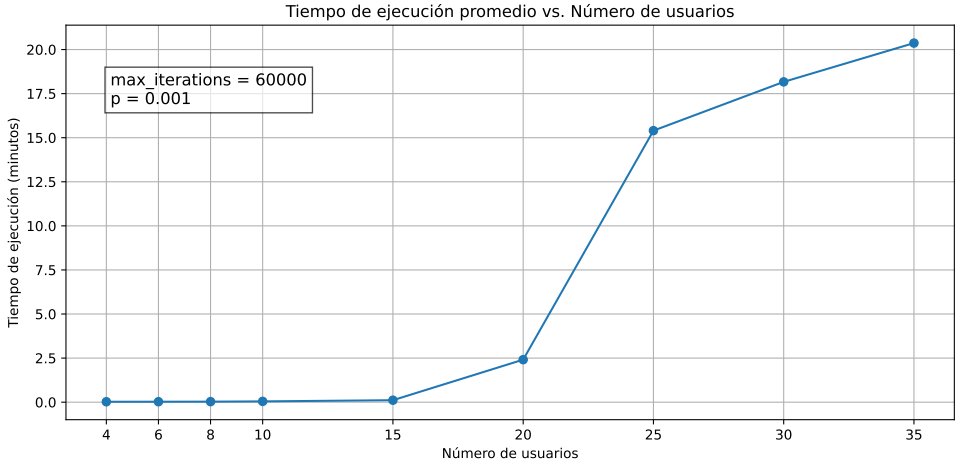


Figura 5: Evolución del tiempo de ejecución

La anterior Figura 4 muestra el error cometido en las restricciones de filas, evaluada para cada número de usuario un total de 20 veces y promediado. Se observa claramente que el método, gracias al suavizado obtiene una solución buena para todos los ejemplos, que va empeorando como verifica esa restricción de filas a medida que se aumenta el número de usuarios. Sin embargo, en términos de error sigue siendo bastante bueno para el caso más elevado, con 35 usuarios contemplados.

Por otro lado, la Figura 5 muestra el comportamiento lineal del algoritmo, que sigue proporcionando tiempos de ejecución buenos para el caso con más usuarios. Para menos de 15 usuarios el método es lo suficientemente rápido y es capaz de proporcionar la solución en segundos.

5.1.2. Estabilidad

Ahora analizaremos la estabilidad del método. La estabilidad de un método nos sirve para ver si, haciendo leves modificaciones en los datos de entrada, el algoritmo se comporta bien, dando una solución parecida o si unos mínimos cambios dan lugar a resultados muy diferentes.

Para llevar a cabo este estudio, realizamos un total de 40 pruebas a distintos usuarios, empleando un número suficiente de iteraciones para que el método proporcione una solución cercana a la óptima.

Se consideraron cuatro niveles de ruido: 0.5, 1.5, 2.5 y 3.5. En cada prueba, se añadió un número aleatorio a cada celda de la matriz de consumo y a cada celda de la matriz de generación. Este número aleatorio se generó en un rango de -0.5 a 0.5 para el primer nivel de ruido, y de -1.5 a 1.5, -2.5 a 2.5, y -3.5 a 3.5 para los niveles subsiguientes, siempre asegurándonos de que los valores resultantes fueran no negativos.

Luego, se compararon los resultados con la solución sin ruido, almacenando el error, definido como el valor absoluto de la diferencia entre el vertido obtenido con ruido y sin ruido. Los puntos en la gráfica representan el promedio de estos errores para cada nivel de ruido, y las barras de error indican la dispersión de los resultados.

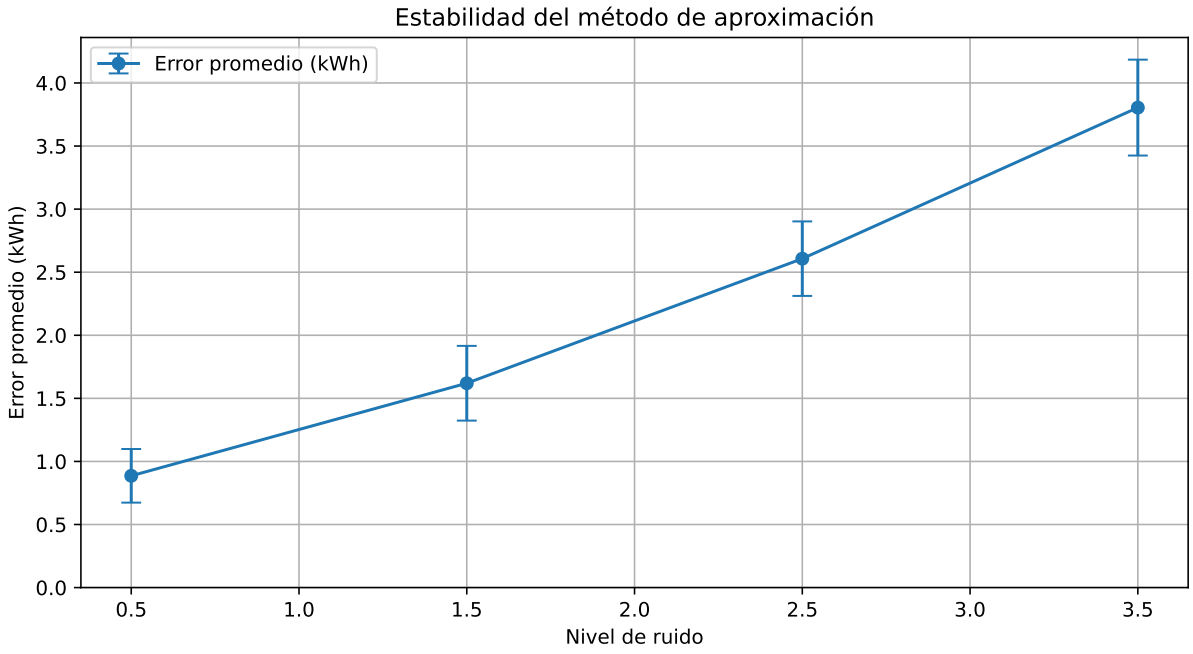


Figura 6: Estabilidad del método de acuerdo a diferentes niveles de ruido

En la gráfica se muestra la diferencia entre el valor sin ruido y el valor con ruido, de forma que, una solución que tenga valor parecido a la función objetivo estará cerca del 0. Podemos observar que el método es estable para variaciones de ruido pequeñas, como se puede ver en la Figura 6. A medida que aumentamos el ruido, el método comienza a presentarse algo más inestable, teniendo picos de variación de hasta 4 kWh en algunas pruebas. A pesar de ello, consideramos que el método es bastante estable, pues, no se ha observado un crecimiento exponencial de variación con respecto al valor esperado.

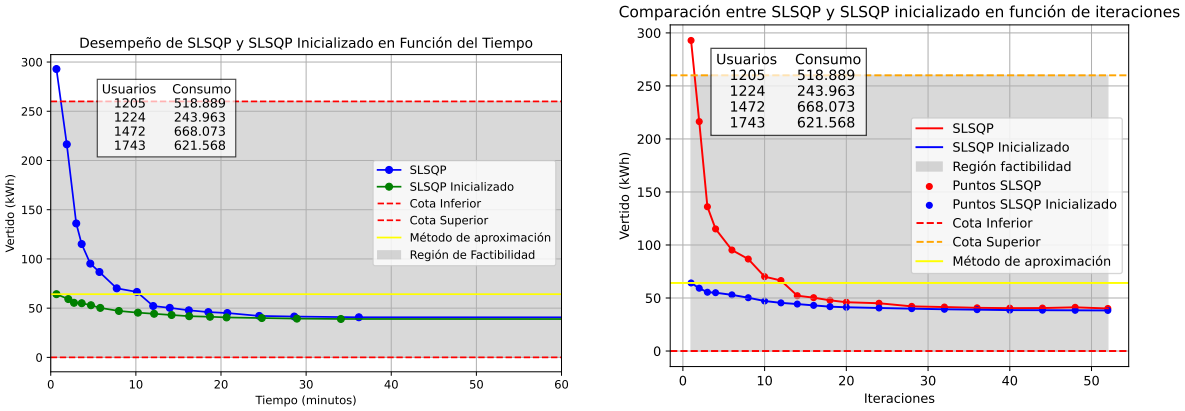
5.2. Método SLSQP y Método SLSQP Inicializado

Como ya se indicó en la sección anterior, si el objetivo es encontrar una solución lo más cercana a la óptima, para ello es conveniente inicializar el método *SLSQP* con la solución que devuelve el método de aproximación. Así, estamos aplicando el método desde un punto más cercano al óptimo.

5.2.1. Evolución con el número de usuarios

Ahora analizaremos varios casos, para ver cómo evoluciona el comportamiento de ambos métodos. En la sección 4.4, se vio que el vertido óptimo se encontraba entre dos cotas, el 0, en caso de que se consiga minimizar el vertido completamente y como cota superior el vertido con coeficientes equitativos. Estas cotas se muestran en las figuras siguientes, y servirán para ver la evolución de ambos métodos. Además, por cada ejemplo estudiado, se muestra su comportamiento frente al tiempo y frente a iteraciones. Estas iteraciones son cada paso de descenso de gradiante que hace el método.

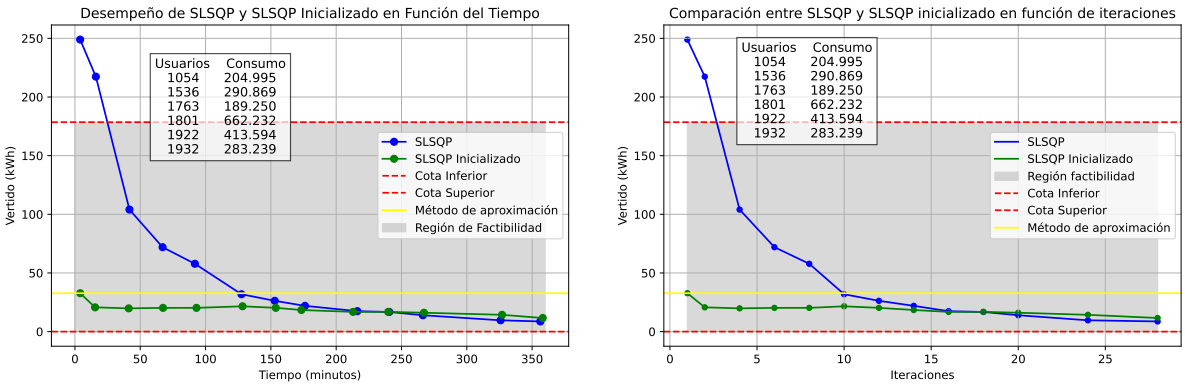
Podemos observar por un lado la relación entre iteración y tiempo, debido a la naturaleza de ambas. Podemos observar la Figura 7b como *SLSQP*, al inicializar en un punto aleatorio, la solución inicial está lejos de la óptima, de hecho, no se encuentra ni en la región de soluciones factibles. Sin embargo, *SLSQP Inicializado* parte de la solución calculada por el método de aproximación, así que ya desde el principio se encuentra en una solución cercana a la óptima. Con el paso de iteraciones del método podemos observar que las dos soluciones tienden al mismo vertido, la diferencia es que *SLSQP Inicializado* comienza a descender levemente desde el principio y *SLSQP* sin inicializar necesita dar más iteraciones.



(a) Tiempo (b) Iteraciones

Figura 7: Evolución de los métodos con 4 usuarios

En búsqueda de la solución óptima no hay diferencias claras en este ejemplo, pero en caso de tener una solución más rápida, con menos iteraciones del método, es claramente visible que es preferible usar el método inicializado. Vamos ahora con una muestra de 6 usuarios:



(a) Tiempo (b) Iteraciones

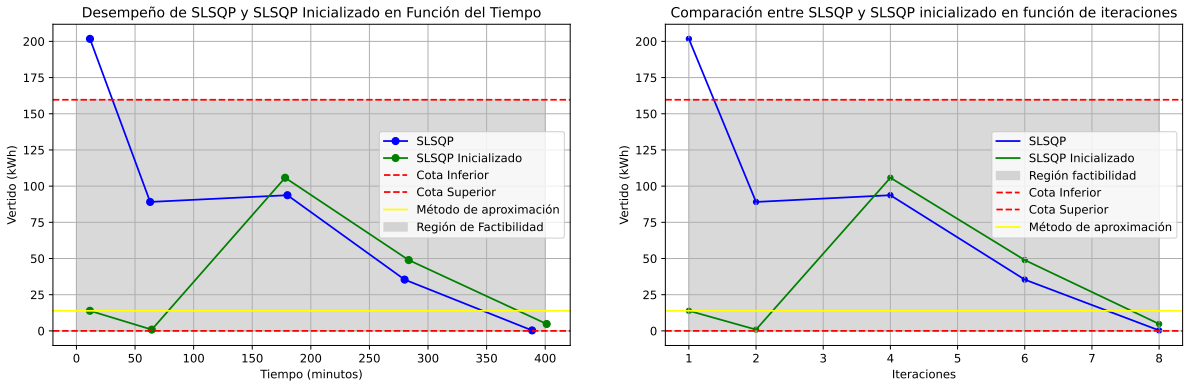
Figura 8: Evolución de los métodos con 6 usuarios

Este ejemplo nos sirve para ilustrar otro comportamiento que puede ocurrir. Al igual que antes, *SLSQP Inicializado* comienza con una solución más cerca de la óptima, y *SLSQP* sin inicializar parte de un punto muy alejado de la óptima. Sin embargo, el camino desde ese punto hacia la solución óptima es más fácil. El algoritmo puede tomar pasos más grandes y acercarse más rápidamente a la óptima. El comportamiento es similar, durante las primeras iteraciones es mucho mejor el método inicializado, pero, a partir de la iteración 18, el método

SLSQP sin inicializar adelanta al método inicializado.

Nótese que en el momento que *SLSQP* cruza la línea amarilla está teniendo un vertido igual a la solución que proporciona el método de aproximación, pero ese vertido igual no significa que tengan los mismos coeficientes. De hecho, en caso de ser el mismo, la gráfica de *SLSQP* sin inicializar continuaría como la inicializada en el momento del corte, pero esto no ocurre.

Este caso nos muestra que, para encontrar la solución óptima, no podemos garantizar en todos los casos que sea más rápido inicializando *SLSQP* a tomando un punto aleatorio. También es conveniente mirar la gráfica de tiempo y la de iteraciones. Debido a la complejidad de *SLSQP* a medida que aumentamos el número de usuarios, cada iteración tarda más en terminar.



(a) Tiempo

(b) Iteraciones

Figura 9: Evolución de los métodos con 10 usuarios

En la figura anterior se puede observar un comportamiento muy diferente al resto. Por un lado, las iteraciones se han disminuido bastante. Esto es debido a que con un tamaño de 10 usuarios los tiempos de ejecución son excesivamente altos. Observes la Figura 9a el eje de tiempo. Además, *SLSQP Inicializado* encuentra una solución óptima en la iteración 2, pero si continuamos iterando, el método sube drásticamente con el objetivo de buscar otra aún mejor. A pesar de ello se mantiene la línea anterior, *SLSQP Inicializado* es mejor para las primera iteraciones, pero para un número alto de iteraciones los resultados son similares.

5.2.2. Estabilidad

Al igual que en la sección anterior, añadiremos ruido a ciertos casos y observaremos cómo se comporta el método *SLSQP Inicializado*. En esta ocasión, se han evaluado un total de 20

casos, con variaciones de 3, 4 y 5 usuarios.

Para cada caso, se han introducido niveles de ruido similares a los utilizados previamente. Específicamente, a cada celda de las matrices de consumo y generación se les ha sumado un número aleatorio generado en un rango ajustado a los niveles de ruido evaluados: -0.5 a 0.5 para el nivel más bajo, -1.5 a 1.5 para un nivel intermedio, y así sucesivamente para niveles más altos, asegurando siempre que los valores resultantes sean no negativos.

La comparación se realizó entre la solución obtenida con ruido y la solución sin ruido, almacenando el error correspondiente. Este error se define como el valor absoluto de la diferencia entre el resultado obtenido con ruido y el resultado sin ruido.

Los resultados se representan en la Figura 10, donde cada punto indica el promedio de los errores obtenidos en las pruebas para cada nivel de ruido y las barras de error reflejan la dispersión de los datos.

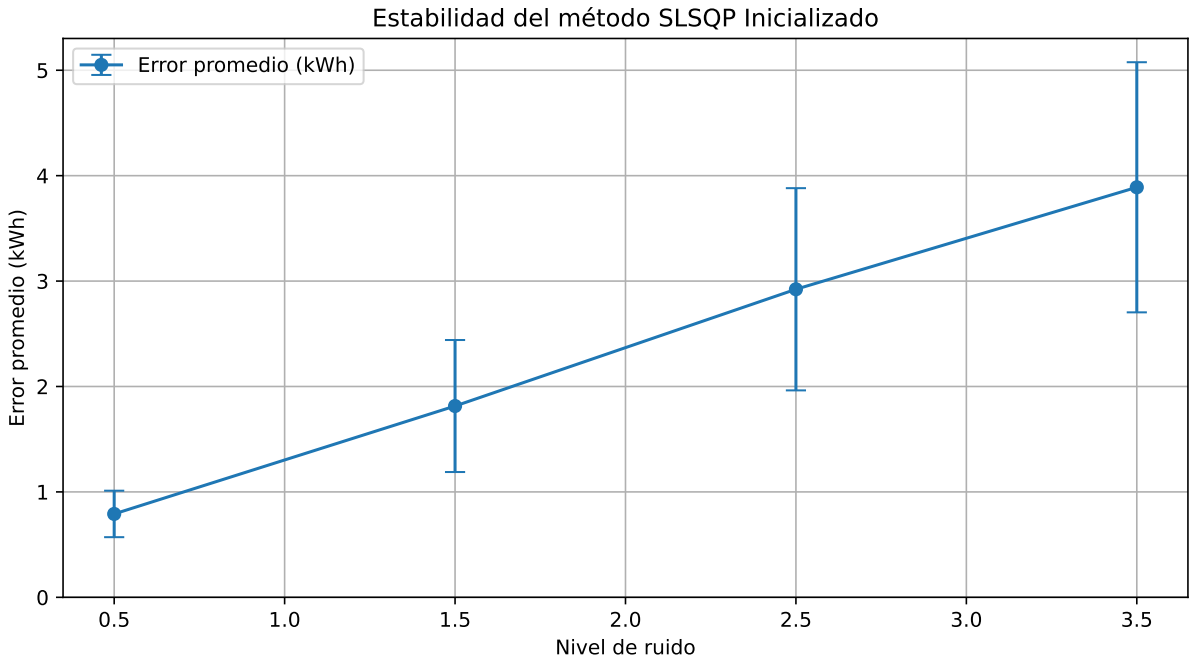


Figura 10: Estabilidad de *SLSQP Inicializado* de acuerdo a diferentes niveles de ruido

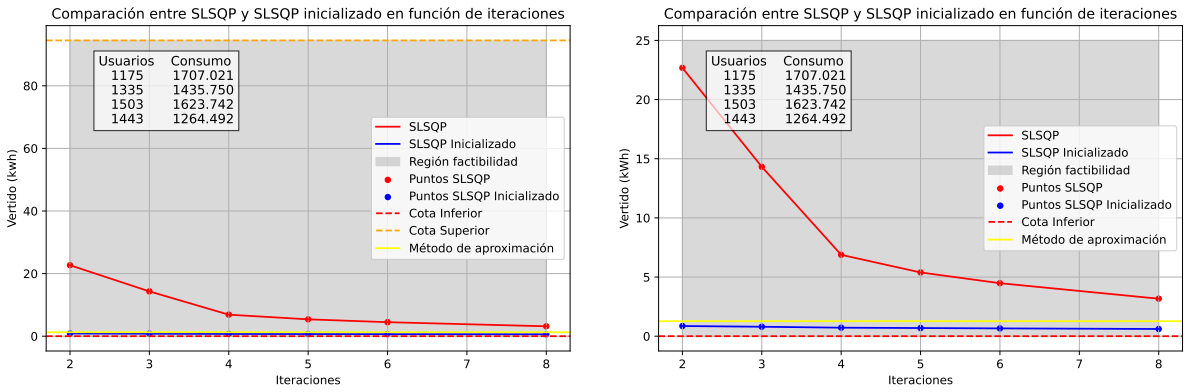
Podemos observar el comportamiento natural del método. A medida que se aumenta el nivel de ruido, el error en el vertido se incrementa, alejándose más de las soluciones sin ruido. Sin embargo, el método puede considerarse estable, ya que las variaciones no crecen de manera exponencial a medida que aumentamos el ruido. Esto indica que, aunque la precisión

disminuye con el incremento del ruido, el método *SLSQP Inicializado* mantiene un desempeño razonablemente consistente.

5.3. Comportamiento frente a distintos perfiles

Ahora trataremos una conclusión esperable del problema de optimización. Es fácilmente previsible que, la optimización sea mejor cuando hay perfiles de consumo dispares. Por ejemplo, que los usuarios consuman a diferentes horas para que se puedan compensar los coeficientes, como se vio en el ejemplo 1. Si estamos en un caso en el que los consumos son similares a las mismas horas, no se va a notar una diferencia sustancial en la optimización.

Para ilustrar esto, usando el script de preprocesado, tomamos dos conjuntos de usuarios distintos, por evitar casos extremos, usuarios que tengan un consumo mensual de entre 2000 kWh y 500 kWh. Y elegiremos aquellos 4 usuarios con mayor desviación estándar y los 4 con menos. Primero, veamos las gráficas de los usuarios con perfiles más dispares:



(a) Gráfica con cota superior

(b) Gráfica ampliada

Figura 11: Evolución de 4 usuarios dispares

En la Figura 11a mostramos la evolución de ambos métodos con 8 iteraciones. La cota superior indica el vertido con coeficientes constantes, y se puede observar que muy rápidamente, los métodos tienden a un vertido de 0. Esto se debe a que, al elegir perfiles sumamente dispares, el método tiene más facilidad para asignar los coeficientes. En la Figura 11b se muestra la misma gráfica pero ampliada. Se puede ver como, ya el método de aproximación da una solución cercana a 0, y *SLSQP Inicializado* la va acercando cada vez más a 0 de vertido con el paso de las iteraciones.

Ahora, veamos qué ocurre eligiendo perfiles muy similares.

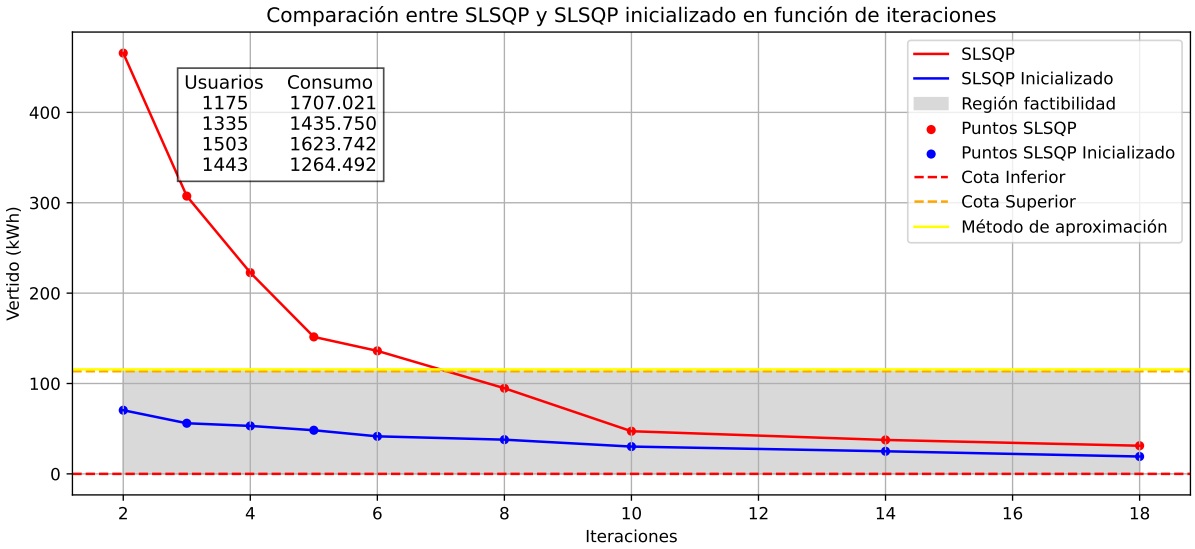


Figura 12: Evolución de 4 usuarios con perfiles similares

El primer aspecto a destacar es que la convergencia es más lenta. Tras cada iteración al método le cuesta mas descender y minimizar el vertido. Esto se ve reflejado en el número de iteraciones en cada gráfica. Para perfiles similares se necesitan muchas iteraciones mientras que para perfiles dispares es posible minimizar prácticamente el vertido a cero a partir de las primeras iteraciones.

Además, el vertido que devuelven la solución con coeficientes no dinámicos y la solución del método de aproximación es la misma. En este caso, el método de aproximación no consigue ninguna mejora. Sin embargo, a pesar de ser perfiles muy similares *SLSQP* sí consigue minimizar. Esto no solo ocurre con este ejemplo, sino con todos los perfiles seleccionados con mayor parecido en consumo.

Debido a esto, podemos concluir que la optimización siempre es beneficiosa. Cuando los perfiles son dispares, como se observa en la Figura 11a, el método de aproximación consigue una solución muy buena, y *SLSQP Inicializado*, con pocas iteraciones se acerca a vertido nulo. Sin embargo, para perfiles similares el método de aproximación no consigue dar una buena aproximación, y se necesitan muchas más iteraciones para obtener un vertido cercano al cero. Se puede ver la Figura 12 como la solución con coeficientes no dinámicos (cota superior) y la solución que proporciona el método de aproximación son similares en vertido.

Por norma general, las gráficas no serán tan extremas como en estos casos. Eligiendo usuarios aleatorios, las gráficas tienden a parecerse más a las de las Figura 7a o Figura 8a.

5.4. Prueba de Wilcoxon

La *prueba de Wilcoxon* es una técnica estadística no paramétrica utilizada para comparar dos muestras relacionadas y determinar si sus poblaciones difieren significativamente. Es útil cuando los datos no cumplen con los supuestos de normalidad requeridos por pruebas paramétricas como la prueba t para muestras pareadas. En esta prueba, se evalúan las diferencias entre pares de observaciones, se les asigna rangos y se analiza si la suma de rangos con signos (positivos y negativos) difiere significativamente de cero.

La *hipótesis nula* (H_0) que probaremos es que no hay diferencia significativa entre los coeficientes tras altas iteraciones obtenidos mediante el método *SLSQP* estándar y el método *SLSQP Inicializado*. Esto implica que cualquier diferencia observada en los coeficientes es debida al azar. En cambio, sí que la prueba de Wilcoxon encuentra diferencias significativas cuando las iteraciones son bajas.

El *p-valor* es una medida que ayuda a determinar la significancia estadística de nuestros resultados. Representa la probabilidad de observar efectos tan extremos como los encontrados en nuestro estudio, bajo la suposición de que la hipótesis nula es cierta. Un p-valor bajo (comúnmente menor que 0.05) indica que es muy improbable obtener un resultado como el observado si la hipótesis nula fuera cierta, y por lo tanto, podemos rechazar la hipótesis nula a favor de la hipótesis alternativa [14].

Para el estudio, hemos tomado diferentes experimentos, y comparamos su valor p-valor. Podemos observar en la Figura 13 cómo el p-valor está por debajo de 0.05 en las primeras 6 iteraciones. En este caso, esta tendencia indica un rechazo fuerte de la hipótesis nula, sugiriendo que hay diferencias estadísticamente significativas entre los métodos comparados durante las etapas iniciales. A medida que aumentan las iteraciones, los p-valores comienzan a elevarse, superando el umbral de 0.05 a partir de la séptima iteración y acercándose o incluso superando 0.5 en iteraciones posteriores. Este aumento en los p-valores sugiere que la diferencia entre los métodos se reduce y se vuelve estadísticamente insignificante, lo cual podría indicar una convergencia en el desempeño de los métodos o una disminución en la variabilidad de los resultados obtenidos. La región de significancia estadística, marcada en amarillo, destaca dónde

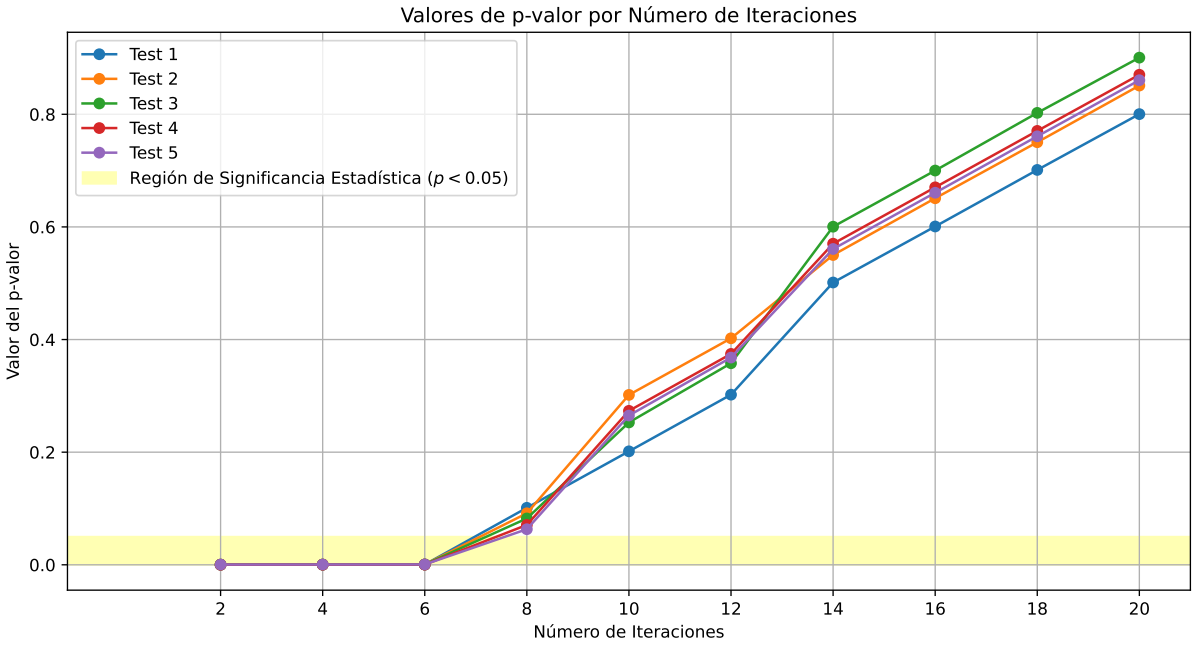


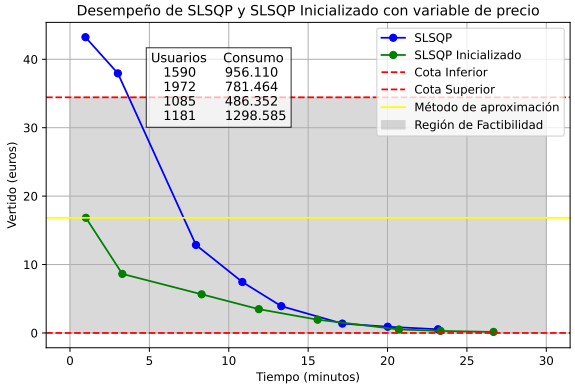
Figura 13: Evolución de 5 experimentos. Comparación del p-valor tras iteraciones.

los resultados son suficientemente robustos para rechazar la hipótesis nula.

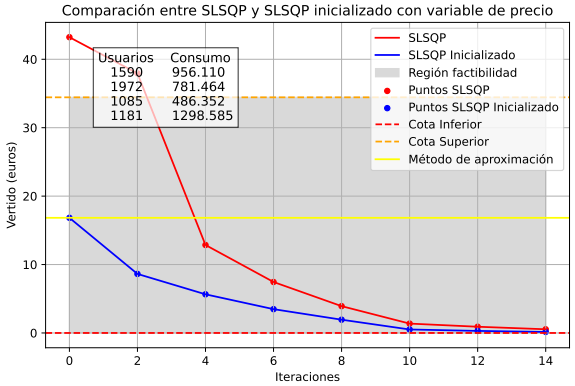
En resumen, la figura anterior traduce la Figura 7a o Figura 8a. A medida que se aumentan el número de iteraciones, los métodos tienden a comportarse de forma parecida.

5.5. Modelos con la variable del precio de la energía

Como ya se indicó en la sección 4.6, el problema de optimización usando la variable de precio es el mismo pero vertiendo euros en vez de kWh. La similitud de ambos problemas trasciende a que, las conclusiones anteriores se pueden trasladar al problema con la variable del precio de la energía. Como se puede observar en la Figura 14, el comportamiento de los métodos *SLSQP* y *SLSQP inicializado* tienen una forma similar a los casos estudiados anteriormente.



(a) Tiempo



(b) Iteraciones

Figura 14: Evolución de los modelos con 4 usuarios

Se observa en la Figura 14 como *SLSQP Inicializado* parte de una mejor solución gracias al método de aproximación, pero con el paso de las iteraciones, ambos métodos terminan convergiendo a una solución muy cercana a la óptima. En general, todas las conclusiones extraídas del problema de vertido en kWh se puede extender al problema de vertido con variable de precio de la energía.

6

Conclusiones y Líneas Futuras

6.1. Conclusiones

Hemos evaluado diferentes métodos para la optimización dentro del marco CRISP-DM, analizando sus ventajas y desventajas, complejidad, tiempo de ejecución y comportamiento frente a distintos perfiles de usuario.

El **método de aproximación**, con una complejidad $O(n)$, mostró ser eficiente en términos de tiempo de ejecución para cualquier número de usuarios. Sin embargo, su precisión en cumplir las restricciones de filas disminuye a medida que el número de usuarios aumenta, aunque sigue siendo aceptable hasta 35 usuarios.

El **método *SLSQP* y *SLSQP Inicializado*** se analizaron bajo diferentes condiciones. Se observó que inicializar *SLSQP* con la solución del método de aproximación mejora significativamente su rendimiento en las primeras iteraciones. Sin embargo, para un número mayor de iteraciones, ambos métodos tienden a converger hacia soluciones similares. Además, el método *SLSQP* sin inicializar puede en ocasiones superar al inicializado en términos de tiempo de convergencia, dependiendo del caso específico.

En términos de **estabilidad**, ambos métodos demostraron ser robustos frente a variaciones en los datos de entrada, manteniendo un comportamiento estable con niveles de ruido bajos. Al elevar demasiado los niveles de ruido, sí que se llegaron a obtener alguna soluciones muy distintas a las esperadas.

Finalmente, se concluyó que la optimización es más efectiva cuando los perfiles de consumo de los usuarios son dispares. En tales casos, el método de aproximación proporciona una solución inicial muy cercana al óptimo, facilitando la convergencia de *SLSQP Inicializado*. En

cambio, para perfiles similares, la mejora es menos significativa y requiere más iteraciones para acercarse al óptimo.

La **prueba de Wilcoxon** confirmó que las diferencias entre los coeficientes obtenidos mediante *SLSQP* y *SLSQP Inicializado* son significativas en las primeras iteraciones, pero tienden a desaparecer con un mayor número de iteraciones, indicando una convergencia en el desempeño de ambos métodos.

Intentos de abordar el problema con **algoritmos genéticos** no fueron exitosos, ya que no lograron proporcionar soluciones óptimas ni eficientes en términos de tiempo de ejecución comparados con los métodos anteriores. Además de la mala verificación de las restricciones.

Adicionalmente, se consideró la **variable del precio de la energía** en el problema. Se determinó que este enfoque es equivalente al problema original y que los resultados de comparación entre los métodos fueron similares, reafirmando la efectividad de los métodos de aproximación y *SLSQP* en la optimización de costos energéticos bajo diversas condiciones.

Una limitación importante identificada es que, cuando se superan los **8-10 usuarios**, la complejidad del método *SLSQP* hace que los tiempos de ejecución se eleven considerablemente, lo que puede ser un inconveniente en aplicaciones prácticas. Sin embargo, el método de aproximación sigue siendo muy rápido incluso con un mayor número de usuarios, proporcionando soluciones iniciales adecuadas para su uso en combinación con *SLSQP*.

6.2. Líneas futuras

En futuras investigaciones, se pueden explorar diferentes enfoques para mejorar la eficiencia y efectividad de los métodos de optimización evaluados en este estudio.

Una posible dirección es el uso de **algoritmos genéticos**. Aunque los intentos iniciales con este enfoque no fueron exitosos, se podría realizar un estudio exhaustivo de los parámetros del algoritmo, como la selección, cruce, mutación y tamaño de la población, para alcanzar una buena solución. Ajustar estos parámetros adecuadamente podría mejorar los resultados y hacer que los algoritmos genéticos sean una alternativa viable.

Otra propuesta interesante es el **Optimizador *SLSQP Inicializado por partes***. Cuando aumentamos el número de usuarios, el problema se vuelve cada vez más costoso computacionalmente, alcanzando tiempos de ejecución demasiado elevados. Para abordar este desafío, se propone una estrategia para conseguir una solución cercana a la óptima, pero manteniendo

unos buenos tiempos de ejecución. La idea es dividir la matriz de consumos en submatrices por filas, con un número de filas más manejable, y ejecutar *SLSQP Inicializado* para cada submatriz, unificando luego las submatrices de coeficientes en una única matriz.

Cuadro 18: Matriz de consumo E

e_{11}	e_{12}	e_{13}	e_{14}	e_{15}
e_{21}	e_{22}	e_{23}	e_{24}	e_{25}
e_{31}	e_{32}	e_{33}	e_{34}	e_{35}
e_{41}	e_{42}	e_{43}	e_{44}	e_{45}
e_{51}	e_{52}	e_{53}	e_{54}	e_{55}
e_{61}	e_{62}	e_{63}	e_{64}	e_{65}
e_{71}	e_{72}	e_{73}	e_{74}	e_{75}
e_{81}	e_{82}	e_{83}	e_{84}	e_{85}

} Matriz de consumo E_1

} Matriz de consumo E_2

Ahora resolvemos *SLSQP Inicializado* con el método de aproximación. Naturalmente, las restricciones de columna deben adaptarse. Para ello, ahora la restricción de columna será $\frac{nUser_i}{nUser}$, donde $nUser_i$ representa el número de usuarios de la matriz i.

Además, lo idóneo es organizar los grupos de matriz de forma que los perfiles dispares estén repartidos entre ambas matrices. Para ello, el script propuesto ordena la matriz por orden de varianza y luego asigna los perfiles dispares a cada grupo por igual, es decir, el más dispar y el menos dispar a una submatriz, el segundo más dispar y el segundo menos dispar a otra submatriz, y así sucesivamente.

Explorar estas líneas futuras podría llevar a mejoras significativas en la eficiencia de los métodos de optimización, permitiendo su aplicación en escenarios con un mayor número de usuarios sin comprometer el tiempo de ejecución.

6.3. Sostenibilidad del autoconsumo compartido

El autoconsumo compartido juega un papel importante en la transición hacia una economía sostenible y baja en carbono. Al optimizar los coeficientes en sistemas de placas solares, se mejora notablemente la eficiencia de estas instalaciones. Esto permite que la energía generada

se utilice de manera efectiva cerca de su punto de producción, optimizando el uso de la energía y fomentando un consumo más eficiente.

El atractivo del autoconsumo compartido aumenta con el uso de coeficientes dinámicos, que no solo mejoran la gestión energética que resulta en un ahorro económico significativo para los usuarios. Este beneficio económico hace que la tecnología sea más atractiva y sirve como un fuerte incentivo para su adopción, impulsando así la expansión y aceptación del autoconsumo compartido en entornos urbanos.

Este enfoque facilita la implementación de sistemas de energía renovable en zonas donde el espacio puede ser escaso. Permitiendo que varios usuarios compartan los beneficios de una instalación solar, se maximiza el uso del espacio disponible y se fomenta la cooperación entre los ciudadanos. Además de los beneficios en términos de costos, el autoconsumo compartido contribuye significativamente a la reducción de la dependencia de los combustibles fósiles y a la minimización de la huella de carbono en la producción energética.

Referencias

[1] *Algoritmos Genéticos*. URL: https://www.cs.us.es/~fsancho/Blog/posts/Algoritmos_Geneticos.md.html (visitado 11-05-2024).

[2] Irish Social Science Data Archive. *Home, Irish Social Science Data Archive*. Irish Social Science Data Archive. Publisher: Irish Social Science Data Archive. URL: https://www.ucd.ie/issda/data/commissionforenergyregulationcer/ (visitado 04-06-2024).

[3] Peter Berzi. "Convergence and Stability Improvement of Quasi-Newton Methods by Full-Rank Update of the Jacobian Approximates". En: *AppliedMath* 4 (26 de ene. de 2024), págs. 143-181. DOI: 10.3390/appliedmath4010008.

[4] Jason Brownlee. *How to Choose an Optimization Algorithm*. MachineLearningMastery.com. 22 de dic. de 2020. URL: https://machinelearningmastery.com/tour-of-optimization-algorithms/ (visitado 05-05-2024).

[5] *Cómo elegir un algoritmo de optimización - Top Big Data*. Section: Machine learning. 24 de dic. de 2020. URL: https://topbigdata.es/como-elegir-un-algoritmo-de-optimizacion/ (visitado 12-05-2024).

[6] *Conceptos básicos de ayuda de CRISP-DM - Documentación de IBM*. URL: https://www.ibm.com/docs/es/spss-modeler/saas?topic=dm-crisp-help-overview (visitado 05-05-2024).

[7] *Degenerate Conic | SLSQP*. URL: https://degenerateconic.com/slsqp.html (visitado 05-05-2024).

[8] *Entrenamiento de Redes Neuronales: mejorando el Gradiente Descendente*. URL: https://www.cs.us.es/~fsancho/Blog/posts/Mejorando_Gradiente_Descendiente.md (visitado 05-05-2024).

[9] Martina Hlatká, Ladislav Bartuska y Ján Ližbetin. "Application of the Vogel Approximation Method to Reduce Transport-logistics Processes". En: *MATEC Web of Conferences* 134 (8 de nov. de 2017), pág. 00019. DOI: 10.1051/matecconf/201713400019.

[10] *JRC Photovoltaic Geographical Information System (PVGIS) - European Commission*. URL: https://re.jrc.ec.europa.eu/pvg_tools/es/ (visitado 04-06-2024).

[11] Yingjie Ma et al. *Improved SQP and SLSQP Algorithms for Feasible Path-based Process Optimisation*. 20 de feb. de 2024. DOI: 10.48550/arXiv.2402.10396.

[12] paulduf. *Answer to "What is the time and space complexity of SLSQP minimize algorithm in Scipy?"*. Stack Overflow. 5 de oct. de 2021. URL: https://stackoverflow.com/a/69452074 (visitado 12-05-2024).

[13] *scipy.optimize.minimize — SciPy v1.13.0 Manual*. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html (visitado 05-05-2024).

[14] *t-Test, Chi-Square, ANOVA, Regression, Correlation...* URL: https://datatab.es/tutorial/wilcoxon-test (visitado 12-05-2024).

[15] Wenjie Tang et al. "Solving Engineering Optimization Problems Based on Multi-Strategy Particle Swarm Optimization Hybrid Dandelion Optimization Algorithm". En: *Biomimetics* 9 (17 de mayo de 2024), pág. 298. DOI: 10.3390/biomimetics9050298.

[16] Toogit. *How to solve non-linear optimization problems in Python - Learn and hire online*. Toogit. Section: Other - Software Development. 1 de abr. de 2019. URL: https://www.toogit.com/tlc/article/how-to-solve-non-linear-optimization-problems-in-python (visitado 05-05-2024).

Apéndice A
Manual de Instalación

A.1. Instalación de librerías de Python

El desarrollo se ha realizado utilizando Python, y se ha hecho uso de archivos Python y Jupyter Notebooks. Asumiendo que ya tiene instalados Python3 y pip3 en su sistema, instale las siguientes librerías utilizando pip:

```
pip3 install pandas==1.5.3 numpy==1.26.4 scipy==1.13.0
```

A.2. Instalación de Jupyter

Jupyter es fundamental para trabajar con notebooks interactivos. Instale Jupyter utilizando pip con el siguiente comando:

```
pip3 install jupyter
```

A.3. Verificación de la instalación

Para verificar que todas las librerías están correctamente instaladas, inicie Jupyter Notebook ejecutando el siguiente comando:

```
jupyter notebook
```

Esto abrirá Jupyter en su navegador web predeterminado. Cree un nuevo notebook y pruebe importar las librerías con el siguiente código:

```
import pandas as pd
import numpy as np
import scipy.optimize as opt
```

Si no encuentra errores al importar estas librerías, entonces la instalación ha sido exitosa.

A.4. Mantenimiento de las librerías

El software ha sido probado con las versiones específicas de las librerías mencionadas anteriormente. Se recomienda mantener sus librerías actualizadas para beneficiarse de mejoras y correcciones de errores, lo cual puede hacerse con el siguiente comando:

```
pip3 install --upgrade pandas numpy scipy jupyter
```

Sin embargo, tenga en cuenta que las futuras actualizaciones de estas librerías podrían hacer que parte del código quede obsoleto.

Apéndice B
Documentación

B.1. Estructura del Proyecto

El proyecto consta de varios componentes estructurados en directorios que facilitan la organización y ejecución del software de optimización. A continuación, se describen los principales directorios y archivos:

B.1.1. Directorios y archivos principales

- **Raíz del Proyecto:** Contiene los archivos `optimization.ipynb` y `energy_data_processing.py`. El notebook `optimization.ipynb` maneja toda la lógica de optimización, mientras que `energy_data_processing.py` se encarga del preprocesamiento de datos.
- **coefficients:** Este directorio almacena archivos CSV con los coeficientes utilizados por cada método de optimización. Los nombres de los archivos reflejan el tipo de optimización y el método utilizado:
  - **optimization1_SLSQP.csv:** Coeficientes para el modelo de minimización de vertido de energía en kWh usando el algoritmo SLSQP.
  - **optimization1_custom.csv:** Coeficientes para el modelo que minimiza el vertido de energía en kWh usando un método de aproximación personalizado.
  - **optimization1_SLSQP_init.csv:** Coeficientes iniciales usados para alimentar la solución SLSQP, basados en la salida de otro método de SLSQP.
  - **optimization2_SLSQP.csv:** Coeficientes para el modelo de minimización de vertido monetario en euros usando el algoritmo SLSQP.
  - **optimization2_custom.csv:** Coeficientes para el modelo que minimiza el vertido monetario en euros usando un método de aproximación personalizado.
  - **optimization2_SLSQP_init.csv:** Coeficientes iniciales usados para alimentar la solución SLSQP, basados en la salida de otro método de SLSQP.

- **grafics:** Contiene scripts de Python que generan las gráficas utilizadas en la documentación del proyecto.
- **output_files:** Guarda los archivos CSV generados con los datos de consumo procesados.
- **source_data:** Incluye archivos de datos en bruto desde `File1.txt` hasta `File6.txt`, así como datos de generación por defecto.

B.2. Notebook de optimización

El archivo `optimization.ipynb` en la raíz del proyecto es fundamental para la ejecución de la optimización. Contiene varios parámetros configurables que el usuario puede ajustar según sea necesario. Los parámetros modificables incluyen:

B.2.1. Parámetros configurables

- **file_number:** Refiere al número de archivo de datos en bruto. Si se importan los 6 archivos `File1.txt` hasta `File6.txt` se podrá seleccionar cualquier archivo usando este parámetro.
- **n_days:** Número de días para los cuales se generan los datos. Fijo a 30 días para evitar errores de generación.
- **n_users:** Número de usuarios para ser tratados
- **selection_type:** Tipo de selección de usuarios. Opciones disponibles: `low`, `top`, `random`. Esta opción busca usuarios con muy poca similitud, mucha similitud o aleatorio, en términos de consumos.
- **optimization1:** Activa la optimización para minimizar el desperdicio de energía.
- **optimization2:** Activa la optimización para minimizar el desperdicio monetario en euros.
- **max_iter:** Número máximo de iteraciones permitidas en la optimización. Aconsejable entre 10 y 20 para muestras pequeñas de usuarios (menos de 8 usuarios).

- **system_coefficients:** Coeficientes de participación de cada usuario. Si se deja vacío, se asume equidistante.

B.2.2. Ejecución

El archivo está diseñado para ser ejecutado de manera secuencial y en el orden establecido. Intentar ejecutar bloques de código de forma aislada puede resultar en errores debido a dependencias entre las secciones del código.

El proceso de preprocesamiento de datos es intensivo en el uso de memoria RAM. Ejecutar este archivo en máquinas con recursos limitados de memoria puede ocasionar problemas de rendimiento o fallos. Se recomienda utilizar un sistema con una capacidad de memoria adecuada. Se recomienda idealmente 16 GB de RAM, y como requisito mínimo 8 GB de RAM.

El número de usuarios y la cantidad de iteraciones configuradas influyen directamente en el tiempo de computación debido a la complejidad temporal del algoritmo SLSQP. Configuraciones con muchos usuarios y un alto número de iteraciones pueden resultar en tiempos de ejecución prolongados. Es aconsejable ajustar estos parámetros según las capacidades del hardware disponible para mantener un balance entre precisión y eficiencia en el tiempo de procesamiento.

B.3. Nota sobre Datos de Consumo

Los datos de consumo en bruto no se pueden proporcionar directamente debido a restricciones del proveedor, pero están disponibles para descarga en la siguiente URL:

`https://www.ucd.ie/issda/data/commissionforenergyregulationcer/`

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA