GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

DESARROLLO DE UN MODELO DE LA PLANIFICACIÓN DE LA OPERACIÓN A MEDIO PLAZO EN PYTHON PYOMO

Autor: Juan Raposo Picos

Director: Javier García González

Madrid

Agosto 2023

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Desarrollo de un modelo de la planificación de la operación a medio plazo en Python Pyomo

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2022/23 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

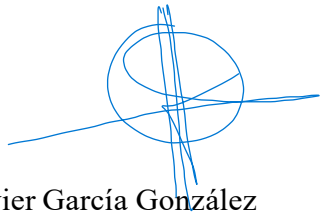
El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Juan Raposo Picos Fecha: 31/ 08/ 2023

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO



Fdo.: Javier García González Fecha: 31/ 08/ 2023

25685421X JAVIER GARCÍA

Firmado digitalmente por 25685421X JAVIER GARCÍA
Fecha: 2023.09.05 09:37:01 +02'00'

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Juan Raposo Picos DECLARA ser el titular de los derechos de propiedad intelectual de la obra: Desarrollo de un modelo de planificación de la operación a medio plazo en Python Pyomo, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente)*.

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.
d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción

de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 31 de agosto de 2023

ACEPTA

Fdo 

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

DESARROLLO DE UN MODELO DE LA PLANIFICACIÓN DE LA OPERACIÓN A MEDIO PLAZO EN PYTHON PYOMO

Autor: Juan Raposo Picos

Director: Javier García González

Madrid

Agosto 2023

DESARROLLO DE UN MODELO DE LA PLANIFICACIÓN DE LA OPERACIÓN A MEDIO PLAZO EN PYTHON PYOMO

Autor: Raposo Picos, Juan**.**
Director: García González, Javier.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas.

RESUMEN DEL PROYECTO

En este proyecto se ha diseñado un modelo de la planificación de la operación de los sistemas de potencia eléctrica implementado en Python Pyomo. Se ha establecido como horizonte temporal el medio plazo el cual, en función de la literatura estudiada, abarca desde unos meses hasta 1 o 2 años por norma general. En este estudio, se pretende comprender el funcionamiento de la regulación de los sistemas de energía eléctrica para las diferentes tecnologías (hidráulica, térmica, eólica y solar) y cuáles son los principales factores que afectan a los costes de producción.

Palabras clave: Modelo de planificación a medio plazo de la generación, coordinación hidrotérmica con generación renovable, optimización.

1. Introducción

El proyecto aborda la gestión eficiente de los recursos energéticos en España, destacando la importancia de herramientas de optimización matemática para el Sistema Eléctrico Español. Se exploran dos modelos económicos: el sistema regulado y el mercado liberalizado. Se propone un modelo regulado de minimización de costes a medio plazo, que abarca la planificación de operaciones a largo y medio plazo, considerando el sistema de generación y el mercado de CO2. Se presenta una estructura detallada, desde introducción hasta conclusiones, que incluye fundamentos teóricos, revisión del estado del arte, diseño e implementación del modelo, evaluación de resultados y posibles líneas futuras de investigación. El objetivo es lograr un desarrollo económico sostenible a través de la gestión óptima del sistema eléctrico.

2. Definición del proyecto

El proyecto consiste en desarrollar un modelo computacional que optimice la gestión de recursos energéticos en un sistema eléctrico. Este enfoque se basa en la programación matemática y en el uso de la biblioteca Pyomo para construir y resolver un modelo de optimización. El objetivo es la minimización de costes, atendiendo a criterios de eficiencia económica y a la utilización de recursos, considerando factores como generación, demanda, reservas hidráulicas y otras restricciones operativas. A través de la formulación de restricciones y objetivos en el lenguaje de programación de Python Pyomo, se busca determinar una estrategia de operación óptima a medio plazo que minimice los costos y maximice el rendimiento del sistema eléctrico, contribuyendo así al desarrollo sostenible y eficiente del sector energético.

3. Descripción del modelo/sistema/herramienta

El desarrollo del modelo se determina en tres bloques de operación:

1. *Modelización de la generación del medio plazo, alineada con los ODS:*
   En esta primera fase, se aborda la modelización de la operación del sector eléctrico a

medio plazo. Para ello, se tomará como referencia el trabajo elaborado en [1]. La motivación de este proyecto reside en que existen varias modelizaciones para la generación de energía eléctrica en el medio plazo, como se describirá con detalle en el capítulo 3. En este proyecto, se tendrán en cuenta las emisiones de dióxido de carbono en las producciones térmicas, a la hora de determinar los costes de producción mínimos. Además, como complemento a las tecnologías implementadas en [1], se añadirán las tecnologías eólica y fotovoltaica, para así desarrollar un modelo lo más próximo a la realidad posible. La combinación de ambas ideas recopila y procesa datos relevantes, como la demanda histórica y prevista de electricidad, la capacidad de generación de las distintas fuentes (hidroeléctrica, térmica, solar y eólica), los costes operativos y de mantenimiento y otras políticas energéticas. En definitiva, se construye un modelo matemático que representa la operación del sistema eléctrico, incluyendo la programación de la generación de energía y la asignación óptima de recursos para satisfacer la demanda proyectada mientras se minimizan los costos y se cumplen las restricciones, técnicas, regulatorias y medioambientales.

2. *Implementación de un modelo abstracto en Python Pyomo:*
   En esta segunda fase, se traduce el modelo matemático desarrollado en la primera fase a un formato implementable en código. De entre las múltiples opciones existentes, se utiliza Pyomo, una biblioteca de optimización en Python, para construir el modelo de programación lineal que refleje las ecuaciones, variables de decisión y restricciones definidas en la modelización de la operación. Esto se debe a que Pyomo permite crear de manera eficiente y estructurada el sistema de ecuaciones que representa la optimización de la generación eléctrica y la toma de decisiones a medio plazo. Cabe destacar, que dentro de Pyomo, de entre las dos opciones de modelado (de parametrización o numérica), se utilizará la abstracto, pues ello facilita una rápida implementación y mejor gestión de los datos a largo plazo. Como se verá en el capítulo 4 de forma exhaustiva, para la ejecución de los modelos abstractos, es necesaria la carga de los datos a través de documentos de datos externos . En el caso de este proyecto, se utilizará el formato de CSV ("Comma-separated values"). Debido a que Pyomo es un lenguaje extremadamente tipado, se realizará una explicación detallada, de la forma de construir los archivos CSV, para los distintos parámetros, variables, etc.

3. *Análisis teoórico de la posible implementación en interfaz web:*
   En esta tercera fase, se busca llevar el modelo de optimización implementado en Pyomo a una interfaz web accesible a cualquier usuario. Sin embargo, tan solo se realiza un esbozo teórico. Se propone el desarrollo de una aplicación web interactiva que permita a los usuarios (operadores del sistema eléctrico, planificadores, reguladores, etc.) ingresar datos relevantes, como las previsiones de demanda actualizadas y parámetros de configuración. La aplicación web utilizaría técnicas de programación y diseño para interactuar con el modelo de Pyomo y ejecutar la optimización en tiempo real o a través de escenarios simulados. Los resultados, como la programación de generación óptima y los costes proyectados, se presentarían de una manera clara y comprensible en la interfaz web, lo que permite a los usuarios tomar decisiones informadas sobre la operación del sistema eléctrico a medio plazo.

En conjunto, este enfoque integra la modelización matemática, la implementación de optimización en Pyomo y la creación de una interfaz web (teórica) amigable para proporcionar una herramienta poderosa y versátil que permita la planificación eficiente de la operación del sector eléctrico a medio plazo.

4. Resultados

En el capítulo 5, se detallan los resultados obtenidos del estudio de un modelo a medio plazo implementado en Python Pyomo. El capítulo se estructura en dos partes: la primera parte aborda la comprobación de posibles fallos y la validación de los resultados, mediante en contraste con los obtenidos en el programa GAMS, mientras que la segunda parte, realiza un análisis exhaustivo de los resultados, incluyendo la incorporación de tecnologías renovables (eólica y fotovoltaica) para mejorar la representación realista del modelo:

1. *Pruebas y validación del sistema:*
   En la primera parte, se procede a verificar la integridad y precisión del modelo implementado. Se lleva a cabo un proceso de comprobación de posibles anomalías en la definición y construcción de elementos como variables, parámetros, ecuación de restricción y función objetivo. Esto implica aislar cada elemento del modelo para asegurar que el programa funcione de forma óptima. Posteriormente, se avanza a la validación del programa comparando los resultados obtenidos en Pyomo con los resultados de un fichero GAMS proporcionado por el director de trabajo. Dado que se utilizan solucionadores idénticos (Gurobi). La coincidencia de resultados se espera (y resulta) ser muy cercana, y cualquier discrepancia se abordará con un análisis detallado de las razones que hay detrás.

2. *Análisis de los resultados obtenidos:*
   En esta segunda parte, se lleva a cabo un análisis más profundo y exhaustivo de los resultados. Se consideran cuatro tecnologías de generación (térmica, hidráulica, eólica y solar) y se incorpora el precio de CO2 por tonelada como factor adicional. Se exploran varias métricas y visualizaciones para obtener una comprensión completa del modelo y sus implicaciones:

   - **Producción Neta Anual por Tecnología**: Se evalúa la producción neta anual para cada tecnología de generación, permitiendo una comparación clara de su contribución a lo largo del año.
   - **Producción Neta Anual Apilada:** Se presenta la producción neta anual total, mostrando cómo se acumulan las contribuciones de todas las tecnologías.
   - **Producción Mensual y Anual apilada en subperíodos y niveles de carga:** Se desglosa la producción mensual y anual apilada, en subperiodos y niveles de carga, lo que proporciona una visión detallada de las fluctuaciones en la generación de energía.
   - **Costes marginales del Sistema:** Se analizan los costes marginales del sistema a lo largo del año, tanto en términos anuales como mensuales, en subperíodos y niveles de carga.
   - **Evolución de las Reservas y su valor marginal:** Se examina cómo evolucionan las reservas de las plantas hidráulicas y se analiza el valor marginal del agua en la ecuación de balance de energía de la reserva hidráulica.
   - **Producción de CO2:** Se investiga la producción de CO2 a lo largo del tiempo, brindando información sobre las emisiones asociadas a la generación de energía.

En cada uno de estos apartados, se exploran las interacciones entre las gráficas y se analiza la sensibilidad de las ecuaciones y la producción ante cambios. En conjunto, estos análisis proporcionan una comprensión completa de los resultados obtenidos, permitiendo a los investigadores y tomadores de decisiones extraer conclusiones significativas y tomar medidas informadas en la planificación y operación del sector eléctrico a medio plazo.

5. Conclusiones

En este Proyecto se ha desarrollado un modelo de optimización a medio plazo para la toma de decisiones en sistemas de potencia eléctrica, implementado en Python Pyomo. El modelo considera aspectos técnicos, legales y medioambientales para estimar los costes de producción de electricidad.

Se han logrado varios hitos importantes en el transcurso del proyecto:

1. **Familiarización con Python:** Se adquirió conocimiento en el uso de Python, incluyendo su sintaxis, estructura de funciones y clases.
2. **Dominio de Pyomo:** Se aprendió y empleó la herramienta Pyomo para la construcción de modelos de optimización. Se exploraron diferentes solucionadores y se estudiaron alternativas similares.
3. **Teorización de una interfaz web:** Se investigó y comprendió los conceptos detrás de la creación de una interfaz web, abarcando aspectos de backend, frontend, bases de datos, seguridad cibernética y diseño web.
4. **Estudio detallado de producción eléctrica:** Se profundizó en el funcionamiento de la producción eléctrica, especialmente en el contexto del medio plazo. Se exploraron decisiones, restricciones y la creciente influencia de las energías renovables.
5. **Conocimiento de optimización matemática:** En menor medida, pero se obtuvo una comprensión básica de las técnicas de optimización lineal fundamentales para abordar problemas complejos.

En términos de limitaciones y futuras investigaciones:

1. **Limitaciones de escalabilidad:** El tiempo de ejecución podría aumentar significativamente al escalar el modelo con más generadores, restricciones y variables. Se sugiere considerar otros lenguajes de programación como alternativas.
2. **Interfaz web:** Una limitación identificada fue la necesidad de enviar todos los archivos con el código. Como futura línea de investigación, se recomienda desarrollar una interfaz web que permita modificar datos de forma dinámica con los archivos almacenados en la nube.

En resumen, este proyecto logró la creación y aplicación de un modelo de optimización para sistemas de potencia eléctrica, así como la adquisición de habilidades en programación, optimización matemática y conceptos de interfaces web. Se sugiere continuar con la implementación de una interfaz web dinámica y explorar enforques para abordar problemas de escalabilidad.

DEVELOPMENT OF A MEDIUM-TERM OPERATION PLANNING MODEL IN PYTHON PYOMO

Author: Raposo Picos, Juan.
Supervisor: García González, Javier.
Collaborating Entity: ICAI – Universidad Pontificia Comillas.

ABSTRACT

In this project a model of the operation planning of electric power systems has been designed and implemented in Python Pyomo. The time horizon has been established as medium term, which ranges from a few months to 1 or 2 years as a general rule, depending on the literature studied. This study aims to understand how the regulation of electric power systems works for the different technologies (hydro, thermal, wind and solar) and which are the main factors affecting production costs.

Keywords: Medium-term generation planning model, hydrothermal coordination with renewable generation, optimization.

1. Introduction

The project addresses the efficient management of energy resources in Spain, highlighting the importance of mathematical optimization tools for the Spanish Electricity System. Two economic models are explored: the regulated system and the liberalized market. A regulated model of medium-term cost minimization is proposed, covering long and medium-term operation planning, considering several factors such as generation, transmission network, CO2 price. A detailed structure is presented, from introduction to conclusions, including theoretical foundations, review of the state of the art, design and implementation of the model, evaluation of results and possible future lines of research. The objective is to achieve sustainable economic development through optimal management of the electricity system.

2. Project definition

The project consists of developing a computational model to optimize the management of energy resources in an electrical system. This approach is based on mathematical programming and the use of the Pyomo library to build and solve an optimization model. The objective is cost minimization, taking into account economic efficiency criteria and resource utilization, considering factors such as generation, demand, hydraulic reserves, capacity expansion and other operational constraints. Through the formulation of constraints and objectives in the Python Pyomo programming language, the aim is to determine an optimal medium-term operating strategy that minimizes costs and maximizes the performance of the electricity system, thus contributing to the sustainable and efficient development of the energy sector.

3. Description of the model/system/tool

The development of the model is determined by three blocks of operation:

1. *Modeling of medium-term generation, aligned with the SDGs*:

   In this first phase, the modeling of the operation of the electricity sector in the medium term is addressed. For this purpose, the work elaborated in [1] will be taken as a reference. The motivation for this project lies in the fact that there are several modelizations for the generation of electric power in the medium term, as will be described in detail in Chapter 3. Therefore, as a novelty in this project, carbon dioxide emissions in thermal production will be considered when determining the minimum production costs. In addition, as a complement to the technologies implemented in [1], wind and photovoltaic technologies will be added, to develop a model as close to reality as possible. The combination of both ideas collects and processes relevant data, such as historical and forecasted electricity demand, generation capacity of the different sources (hydro, thermal, solar, and wind), operating and maintenance costs, transmission constraints, and other energy policies. In short, a mathematical model is built to represent the operation of the power system, including the scheduling of power generation and the optimal allocation of resources to meet the projected demand while minimizing costs and complying with technical, regulatory, and environmental constraints.

2. *Implementation of an abstract model in Python Pyomo*:

   In this second phase, the mathematical model developed in the first phase is translated into an implementable code format. Among the many options available, Pyomo, a Python optimization library, is used to build the linear programming model that reflects the equations, decision variables and constraints defined in the modeling of the operation. This is because Pyomo allows to create in an efficient and structured way the system of equations that represents the optimization of the electrical generation and the decision making in the medium term. It should be noted that within Pyomo, between the two modeling options (parameterization or numerical), the abstract one will be used, as it facilitates a quick implementation and better data management in the long term. As will be seen in chapter 4 in detail, for the execution of the abstract models, it is necessary to load the data through external data documents. In the case of this project, the CSV ("Comma-separated values") format will be used. Since Pyomo is an extremely typed language, a detailed explanation of how to build the CSV files for the different parameters, variables, etc. will be provided.

3. *Theorization of the possible implementation in a web interface:*

   In this third phase, we seek to bring the optimization model implemented in Pyomo to a web interface accessible to any user. However, only a theoretical outline is made. It is proposed to develop an interactive web application that allows users (power system operators, planners, regulators, etc.) to enter relevant data, such as updated demand forecasts and configuration parameters. The web application would use programming and design techniques to interact with the Pyomo model and run optimization in real time or through simulated scenarios. The results, such as the optimal generation schedule and projected costs, would be presented in a clear and understandable manner on the web interface, allowing users to make informed decisions about the operation of the power system over the medium term.

Overall, this three-phase approach integrates mathematical modeling, optimization implementation in Pyomo, and the creation of a user-friendly (theoretical) web interface to provide a powerful and versatile tool for efficient medium-term power sector operation planning.

4. Results

Chapter 5 details the results obtained from the study of a medium-term model implemented in Python Pyomo. The chapter is structured in two parts: the first part deals with the verification of possible failures and the validation of the results, by contrasting them with those obtained in the GAMS program, while the second part performs an exhaustive analysis of the results, including the incorporation of renewable technologies (wind and photovoltaic) to improve the realistic representation of the model:

1. *System testing and validation:*

   In the first part, we proceed to verify the integrity and accuracy of the implemented model. A process of checking for possible anomalies in the definition and construction of elements such as variables, parameters, constraint equation and objective function is carried out. This involves isolating each element of the model to ensure that the program works optimally. Subsequently, we proceed to the validation of the program by comparing the results obtained in Pyomo with the results of a GAMS file provided by the work manager. Since identical solvers (Gurobi) are used. The matching of results is expected (and turns out) to be very close, and any discrepancy will be addressed with a detailed analysis of the reasons behind it.

2. *Analysis of the results obtained:*

   In this second part, a deeper and more comprehensive analysis of the results is carried out. Four generation technologies are considered (thermal, hydro, wind and solar) and the CO2 price per ton is incorporated as an additional factor. Various metrics and visualizations are explored to gain a complete understanding of the model and its implications:

   - **Annual Net Production by Technology:** annual net production is evaluated for each generation technology, allowing for a clear comparison of their contribution throughout the year.
   - **Stacked Annual Net Production:** Total annual net production is presented, showing how the contributions of all technologies stack up.
   - **Monthly and Annual Stacked Production in sub-periods and load levels:** Monthly and annual stacked production is broken down into sub-periods and load levels, providing a detailed view of fluctuations in power generation.
   - **Marginal costs of the System:** The marginal costs of the system are analysed throughout the year, both in annual and monthly terms, in sub-periods and load levels.
   - **Evolution of Reserves and their marginal value:** The evolution of the reserves of the hydroelectric plants is examined and the marginal value of water in the hydroelectric reserve energy balance equation is analysed.
   - **CO2 production:** CO2 production over time is investigated, providing information on emissions associated with power generation.

In each of these sections, the interactions between the graphs are explored and the sensitivity of the equations and production to changes is analysed. Together, these analyses provide a comprehensive understanding of the results obtained, allowing researchers and decision makers to draw meaningful conclusions and take informed action in the medium-term planning and operation of the power sector.

5. Conclusions

In this project, a medium-term optimization model for decision making in electric power systems, implemented in Python Pyomo, has been developed. The model considers technical, legal and environmental aspects to estimate electricity production costs.

Several important milestones have been achieved during the project:

1. **Familiarization with Python:** Knowledge was acquired in the use of Python, including its syntax, function structure and classes.
2. **Proficiency with Pyomo:** The Pyomo tool for building optimization models was learned and used. Different solvers were explored, and similar alternatives were studied.
3. **Theorization of a web interface:** Researched and understood the concepts behind the creation of a web interface, covering aspects of backend, frontend, databases, cyber security and web design.
4. **Detailed study of electricity production:** In-depth study of the operation of electricity production, especially in the context of the medium term. Decisions, constraints and the growing influence of renewable energies were explored.
5. **Mathematical optimization knowledge:** To a lesser extent, but a basic understanding of linear optimization techniques fundamental to tackling complex problems was obtained.

In terms of limitations and future research:

1. **Scalability limitations:** Execution time could increase significantly by scaling the model with more generators, constraints and variables. It is suggested to consider other programming languages as alternatives.
2. **Web interface:** A limitation identified was the need to send all files with the code. As a future line of research, it is recommended to develop a web interface that allows modifying data dynamically with the files stored in the cloud.

In summary, this project achieved the creation and application of an optimization model for electric power systems, as well as the acquisition of skills in programming, mathematical optimization, and web interface concepts. It is suggested to continue with the implementation of a dynamic web interface and explore approaches to address scalability issues.

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

Los recursos energéticos españoles son limitados, por lo tanto, es necesario gestionarlos de manera eficiente para el correcto desarrollo económico del país. Para lograr una planificación óptima del Sistema Eléctrico Español, es indispensable contar con herramientas computacionales que utilicen métodos de optimización matemática eficientes.

Existen dos modelos económicos principales para la gestión de los recursos eléctricos: el sistema regulado, basado en la optimización de costes de todo el sistema, y el modelo de mercado liberalizado, que se fundamenta en la introducción de la competencia en aquellas actividades que lo permitan, como por ejemplo el negocio de generación y el de comercialización. En ambos modelos, el problema de la planificación operativa de un sistema eléctrico de potencia se resuelve descomponiendo el horizonte de planificación en diferentes escalas de tiempo y estableciendo una jerarquización entre dichas escalas. El objetivo de este proyecto consiste en el desarrollo de un modelo de optimización para determinar la política de explotación de los recursos de generación disponibles (generación nuclear, térmica, hidráulica y renovables) que minimice el coste total de explotación en un horizonte de medio plazo (un año).

El problema a nivel anual o de medio plazo, puede ser dividido en intervalos mensuales o semanales y sirve como guía para las variables de decisión del problema a corto plazo (hasta una semana). Este último, consiste en la planificación semanal del modelado de costes de arranque y parada, las rampas de generación de los generadores térmicos, considerando también la disponibilidad de energía en los generadores hidroeléctricos y las consignas de explotación establecidas en el medio y largo plazo. A este modelado se le denomina *unit commitment*.

Además, tras la resolución de este problema, se lleva a cabo el despacho económico, que consiste en asignar de manera óptima la demanda total del sistema entre las distintas unidades de generación que estén acopladas a la red eléctrica.

El trabajo que aquí se expone está estructurado como sigue: el primer capítulo realiza una introducción del problema que se aborda, seguida de la motivación, objetivos, metodología y recursos del proyecto. A continuación, se elabora una exposición concisa de los fundamentos teóricos sobre los que se fundamenta este proyecto: técnicas de optimización lineal, de programación en Python y diseño web (capítulo segundo). Posteriormente, se realiza una revisión del estado del arte de trabajos de índole similar que han tratado de modelar el problema de optimización en distintos horizontes temporales, junto con un análisis crítico de las soluciones existentes (tercer capítulo). En el cuarto capítulo, se detalla el diseño e implementación del modelo de planificación de la operación a medio plazo, siendo esto la parte nuclear del proyecto realizado. Además, se expondrán algunas directrices teóricas para el desarrollo deuna interfaz web. En el quinto capítulo se realiza una evaluación y análisis de los resultados obtenidos tras la ejecución del programa informático, mediante el uso de herramientas de visualización las cuales se explicarán con detalle en el capítulo 4. Este capítulo consistirá enun primer bloque de ensayo y validación, seguido de un segundo bloque de análisis de resultados. Finalmente, en el sexto capítulo, se elaborará un resumen de logros, unasconclusiones sobre las contribuciones del trabajo y una exposición de las limitaciones de este, junto con posibles futuras líneas de investigación.

1.1 MOTIVACIÓN DEL PROYECTO

La motivación de este proyecto se puede analizar desde múltiples perspectivas. En primer lugar, presenta un valor teórico-práctico. El proyecto proporciona una síntesis de algunos estudios anteriores sobre modelos de optimización de la planificación a medio plazo de los sistemas de potencia eléctrica, tomando como punto de partida y fundamento, el modelo desarrollado en el trabajo [1] (se exponen las técnicas de optimización, restricciones, variables de decisión y funciones objetivo a utilizar). Como complemento meramente informativo, se recurre a los proyectos [2], (representativo de la planificación de la operación a corto y medio plazo en México,) [3] (en el que se expone de forma clara y concisa un análisis de diversos procesos de optimización matemática para modelos de medio plazo), [4] (presenta una explicación sobre las funciones de planificación y operación para los distintos horizontes temporales) y [5]. Todo ello se expondrá en mayor profundidad en la revisión del estado del arte (capítulo 3). Por otro lado, estas digresiones teóricas se complementan con la construcción del modelo de optimización en un lenguaje de código abierto como es Python y el uso de un paquete de optimización matemática conocido como *Pyomo.* La potencialidad de este último reside en que tiene una gran aplicabilidad real. Como se indicará en la explicación teórica de *Pyomo* (capítulo 3), este se puede utilizar para la resolución de problemas reales como pueden ser la minimización de coste en el transporte de mercancías, modelos de despacho económicos, representación de variables duales, generación de árboles de expansión (también conocidos como *spanning trees* en la literatura anglosajona)… los cuales debido a su enorme complejidad y número de variables, sin el uso de herramientas computacionales serían inabarcables para el ser humano. Por último, el desarrollo de una interfaz web tendrá gran potencialidad para los operadores del sistema, pues permitirá evaluar de forma interactiva y sencilla, el coste de generación de la electricidad, mediante un tratamiento dinámico de diversas bases de datos (capítulo 4).

A estas motivaciones, se le puede añadir una intrínseca al objetivo del proyecto: el valor económico. El modelo que aquí se propone considera una minimización de costes de los sistemas de generación eléctrica a medio plazo (entre los que se encuentran generadores hidroeléctricos, térmicos, al igual que en [2]).

Por último, mencionar un claro valor social del proyecto, pues un aumento de rapidez, eficiencia y facilidad en el cálculo del coste de la producción eléctrica, supone un enorme beneficio social, pues este proyecto, propone un acceso universal al tratamiento de datos de la producción, algo que no se contempla en la mayor parte de los modelos revisados en la revisión del estado del arte (capítulo 3) y supone una mayor transparencia y una mayor comprensión ciudadana del coste de la producción eléctrica.

El objeto de estudio de este proyecto se expondrá en el siguiente apartado, pudiéndose destacar, los siguientes casos de estudio:

1) Análisis y visualización de la producción termal y eólica desglosado en mensualidades, días laborales y festivos, y niveles de carga. Como se expondrán en los capítulos 2 y 4, la gestión del combustible (carbón, gas, petróleo) requiere una planificación cuidadosa, junto con un análisis del uso del agua, como un combustible más. En conjunto, este estudio permitirá realizar una compra de combustible y uso del valor del agua más económico
2) El estudio de las principales variables duales asociadas a las restricciones destacando: el coste marginal del sistema (variable dual de la ecuación de demanda) y la evolución del valor marginal del agua década embalse (variable dual de la ecuación de balance hidráulico). En definitiva, las variables duales, como se indicará en el capítulo posterior, permiten expresar valores marginales o precios sombra, además de darnos una percepción de la sensibilidad de la solución a cambios en las restricciones.
3) Estudiar la evolución de las reservas de energía equivalente en cada embalse hidráulico (teniendo en cuenta el volumen inicial) y añadiendo el valor máximo y mínimo de las reservas.

1.2 OBJETIVOS

Teniendo en cuenta lo expuesto en el apartado anterior, se puede afirmar que el objeto principal de este estudio es estimar la producción necesaria de cada generador, para satisfacer la demanda del sistema, pero siempre bajo el estricto cumplimiento de las restricciones propuestas, en el marco temporal del medio plazo (desde unas semanas hasta un año). Como objetivos más particulares, se pretende dar respuesta a los casos de estudio propuestos con anterioridad, mediante un exhaustivo análisis de las sensibilidades, costes marginales, del "valor del agua", variaciones de producción mensuales, etc.

Para el cumplimiento de estos objetivos, es necesario un estudio previo y revisión cuidadosa de las siguientes cuestiones:

1) El estado del arte (capítulo 3), el cual permitirá conocer las perspectivas y enfoques realizados a problemas similares o idénticos, además de madurar los conceptos teóricos que permitirán al alumno una mayor comprensión y facilidad para la realización del proyecto.
2) El modelo de partida facilitado al alumno. El director del proyecto facilita al alumno un modelo de despacho económico, codificado en *Python Pyomo*, el cual permitirá al alumno una mayor facilidad para la comprensión de conceptos de la optimización lineal y la programación informática. Además, también recibirá un código del modelo de operación a medio plazo en el entorno gráfico GAMS.

En resumen, como objetivos que se persiguen resolver con el proyecto, se pretende:

1) Realizar un programa de Python que permita el análisis y la visualización del problema de optimización de la generación en el medio plazo (MMP) mediante el estudio de datos de generación hidroeléctrica y térmica, demanda, producción solar y eólica, etc. con el objetivo de automatizar el estudio de la minimización de costes, pues tiene mayor portabilidad que el realizado previamente en el entorno gráfico GAMS.

2) Conseguir un programa de Python ambivalente que admita cualquier tipo de archivo de datos (CSV, XLS, JSON, etc.)

1.3 METODOLOGÍA

La resolución del proyecto consta de 4 fases:

En primer lugar, para alcanzar los objetivos expuestos en el apartado anterior, es necesario un estudio y comprensión de ciertas nociones sobre la generación eléctrica (funcionamiento, regulación) junto con conceptos de optimización matemática asociados. En este caso, como se expondrá en el capítulo 4, se trata de un MILP[1].

Tras una revisión bibliográfica exhaustiva, se procede a la segunda fase del proyecto, orientada al modelado y la programación. En ella se pretende modelar y comprender dicho problema, por un lado, y exportar el problema desarrollado en GAMS, al entorno de Python, mediante el uso del paquete de optimización *Pyomo*. Al carecer de experiencia con este último, se precisa de una familiarización con el entorno de programación, razón por la cual el director facilita una plantilla en Python, que puede servir como referencia para el desarrollo del programa.

En la tercera fase, se pretende en primer lugar, acomodar las bases de datos de GAMS, para la lectura de los datos, en un formato de archivo CSV[2]. Mediante el uso de funciones internas al programa de *Pyomo*, se podrá extender la lectura de archivos a otros formatos (JSON, .dat, EXCEL, etc.)

Tras la validación y carga de datos en el modelo, se procede a plantear los casos de estudio expuestos con anterioridad, y se analizan sus resultados para responder a las cuestiones también planteadas.

[1] Acrónimo del inglés: ”Mixed Integer Linear Problems (MILP)”
[2] Archivo de valores separado por comas.

1.4 RECURSOS

Se parte de una exhaustiva revisión bibliográfica, fruto de la lectura de manuales, libros, enlaces web y otra documentación encontrada bien en internet, bien en la biblioteca de la Universidad Pontificia de Comillas y/o University of Illinois Urbana-Champaign. Todos estos recursos aparecerán documentados en el apartado de referencias bibliográficas.

El desarrollo del modelo, tomando como referencia el modelo de medio plazo facilitado al alumno en GAMS, y el modelo de despacho económico, desarrollado en Python, se elaborará mediante el uso de del software de ciencia de datos Anaconda, el cual incorpora *Jupyter Notebook*, una interfaz web de código abierto que permite la inclusión de texto, video, audio, imágenes, así como la ejecución de código a través del navegador en múltiples lenguajes. En este caso el lenguaje utilizado será *Pyomo*, ideal para el desarrollo de modelos de optimización.

Finalmente, para el análisis de los resultados se utilizan los paquetes de Python *Matplotlib*, *Pandas*, *Scipy*. Estos son herramientas de ciencia de datos los cuales facilitan su tratamiento y visualización. Para la elaboración de la memoria y presentación del proyecto se emplearán los siguientes programas de Microsoft Office: Word, Excel y Power Point.

Capítulo 2. FUNDAMENTOS TEÓRICOS

2.1 MODELOS DE OPTIMIZACIÓN MATEMÁTICA: PLANIFICACIÓN DE LA OPERACIÓN PARA LOS DISTINTOS HORIZONTES TEMPORALES

La gestión económica de un sistema eléctrico está compuesta de un entramado de decisiones complejas, pues pueden llegar a intervenir aspectos financieros (gestión de riesgo, índices macroeconómicos), empresariales (compraventa de combustibles fósiles), medioambientales (restricciones de emisiones), sociales (ayudas, subvenciones), además de todo lo relacionado con la planificación y operación del sistema. El alcance de este proyecto se centrará en la planificación y operación del sistema en el medio plazo, teniendo en cuenta las restricciones impuestas de diversa índole (las cuales se desarrollarán en el capítulo 4 de forma exhaustiva).

En este apartado se desarrollará el marco temporal de la toma de decisiones para la producción de energía eléctrica, matizando la jerarquía de las decisiones, fuentes de producción de energía, modelado del problema de optimización (variables de decisión, función objetivo, restricciones, tipo de optimización y análisis de las variables duales), etc. Todo ello se sustentará en los siguientes pilares o principios fundamentales:

- Minimización del coste (tanto de inversión, como de producción)
- Garantía de la fiabilidad del sistema: Es de suma importancia la reducción del número de fallos de la red, para garantizar una mejor operatividad y a su vez, una mayor satisfacción del cliente o consumidor.
- Satisfacción de la demanda del consumidor.
- Cumplimiento del marco normativo o legal establecido en el sistema.

De lo citado con anterioridad se deduce, que en la toma de decisiones existe una constante tensión entre el factor económico, técnico y regulador, cuya simbiosis varía en función de la etapa temporal en la que nos encontremos, como se indicará a continuación.

La gestión de un sistema de potencia eléctrica, dada su complejidad requiere, de dividir y jerarquizar el problema en un horizonte temporal, para así simplificarlo. La siguiente tabla (Tabla 1) muestra dicho horizonte temporal, junto con comentarios adicionales acerca de los objetivos, duración, predomino económico, técnico o regulador.

MODELOS ECONÓMICOS PARA LA PLANIFICACIÓN Y MODELADO	**HORIZONTE TEMPORAL**	**DURACIÓN DEL HORIZONTE TEMPORAL**	**PLANIFICACIÓN DE LA OPERACIÓN**	**PREDOMINIO FACTOR TÉCNICO, ECONÓMICO O REGULADOR**
Sistema regulado (Criterio de minimización de costes)	*Largo plazo*	2-3 años hasta los 10-15 años.	-Instalación de nuevos equipos de generación y transporte. -Gestión energética: ciclo nuclear y reservas hiperanuales.	Predominio factor económico. El marco regulador afecta sobre todo a la localización de los nuevos equipos de generación.
	Medio Plazo	En el ámbito de los meses: 1-3 años.	-Gestión de combustibles fósiles: gas y carbón -Coordinación hidrotérmica anual: Determinación valor del agua. -Mantenimiento de centrales.	Mayor predominio del factor técnico. Importante presencia del marco regulador en relación con los embalses.
	Corto Plazo	En el ámbito de la semana: desde días hasta un mes.	-Asignación de unidades o unit commitment. - Coordinación hidrotérmica mensual o semanal. -Despacho económico	Predominio factor técnico. El marco regulador afecta a los límites de producción térmica e hidráulica.

			-Gestión unidades de bombeo	
Mercado Liberalizado (Criterio de maximización de beneficios)	*Largo Plazo*	Ídem sistema regulado	-Evaluación de la inversión en nuevas instalaciones. - Gestión de riesgo. -Análiss contratos largo plazo: compra de combustibles y mercados derivados eléctricos.	Ídem sistema regulado
	Medio Plazo	Ídem sistema regulado	-Estimaciones presupuestarias -Subastas en mercados derivados. -Elaboración de estrategias relacionadas con el precio y la cuota de mercado.	Ídem sistema regulado
	Corto Plazo	Ídem sistema regulado	-Subastas estratégicas: energía y servicios auxiliares o secundarios.	Ídem sistema regulado

Tabla 1. Resumen del marco temporal de la planificación de la operación.

Los sistemas de energía eléctrica se estructuran en centros de producción (generación), de transporte (red de alta tensión), de distribución (red de media y baja tensión), y de consumo, además de los sistemas asociados de protección y control. A efectos de este estudio, nos centraremos en los sistemas de producción.

A su vez, como se indica en la Tabla 1, los centros de generación desarrollan su actividad en función del modelo económico al que están sujetos: bien el modelo tradicional regulado, bien el modelo liberalizado, o un modelo mixto. Actualmente la mayoría de los países, presentan una tendencia liberalizadora. A efectos de este estudio, nos centraremos en un contexto regulado.

En el sistema regulado, la planificación de la operación, basada en la optimización de costes, tiene como objeto la minimización de los costes de producción de energía de las centrales térmicas, centrales de bombeo, hidroeléctricas[3] y nucleares, de forma que se respeten todas las restricciones de explotación (técnicas, legales y medioambientales). Todo ello, tiene además como objetivo último, satisfacer la demanda del consumidor, bajo el cumplimiento de ciertos criterios de fiabilidad y seguridad[4].

En este sistema, la actividad queda como una función centralizada de la operación, en la que un operador centralizado, comúnmente denominado operador del sistema, vela por la seguridad del sistema, manteniendo un esquema de control y supervisión para el entorno tradicional, que se describirá a continuación, centrándose en el caso español.

Para el caso específico de España, (y en general para el resto de países) se establece un horizonte temporal y una jerarquía de la toma de decisiones. El horizonte temporal de un modelo de planificación de energía eléctrica en un entorno regulad sigue, en el contexto

[3] El funcionamiento de los elementos de generación se explicará con sumo detalle en la revisión sobre el diseño e implementación del modelo de planificación (Capítulo 4).
[4] La fiabilidad y seguridad son en grandes rasgos sinónimos. Sin embargo, al tratar el horizonte temporal, fiabilidad se refiere al ámbito del largo plazo y seguridad al del corto.

español, la siguiente estructura, en la que las decisiones a más largo plazo influyen y retroalimentan a las más cortoplacistas:

1) Largo plazo (meses a años):
   a) Planificación anual: Se realiza una planificación de la producción de energía eléctrica para un año completo. Se consideran las previsiones de demanda a largo plazo, las políticas energéticas nacionales y europeas, los objetivos de integración de energías renovables y las necesidades de inversión en nuevas capacidades de generación.
   b) Evaluación de escenarios: Se realizan análisis de escenarios para evaluar el impacto de diferentes factores, como cambios en la demanda, la evolución de los precios del combustible, la incorporación de nuevas tecnologías y las políticas de reducción de emisiones. Esto permite tomar decisiones estratégicas en la planificación, como la construcción de nuevas plantas de energías renovables, la modernización de infraestructuras existentes o la implementación de políticas de almacenamiento de energía. Por tanto, hay una enorme influencia de factores económicos y de la incertidumbre.
2) Medio plazo (semanas a meses):
   a) Programación semanal: se realiza la programación de la generación eléctrica para una semana completa, teniendo en cuenta las previsiones de demanda, los mantenimientos programados de las centrales y las restricciones de recursos. También se consideran los intercambios de energía con otros países, a través de las interconexiones existentes y la compra de combustibles fósiles (gas, carbón y petróleo principalmente). Además, se realiza una planificación conocida como coordinación hidrotérmica y la determinación del valor del agua (explicación más exhaustiva en el capítulo 4).
   b) Planificación de mantenimiento: se establecen los períodos de mantenimiento y las paradas programadas de las unidades de generación para garantizar su adecuado funcionamiento y minimizar los impactos en la producción eléctrica. Esto incluye tanto las centrales nucleares, hidroeléctricas, de ciclo combinado, eólicas y solares, entre otras.

3) Corto plazo (horas a días):
   a) Programación horaria: se establece la programación horaria de la generación eléctrica para las próximas 24-48 horas (arranque y paradas de centrales de generación para satisfacer la demanda, también conocido como *unit commitment*). Se consideran las previsiones de demanda, la disponibilidad de recursos (como centrales térmicas, hidroeléctricas, eólicas, solares, etc.) y las restricciones operativas establecidas por el Operador del Sistema Eléctrico.
   b) Gestión de demanda: se evalúa la respuesta de la demanda y la capacidad de ajustar el consumo en función de las señales de precios o la disponibilidad de energía (despacho económico o *economic dispatch* en la literatura anglosajona). Se pueden implementar estrategias de gestión de la demanda, como programas de eficiencia energética o incentivos para el consumo en horarios de menor demanda.

Es importante tener en cuenta que el sector eléctrico en España está regulado por la Comisión Nacional de los Mercados y la Competencia (CNMC) y el Operador del Sistema Eléctrico (Red Eléctrica de España), quienes establecen las pautas y requisitos específicos para la planificación y operación del sistema eléctrico en el país. Los horizontes temporales mencionados anteriormente, siguen las regulaciones y políticas energéticas establecidas en la Ley del Sector Eléctrico, Ley 24/2013, artículo 4.

Para lograr una planificación óptima del Sistema Eléctrico Español, es indispensable contar con herramientas computacionales (se revisarán en los próximos apartados) que utilicen métodos de optimización matemática eficientes.

A continuación, se procederá a realizar una revisión resumida del modelado de un problema de optimización matemática:

En el contexto de la toma de decisiones y la resolución de problemas, la optimización desempeña un papel fundamental para la obtención de los mejores resultados posibles. Los modelos de optimización permiten tomar decisiones informadas basadas en algoritmos matemáticos y restricciones. En este trabajo, se pretende explicar los principios fundamentales y el funcionamiento de los

modelos de optimización, destacando su relevancia en diversos ámbitos como el del transporte, gestión económica de recursos, etc.

Los modelos deterministas, son, representaciones matemáticas de problemas del mundo real que asumen que todos los parámetros de entrada y las restricciones se conocen con certeza. Estos modelos se basan en relaciones matemáticas y ecuaciones específicas para optimizar una determinada función objetivo, la cual o bien se maximiza (generalmente suele estar vinculada a los beneficios, pero no necesariamente) o se minimiza (generalmente se suele asociar con los costes).

Algunos componentes de los modelos deterministas son:

1) Variables de decisión: representan aquello desconocido que desea optimizarse. Por ejemplo, una variable de decisión podría ser en el caso que nos concierne de producción eléctrica, si se enciende una central térmica o no en un período determinado (variable binaria). Existen otros tipos de variables en función del dominio matemático que abarquen.
2) Función objetivo: función que se optimiza, ya sea maximizando o minimizando su valor. En lo que atañe a este proyecto, un ejemplo de función objetivo podría ser la minimización de los costes de producción de energía eléctrica para un Operador del Sistema Centralizado.
3) Restricciones: límites o cotas dentro de las cuales debe resolverse el problema de optimización. Estas limitaciones, pueden ser ecuaciones o desigualdades matemáticas, condiciones lógicas, las cuales restringen valores factibles de las variables de decisión.
4) Variables duales: también conocidas como multiplicadores de Lagrange o precios sombra. Estas, están asociadas con las restricciones del problema de optimización y dan información fundamental acerca de la sensibilidad de la función objetivo a cambios en las restricciones. Un ejemplo de variables duales podría ser el incremento de un coste marginal asociado a una restricción de producción de un bien, es decir, que por cada unidad de producción del bien que se añada, el incremento de dicho coste vendrá determinado por su variable dual asociada a la restricción.

A partir de la construcción de las 3 primeras componentes descritas del modelo de optimización, se procede a la resolución del problema, mediante la aplicación dealgoritmos matemáticos. El algoritmo explora iterativamente las diferentes soluciones factibles hasta encontrar una solución óptima. Tras obtener una solución óptima, se evalúa la viabilidad y calidad de esta mediante el contraste con [1] y, si esta es válida se aplica en el campo de trabajo respectivo.

En definitiva, como cierre de este apartado, para el desarrollo de este proyecto, teniendo en cuenta los conceptos anteriores, se puede proceder a la explicación del modelo de planificación que se estudiará con mayor detenimiento en los capítulos posteriores:

En este proyecto, se va a estudiar un modelo lineal, determinista, de la planificación de la operación en el horizonte temporal del medio plazo. Dado que se utilizará también el modelo en un contexto centralizado, se asume que existe un único Operador Centralizado del Sistema, que tiene como objetivo la minimización de los costes de producción de los sistemas de producción de energía eléctrica objetos de este estudio (termoeléctrica e hidráulica), pero siempre bajo el estricto cumplimiento de las restricciones técnicas, medioambientales y legales pertinentes, además de la satisfacción de la demanda del consumidor de energía eléctrica (particulares, empresas, etc). Debido a la complejidad del problema se necesitará el uso de optimizadores comerciales, resultando codificados en algún lenguaje específico, para resolver el problema de optimización. En los apartados siguientes, se explicarán las herramientas informáticas para la realización de este proyecto.

2.2 PYTHON COMO LENGUAJE DE PROGRAMACIÓN

Python es un lenguaje de programación de alto nivel, interpretado, desarrollado bajo la premisa de ser fácil tanto para el aprendizaje como para el desarrollo de la programación por parte del usuario. Fue desarrollado por el ingeniero holandés Guido Van Rossum a finales de los 80 y se ha convertido en uno de los lenguajes más populares en la industria del desarrollo de software. Algunas características del lenguaje de programación son las siguientes:

- ***Multiparadigma:*** Python es un lenguaje que soporta parcialmente la programación orientada a objetos (POO) [8], programación imperativa [9] y, en menor media, programación funcional[5] [10].
- ***Interpretado:*** A este tipo de lenguajes también se le conocen como *lenguaje scripting*, es decir, el código escrito no se traduce realmente a un formato legible por el ordenador en tiempo de ejecución.
- ***Dinámico:*** "Un lenguaje de programación es dinámicamente tipado si una variable puede tomar valores de distintos tipos. La mayoría de los lenguajes de tipado dinámico son lenguajes interpretados, como Python o Ruby. Un lenguaje que no es dinámicamente tipado se dice que es de tipado estático, o estáticamente tipado." [11]
- ***Multiplataforma:*** Concepto también empleado a través de su sinónimo: portabilidad. Hace referencia a la capacidad del lenguaje de poder ejecutarse en distintos dispositivos y/o sistemas operativos. Entre otras implicaciones, destacan que un mismo código puede ejecutarse en Windows, Linux, y Macintosh.

Para entender mejor el funcionamiento de Python, que se desarrollará a continuación, es necesario dar un paso atrás y mencionar brevemente un documento que realiza de forma sinóptica una exposición de 20 de los principales principios para la programación en Python. Se trata de la obra *The Zen of Python* de Tim Peters [12], entre cuyos principios destacan:

[5] Ver bibliografía para una explicación detallada de los 3 conceptos mencionados en este apartado.

<<Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Readability counts.

Although practicality beats purity:

Explícito es mejor que implícito

Simple es mejor que complejo

Complejo es mejor que complicado

La legibilidad cuenta

Lo práctico le gana a la pureza >>

Estos mantras facilitarán la comprensión del funcionamiento de Python: El lenguaje de programación Python utiliza módulos de código que son intercambiables en lugar de una lista de instrucción (convención estándar para los lenguajes de programación funcional). Por ello, la implementación estándar de Python no convierte su código en uno directamente interpretable para la máquina o hardware. En vez de eso, lo convierte en lo denominado "código de byte", así que como ocurre cuando hay personas que se intentan comunicar en distintos idiomas, al utilizar códigos, necesitamos también un intérprete. En este caso, al intérprete (invisible para el usuario) se le denomina Máquina Virtual Python (PVM), que ejecuta los códigos de bytes.

El intérprete de Python realiza las siguientes tareas o pasos para la ejecución de un programa determinado:

1. El intérprete lee un código o instrucción de Python realizado en un editor (*PycharmProjects, Jupyter Notebook, Sublime, Visual Studio*,etc) verifica la sintaxis, línea por línea; de encontrar algún error, detiene inmediatamente la

traducción y muestra un mensaje de error[6]. Es en este paso, en el que se importan las librerías de Python que no estén instaladas por defecto (*Matplotlib, Pandas, SciPy, Pyomo*, etc) en función de las distintas funcionalidades que se necesiten.

2. De no haber ningún error, es decir, el código presenta una sintaxis correcta, se procede a la traducción de la instrucción al lenguaje intermedio "código byte".
3. Se procede al envío del código byte a la Máquina Virtual Python, en donde se ejecuta el código del byte en PVM. Si se produce un error durante esta ejecución, ésta se detiene con un mensaje de error. (Típicamente a estos tipos de error se les denomina *RunTimeErrors*).

La figura 1, expone a modo de resumen gráfico el proceso de compilación y ejecución de Python.

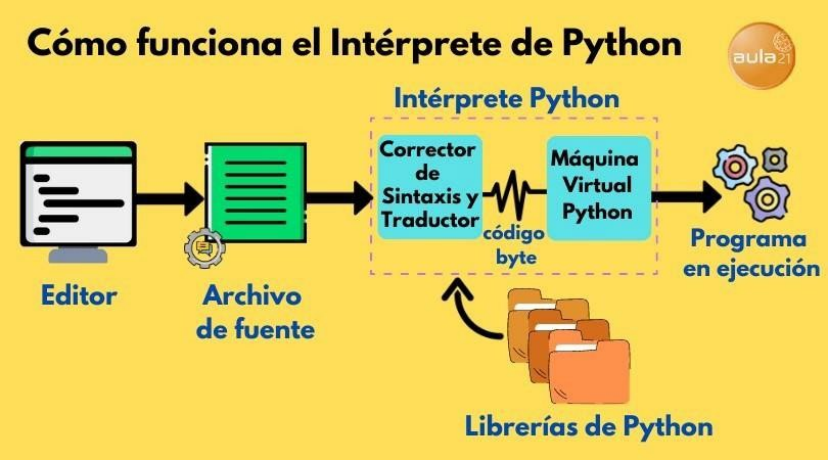


Figura 1. Funcionamiento compilación en Python. Fuente: [13]

Finalmente, en este subapartado, es preciso realizar un análisis de las ventajas y desventajas del programa Python respecto de otros lenguajes como pueden ser Java, C, C++, Ruby…

[6] Para ver los principales tipos de error en Python consulte Anexo I.

seguido, de una breve explicación de aplicaciones y ejemplos de uso de este lenguaje, además de una explicación del porqué de la selección de este lenguaje.

USO DE PYTHON FRENTE A OTROS LENGUAJES DE PROGRAMACIÓN	
VENTAJAS	*INCONVENIENTES*
-Lenguaje en alto nivel. -Polivalente. -Bibliotecas y *frameworks*. -Portabilidad. -Gratuito y código abierto. -Baja curva de aprendizaje. -Comunidad muy activa	-Lentitud de ejecución -Alto consumo de memoria. -Desarrollo móvil. -Excesiva sencillez. -Limitado para representación 3D.

Tabla 3. Ventajas y desventajas en el uso de Python.

Entre las aplicaciones principales para el uso de Python destacan:

- Análisis de datos: Python es ampliamente utilizado para el análisis de datos, teniendo la biblioteca Pandas, como su principal exponente, dado que proporciona una estructura de datos flexible y herramientas para realizar operaciones complejas en conjuntos de datos.
- Automatización de tareas: Python es una opción popular para automatizar tareas repetitivas, debido a su facilidad de uso y aprendizaje.
- Inteligencia Artificial (IA): Python se ha convertido en uno de los lenguajes dominantes en el campo de la inteligencia artificial, con bibliotecas como *TensorFlow* y *PyTorch*, las cuales permiten realizar problemas avanzados de aprendizaje avanzado y redes neuronales.

- Creación de sitios web y aplicaciones: destacando el uso de *frameworks* como *Django.* Ejemplos de aplicaciones y sitios web destacados podrían ser YouTube, Instagram, el motor de búsqueda de Google o bien la Bolsa de Nueva York (NYSE), todas ellas dependen en cierta medida de instrucciones de Python.

En definitiva, teniendo en cuenta lo expuesto en el presente apartado y en el apartado 2.1, resulta entendible y lógico, entender la decisión para realizar la automatización de un problema de optimización de la planificación (es decir, un programa de ciencia de datos), mediante el uso del lenguaje de programación Python. A pesar de las desventajas que acarrea una lentitud de ejecución (que puede ser optimizada en cierto modo), y el gran uso de memoria, dado que el alcance de este programa es pequeño y puramente académico, las numerosas ventajas ofertadas (expuestas en la Tabla 3) se sobreponen con creces a los inconvenientes.

Para la realización de este programa se utilizará el editor dinámico de *Jupyter Notebook* integrado en la distribución de lenguajes para la computación científica, Anaconda. Entre las posibilidades existentes (*Pycharm*, editores de texto como *Sublime* invocados por la consola de *Windows*, *VisualStudio*…), *Anaconda* no solo engloba a algunos de ellos (además del lenguaje R), sino que tiene como ventaja la simplificación de la gestión e implementación de paquetes como se expondrá en el apartado siguiente. Por otro lado, la elección de *Jupyter Notebook* como editor, obedece a principios de pragmatismo, pues su carácter interactivo y dinámico, permite al programador compilar el programa por bloques, además de la posibilidad de integrar anotaciones de texto, imágenes y otros tipos de contenido multimedia.

La versión de Python utilizada es la 3.10, y se utilizan los paquetes (además de los ya integrados) de: *Matplotlib, Pandas, os* y *Pyomo*, entre otros. A modo de introducción o manual, en el siguiente apartado se expondrán las principales características de Pyomo, facilitando así una mejor comprensión del estudio realizado en este proyecto.

2.3 PYOMO: INTRODUCCIÓN, CONCEPTOS BÁSICOS Y EJEMPLOS

Antes de comenzar la exposición teórica de Python Pyomo, es preciso realizar una breve explicación de:

1. Paquetes informáticos requeridos
2. Opciones para la realización del proyecto.
3. Bibliografía y medios para realizar preguntas.
4. Recomendaciones personales para la realización de proyectos similares.

En primer lugar, es preciso indicar que todos los programas realizados o expuestos siguen las siguientes versiones de los softwares o librerías indicados a continuación:

- Python: v. 3.9.15
- *Pyomo*: v. 6.5.0
- *Matplotlib*: v. 3.5.1
- *Pandas*: v. 1.5.3

Por tanto, en caso de ser utilizado este estudio en un futuro, el lector deberá tener en cuenta las implicaciones, en términos de desuso y/o actualización, añadido, eliminado o supresión, de funciones, módulos, objetos utilizados en el presente proyecto.

En segundo lugar, se procede a la explicación del porqué de la elección de los medios informáticos (Anaconda y *Jupyter Notebook*) usados en este proyecto, mediante unanálisis del estado del arte de las posibles opciones disponibles. En primera instancia, aquellas personas con cierto nivel de programación, puesto que ya tienen determinado conocimiento sobre los procesos de ejecución y *debugging* de los programas informáticos, recurrirán a la opción del uso de la ventana de comando o *Command Prompt,* en consonanciacon un editor de texto convencional (i.e *Notepad, Notepad++, Sublime*). Esta, es la opción más primitiva en cuanto a visualización y facilidad para encontrar errores. Sin embargo, es la más rápida entre las opciones que se evaluarán. En la figura 2 se observa un posible *display*de la primera opción tratada.

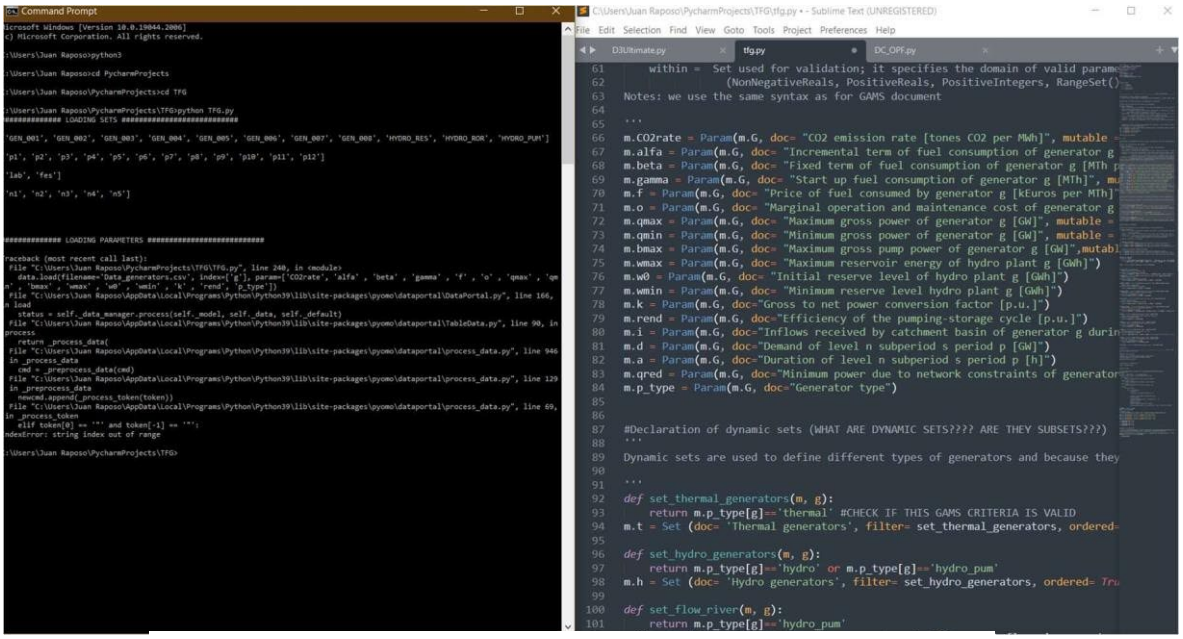


Figura 2. Posible visualización del proyecto mediante el uso del *Command Prompt* (izq) ligado al editor de texto *Sublime* (dcha). Fuente: Elaboración propia

Por otro lado, existe una opción más atractiva para un recién iniciado en programación en *Python*. Se trata del uso de Entornos de Desarrollo Integrados o bien en inglés (pues sus siglas están más extendidas en la literatura): *Integrated Development Environment* (IDE). Una IDE es un software que proporciona a los desarrolladores una plataforma para escribir, probar y depurar programas. Esta herramienta, combina un editor de código, un compilador o intérprete (revísese el apartado 2.2 si no se recuerda su significado), un depurador, además de otras utilidades menores, todo ello dentro de una interfaz unificada. El objetivo de las IDEs es mejorar la productividad de los programadores al integrar todas las herramientas descritascon anterioridad. Entre las principales IDEs, destacan: *Visual Studio, Eclipse y PyCharm*. Elprogramador que se enfrente a un problema determinado, y escoja la opción de las IDEs,

deberá escoger la que mejor se adapte a sus necesidades y preferencias. En la figura 3 se observa una posible visualización del Trabajo de Fin de Grado en la IDE *PyCharm*.

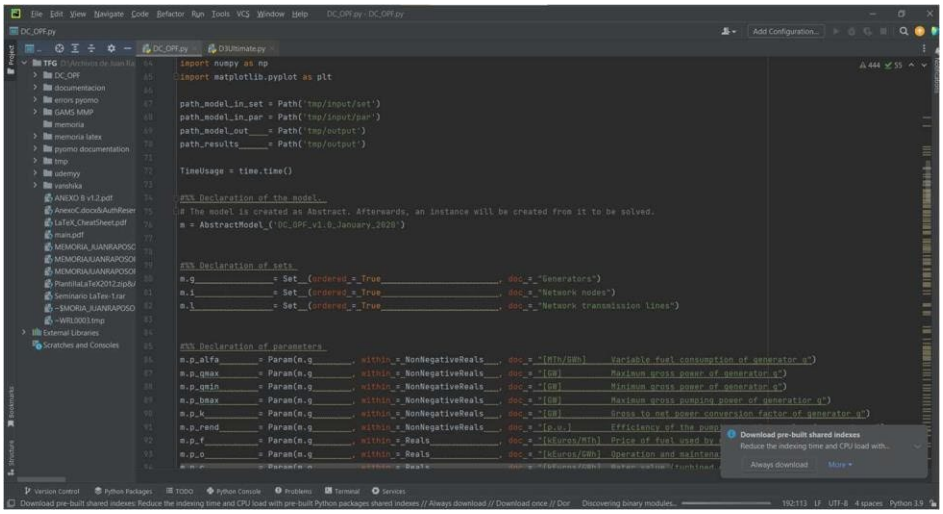


Figura 3. Posible visualización del proyecto mediante el uso de *PyCharm*.

Fuente: Elaboración propia

Que el autor de la presente obra no seleccione esta opción, obedece a criterios totalmente arbitrarios del mismo, pues considera, que la interfaz, presenta un exceso de información en la pantalla, lo cual, a efectos de productividad de programación, esta puede verse mermada significativamente.

Finalmente, se trata la última opción posible (conocida y estudiada por el autor de este proyecto): *Anaconda* y *Jupyter Notebook*. La elaboración de programas utilizando *Anaconda* y *Jupyter Notebook* proporciona un entorno de desarrollo interactivo y potente (menos saturado, en opinión del autor de este trabajo, que las IDEs) para la programación en diversos lenguajes, como Python, R y Julia. Véase cómo funciona y por qué es una opción popular para muchos desarrolladores y *DataAnalysts*.

Anaconda es una plataforma de código abierto que facilita la gestión de paquetes y entornos de desarrollo. Proporciona una forma conveniente de instalar y administrar diferentes lenguajes de programación, bibliotecas y herramientas necesarias para el desarrollo de software y análisis de datos. Además, *Anaconda* incluye su propio gestor de paquetes

denominado "conda", que permite instalar, actualizar y gestionar fácilmente las dependencias del proyecto, como se muestra en la figura 4.

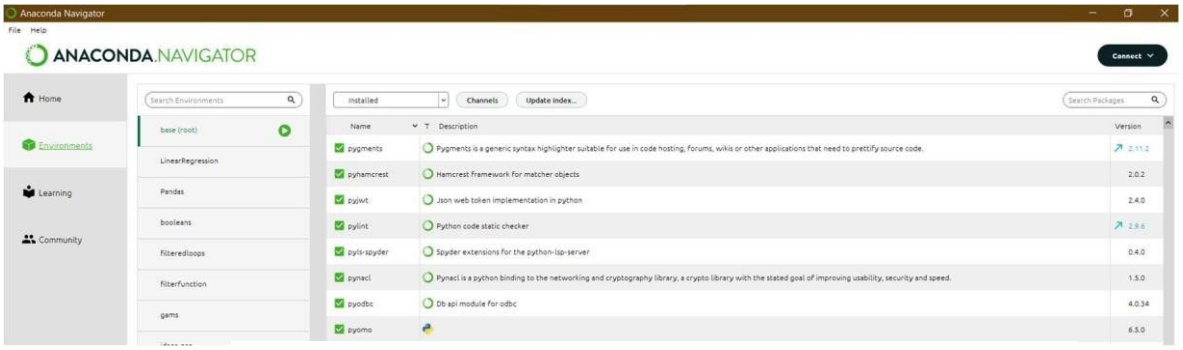


Figura 4. Visualización de los paquetes instalados en *Anaconda*. Fuente: Elaboración propia

Dentro de Anaconda, *Jupyter Notebook* es una aplicación web que permite crear y compartir documentos llamados "notebooks". Estos notebooks contienen código ejecutable, visualizaciones interactivas y texto enriquecido, lo que permite combinar código, resultados y explicaciones (imágenes, texto y /o vídeo) en un único documento. La popularidad de *Jupyter Notebook* reside en su capacidad para realizar análisis exploratorio de datos y crear informes interactivos.

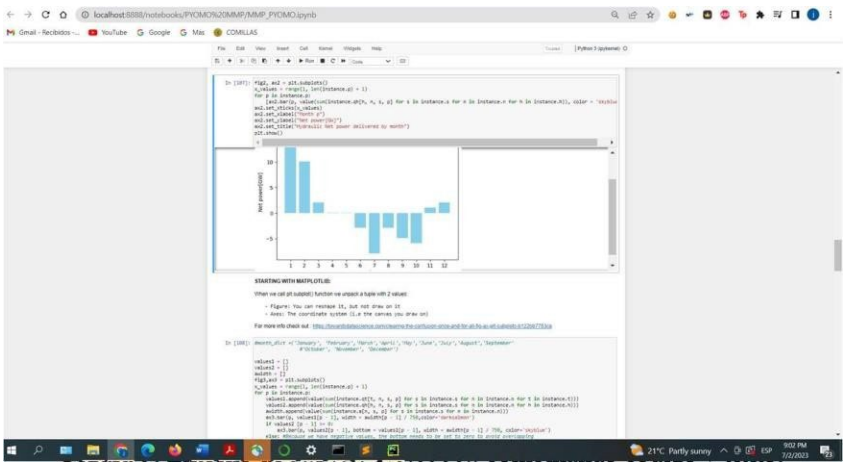


Figura 5. Entorno de *Jupyter Notebook* combinando código y texto. Fuente: Elaboración propia

Una de las ventajas de utilizar *Anaconda* y *Jupyter Notebook* se debe a su facilidad de uso y configuración. *Anaconda* simplifica enormemente tanto la instalación como la gestión de paquetes, lo cual en ciertos entornos tradicionales como los descritos con anterioridad, puede implicar complicaciones durante el desarrollo del programa. Además, *Jupyter Notebook* proporciona una interfaz web intuitiva consistente en la creación y ejecución en celdas individuales, lo que permite un enfoque interactivo y experimental en el desarrollo (figura 5).

En resumen, la elaboración de programas utilizando *Anaconda* y *Jupyter Notebook* ofrece un entorno de desarrollo interactivo, flexible y potente. La combinación de la gestión de paquetes de *Anaconda,* la facilidad de uso de *Jupyter Notebook* y su capacidad para integra, código visualizaciones y documentación en un solo lugar hace que esta opción sea atractiva para desarrolladores y analistas de datos y, en este caso, para el autor de este Trabajo de Fin de Grado.

Es preciso mencionar, que existen otras opciones no exploradas en este proyecto, pero de posible gran potencialidad como son *Google Colab* y *Neos.*

La documentación sobre Python *Pyomo* (tercer bloque de este apartado), es su principal talón de Aquiles, al estar dispersa, oculta y en la mayoría de las ocasiones desactualizada. Por motivos que se expondrán, se recomienda encarecidamente que cualquier programador que codifique en Python *Pyomo* rehuse de usar la herramienta actual ChatGPT 3.0 para la consulta de dudas, específicas, pues sus análisis pueden ser erróneos o bien usar documentación desactualizada[7].

La primera fuente de información que se recomienda visitar es la página oficial de Pyomo [14], pues contiene la información más actual y revisada. El software de código abierto de Pyomo está alojado en GitHub [15]. Por tanto, es recomendable como iniciación revisar esta documentación en la que se encuentran multitud de ejemplos con los que empezar a

[7] Esto se debe, a que la base de datos de ChatGPT se remonta hasta 2021 únicamente y, como se indicará, Pyomo se caracteriza por ser código abierto, es decir, está sujeto a constantes actualizaciones y revisiones.

familiarizarse con la herramienta de *Pyomo*. Seguido de este repositorio, se recomiendan también otros, entre los que destacan:

- OptimizationExpert [16]: este repositorio contiene ejemplos de gran complejidad. Lo más recomendable es acceder a él cuando el alumno o programador haya alcanzado cierto nivel y manejo de la herramienta de *Pyomo.*
- Repositorios del IIT de ICAI [17]: en él se pueden encontrar varios ejemplos de cierta complejidad como puede ser un problema de *unit commitment*.
- Repositorio personal [18]: en él se encuentran desde problemas sencillos e introductorios a la herramienta, como pueden ser optimizar el número máximo de damas en un tablero de ajedrez sin que estas se puedan comer, hasta el programa para el proyecto en cuestión.

Estas recomendaciones prácticas permiten al programador conocer las principales funcionalidades y estructuras de los programas en alto nivel. Sin embargo, para completar y complementar este aprendizaje, es necesario realizar un análisis más exhaustivo, más teórico, para que el alumno y/o programador pueda empezar a elaborar programas por cuenta propia y demostrarse así a sí mismo que domina completamente los conceptos trabajados. Para ello, se recomienda una revisión de [19][20] y [21]. Estos y algunos títulos más se tratarán con mayor exhaustividad en la revisión del estado del arte (capítulo 3). Finalmente, se recomienda el uso de materiales, como los que se encuentran en YouTube (de suma relevancia la línea de trabajo desarrollada por la Dr Katie, Profesora Asociada del Instituto Tecnológico de Rochester) y algún curso en plataformas online (la plataforma Udemy, avalada por entidades como el Nasdaq o Volkswagen, ofrece un buen repertorio formativo). Mencionar también con cierto énfasis, y reiterando la idea mencionada al inicio de este apartado, que el uso de la herramienta de ChatGPT, puede ocasionar numerosas confusiones, lo que puede conllevar a un estancamiento y retraso en la elaboración de un programa. Ello no quiere decir que todas las respuestas a las preguntas realizadas sean incorrectas, pero que, si podría ser necesario consultar una segunda fuente como respaldo, pues existen 2 posibles fuentes de error:

- La cuestión realizada, puede estar mal formulada por parte del programador.
- La información encontrada por el algoritmo de ChatGPT es incorrecta o desactualizada.

Formas alternativas de contraste de información, a la bibliografía mencionada, podría ser la consulta de información en distintas plataformas web como las que se proceden a explicar.

En primer lugar, se encuentra *Stack Overflow*. Stack Overflow es una plataforma de preguntas y respuestas orientada a la programación y desarrollo de software. Esta *grosso modo* funciona mediante un sistema de publicación y etiquetado de preguntas, con su correspondiente realimentación en forma de respuesta, o votación de la relevancia de la pregunta. En función de la reputación de la pregunta y sus respuestas, el usuario de la plataforma puede ganar medallas y privilegios que le permitirán participar más activamente.

Otro método muy útil de comunicación directa del programador con expertos, autores de publicaciones de *Pyomo*, es el foro de *Pyomo* dentro de una herramienta conocida como *Google Groups*. Google cuenta con diversos foros de soporte para sus productos y servicios, en el que de forma similar a *Stack Overflow* se categorizan preguntas por tema y existe una moderación y respuestas a las mismas.

En definitiva, tanto Stack Overflow como los foros de Google de *Pyomo,* son plataformas donde los usuarios pueden buscar respuesta a sus preguntas y obtener ayuda de la comunidad de programadores y usuarios.

Como cierre a este bloque de preparación previa, me gustaría añadir algunas pautas personales, que recomendaría para futuros programadores que se inicialicen en la programación en *Pyomo*. La idea de estos principios sigue el concepto desarrollado por el célebre arquitecto Mies Van der Rohe de “Less is more” unida a los principios para la programación en Python descritos en [12]:

- Empezar a programar casos pequeños: lo principal en primera instancia, es familiarizarse con la herramienta y comprender en alto nivel cómo funcionan tanto Python como *Pyomo*.
- Constante comprobación y revisión de resultados: es crucial para el éxito en cualquier proyecto de programación, comprobar constantemente cómo evoluciona la ejecución del proyecto. Para ello, es recomendable servirse de las herramientas de "printeo" proporcionadas.
- Proceso anidado: dividir el proyecto en pequeños bloques, facilita su comprensión y detección de errores.
- Equivocarse: la clave del aprendizaje en programación es el aprender del error, y comprender el porqué, pues acelerará la curva de aprendizaje del programador.

Como segunda parte del apartado 2.3 sobre *Pyomo*, se procede a la explicación de la mencionada herramienta.

En este proyecto se describe una herramienta para un modelado matemático: el paquete de software "the Python Optimization Modeling Objects" (Pyomo). Pyomo permite la formulación y análisis de modelos matemáticos para problemas avanzados de optimización. Esta capacidad se suele asociar con lenguajes de modelización algebraica (AMLs en inglés), pudiendo destacar AMPL [22] y GAMS [23].

Los AMLs son lenguajes de alto nivel, utilizados para describir y resolver problemas de optimización. La importancia de este matiz reside en que, el alto nivel minimiza las dificultades asociadas con el análisis de problemas de optimización, pues implementan una sintaxis concisa e intuitiva, en términos de definición de variables, restricciones y funciones objetivo. Además, integran en la propia herramienta solucionadores o "solvers" tanto de código abierto(GLPK) como de uso comercial (CPLEX, Gurobi). Sin embargo, al no estar integrados en un lenguaje de programación de bajo nivel (véase Java, C++), su proceso de depuración o "debbuging" diverge de dichos lenguajes. Por tanto, sería ideal, que existiese un AML integrado en un lenguaje de programación de bajo nivel: Pyomo, la cual está integrada en el lenguaje de software Python, proporcionando a esta herramienta, numerosas funcionalidades, que se

tratarán a continuación. Otro ejemplo de paquete de optimización integrado es TOMLAB Optimization Environment [24], integrado en la herramienta de MATLAB. Estos dos ejemplos representan un enfoque de dos lenguajes que aúnan la flexibilidad de los lenguajes en alto nivel, con la eficiencia de los lenguajes en bajo nivel para las computaciones numéricas.

Por este motivo, para el desarrollo de este proyecto se utilizará la herramienta de Pyomo. Sin embargo, hay que evitar caer en idealizaciones pues este sistema mixto tiene alguna contraprestación.

En primer lugar, el tiempo de creación de instancias (su significado se describirá más adelante, pero se refiere a la unidad de datos que Pyomo crea para la lectura y procesamiento de archivos externos). En función del tipo de problema, el tiempo varía, pero como se observa en la figura 6.

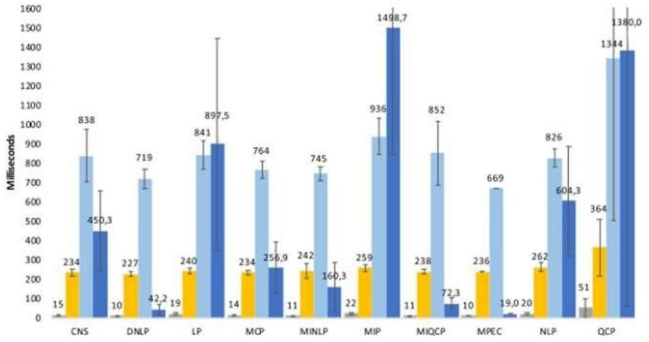


Figura 6. Tiempo de generación medio de instancias de un modelo prototípico. Fuente: [25]

En el caso de este proyecto, se podría tomar como referencia el caso de programación lineal (LP). Se observa que el tiempo de generación de Pyomo cuadriplica al de GAMS y es del orden de 40 veces superior al de AMPL.

En caso de querer tratar de escalar el modelo, para un número elevado de variables, estas diferencias varían significativamente, una respecto a otra, como se puede observar en la figura 7.

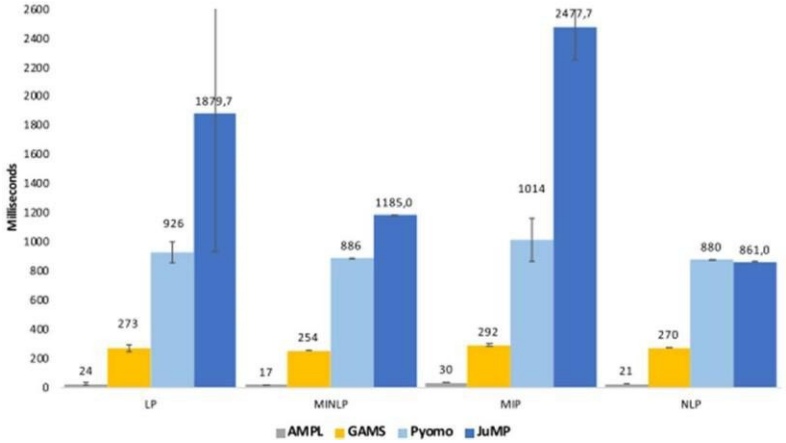


Figura 7. Tiempo de generación medio de instancias de un modelo a gran escala. Fuente: [25]

En el caso del LP, se observa que el tiempo de ejecución del AMPL ha aumentado en un 26%, mientras que en el caso de GAMS y Pyomo, un 13,75 % y 10,10% respectivamente. Aunque en términos absolutos, este aumento es menor al escalar un modelo, las diferencias entre AMLs no son despreciables.

Sin embargo, la opción escogida para este proyecto, por ser de tamaño reducido, seguirá siendo Pyomo, pues las diferencias de tiempos a pequeña escala, podemos considerarlas despreciables (recuerde que las medidas de las figuras 6 y 7 son del orden de milisegundos), y sus características particulares por implementar dos niveles de lenguaje esta opción, a juicio crítico del autor de este proyecto, la óptima.

La razón de que este lenguaje sea el más válido para la realización de este proyecto reside en diversos factores, los cuales necesitan ser contextualizados previamente explicando cual es la función de la modelización matemática.

El modelado es fundamental en la investigación científica, la ingeniería, los negocios, etc. Modelar implica formular una representación simplificada de un sistema o de un objeto de la realidad. El uso de estas simplificaciones permite al investigador, al científico, ingeniero,

etc. obtener una representación estructurada de cierta información acera del sistema original. Esta información de los modelos se utiliza para:

- Explicar fenómenos que surgen en un sistema.
- Predecir comportamientos de estados futuros de un sistema.
- Entender las causas que influencian fenómenos determinados en un sistema (i.e. disminución del coste de producción de un sistema al añadir energías renovables).
- Identificar casos extremos.
- Analizar el coste de oportunidad para ayudar en la toma de decisiones de un operador humano.

Debido a que las decisiones que se realizan en el mundo de la ingeniería, científico, de los negocios, el volumen de información a analizar es muy vasto, el ser humano precisa de la ayuda de la computadora, para agilizar el proceso de la toma de decisiones y que este sea eficiente. Para ello, en el caso de los modelos de optimización matemática, es decir, la gestión más eficiente (en general en términos económicos), precisa de herramientas informáticas conocidas como "solvers". Algunos ejemplos de solvers son: CMPLEX, GUROBI, IPOPT, GLPK, etc). Un solver, no es más que una arquitectura, conformada por una combinación de algoritmos que buscan solucionar el modelo de un problema, de la forma más eficaz posible. Las diferencias entre cada uno de los solucionadores exceden los objetivos de este apartado (ver 4.2.3 para un análisis detallado). Por tanto, es necesario una herramienta como Pyomo que actúe como "traductor", entre los modelos de optimización matemática y los solvers.

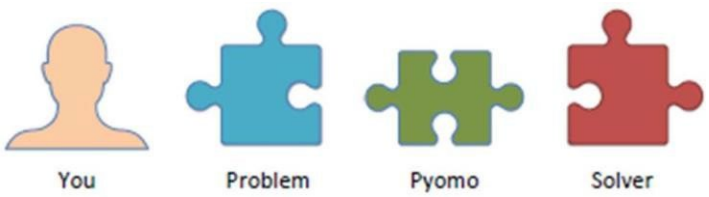


Figura 8. ¿Qué es Pyomo? Fuente: [26]

Una vez hemos entendido el significado de Pyomo, es preciso especificar susrasgos definitorios, sus características principales:

1. Python: Pyomo está integrado dentro del entorno Python, lo cual otorga a Pyomo una enorme potencialidad, dado que les permite acceso a todas las librerías disponibles en Python para el análisis de datos, destacando las librerías de Matplotlib, Pandas, Scipy, NumPy, etc. Que Python esté embebido en Pyomo requiere que el programador, esté familiarizado con algunos conceptos básicos de Python, pues así se puede explotar la documentación de Python y explorar numerosas opciones disponibles para el desarrollo.
2. Código abierto: El desarrollo de Pyomo se encuentra en GitHub [15] y sus funciones, módulos, son manejadas en los grupos de Google ("Google Groups") citados con anterioridad. Esta flexibilidad, puede generar aparentes dudas, pues la integridad de Pyomo se puede ver alterada. Sin embargo, existen una serie de protocolos que garantizan la robustez y la confianza de Pyomo. Por tanto, como consecuencias lógicas de esta característica, Pyomo es un AML flexible y personalizable.
3. POO (Programación Orientada a Objetos): Pyomo emplea un diseño de librerías orientada a objetos. Los modelos desarrollados en Pyomo son objetos de Python y las componentes del modelo son atributos.
4. Ambivalente: Las componentes de Pyomo, pueden ser utilizadas para expresar una gran variedad de problemas de optimización entre las que destacan: programas lineales, cuadráticos, no lineales, MILP[8], DP con ecuaciones diferenciales, etc.
5. Fácil integración de solvers: Las interfaces de Python, facilitan su integración.

Estas son algunos de los rasgos definitorios de Pyomo. Una vez explicado el significado, y las características principales, parece necesario realizar un análisis de la toma de decisiones o pasos a seguir durante la programación. Para ello, se hará una descripción

[8] *Mixed-Integer Linear Programming*

general, de la estructura, atributos y pasos de modelado. Esto será seguido de una explicación de las reglas de construcción (o protocolo de la sintaxis utilizada), una discusión de las diferencias entre un modelo abstracto y uno concreto, explicaciones sobre cómo modelar las componentes de Pyomo y finalmente un análisis de la codificación de los solucionadores y la exposición de los resultados.

En primer lugar, la estructura de un código de Pyomo es similar a la de Python, con una declaración de los módulos o librerías inicialmente, un desarrollo medio del código y una presentación de los resultados. El flujo de código en Pyomo, tiene la particularidad, de que va organizado en función de dos aspectos: las clases de Pyomo y el tipo de modelo a utilizar (concreto o abstracto).

Las clases de Pyomo, son objetos de Python, que permiten el modelado de los distintos elementos de un problema de optimización. Para facilitar la comprensión de este concepto, las explicaciones se apoyarán en pequeños casos prácticos realizados durante el alumno, previo a la programación del proyecto final.

Se tiene el siguiente modelo de optimización:

$$\min \sum_{i} c_i \, x_i \qquad (1)$$

$$s.t \sum_{i} a_{mi} \, x_i \geq b_m \qquad \forall m \qquad (2)$$

Si se organiza cada uno de los elementos en formato tabular:

ELEMENTO DEL PROBLEMA	EXPRESIÓN MATEMÁTICA	NÚMERO DE ELEMENTOS
Parámetros	a	m * i
	b	m
Variables	x	i
Restricciones	$\sum_{i} a_{mi}\, x_i \geq b_m$	m
Función objetivo	$\min \sum_{i} c_i\, x_i$	1

Tabla 4. Tabla resumen de un modelo de optimización matemática.

Las componentes del modelo están definidas en Pyomo mediante las siguientes clases de Python:

Set	Conjunto cuyos elementos son usados para definir una instancia del modelo. Es el índice o número de elementos de una variable, parámetro, restricción, función objetivo, etc.
Param	Datos de los parámetros que son utilizados para definir una instancia del modelo.
Var	Variables de decisión.
Constraint	Restricciones del modelo.
Objective	Expresión que es maximizada o minimizada en el modelo.

Tabla 5. Componentes del modelo.

Por tanto, una definición del modelo en Pyomo podría ser de la siguiente manera:

```
from pyomo.environ import*

model = AbstractModel()
model.N = Set() # N -> number of variables
model.M = Set() # M -> number of constraints
model.a = Param(model.M, model.N)
model.b = Param(model.M)
model.c = Param(model.N)
model.x = Var(model.N,within= NonNegativeReals)
def con_rule(model,m):
    return sum(model.a[m, i] * model.x[i] for i in model.N) >= model.b[m]
def obj_rule(model):
    return sum(model.c[i]*model.x[i] for i in model.N)
model.C1 = Constraint(model.M, rule= con_rule)
model.OF = Objective(rule= obj_rule,sense= minimize)
model.dual = Suffix(direction=Suffix.IMPORT)
```

Figura 9. Código en Pyomo del modelo. Fuente: [18] (Elaboración propia)

En el primer bloque, se importan las librerías necesarias. En este caso, se importan, de la librería de Pyomo, todos los elementos que se encuentran en ella (Set, Var, Objective, AbstractModel, ConcreteModel, etc) mediante el uso de la notación args*. Este tipo de práctica, en general, no es recomendable en programación en Pyomo, pues no asignar la librería a una variable de la forma estándar: "import pyomo.environ as pyo", puede provocar problemas de "Override", es decir, existe el peligro de sobrescribir otros métodos.

En el segundo bloque se realiza una codificación del modelo, mediante una analogía entre elementos del modelo y las componentes de Pyomo. Como se puede observar en el código (figura 9), todas las componentes (atributo de un objeto en Python) emanan de la definición del modelo (objeto de Python, en este caso un modelo abstracto) a modo de estructura en árbol.

Existen una serie de notaciones o protocolos a seguir en cuanto a notación del código, para facilitar su legibilidad, las cuáles pueden variar en función del programador, pero suelen ser comunes a todos los programadores las siguientes:

1. Para la declaración de las restricciones, es notación estándar escribir la expresión matemática, en una función inmediatamente antes de usar el comando Constraint.
2. También es recomendable, realizar una distinción en la declaración de las variables, parámetros, restricciones, etc. (en la escasa literatura existente, se suelen escribir los sets en mayúscula y el resto de los elementos en minúscula. En otros casos, se precede

al nombre de una componente, de una letra significativa, "v" en el caso de variables, "p" en el caso de parámetros y "c" en el caso de constraint[9].

3. En función de si el modelo es concreto o abstracto, el flujo de código varía ligeramente.

Los pasos básicos en un proceso de modelización sencillo son los siguientes:

1. Crear una instancia de un modelo usando las componentes de modelización de Pyomo.
2. Pasar la instancia a un solucionador para encontrar una solución.
3. Realizar un informe y analizar los resultados del solucionador

A la hora de crear la instancia (bloque de datos a analizar por Pyomo) debemos seleccionar el tipo de modelo. En Pyomo, un modelo concreto y un modelo abstracto se refieren a dos formas de construir y resolver modelos de programación matemática.

Se entiende como modelo concreto en Pyomo a todo aquel en el que se definen y especifican todas las componentes (variables, parámetros, restricciones, índices, etc) de forma explícita previo a la resolución del problema. Esto quiere decir, que un modelo concreto utiliza datos específicos integrados dentro de la codificación del modelo. Por tanto, se construye la instancia y se soluciona el modelo, con los solucionadores externos ya citados (Figura 10).

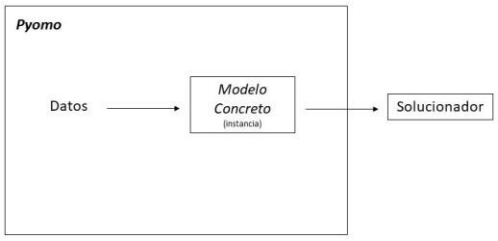


Figura 10. Modelo concreto Pyomo. Fuente: Elaboración propia

[9] Esta idea es tomada de [27]

Para complementar la comprensión del concepto se adjunta un ejemplo a continuación (Figura 11).

```
In [13]: from pyomo.environ import*

In [14]: model = ConcreteModel()
         model.L= Param(initialize= 10, mutable=True, within= NonNegativeIntegers)
         model.H1= Param(initialize= 2, mutable=True, within= NonNegativeIntegers)
         model.H2= Param(initialize= 4, mutable=True, within= NonNegativeIntegers)
         model.d1= Var(bounds= (0,model.L), initialize= 0, within= NonNegativeReals)
         model.d2= Var(bounds= (0,model.L), initialize= 0, within= NonNegativeReals)
         model.x= Var(bounds=(0, model.L), initialize= 0, within= NonNegativeReals)
         model.C1= Constraint(expr= model.d1**2 == model.x**2 + model.H1**2)
         model.C2= Constraint(expr= model.d2**2 == (model.L - model.x)**2 + model.H2**2)
         model.OF= Objective(expr= model.d1 + model.d2, sense= minimize)

In [15]: solver = SolverFactory('ipopt')
         results= solver.solve(model)

In [16]: print('d1=' , round(value(model.d1),2))
         print('d2=' ,round(value(model.d2),2))
         print('x=' ,round(value(model.x),2))
         print('OF=' ,round(value(model.OF),2))

         d1= 3.89
         d2= 7.77
         x= 3.33
         OF= 11.66
```

Figura 11. Ejemplo de modelo concreto en Pyomo. Fuente: Elaboración propia

En este ejemplo, se puede observar cómo este modelo no lineal (de ahí que el solucionador utilizado se IPOT) por ser concreto, presenta todos los datos integrados en la codificación.

Por otro lado, un modelo abstracto (el que se utilizará en este proyecto) se construye utilizando una estructura más flexible y generalizada. En este caso, en lugar de definir todos los detalles concretos del modelo, las componentes definidas son abstractas, y se inicializarán tras la carga de archivos de datos externos (.dat, .xls, .csv, .json, .yaml, etc). Por ejemplo, se pueden definir conjuntos de índices de longitud variable y sin definir valores exactos, establecer parámetros, variables, restricciones y funciones objetivo, sin asignar valores numéricos.

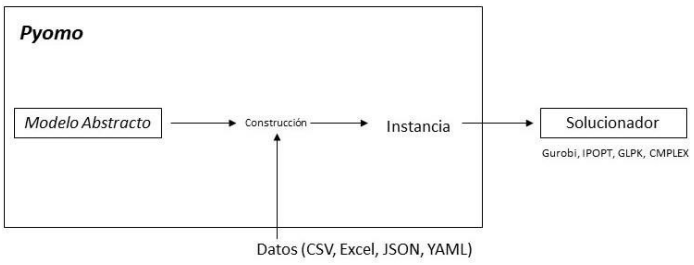


Figura 12. Modelo abstracto en Pyomo. Fuente: Elaboración propia

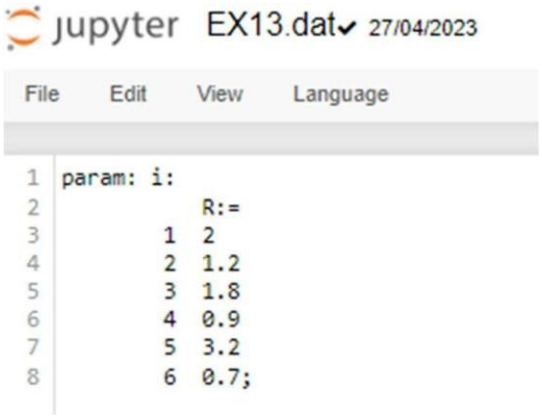
La principal ventaja de utilizar un modelo abstracto en Pyomo, frene a uno concreto, es que permite la independencia de la estructura del modelo de los datos específicos, lo que facilita la reutilización y modificación del modelo para distintos escenarios. Además, permite utilizar técnicas de programación orientada a objetos para definir componentes genéricos y así asignar a continuación, valores específicos a medida que se resuelve el modelo.

```
from pyomo.environ import*
import matplotlib.pyplot as plt
import numpy as np
```

```
model = AbstractModel()
model.i = Set()
model.j = Set(initialize= model.i)
model.R = Param(model.i, within= NonNegativeReals)

def lowband(model, i):
    return (model.R[i], 100)
model.x = Var(model.i, bounds=lowband, within=NonNegativeReals, initialize=0)
model.y = Var(model.i, bounds= lowband, within= NonNegativeReals, initialize=0)
model.W = Var(bounds=(0,100), within=NonNegativeReals, initialize=0)
model.L = Var(bounds=(0,100), within=NonNegativeReals, initialize=0)

def rule_eq1(model, i, j):
    if i>j:
        return (model.x[i]-model.x[j])**2 + (model.y[i]-model.y[j])**2 >= (model.R[i]+model.R[j])**2
    else:
        return Constraint.Skip
model.eq1 = Constraint(model.i,model.j,rule= rule_eq1)

def rule_eq2(model, i):
    return model.x[i] <= model.W - model.R[i] # You cant write two constraints at the same time it will cause error. Minor constr

model.eq2 = Constraint(model.i, rule= rule_eq2)

def rule_eq3(model, i):#For constraint rule we pass the model and the index that indicates the number of constraints
    return model.y[i] <= model.L-model.R[i] #idem eq2

model.eq3 = Constraint(model.i, rule= rule_eq3) #Here we just have to indicate the exact number of constraints

model.obj = Objective(expr= model.L*model.W, sense= minimize) #Objective function rule we just give the model and no indexes, be
```

```
solver = SolverFactory('ipopt')
instance = model.create_instance('EX13.dat') #create_instance reads independently LowerCase and UpperCase
results = solver.solve(instance)
instance.pprint()
```

Figura 13. Ejemplo de modelo abstracto en Pyomo. Fuente: Elaboración propia



	A	B
1	i	R
2	1	2
3	2	1.2
4	3	1.8
5	4	0.9
6	5	3.2
7	6	0.7
8		

Figura 14. Ejemplo de archivo de datos .dat y .csv. Fuente: Elaboración propia

En definitiva, la diferencia entre un modelo concreto y uno abstracto, reside en el nivel de especificidad de los datos utilizados para definir el modelo. Existen, modelos híbridos que realizan una combinación de los conceptos tratados, aunque esta variante no suele ser una buena práctica de programación.

Es preciso mencionar una herramienta conocida como "DataPortal". DataPortal es una herramienta utilizada en Pyomo para la carga y gestión de datos en modelos de optimización. Para el uso de este objeto, es necesario asignarlo a una variable. Típicamente, el nombre utilizado para la variable es "data". Un objeto DataPortal, puede cargar datos de las siguientes fuentes de datos:

- TAB.
- CSV.
- JSON.
- YAML
- XML
- Excel
- DAT

La mayoría de estos formatos de datos permiten la expresión de datos en formato tabular, lo cual será de vital importancia para el desarrollo de este proyecto como se explicará con mayor detenimiento en el Capítulo 4. Para ver un ejemplo de uso del comando DataPortal, consulte el apartado 4.2.2.

Por tanto, en este punto el lector tiene las herramientas suficientes para poder estructurar el programa de Pyomo, para cualquier modelo de optimización, además de ser capaz de discernir en qué caso conviene utilizar un modelo abstracto y en cual uno concreto. Sin embargo, todavía es necesario aprender cómo modelar las distintas componentes de Pyomo, cómo utilizar los solver, visualizar datos y presentar resultados, sea en formato gráfico o bien en un archivo de texto.

De forma sinóptica, todos los elementos de Pyomo se estructuran de la siguiente forma (Figura 14).

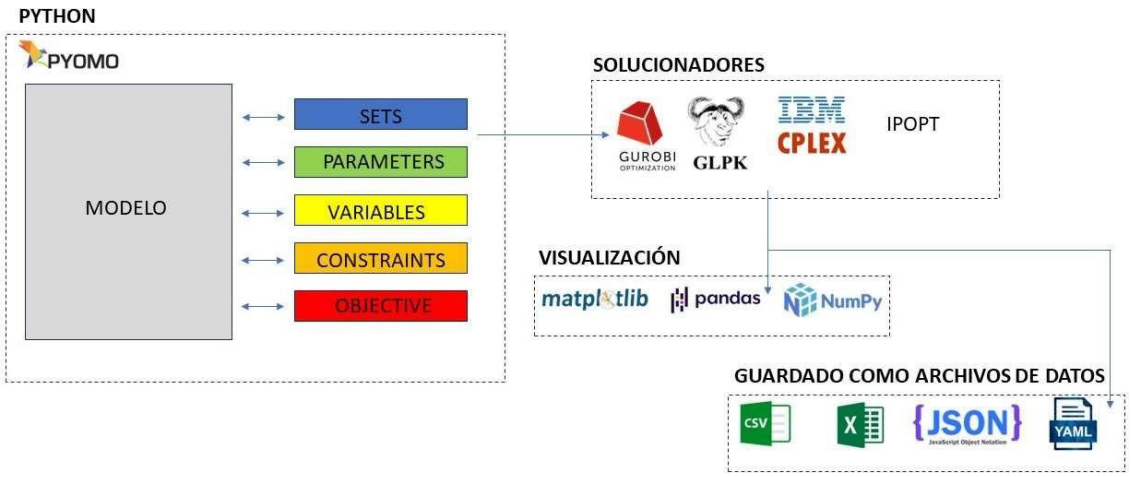


Figura 15. Esquema de Pyomo. Fuente: Elaboración propia

En primer lugar, se procede a realizar una explicación de las distintas clases de Python utilizadas, siguiendo el orden utilizado en los códigos que aparecen explicados a lo largo de este texto y en la figura 15 (aunque este orden es flexible):

1. Sets: Pueden ser declarados utilizando las declaraciones de: Set () o RangeSet(). Algunos argumentos destacados utilizados como *flags*[10], documentación o dimensionado se pueden encontrar en [14]. Posibles ejemplos de código para la creación de sets son los siguientes (Figura 16):

```
model.i = RangeSet(model.N)
model.j = Set(initialize=model.i)
```

Figura 16. Sets en Pyomo. Fuente: Elaboración propia

2. Parámetros: Sus declaraciones y argumentos son muy similares a los sets, como se expone en [14].

```
model.N = Param(mutable=True, initialize=40)
```

Figura 17. Parámetros en Pyomo. Fuente: Elaboración propia

[10] Se entiende como flags a aquellas herramientas utilizadas para la validación de los datos introducidos por el usuario.

De las figuras 16 y 17, se recaba que se ha definido en Pyomo un parámetro de valor 40 (initialize= 40), cuyo valor puede ser modificado (mutable= True). Además, se han establecido dos índices "i" y "j", de valores desde 1 a 40.

3. Variables: Al declararlas suele indicarse un dominio (argumento" within"), unos límites numéricos ("bounds") y además, para la búsqueda iterativa dl valor ópitmo, se suele inicializar en un valor determinado ("initialize").

```
model.R = Var(bounds=(0.001,2), within=NonNegativeReals, initialize=random.uniform(0.001,1))
model.x = Var(model.i, bounds=(0.001,2), within=NonNegativeReals, initialize=random.uniform(0.001,2))
model.y = Var(model.i, bounds=(0.001,2), within=NonNegativeReals, initialize=random.uniform(0.001,2))
```

Figura 18. Variables en Pyomo. Fuente: Elaboración propia

En la figura 18, se puede observar, que la variable R, por ejemplo, está limitada dentro del dominio de los números reales no negativos[11], estando sus valores numéricos entre 0.001 y 2. Como semilla para comenzar el proceso iterativo, se establece un número aleatorio entre 0.001 y 1. (El valor inicial afecta al tiempo y número de iteraciones de un problema de optimización mediante el uso de métodos numéricos).

4. Restricciones: Las condiciones de las restricciones, por convención, se suelen declarar previamente en forma de funciones. Si se tienen las siguientes restricciones:

$$(x_i - x_j)^2 + (y_i - y_j)^2 \geq (R + R)^2 \quad \forall i,j \qquad (3)$$

$$(x_i - 1)^2 + (y_i - 1)^2 \leq (1 - R)^2 \; \forall i \qquad (4)$$

Una posible forma de codificar las desigualdades podría ser la siguiente (Figura 19):

```
def c1_eq_rule(model, i, j):
    if i > j:
        return (model.x[i] - model.x[j])**2 + (model.y[i] -model.y[j])**2 >= (model.r + model.r)**2
    else:
        return Constraint.Skip
model.c1 = Constraint(model.i, model.j, rule= c1_eq_rule)

def c2_eq_rule(model, i):
    return (model.x[i] - 1)**2 + (model.y[i] - 1)**2 <= (1 - model.r)**2

model.c2 = Constraint(model.i, rule= c2_eq_rule)
```

Figura 19. Restricciones en Pyomo. Fuente: Elaboración propia

[11] Consulte el anexo 3 con todos los dominios posibles.

Si se analiza la declaración de funciones en primer lugar, se observa que los parámetros de entrada corresponden a la instancia del modelo y los índices o sets utilizados. Para facilitar la legibilidad del código se les ha denominado de la misma forma a los datos de entrada y en la declaración de sets, aunque esto no sea estrictamente necesario. Por otro lado, las funciones tienen como variables de retorno las restricciones.

Si se analiza la instancia de la restricción, se observa que los argumentos de esta corresponden a la representación del "para todo" matemático y a la restricción a seguir ("rule")[12].

5. Función objetivo: Expresión que devuelve un valor que se desea minimizar o maximizar. Su argumento y construcción es muy similar al de las restricciones. Si se tiene una función objetivo sencilla de la forma:

$$\text{Max } R \quad (5)$$

Esta, se podría codificar de la siguiente manera (Figura 20):

```
model.obj = Objective(expr = model.r, sense= maximize)
```

Figura 20. Función objetivo en Pyomo. Fuente: Elaboración propia

Existe una clase más denominada "Suffix", la cual se utiliza para visualizar las variables duales del problema de optimización. En Pyomo, existe una problemática con las variables duales, que reside en que, en aquellos problemas con variables binarias (MIP), los comandos para desplegar variables duales no funcionarán. Para solucionar este problema, es necesario fijar las variables binarias a su valor óptimo y volver a ejecutar el problema. Por tanto, se transforma el problema de un MIP a un LP para poder ejecutarlo. En el capítulo 4 de este proyecto, se tratará esta circunstancia.

[12] En este caso particular, en la primera condición ha sido necesario aplicar un filtro para evitar un solapamiento de datos.

Tras construir la instancia e inicializar las variables del modelo, se procede a la resolución del modelo. Para ello, se realiza una codificación similar a la siguiente:

```
opt = SolverFactory('ipopt')
instance = model.create_instance()
results = opt.solve(instance)
```

Figura 21. Solvers en Pyomo. Fuente: Elaboración propia

Se observa en la figura 21 tres líneas de código, en las que se declara el solucionador a utilizar (en este caso, la elección del solucionador IPOPT, se debe a que es de uso común para la resolución de modelos no lineales). Por tratarse de un modelo abstracto, tras la carga de datos es necesario crear una instancia. Finalmente, en la última declaración se le pasa a Pyomo dicha instancia para así resolverla y obtener los resultados del problema. La revisión sobre los distintos solucionadores, sus funcionalidades, ventajas y desventajas, se tratarán en el apartado 4.2.3.

Tal y como ya se ha explicado, la fase final de cualquier programa en Pyomo, tras su correcta compilación, corresponde a la visualización y presentación de resultados. En Python, existen numerosas herramientas de representación gráfica, pero a efectos de este estudio se utilizará la librería de Matplotlib, pues es de las herramientas de uso más extendido. Además, para organizar matricialmente los datos, serán de gran utilidad las librerías de Pandasy Numpy. Continuando con el ejemplo utilizado en las últimas figuras, una posible representación de los resultados podría ser la siguiente:

```
fig = plt.figure(figsize=(6,6)) #Opening a frame
theta = np.linspace(0, 2*np.pi, 100)
for i in instance.i:
    Xc = value(instance.x[i]) + value(instance.r)*np.cos(theta) #drawing a circle
    Yc = value(instance.y[i]) + value(instance.r)*np.sin(theta)
    plt.text(value(instance.x[i]), value(instance.y[i]), str(i), fontweight= 'bold')
    plt.plot(Xc,Yc)

Xc = 1 + np.cos(theta)
Yc = 1 + np.sin(theta)
plt.plot(Xc, Yc)#plot big circle
#plt.plot(1 + 1*np.cos(theta), 1 + 0.5*np.sin(theta))
plt.title('N= ' + str(value(instance.N)) + ' Radius= ' + str(round(value(instance.r), 2)))
plt.xlabel('X')
plt.ylabel('Y')
plt.show() #Show means the end of the plotting (this means if you plt anything after it will be in another frame)
```

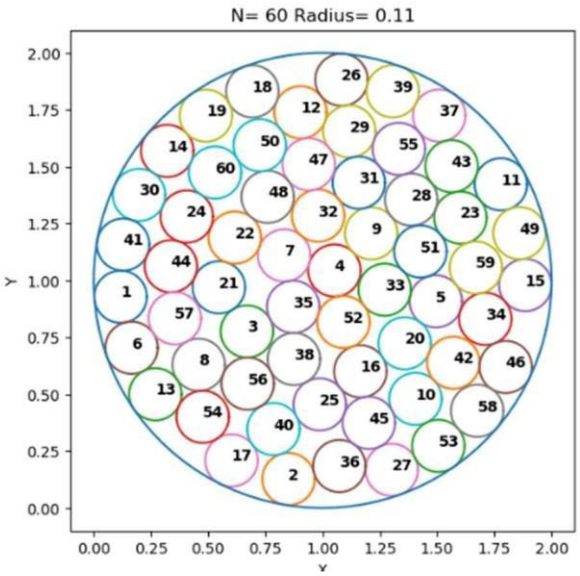


Figura 22. Visualización en Pyomo. Fuente: Elaboración propia

Este fragmento de código utiliza la biblioteca Matplotlib para crear un gráfico. Si se explica cada paso del código:

1. **'fig = plt.figure(figsize= (6,6))'**: Esta línea crea un nuevo objeto o figura con un tamaño de 6x6 pulgadas. La figura sirve como marco general para el gráfico.

2. **'theta = np.linspace(0, 2*np.pi, 100)'**: En este caso, la función 'np.linspace' de la biblioteca NumPu se utiliza para la generación de una matriz de 100 valores equiespaciados entre 0 y 2π. Por tanto, esta matriz es la presentación de los ángulos necesarios para dibujar círculos.

3. **'for i in instance. i'**: Esta línea indica que el bloque de Código identado a continuación se ejecutará para cada valor de 'i' en 'instance.i', siendo 'instance.i' un interable que contiene índices.

4.**'Xc = value(instance.x[i]) + value(instance.r)*np.cos(theta)**

Yc = value(instance.y[i]) + value(instance.r)*np.sin(theta)

plt.text(value(instance.x[i]), value(instance.y[i]), str(ii) fontweight='bold')

plt.plot(Xc, Yc)': Este bloque de Código corresponde a las instrucciones dentro del bucle 'for'. Las dos primeras líneas, calculan las coordenadas X e Y del círculo sumando el valor de la posición del centro de X e y al producto del radio por el coseno y el seno de cada ángulo respectivamente. La siguiente línea agrega texto al gráfico en la posición X e Y, con el valor de 'i' convertido a una cadena de texto. El texto se muestra en negrita. Finalmente, se traza un círculo utilizndo las coordenadas X e Y calculadas anteriormente, conectando los puntos para crear una forma circular.

Después del bucle, el código continúa con la creación de un círculo externo centrado en (1,1).

El último bloque de código corresponde a varias funciones de formato (título, etiqueta del eje X la etiqueta del eje Y para el gráfico, respectivamente. Finalmente el comando 'plt.show()' muestra el gráfico en pantalla.

En definitiva, en este código se generan múltiples círculos centrados en distintas posiciones con un radio determinado, todos ellos inscritos en un círculo centrado en (1,1). El objetivo es maximizar el radio de los círculos internos, para un número N de ellos, pero evitando que estos se corten, tan solo siendo posible la tangencia entre los mismos.[13]

Como cierre a esta sección teórica sobre Pyomo y sus características principales, se añaden unos comentarios finales, que indican algunos comandos útiles para facilitar el desarrollo de la compilación del código, junto con un análisis de los principales errores cometidos en la programación en este lenguaje.

[13] Si desea ver el código completo, consulte el anexo X.

COMANDO	FUNCIÓN
instance.pprint()	Se utiliza para realizar un "display()" de la instancia.
instance.write()	Se utiliza para guardar la instancia en un archivo.
results.write()	Realiza un informe del problema de optimización y de la solución evaluada por el solucionador.
instance.i.at()	Muestra la variable dentro de la instancia para un índice determinado.
data['index']	Su funcionamiento, es similar al de un mapa de Java. En el que se introduce la clave o índice y te devuelve el valor correspondiente.
type()	Determina el tipo de clase a la que pertenece el objeto u variable a analizar.
value()	Permite convertir las variables pertenecientes a clases de Pyomo, en variables primitivas (int, float, etc) interpretables por Python.

Tabla 6. Comandos principales utilizados para la ejecución y revisión del modelo.

La tabla 6 muestra algunos de los principales comandos utilizados para la elaboración de este proyecto, para "debuggear" los principales errores cometidos durante la escritura y ejecución del código.

Para evitar algunos de los principales obstáculos encontrados en este proyecto, se proponen una serie de consejos para las implementaciones de futuros programadores en Pyomo. Para ello, se utilizarán algunos ejemplos usados para la comprensión de Pyomo por parte del autor.

Para facilitar la asimilación de los errores, se seguirá un análisis secuencial, manteniendo el orden utilizado y explicado en ejemplos anteriores y durante las explicaciones teóricas. Por tanto, se comienzan explicando la implementación de las librerías.

En las librerías, existe poco margen de error. El principal error cometido suele ser tipográfico o de escritura, mostrándose en pantalla un error similar al siguiente:

```
In [12]: from pyomo.enviro import*

---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[12], line 1
----> 1 from pyomo.enviro import*

ModuleNotFoundError: No module named 'pyomo.enviro'
```

Figura 23. Errores en la importación de librerías. Fuente: Elaboración propia

Por simple inspección, se observa que el comando "from pyomo.environ import*" está mal escrito, por lo que el programa Kernel, es incapaz de encontrar el módulo seleccionado en Jupyter Notebook. Otro posible error, podría ser no tener cargado alguno de los paquetes necesarios. Por ejemplo, imagínese el lector, que desea instalar la librería asociada con TensorFlow (utilizada en Data Science para la construcción de redes neuronales principalmente). Al no estar instalada en el entorno de Anaconda, aparecerá un error del estilo al mostrado en la figura 23. Para solucionar este problema, será necesario instalar los paquetes necesarios en la consola de Anaconda.

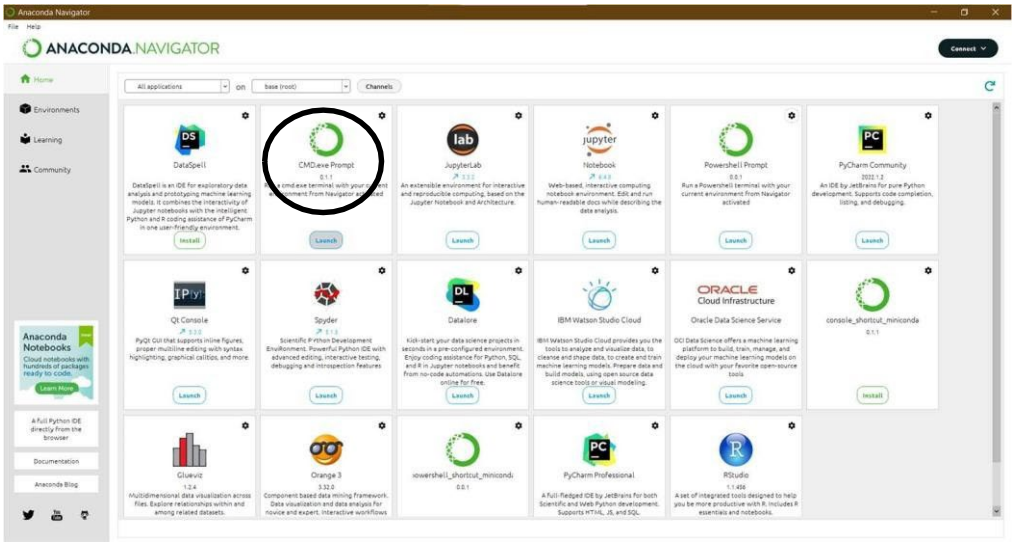


Figura 24. Directorio de Anaconda. Fuente: Elaboración propia

Para ello, es necesario seleccionar la consola rodeada en la figura 24 y ejecutar el siguiente comando:

```
conda install -c conda-forge tensorflow
```

Una vez ejecutado ese comando, en la ventana de comandos surgirá un texto de formato similar al de la figura 25.

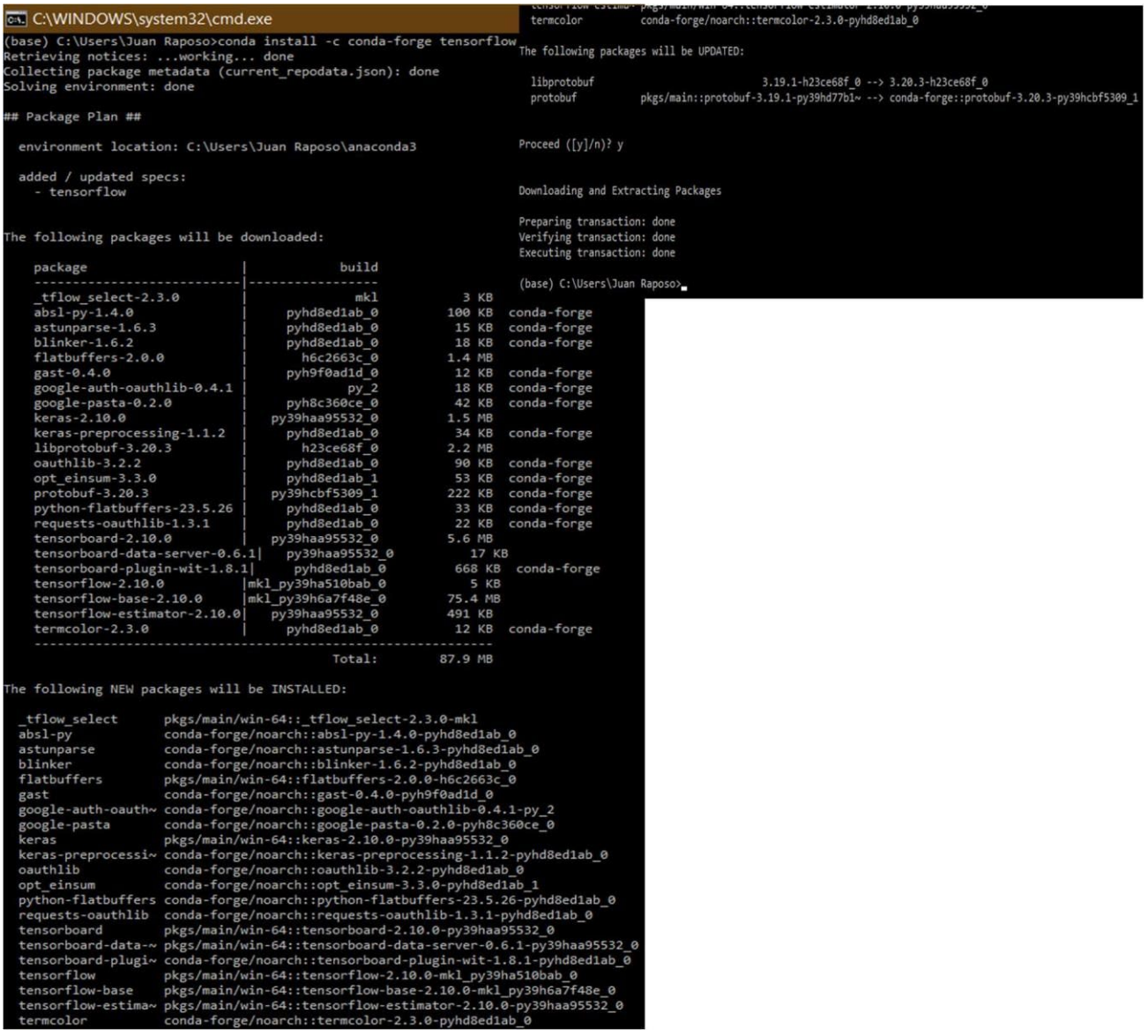


Figura 25. Ventana de comandos de Anaconda. Fuente: Elaboración propia

Tras ejecutar los comandos observados con anterioridad se finaliza la instalación del módulo de TensorFlow y podremos ejecutar el código en Jupyter.

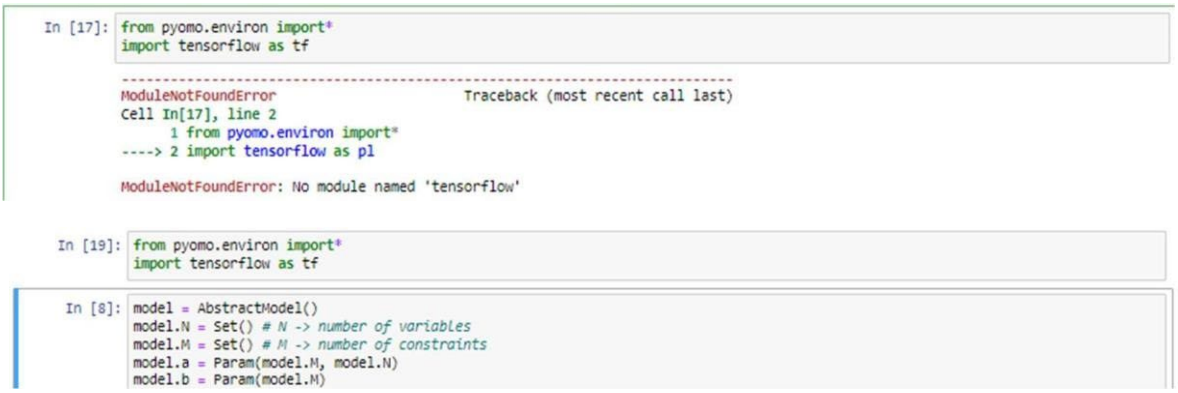


Figura 26. Implementación del módulo antes de instalar el paquete TensorFlow (arriba) y después (abajo). Fuente: Elaboración propia

Otro error, digno de mención al importar librerías hace referencia a un concepto conocido en la literatura inglesa como “override”. En programación, “override” (sobrescribir) es un concepto que se refiere a la capacidad de una clase hija de proporcionar una implementación diferente para un método ya definido en su clase padre. Sin embargo, este concepto es más habitual en programación en Java. En Python, el override se produce, cuando una librería reemplaza o sobrescribe las funcionalidades de otra librería.

Para simplificar la explicación se utilizará un ejemplo sencillo usado para iniciarme en la formulación de problemas en Pyomo. Se tiene el código que aparece en la figura 27.

```
In [32]: from pyomo.environ import*
         from gurobipy import*

In [33]: model = AbstractModel()
         model.N = Set() # N -> number of variables
         model.M = Set() # M -> number of constraints
         model.a = Param(model.M, model.N)
         model.b = Param(model.M)
         model.c = Param(model.N)
         model.x = Var(model.N,within= NonNegativeReals)
         def con_rule(model,m):
             return sum(model.a[m, i] * model.x[i] for i in model.N) >= model.b[m]
         def obj_rule(model):
             return sum(model.c[i]*model.x[i] for i in model.N)
         model.C1 = Constraint(model.M, rule= con_rule)
         model.OF = Objective(rule= obj_rule,sense= minimize)
         model.dual = Suffix(direction=Suffix.IMPORT)

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[33], line 7
      5 model.b = Param(model.M)
      6 model.c = Param(model.N)
----> 7 model.x = Var(model.N,within= NonNegativeReals)
      8 def con_rule(model,m):
      9     return sum(model.a[m, i] * model.x[i] for i in model.N) >= model.b[m]

File src\gurobipy\var.pxi:29, in gurobipy.Var.__init__()

TypeError: __init__() takes exactly 3 positional arguments (2 given)
```

Figura 27. Error del override. Fuente: Elaboración propia

En él, se observa un problema de optimización el cual se resuelve con el solucionador de código abierto GLPK. Sin embargo, como se explicará en el capítulo 4, el solucionador Gurobi es mucho más potente. Por ello, se decide importar la librería "gurobipy", módulo que nos permite resolver con facilidad MIPs y problemas cuadráticos, entre otros. Se invita al lector que trate de comprender con lo explicado hasta ahora, cómo se puede solucionar el problema en forma de TypeError (ver anexo I) observado en la figura 27.

Como habrá deducido, lo que ocurre, indicado en el mensaje de error de la consola, es que la función de inicialización de gurobipy está mal construida, pues le estamos pasando elementos con la sintaxis de construcción del entorno de Pyomo. Por tanto, existen dos posibles soluciones a este problema. La primera, eliminar el comando de gurobipy y trabajar con el módulo de Pyomo y su correspondiente sintaxis. La segunda, adaptar la sintaxis, para que esta pueda ser leída por el módulo de gurobipy. Se invita al lector a valorar esta segunda opción, pues en ciertos casos, puede compensarle. A efectos de este estudio, se elegirá la primera opción.

Por tanto, cerrado el análisis de posibles errores en la importación de módulos, se procede a analizar los errores en la descripción y carga de datos del modelo.

En este caso, los errores variarán, en función de si el modelo es concreto o abstracto, pues el orden y la carga de datos es de distinta forma. Se enumeran algunos de errores más comunes que se encuentran al trabajar con Pyomo:

a) Error de sintaxis: Este error surge al existir errores tipográficos, falta de comillas, paréntesis o bien una estructuración incorrecta del modelo. Es de suma importancia revisar la sintaxis y corregir cualquier error que produzca un fallo en la compilación.
b) Error de tipo de datos: Pyomo es fuertemente "tipado", lo que significa que los tipos dedatos deben ser compatibles en las operaciones matemático-lógicas. Por tanto, es necesario garantizar una consistencia y compatibilidad entre los mismos.
c) Error de referencia una variable o conjunto inexistente: Si se hace referencia a una variable o conjunto que no ha sido definido en el modelo, se producirá un error.
d) Error de índices o dimensiones incorrectas: Pyomo utiliza índices dentro de corchetes, para acceder a variables, parámetros y restricciones. Si se utilizan índices incorrectos o se accede a elementos fuera de los límites. Se producirá un error.
e) Error de restricciones inconsistentes: Si se definen restricciones contradictorias o inconsistentes en el modelo, puede conducir a problemas durante la solución del modelo.
f) Error en la formulación matemática del modelo.

Estos son algunos de los errores más generales cometidos durante la formulación del modelo, tanto concreto, como abstracto. Para agilizar la depuración del código, se recomienda utilizar algunas de las herramientas mencionadas en la tabla 6.

Finalmente, y como cierre a este apartado introductorio de Pyomo, se procede a indicar posibles errores, relativos al solucionador y a la visualización de datos.

En relación con el primer caso, el primer error más repetido, está relacionado con el descrito en la casuística de los módulos (no tenerlo instalado el solucionador en el entorno de trabajo). Otro posible error, tiene que ver con no disponer de la licencia del solucionador. En el caso que nos concierne, como se describirá en el capítulo 4, Gurobi precisa de una licencia válida para su uso, por lo que tiene que ser configurada correctamente. También puede ocurrir algún

de integración entre el solucionador y Pyomo. Esto puede deberse a incompatibilidades en las versiones, por lo que, ante esta situación, es necesario leer con minuciosidad ls especificaciones tanto de Pyomo (disponibles en su web), como de los solucionadores.

Por último, conviene revisar los posibles errores de visualización. Los *displays* integrados en Pyomo (*pprint(), write(),* etc) son extremadamente rudimentarios, por lo que se precisa del uso de bibliotecas externas para una correcta visualización de los resultados. Aunque que sea bastante reiterativo, algunos de los errores comunes que se pueden encontrar son:

i) Errores en la instalación de la biblioteca de visualización.
ii) Errores en la configuración del entorno de visualización.
iii) Errores en el acceso a los resultados del modelo.
iv) Errores en la interpretación o presentación de los resultados.
v) Errores en la generación de gráficos o visualizaciones específicas.

En definitiva, Pyomo es una herramienta muy intuitiva y flexible, puyes integra las funcionalidades de Python en consonancia con las de un AML. La dificultad en su aprendizaje no reside en la dificultad de sus conceptos, sino en la escasa información de calidad disponible para mejorar la calidad de nuestros programas, teniendo que recurrir a foros de internet, para desbloquear los distintos problemas que surgen durante el desarrollo del código, pues al ser de código abierto, este está en constante desarrollo y ciertas funcionalidades se vuelven arcaicas con facilidad. Además, por ser un lenguaje extremadamente tipado, resulta de vital importancia para encontrar los errores en el código, entender a la perfección el funcionamiento de la herramienta y cómo se integra, tanto en el lenguaje de Python como con los distintos solucionadores y /o herramientas de visualización convenientes.

2.4 DESCRIPCIÓN TEÓRICA DE LA INTERFAZ WEB

Retomando lo expuesto en la introducción, implementar el programa de Pyomo en un entorno web, consiste en desarrollar una interfaz de usuario en la cual se puede acceder mediante el uso de navegadores (Google Chrome, Mozilla, Internet Explorer, Opera, etc), se aloja el código pertinente, y se permite la interacción con los datos disponibles, además de la correspondiente visualización de los resultados. Todo su funcionamiento, además, debe de estar integrado dentro de un servidor determinado.

Para crear una interfaz web, existen numerosas tecnologías y enfoques que se pueden utilizar. Se esboza una descripción general de posibles pasos a seguir:

1. Diseño de la interfaz: previo al desarrollo del código, es útil tener un diseño visual de tu interfaz web. Entre las herramientas disponibles, se encuentran, las de diseño gráfico o bocetos de papel, los cuales pueden resultar de utilidad para la definición de la estructura, el diseño y la apariencia de la interfaz.
2. Selección de una tecnología de desarrollo web: existen numerosas tecnologías y lenguajes de programación para desarrollar interfaces web. Entre las principales opciones se encuentran HTML, CSS y JavaScript para la parte del cliente, y frameworks de desarrollo web como Flask o Django para la parte del servidor. Se utilizará la que mejor se adapte a las necesidades y conocimientos del autor.
3. Desarrollo de la interfaz de usuario: se utilizará HTML para estructurar el contenido de la interfaz web y CSS para estilizarlo. Se recomienda el uso de JavaScript para la agregación de interactividad y funcionalidades dinámicas a la interfaz. Existen frameworks de JavaScript como React, Angular o Vue.js que pueden facilitar el desarrollo de la interfaz de usuario.
4. Desarrollo del *backend*: se utilizará como lenguaje de programación Python o Java, para la creación de la lógica del servidor y así poder manejar las solicitudes y respuestas entre el cliente y el servidor. Se recomienda de nuevo el uso de un framework como Flask, Django, Express,js u otros para facilitar el desarrollo del backend y la comunicación con la interfaz de usuario.

5. Almacenamiento y gestión de datos: en el caso que nos atañe, se necesita almacenar y gestionar datos, por lo que se pueden utilizar bases de datos como MySQL o MongoDB. Es importante el aprendizaje sobre la interacción con bases de datos utilizando el lenguaje de consulta correspondiente (por ejemplo, SQL para bases de datos relacionales) y elegir una biblioteca o framework que ayude a la conexión y realización de operaciones en la base de datos desde el backend.
6. Implementación de la seguridad: la seguridad es un aspecto fundamental en las aplicaciones web. Entre las medidas de seguridad necesarias, se encuentran, la autenticación de usuarios, protección contra ataques de inyección de código, validación de datos de entrada y protección contra ataques de falsificación de solicitudes entre sitios (CSRF, por sus siglas en inglés).
7. Implementación del despliegue: selección de una plataforma de hosting o un servicio de alojamiento en la nube para el despliegue de la aplicación web. Configuración del servidor web y las bases de datos, según las necesidades personales y asegurarse de que la aplicación esté accesible a los usuarios.
8. Prueba y depuración: realización de pruebas exhaustivas para asegurarse de que la interfaz web funciona correctamente para los diferentes navegadores y dispositivos. Son de extremada conveniencia, la realización de pruebas de usabilidad además de resolver cualquier problema o error encontrado.

En esta sección, se esbozan de forma genérica algunos de los pasos a seguir para la creación de una interfaz web. En función de las especificidades del programa de Pyomo y tipos de datos a estudiar, se realizará un estudio de los conceptos a estudiar, tecnologías a implementar, las cuales se adapten a las necesidades específicas del proyecto. De forma sinóptica, se expone en la figura 28, un esquema del funcionamiento del *frontend* y el *backend*.

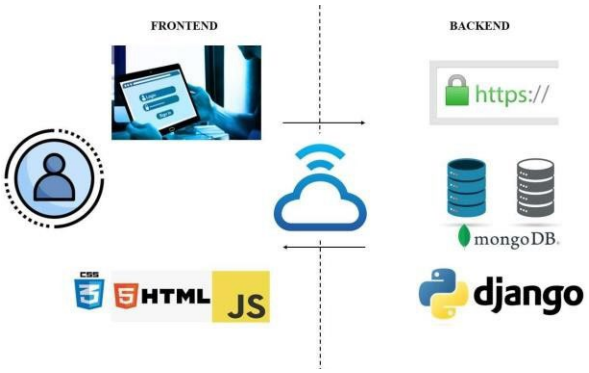


Figura 28. Interacción entre el frontend y el backend en el desarrollo de una web. Fuente: Elaboración propia

Capítulo 3. ESTADO DE LA CUESTIÓN

Existen dos perspectivas a tener en cuenta al revisar el estado del arte. En relación con el modelado de explotación de la generación, numerosos trabajos que han abordado el problema de predecir el comportamiento del sistema eléctrico, desde múltiples perspectivas. En lo relativo a la programación de modelos de explotación en Pyomo, la literatura es escasa, prácticamente inexistente, por lo que también se revisará para facilitar su búsqueda y estudio. Todos los trabajos aquí citados, ofrecen un punto de partida para el proyecto que aquí se quiere realizar.

La labor de investigación, previa a la realización del proyecto, comienza con el estudio del funcionamiento de los sistemas de energía eléctrica. El trabajo de [1], propone un análisis desde el punto de vista de un contexto regulado, de la planificación temporal de la toma de decisiones en modelos de sistemas de potencia eléctrica. Para ello, mediante una serie de simplificaciones y desde un enfoque determinista, propone un horizonte temporal dividido en 3 plazos: corto, medio y plazo, en los que desarrolla unos modelos de optimización lineales en forma de MILP. Este proyecto se fundamenta en el modelo del medio plazo recogido en [1].

Navarro desarrolla en [2] un trabajo centrado en la planificación de la operación a corto y medio plazo en los sistemas eléctricos de potencia. En el contexto de la industria eléctrica, la planificación a corto plazo se entiende como la optimización de la generación y el suministro de energía eléctrica en un horizonte temporal que abarca desde unas horas hasta varios días, mientras que la planificación a medio plazo extiende un horizonte desde unas semanas hasta meses. El autor resalta la importancia de una planificación eficiente para los distintos horizontes temporales, pues permite mayor confianza en la operación del sistema eléctrico, al conseguir que esta sea más económica. Se menciona, una creciente relevancia en la penetración de las energías renovables en el mercado eléctrico, junto con la necesidad de reducir las emisiones de gases de efecto invernadero, los cuales han planteado nuevos desafíos en la planificación de la operación. Por último, he de destacar varios aspectos clave

de la planificación a corto y medio plazo, como la modelización del sistema eléctrico, su correspondiente optimización de generación y despacho de carga, la consideración de las restricciones operativas y la gestión de los riesgos pertinentes. Todo ello, gestionado mediante el uso de un paquete informático conocido como REDPLA, el cual es usado diariamente en la planificación de la operación de la Red Eléctrica Mexicana, que permite determinar el esquema de generación para unidades hidráulicas y térmicas.

En [3] se plantea un modelo, que aborda la aplicación del método de cobertura progresiva (PH, por sus siglas en inglés) para resolver el problema de planificación de la operación a medio plazo en sistemas hidrotérmicos de generación de energía. Dos Santos plantea un problema lineal, estocástico, a gran escala y con incertidumbre, que tiene como objetivo examinar aspectos prácticos relevantes al aplicar el PH en este contexto. Por tanto, en el caso de los sistemas hidrotérmicos, para determinar los costes de operación, se determinan cuáles son los niveles de generación óptimos, teniendo en cuenta las restricciones pertinentes. En el estudio se destacan dos aspectos prácticos clave: el uso de un "inicio en caliente" (warm start) junto con la elección del parámetro de penalización apropiado. Los resultados obtenidos, demuestran que el PH es una herramienta efectiva y competitiva para resolver el problema de la planificación de la operación a medio plazo en sistemas hidrotérmicos.

Desde otro enfoque más generalista y conceptual, en [4] se aborda el contexto tecnológico, económico y regulador de los sistemas de energía eléctrica. Este estudio, en clave universitaria, permite una comprensión más profunda y general, del funcionamiento de los sistemas de energía en los países hispanoamericanos. Son de especial utilidad, los apéndices relativos a las funciones de planificación y operación, el estudio del contexto regulador (en el que se realiza una excelente explicación acerca de la clasificación de las actividades eléctricas: generación, red, transacción y coordinación). Sin embargo, este estudio, carece de modelizaciones por lo que, a efectos prácticos, no nos permite obtener una visión completa de la situación del estado del arte.

Merecen especial mención como revisión de la cuestión [5] y [6]. En el primero de ellos, se describen numerosos ejemplos de planificación de la operación en sistemas de energía

eléctrica a medio plazo, utilizando técnicas de Optimización Robusta Adaptativa (ARO por sus siglas en inglés). De forma sinóptica, ARO modela situaciones en las cuales los operadores diferencian entre dos tipos de decisiones: aquellas que se realizan de forma inmediata, y aquellas que pueden ser tomadas en algún punto en el futuro. El segundo de ellos aborda el problema de la programación a medio plazo para sistemas eléctricos de potencia, dentro de un contexto de mercado mayorista. Aunque el contexto del proyecto está enmarcado en un contexto de regulación, estudiar otras perspectivas, permiten obtener una visión más global y transversal del mercado eléctrico. Existen otros numerosos estudios acerca de la gestión de la producción eléctrica a medio plazo, pero a efectos de este proyecto, se considera suficiente esta revisión para obtener una visión global acerca de la toma de decisiones en sistemas de potencia eléctrica.

Otra revisión necesaria, tiene que ver con Pyomo y la incorporación en dicho lenguaje de programación de modelos de tomas de decisiones de sistemas eléctricos de potencia. El estado de la cuestión en este caso se caracteriza por su escasez y falta de proyectos. Por tanto, conviene también revisar la bibliografía exclusiva de Pyomo para facilitar su implementación. Comenzaré realizando un análisis de la bibliografía más general. Como ya se ha mencionado con anterioridad, [14], [19], [20] y [21] son indispensables para realizar una primera toma de contacto en el entorno de Pyomo. En primer lugar, [14] representa la documentación oficial acreditada y validada por los desarrolladores de Pyomo (Sandia LLC). En esta se describen de forma muy esquemática los conceptos fundamentales, posibles errores, técnicas de programación, junto con algún ejemplo y opciones para desarrolladores (recuerde que Pyomo es un lenguaje de código abierto). [19] y [20] suponen un análisis más exhaustivo de la herramienta. Además de la explicación de las componentes de Pyomo (Set, Variable, Suffix, etc), los libros realizan un análisis de los distintos comandos para el análisis de datos, explicaciones de extensiones y características avanzadas, además de un apéndice con una breve introducción para aquellos que se inician por primera vez en el lenguaje de Python. Como complemento a [19] y [20], se recomienda la lectura de la documentación recogida en [21].

Por otro lado, para un análisis más alineado con los objetivos de este proyecto, se recomienda la consulta de [15], [16], [17], [18], [26] y [27] respectivamente. El bloque correspondiente a la primera decena incluye algunos repositorios disponibles en GitHub, que pueden ser de sumo interés para desarrolladores en Pyomo (entre ellos el [18], de elaboración propia, y en el que se puede consultar el código de este proyecto, pero sería de mayor importancia revisar [16] y [17] debido a la complejidad de los conceptos desarrollados en algunos de ellos). En cambio, [27] ofrece unas excelentes explicaciones junto con ejemplos de *unit commitment*, de la mano del profesor de la Universidad Pontifica de Comillas D. Andrés Ramos. Si se quiere aprender de forma más interactiva se recomienda la consulta de algunos de los cursos disponibles en la plataforma Udemy [26]: desde cursos de iniciación en Pyomo, hasta algunos en el que existen modelos de despacho económico.

En relación con la implementación del código en una interfaz web, existen múltiples perspectivas y/o enfoques que debido a su extensión, queda sujeto a futuras investigaciones.

Capítulo 4. DISEÑO E IMPLEMENTACIÓN DEL MODELO DE PLANIFICACIÓN

A la vista del capítulo anterior, se concluye que existen 3 bloques fundamentales en el estado del arte que determinan el porqué de este proyecto:

1) Modelización de la generación de medio plazo, alineado con los ODS: como se ha descrito en el capítulo 3, se modela el equipo de generación teniendo en cuenta la generación renovable y el mercado de emisiones de CO2.
2) Implementación del modelo en Pyomo: si se revisa minuciosamente la literatura mencionada en la revisión del estado de la cuestión (capítulo 3), se deduce que, a raíz de lo observado, es necesario la implementación de un programa para la modelización a medio plazo en Pyomo.
3) Creación de una interfaz web: la creación de un entorno web, que permita al usuario de internet acceder libremente a los datos y modificarlos libremente y lanzar ejecuciones de manera cómoda.

Estas tres cuestiones construyen el fundamento o motivación extrínseco de este proyecto, por lo que se procede al desglose de este, manteniendo la estructura descrita en los anteriores párrafos.

4.1 DISEÑO DEL MODELO

El diseño del modelo de modelización para la optimización de la generación de energía eléctrica en el medio plazo, en un entorno regulado, es decir, en el que se supone la existencia de un único operador del sistema, sigue la estructura descrita a continuación.

4.1.1 HIPÓTESIS

En esta sección se describen las hipótesis que se han asumido en el modelado de este proyecto. El objetivo de un modelo es expresar mediante la formulación matemática el funcionamiento de un proceso que ocurre en la realidad circundante. Por su parte, las hipótesis de modelado tratan de simplificar una realidad física, debido a su enorme complejidad, hasta el punto en el que se pueda seguir obteniendo información fidedigna, a pesar de la reducción de complejidad. Por tanto, las hipótesis permiten un balance entre la complejidad de la realidad, y la utilidad de la información obtenida de la misma. No se debe confundir las hipótesis de modelado con las hipótesis de ejecución pues estas últimas se diseñan en base normalmente a motivos computacionales.

Las hipótesis del modelado se pueden clasificar en los siguientes bloques:

Hipótesis intrínsecas de la modelización:

1) El enfoque utilizado será puramente determinista, se despreciará todo elemento estocástico (aunque podría tenerse en cuenta). Esto se debe a que tener en cuenta el factor de incertidumbre, aumenta considerablemente el nivel de complejidad.
2) El espacio temporal objeto de estudio, será atendiendo al criterio de nivel de carga, siguiendo una representación conocida como curva monótona, en detrimento de la linealidad cronológica tradicional. El tiempo se dividirá en 5 niveles de carga ("superpico", pico, llanura, valle y "supervalle"), 2 subperíodos (días laborales y festivos) y 12 períodos (12 meses de un año natural).

Hipótesis asociadas a la generación en el sistema eléctrico:

1) Las centrales termoeléctricas tan solo pueden encenderse o apagarse durante los cambios entre subperíodos, debido a la pérdida de cronología.
2) Por simplicidad, para las centrales hidroeléctricas, se utilizará un modelo agregado, en sustitución de las cuencas de captación previas.

Hipótesis en el modelado del transporte:

1) La red de transporte no se incluirá en el modelo. A esta técnica se le conoce como "nodo único".

En los siguientes apartados del capítulo se presentará la formulación del modelo: variables de decisión, función objetivo, restricciones, etc. Para facilitar su comprensión, se presentará la nomenclatura empleada en primera instancia.

4.1.2 NOMENCLATURA

En esta sección se recoge la nomenclatura empleada en la formulación del modelo. En primer lugar, se describen los índices, seguido de los parámetros y las variables de decisión.

Índices	Significado
t	Generador termoeléctrico t (GEN_001, …, GEN_008)
h	Generador hidroeléctrico h (HYDRO_RES, HYDRO_ROR, HYDRO_PUM)
n	Nivel de carga n (n1, n2, n3, n4, n5)
s	Subperíodo s (Lab, Fes)
p	Período p (1,2, …, 12)

Tabla 7. Subíndices.

Parámetros	Significado
$CO2rate_t$	Tasa de emisión de C02 para el generador térmico t [ton CO2 / MWh]
α_t	Término incremental del consumo de combustible para el generador t. [MTh / GWh]
β_t	Término fijo del consumo de combustible del generador térmico t. [MTh / h]
γ_t	Consumo de combustible para el encendido del generador térmico t. [MTh]
f_t	Precio del combustible consumido por el generador térmico t. [k€ / MTh]
o_t	Coste marginal de la operación y de mantenimiento para cada generador térmico t. [k€ / GWh]

Parámetros	**Significado**
$qmax_t$, $qmax_h$	Máxima potencia bruta para el generador térmico t e hidráulico h. [GW]
$qmin_t$, $qmin_h$	Mínima potencia bruta para el generador térmico t e hidráulico h. [GW]
$bmax_h$	Máxima potencia bruta de bombeo del generador hidráulico h. [GW]
$wmax_{h,p}$	Máxima reserva energética del generador hidráulico h en período p. [GWh]
$w0_{h,p}$	Reserva inicial del generador hidráulico h en período p. [GWh]
$wmin_{h,p}$	Nivel de reserva mínimo del generador hidráulico h. [GWh]
k_t,k_h	Factor de conversión de potencia bruta a neta para t y h. [p.u]
$rend_h$	Rendimiento del ciclo de bombeo-almacenamiento del generador h. [p.u]
$i_{h,p}$	Entradas recibidas por el generador hidráulico h en el período p. [GWh]
$d_{n,s,p}$	Demanda del nivel n en el subperíodo s del período p. [GW]
$a_{n,s,p}$	Duración del nivel n en el subperíodo s del período p. [h]
$qred_{t,n,s,p}$	Potencia mínima debido a las restricciones de red del generador térmico t en el nivel n del subperíodo s del período p. [GW]

Tabla 8. Parámetros.

Variables	Significado
$u_{t,s,p}$	Decisión de conectar el generador térmico t en el subperíodo s del período p. [0 apagado, 1 conectado]
$y_{t,s,p}$	Decisión de puesta en marcha del generador térmico t en el subperíodo s del período p. [0 no arrancar, 1 arrancar]
$z_{t,s,p}$	Decisión de puesta apagar el generador térmico t en el subperíodo s del período p. [0 no apagar, 1 apagar]

Tabla 9. Variables binarias.

Variables	Significado
$q_{t,n,s,p}$, $q_{h,n,s,p}$	Potencia neta suministrada por el generador t/h en el nivel n en el subperíodo s del período p. [GW]
$b_{h,n,s,p}$	Consumo de la bomba en la cuenca hidrográfica h en el nivel n en el subperíodo s del período p. [GW]
$w_{h,p}$	Energía almacenada en el depósito h al final del período p. [GWh]

Tabla 10. Variables reales positivas.

4.1.3 FUNCIÓN OBJETIVO

El modelo de optimización de costes propuesto minimiza el sumatorio de los costes para todos los generadores térmicos *t*, períodos *p* y subperíodos *s*, como si el sistema estuviese controlado por un único operador (contexto regulado). Por tanto, el objetivo del modelo a medio plazo desarrollado en esta sección es estimar la producción requerida por cada generador, para así satisfacer la demanda a mínimo coste, siempre y cuando se garanticen las restricciones del sistema (técnicas, medioambientales y legales).

Para evitar técnicas de descomposición matemática y problemas de escalabilidad en el problema, no se tendrán en cuenta ninguna componente estocástica.

Los costes de la fusión objetivo se dividen en dos bloques diferenciados:

Costes asociados al consumo de combustible:

Los costes por minimizar son los variables y fijos asociados al consumo de combustible.

En primer lugar, se encuentran los costes asociados al combustible consumido para el encendido de una central térmica t (γ_t), necesario para que la caldera de una central térmica esté en condiciones para la realización del ciclo Rankine (básico, con recalentamiento, etc.) u otro durante el arranque de la misma ($y_{t,s,p} = 1$).

Por otro lado, se encuentra el sumatorio de todos los niveles de carga *n*, en el cual para la duración de producción para cada nivel de carga ($a_{n,s,p}$), se establece, la relación entre la energía bruta producida ($q_{t,n,s,p}$, $q_{h,n,s,p}$, siendo esta la potencia neta, dividida entre el correspondiente coeficiente de conversión bruto-neto k_t o k_h), y el coeficiente incremental (α_t), todo ello sumado al producto entre el consumo fijo de combustible (β_t) y la variable binaria encargada de conectar el generador termal t ($u_{t,s,p}$). La conceptualización del consumo fijo e incremental aparece reflejada en la siguiente gráfica (figura 29).

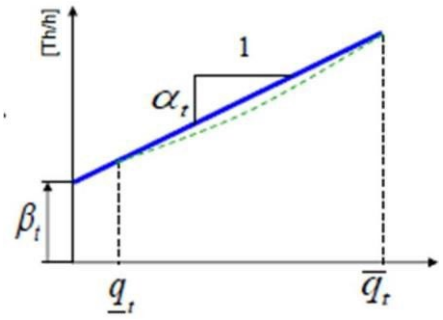


Figura 29. Curva linealizada de entrada-salida, que relaciona la producción bruta de energía con el consumo de combustible en termias por hora. Fuente: [1]

Costes asociados a la operación y al mantenimiento:

Los costes de operación y mantenimiento (O&M, o_t en la literatura), dependen linealmente de la producción bruta de energía y de la duración del funcionamiento para los distintos niveles de carga n de los generadores térmicos t.

Es destacable mencionar, que se ha despreciado el coste de parada de los generadores térmicos *t*. La cantidad de tiempo requerido para iniciar una central termoeléctrica desde un estado en frío puede variar considerablemente debido a una serie de factores, como el tipo de combustible utilizado y el tamaño de la central. Esta variación puede abarcar desde varias horas hasta incluso varios días. Sin embargo, esta situación resulta poco favorable para hacer frente a las fluctuaciones en las curvas de demanda de energía eléctrica. Además, también afecta negativamente el rendimiento óptimo de la central, el cual se alcanza en condiciones de operación continua y carga completa. Es importante destacar que las centrales térmicas convencionales de ciclo combinado, que utilizan carbón como combustible, generalmente presentan un rendimiento mínimo en un rango de 30% al 40%, aunque este valor puede variar debido al tipo específico de combustible empleado.

Finalmente, se expone la función objetivo:

$$Min \sum_{t} \sum_{p} \sum_{s} \{f_t [\gamma_t y_{tsp} + \sum_{n} a_{nsp} [\alpha_t \frac{q_{tnsp}}{k_t} + \beta_t u_{tsp}]] + \sum_{n} a_{nsp} o_t \frac{q_{tnsp}}{k_t}\} \quad (5)$$

4.1.4 RESTRICCIONES

Las restricciones del sistema eléctrico se clasifican en técnicas, legales y medioambientales. Estas, se detallan en [1], pero a continuación se realizará una revisión resumida de las ecuaciones, atendiendo una estructura deductiva (ecuación-explicación).

$$\sum_t q_{t,n,s,p} + \sum_h \left(q_{h,n,s,p} - b_{h,n,s,p}\right) + \sum_{eo} q_{eo,n,s,p} + \sum_{pv} q_{pv,n,s,p} = d_{n,s,p} \quad \forall n \forall s \forall p \ (6)$$

La ecuación (6) describe que la producción neta de energía eléctrica para todo nivel de carga, en el subperíodo s y el período p, se obtiene como la producción de energía térmica, hidroeléctrica menos la potencia de bombeo consumida, eólica, solar y debe coincidir e igualar a la demanda.

Por su importancia cardinal en los sistemas de energía eléctricos, se estudiará su valor dual, pues determina el coste marginal de todo el sistema, ocasionado por aumentar una unidad de la demanda para ese nivel de carga. Por tanto, está información es vital, para observar la modificación de la función objetivo y se analizará en el capítulo 5.

$$w_{\mathrm{h,p}} + \sum_{\mathrm{n,s}} a_{\mathrm{n,s,p}} \cdot [q_{\mathrm{h,n,s,p}} - \eta_{\mathrm{h}} \cdot b_{\mathrm{h,n,s,p}}] \leq w_{\mathrm{h,p-1}} + i_{\mathrm{h,p}} (13)$$

La ecuación (13) representa la ecuación del balance del agua, en términos de balance de energía, como se indica en la figura 30.

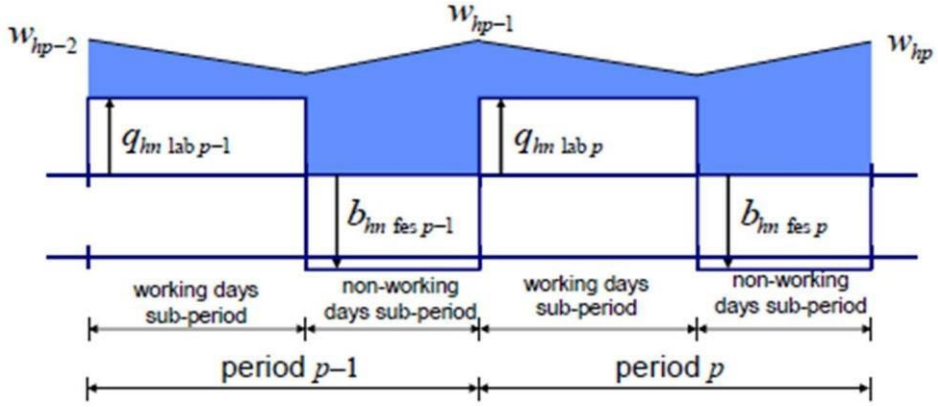


Figura 30. Balance de energía en cada reserva equivalente. Fuente: [1]

Es preciso mencionar que las cuencas hidrográficas, para simplificar el modelo, se representan en términos de balance energético, mediante el uso de un modelo agregado como el de la figura 31.

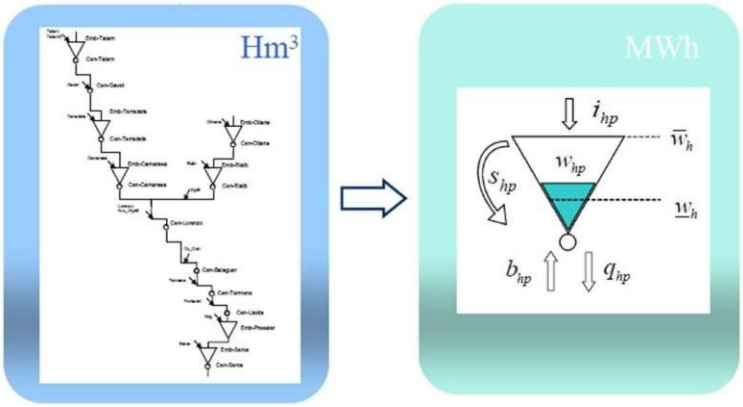


Figura 31. Modelo agregado para una cuenca hidrográfica. Fuente: [1]

Por último, si se reformula la ecuación (13) de la siguiente forma:

$$w_{\mathrm{h,p}} - w_{\mathrm{h,p-1}} + \sum_{\mathrm{n,s}} a_{\mathrm{n,s,p}} \cdot [q_{\mathrm{h,n,s,p}} - \eta_{\mathrm{h}} \cdot b_{\mathrm{h,n,s,p}}] \leq i_{\mathrm{h,p}} \quad : \mu^{\mathrm{w}}_{\mathrm{hp}} \quad (20)$$

Si se estudia la variable dual, se puede determinar el concepto conocido como "valor del agua". La variable $\mu^{\mathrm{w}}_{\mathrm{hp}}$ indica cuanto varía la función objetivo si una unidad adicional de agua entrante estuviese disponible.

Se podría estudiar, todas las variables duales asociadas a las distintas restricciones, pero a efectos de este estudio, tan solo estas dos son de relevancia.

Finalmente, el resto de las restricciones, están asociadas, a las limitaciones teóricas de las variables (límites no negativos, ecuaciones (11) y (12)), a los límites de capacidad (térmicos, hidráulicos y de bombeo, ecuaciones (7), (8), (9) y (10)), de almacenamiento (ecuaciones (14) y (15) y a las restricciones asociadas al mínimo de horas de operación y mínimo de producción para satisfacer la demanda de la red (ecuaciones (16) y (17) respectivamente). Por último, se indican las ecuaciones asociadas a las transiciones de los intraperíodos (ecuaciones (18) y (19)).

Finalmente, se han añadido las tecnologías renovables y, en consecuencia, sus cotas de producción máximas y mínimas (ecuación (20) asociada a las eólicas y (21) asociada a las fotovoltaicas). Además, se ha modificado la función objetivo para añadir el cálculo económico asociado a las emisiones de CO2 y se ha añadido una restricción asociada a dichas emisiones (22).

De forma compacta, se expone el planteamiento del problema:

$$Min \sum_t \sum_p \sum_s \left\{ f_t \left[\gamma_t y_{tsp} + \sum_n a_{nsp} \left[\alpha_t \frac{q_{tnsp}}{k_t} + \beta_t u_{tsp} \right] \right] + \sum_n a_{nsp} o_t \frac{q_{tnsp}}{k_t} \right\} + \sum_t \sum_n \sum_s \sum_p CO2out_{tnsp}\ CO2euros_p\ 0.001 \quad (5)$$

s.t.

$$\sum_t q_{t,n,s,p} + \sum_h (q_{h,n,s,p} - b_{h,n,s,p}) + \sum_{eo} q_{eo,n,s,p} + \sum_{pv} q_{pv,n,s,p} = d_{n,s,p} \quad : \mu^d_{n,s,p} \quad \forall n \forall s \forall p \ (6)$$

$$q_{t,n,s,p} \leq u_{t,s,p} \cdot k_t \cdot q_t^{max} \quad : \mu^{q^{max}}_{t,n,s,p} \quad \forall t \forall n \forall s \forall p \ (7)$$

$$q_{h,n,s,p} \leq k_h \cdot q_h^{max} \quad : \mu^{q^{max}}_{t,n,s,p} \quad \forall h \forall n \forall s \forall p \ (8)$$

$$b_{h,n,s,p} \leq k_h \cdot b_h^{max} \quad : \mu^{b^{max}}_{t,n,s,p} \quad \forall h \forall n \forall s \forall p \ (9)$$

$$q_{t,n,s,p} \geq u_{t,s,p} \cdot k_t \cdot q_t^{min} \quad : \mu^{q^{min}}_{t,n,s,p} \quad \forall t \forall n \forall s \forall p \ (10)$$

$$q_{h,n,s,p} \geq 0 \quad : \mu^{q^{min}}_{h,n,s,p} \quad \forall h \forall n \forall s \forall p \ (11)$$

$$b_{h,n,s,p} \geq 0 \quad : \mu^{b^{min}}_{h,n,s,p} \quad \forall h \forall n \forall s \forall p \ (12)$$

$$w_{h,p} + \sum_{n,s} a_{n,s,p} \cdot \left[\frac{q_{h,n,s,p}}{k_h} - \eta_h \cdot b_{h,n,s,p} \right] \leq w_{h,p-1} + i_{h,p} \quad : \mu^{b^{min}}_{h,n,s,p} \quad \forall h \forall n \forall s \forall p \ (13)$$

$$w_{h,p} \leq w_h^{max} \quad : \mu^{w^{max}}_{h,p} \quad \forall h \forall p \ (14)$$

$$w_{h,p} \geq w_h^{min} \quad : \mu^{w^{min}}_{h,p} \quad \forall h \forall p \ (15)$$

$$\sum_{n,s,p} a_{n,s,p} \cdot q_{t,n,s,p} \geq k_t \cdot q_t^{max} \cdot e \quad : \mu_t^{GdP} \quad \forall t \ (16)$$

$$q_{t,n,s,p} \geq q^{red}_{t,n,s,p} \quad : \mu^{red}_{h,p} \quad \forall t \forall n \forall s \forall p \ (17)$$

$$u_{t\ fes\ p} = u_{t\ lab\ p} + y_{t\ fes\ p} - z_{t\ fes\ p} \quad \forall t \forall p \ (18)$$

$$u_{t\ lab\ p} = u_{t\ fes\ p-1} + y_{t\ lab\ p} - z_{t\ lab\ p} \quad \forall t \forall p \ (19)$$

$$q_{eo,n,s,p} \leq q^{max}_{eo,n,s,p} \quad : \mu^{q^{max}}_{eo,n,s,p} \quad \forall eo \forall n \forall s \forall p \ (20)$$

$$q_{pv,n,s,p} \leq q^{max}_{pv,n,s,p} \quad : \mu^{q^{max}}_{pv,n,s,p} \quad \forall pv \forall n \forall s \forall p \ (21)$$

$$CO2out_{t,n,s,p} = q_{t,n,s,p} \cdot a_{n,s,p} \cdot CO2rate \cdot 1000 \quad \forall t \forall n \forall s \forall p \ (22)$$

$$q_{eo,n,s,p} \geq 0 \quad : \mu^{q^{min}}_{eo,n,s,p} \quad \forall eo \forall n \forall s \forall p \ (23)$$

$$q_{pv,n,s,p} \geq 0 \quad : \mu^{q^{min}}_{pv,n,s,p} \quad \forall pv \forall n \forall s \forall p \ (24)$$

4.2 IMPLEMENTACIÓN EN PYTHON USANDO PYOMO

4.2.1 DATOS DE ENTRADA: CONSTRUCCIÓN DEL ARCHIVO CSV

La construcción de los archivos CSV, presenta una complicación para el caso de la construcción de parámetros (la construcción de sets es más sencilla): no existe bibliografía asociada en el que aparezcan archivos con multi-índices. Por tanto, para abordar este problema, se realizaron dos análisis:

1) Comprensión del funcionamiento de los archivos .dat.
2) Comprensión del funcionamiento de DataPortal

Para la construcción de los .dat, si tenemos una serie de parámetros, dependientes de un solo índice, siguiendo la documentación de Pyomo se establece los índices en la primera columna y los parámetros por filas, de forma similar a los datos ejemplificados en la siguiente figura:

```
param:  i:
                housingspace damage Trainingcostred TrainingcostDark Hitpoint Trainingtime:=
        barbarian 1          34     300            0                205      5
        Archer    1          28     600            0                52       6
        Giant     5          130    4200           0                1850     60
        Goblin    1          52     200            0                101      7
        drag      20         310    22000          0                3900     180
        edrag     30         300    36000          0                4200     360
        baloon    5          236    5500           0                840      30
        pekka     25         310    18000          0                3500     180
        wizard    4          230    4200           0                230      30
        Bowler    6          90     0              140              390      60
        Lava      30         16     0              570              7200     300
        Valkyrie  8          178    0              190              1450     90
        Hog       5          148    0              100              810      45
        Golem     30         40     0              250              5400     300
        Witch     12         180    0              275              480      180
        Minion    2          46     0              7                72       18;
param camp:=286;
param RedElexir:=2000000;
param DarkElexir:=200000;
```

Figura 32. Construcción de parámetros tabulares y unidimensionales.

Fuente: Elaboración propia

En dicha figura, se definen los parámetros en formato tabular y unidimensional.

Por otro lado, si se analiza el funcionamiento de DataPortal, este realiza una lectura de datos similar a los DataFrame de la librería pandas. Por tanto, para la construcción de los índices,

estos serán por columnas, situándose más a la derecha, los índices que se recorren más rápido.

A modo de ejemplo se expone como se crea el parámetro $i_{h,p}$ y del parámetro $qred_{t,n,s,p}$:

1	h	p	i
2	HYDRO_RES	1	400
3	HYDRO_RES	2	320
4	HYDRO_RES	3	340
5	HYDRO_RES	4	260
6	HYDRO_RES	5	200
7	HYDRO_RES	6	150
8	HYDRO_RES	7	100
9	HYDRO_RES	8	100
10	HYDRO_RES	9	260
11	HYDRO_RES	10	280

1	t	n	s	p	qred
2	GEN_001	n1	lab	1	0
3	GEN_001	n1	lab	2	0
4	GEN_001	n1	lab	3	0
5	GEN_001	n1	lab	4	0
6	GEN_001	n1	lab	5	0
7	GEN_001	n1	lab	6	0
8	GEN_001	n1	lab	7	0
9	GEN_001	n1	lab	8	0
10	GEN_001	n1	lab	9	0
11	GEN_001	n1	lab	10	0

Figura 33. Construcción de parámetros tabulares y unidimensionales en CSV. Fuente: Elaboración propia

La construcción de los sets es más sencilla, pues tan solo hay que expresarlo en una columna como se indica a continuación:

1	g
2	GEN_001
3	GEN_002
4	GEN_003
5	GEN_004
6	GEN_005
7	GEN_006
8	GEN_007
9	GEN_008
10	HYDRO_RES
11	HYDRO_ROR
12	HYDRO_PUM

Figura 34. Construcción de un set de generadores en CSV. Fuente: Elaboración propia

4.2.2 CONSTRUCCIÓN DEL MODELO ABSTRACTO

Teniendo en cuenta, los fundamentos teóricos desarrollados en el capítulo 2, se procede al desglose sistemático del problema.

En primer lugar, se importan todas las librerías necesarias: (consultar anexo para ver el código completo). En primer lugar, para evitar cambios en el comportamiento de la división entre números enteros en versiones anteriores de Python se utiliza la siguiente línea de comandos.

from__future__import division

Para poder trabajar con las componentes de Pyomo:

*from pyomo.environ import **

Finalmente, se importantan las librerías utilizadas para el tratamiento de datos tabular, matricial y para su visualización, respectivamente.

Import pandas as pd

```
import numpy as np

import matplotlib.pyplot as plt
```

A continuación, se procede a la declaración del modelo abstracto. En este caso particular, la carga de datos de los parámetros será previa a la definición de sus componentes, debido a los errores que causaban otras configuraciones.

```
param_data = pd.read_csv('Data_duration_demand1.csv', header=0, index_col=[0, 1, 2])

m.a = Param(m.n, m.s, m.p, initialize=param_data['a'].to_dict())

m.d = Param(m.n, m.s, m.p, initialize=param_data['d'].to_dict())
```

Se observa, que para la lectura de parámetros se obviarán los encabezados (fila 0 en Python, 1 en Excel) y se realiza la carga de los parámetros 'a' y 'd' mediante el uso de diccionarios, es decir, establece como clave los índices y como valor los correspondientes valores numéricos.

Obviando esta particularidad, la carga de sets y parámetros es estándar, siguiendo las reglas descritas en capítulos previos.

```
m.G = Set()

m.p = Set()

m.s = Set()

m.n = Set()

m.t = Set()

m.h = Set()

m.hf = Set()
```

```
data = DataPortal()

data.load(filename= 'g.csv', set= m.G)

data.load(filename= 'p.csv', set= m.p)

data.load(filename= 's.csv', set= m.s)

data.load(filename= 'n.csv', set= m.n)

data.load(filename= 't.csv', set= m.t)

data.load(filename= 'h.csv', set= m.h)

data.load(filename= 'hf.csv', set= m.hf)
```

La carga de parámetros dependientes de un solo índice es idéntica.

```
data.load(filename="Data_generators1.csv", param=(m.CO2rate, m.alfa , m.beta ,
m.gamma , m.f , m.o , m.qmax , m.qmin , m.bmax , m.wmax , m.w0 , m.wmin , m.k ,
m.rend))
```

Finalmente, la construcción de las restricciones sigue la notación estándar, tan solo teniendo un pequeño matiz la ecuación (7).

Para el primer período, es necesario obviar la condición, por lo que se usará el controlador de flujo if…else. Por otro lado, como se analiza la energía almacenada en un estado anterior ($w_{h,p-1}$) es necesario convertir la variable p-1 (de clase pyomo) a una de tipo int, por lo que se realiza un concepto conocido como “castear” la variable.

```
def E_Bal_rule(m,h,p):

if p == m.p.first():

return Constraint.Skip
```

else:

*return m.w[h,p] + sum(m.a[n,s,p] * (m.qh[h,n,s,p] - m.rend[h] * m.b[h,n,s,p]) for n in m.n for s in m.s) <= m.w[h,int(p) - 1] + m.i[h,p]*

m.E_Bal = Constraint(m.h, m.p, rule= E_Bal_rule)

Finalmente, se declara la componente asociada a las variables duales y se procede a la fase de solucionar el modelo.

4.2.3 SOLUCIONADOR DEL MODELO: GUROBI

Existen dos grandes grupos de solucionadores en Pyomo: solucionadores sin licencia y solucionadores con licencia.

Los solucionadores sin licencia, son aquellos a los que se puede acceder de forma totalmente gratuita. Entre los principales, destacan:

1) GLPK (GNU Linear Programming Kit): GLPK es un solucionador de Código abierto que se utiliza para resolver problemas de programación lineal y mixta. Proporciona algoritmos de ramificación y corte, así como métodos de resolución simples. GLPK es fácil de usar y se integra bien con Pyomo. Sin embargo, puede tener un rendimiento relativamente más lento en comparación con otros solucionadores comerciales.
2) IPOPT (Interior Point OPTimizer): IPOPT es un solucionador NLP también de código abierto que se enfoca en resolver problemas de programación no lineal continuos. Utiliza el método de puntos interiores para encontrar soluciones óptimas. Este método se basa en iteraciones que se acercan a la solución mediante el movimiento desde el espacio exterior al interior de las restricciones. IPOPT es eficiente para problemas no lineales de tamaño mediano a grande y puede tratar

restricciones lineales como no lineales, aunque se utiliza principalmente para el último caso.

Por otro lado, entre los solucionadores de uso comercial, destacan:

1) CPLEX: CPLEX es un solucionador de optimización comercial desarrollado por la empresa informática IBM. Su funcionamiento consiste en el uso de algoritmos avanzados para resolver modelos de programación lineal, programación entera mixta y programación no lineal, entre otros. Presenta una amplia gama de parámetros configurables lo que permite ofrecer soluciones óptimas o de alta calidad para problemas de optimización, mediante el modelado eficiente por parte del usuario.
2) GUROBI: Gurobi es el solucionador comercial más potente de optimización más potente, muy utilizado en la industria y la academia. Gurobi permite la creación de un modelo propio en Python, sin necesidad de usar el paquete de Pyomo. Sin embargo, presenta muchas menos funcionalidades y facilidad de aprendizaje. Al igual que CPLEX, Gurobi permite ajustar una variedad de parámetros para controlar el comportamiento del solucionador. Estos parámetros incluyen tolerancias de convergencia, estrategias de búsqueda y heurísticas, etc. Los parámetros pueden ajustarse para adaptarse a las características específicas del problema. Gurobi utiliza algorimos de vanguardia para buscar soluciones óptimas o de alta calidad en función de la configuración dada. Los algoritmos incluidos utilizan técnicas de programación lineal, programación entera mixta, programación no lineal y programación cuadrática, entre otras.

   Por tanto, la razón para usar Gurobi en este proyecto, se debe a que este solucionador utiliza técnicas avanzadas de optimización, como pueden ser la relajación lagrangiana, técnicas de cortes y heurísticas inteligentes, con el objetivo de mejorar la eficiencia y calidad de la solución. Además, la interfaz de Gurobi es de fácil uso e integración en lenguajes como Python.

Capítulo 5. EVALUACIÓN Y RESULTADOS

En este penúltimo capítulo, se procede a la descripción de los resultados obtenidos del estudio de un modelo a medio plazo, implementado en Python Pyomo. El capítulo se estructurará en dos partes: una primera, en la que se comprueban posibles fallos y se valida el resultado. Una segunda parte en la que se procede a analizar los resultados de forma más exhaustiva, implementando, además, el uso de tecnologías renovables (eólica y fotovoltaica), pues así el modelo presentará una caracterización más realista

5.1 PRUEBAS Y VALIDACIÓN DEL SISTEMA

En primer lugar, para la comprobación de que el modelo es correcto, es necesario comprobar (de no ser capaz de ejecutar el programa correctamente) la existencia de alguna anomalía en el modelo: en la definición y construcción de los sets, parámetros, ecuaciones de ligadura, función objetivo, etc. Para ello, es preciso que el programador aísle alguno de los elementos del modelo (multiplicando el resto de los elementos por cero) y compruebe el correcto funcionamiento del programa. Una vez realizada esta comprobación, se procede al proceso de validación del programa. Por motivos de evitar sobrecargar este estudio, se invita al lector a realizar estas comprobaciones por cuenta propia.

En segundo lugar, se procede a la realización de la fase de validación. Ella consiste en, utilizando un fichero GAMS facilitado por el director del trabajo, comprobar que los resultados obtenidos en este fichero coinciden con los obtenidos en Pyomo. La tolerancia admitida debe de ser mínima, pues los solucionadores utilizados en ambos programas son idénticos (Solucionador comercial Gurobi). La licencia utilizada por el solucionador es académica y ha sido facilitada por la Universidad Pontificia Comillas.

```
CPU model: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, instruction se
Thread count: 6 physical cores, 12 logical processors, using up to

Optimize a model with 2067 rows, 2115 columns and 7393 nonzeros
Model fingerprint: 0xaf00023e
Coefficient statistics:
  Matrix range     [1e-01, 2e+02]
  Objective range  [1e+01, 1e+04]
  Bounds range     [6e-02, 1e+03]
  RHS range        [2e+00, 2e+03]
Presolve removed 1931 rows and 1052 columns
Presolve time: 0.00s
Presolved: 136 rows, 1063 columns, 1730 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    2.4487043e+05   1.453062e+03   0.000000e+00      0s
     309    4.4825063e+05   0.000000e+00   0.000000e+00      0s

Solved in 309 iterations and 0.02 seconds (0.00 work units)
Optimal objective  4.482506306e+05

User-callback calls 1503, time in user-callback 0.00 sec
Fixed MIP status(2): Model was solved to optimality (subject to tol

MIP   Solution:     448250.630574    (9518 iterations, 328 nodes)
Final Solve:        448250.630574    (309 iterations)

Best possible:      448250.630574
Absolute gap:           -0.000000
Relative gap:            0.000000
```

Figura 35. Resultados MMP en GAMS. Fuente: [1]

En la figura 35, se observa que el problema de *Mixed Integer Programming* (MIP) presenta, tras 309 iteraciones, la solución: 448250.630754 k€, es decir, un coste de aproximadamente medio millón de euros para la planificación a medio plazo.

Si acudimos al programa realizado en Python Pyomo se obtiene el resultado de la figura 36.

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Name: x2293
  Lower bound: 448250.63057426544
  Upper bound: 448250.63057426544
  Number of objectives: 1
  Number of constraints: 3940
  Number of variables: 2284
  Number of binary variables: 567
  Number of integer variables: 567
  Number of continuous variables: 1717
  Number of nonzeros: 9266
  Sense: minimize
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Return code: 0
  Message: Model was solved to optimality (subject to tolerances), and an optimal solution is available.
  Termination condition: optimal
  Termination message: Model was solved to optimality (subject to tolerances), and an optimal solution is available.
  Wall time: 1.7409999370574951
  Error rc: 0
  Time: 1.9913809299468994
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```

```
if (results.solver.status == SolverStatus.ok) and (results.solver.termination_condition == TerminationCondition.optimal):
    print('feasible')
    instance.write('MMP.lp',io_options={'symbolic_solver_labels': True})
elif (results.solver.termination_condition == TerminationCondition.infeasible):
    print('infeasible')
else:
    print ('Solver Status:',  result.solver.status)
```

```
feasible
```

```
print('OBJ=',round(value(instance.obj),2), "keuros")
```

```
OBJ= 448250.63 keuros
```

Figura 36. Resultados MMP en Pyomo. Fuente: Elaboración propia

Y como era esperado, se observa una coincidencia exacta en los resultados obtenidos para cada programa ejecutado.

Por tanto, una vez verificada la validez de los resultados obtenidos en Pyomo, se procede al un estudio más exhaustivo de los mismos en el siguiente subcapítulo, en el que se analizarán de forma gráfica, algunas de las ecuaciones y variables más relevantes del problema, para poder tener una perspectiva global y completa del estudio realizado.

5.2 ANÁLISIS DE LOS RESULTADOS OBTENIDOS

En esta sección, se analizarán, teniendo en cuenta 4 tecnologías (térmica, hidráulica, eólica y solar), además del precio de CO2 por tonelada[14]:

1) La producción neta anual por tecnología.
2) La producción neta anual apilada.
3) La producción mensual apilada desglosada en subperíodos s y niveles de carga n.
4) La producción anual apilada desglosada en subperíodos s y niveles de carga n.
5) Estudio de la variable dual de la ecuación de balance de demanda (6): Costes marginales del sistema.
    a. Costes marginales anuales
    b. Costes marginales mensuales divididos en subperíodos s y niveles de carga n
6) Estudio de la evolución anual de las reservas para las distintas plantas hidráulicas.
7) Estudio de la evolución de la variable dual de la ecuación de balance de energía de la reserva hidráulica (13): El valor marginal del agua.
8) Análisis de la producción de CO2.

En cada uno de estos apartados, se analizarán también las interacciones entre las gráficas, la sensibilidad de las ecuaciones y de la producción ante cambios, etc.

[14] Para realizar un análisis realista del mercado, se tomarán como referencia el precio del CO2 indicado en el portal web de SENDECO2, empresa de compraventa de derechos de emisión.

1) Producción neta anual por tecnología

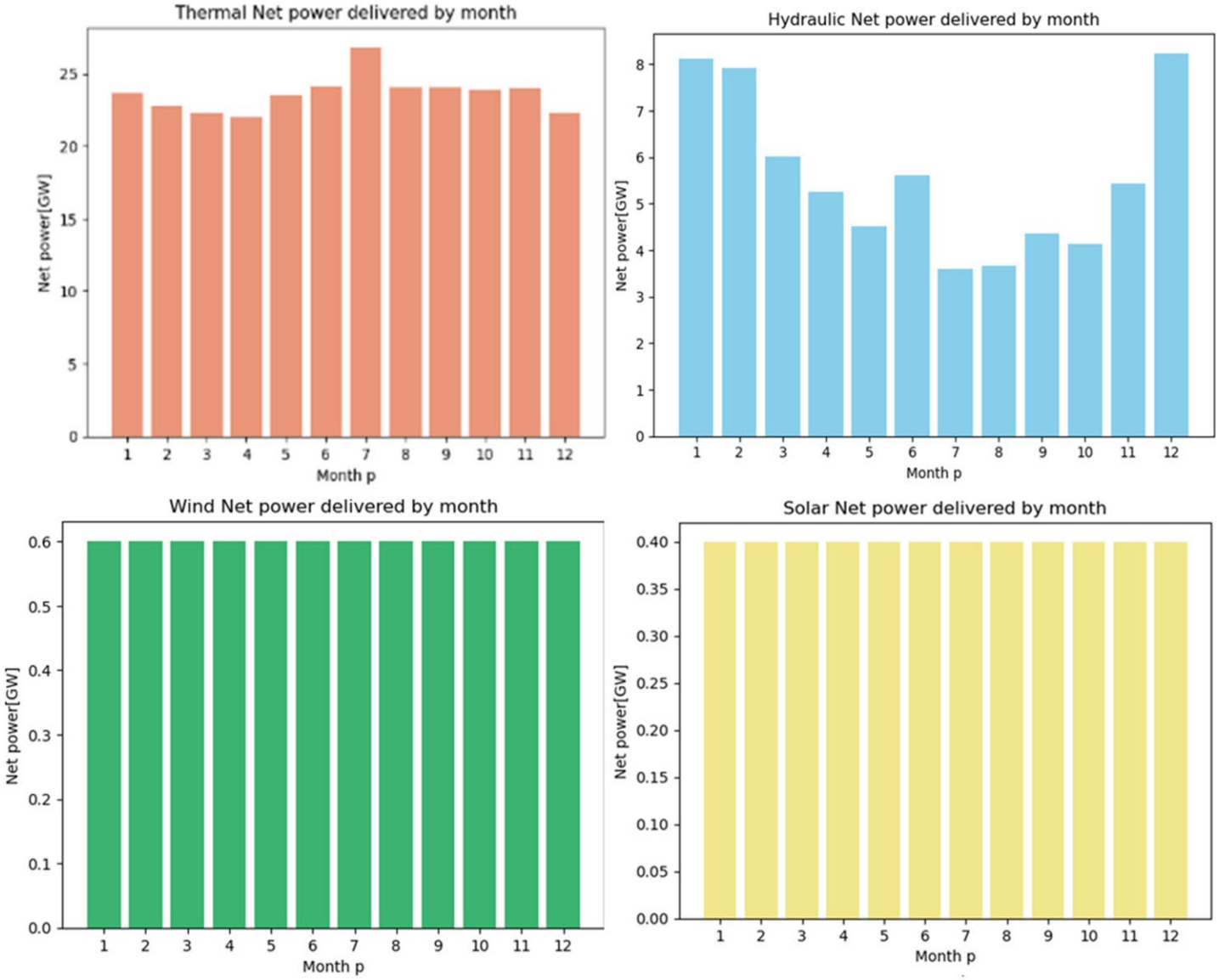


Figura 37. Resultados MMP de la producción anual por tecnologías en Pyomo. Fuente: Elaboración propia

Como se aprecia en la figura 37, la producción anual para cada una de las tecnologías. Esto es fruto de las restricciones descritas en el apartado 4.1.4. Se observa, en primer lugar, que la producción térmica aumenta en los meses de verano mientras que, en el gráfico de barras de producción hidráulica, se observa el efecto contrario, pues la producción es mayor a principios y finales de año, mientras que conforme nos acercamos al ecuador, esta

disminuye. Como se observará en análisis posteriores, estas decisiones están estrechamente vinculadas con el coste marginal del sistema. Además, los meses de invierno (desde diciembre hasta marzo) son los meses en los que se experimentan temperaturas más bajas. Esto implica desde un análisis del consumo particular, mayor demanda de electricidad doméstica para el uso de calefacciones, vida realizada en el hogar, etc. Por tanto, atendiendo a las leyes de oferta y demanda, el precio de la electricidad será más elevado en los meses de invierno. Para minimizar los costes, por tanto, se aumenta la producción de energía hidráulica, en detrimento de la térmica. Finalmente, se encuentran las energías renovables, las cuales abaratan el coste de producción de energía sustancialmente. Debido a unas restricciones laxas (para la producción eólica y fotovoltaica, tan solo se establecen cotas máximas[15] y se añade su producción a la ecuación de balance de energía) en el caso de la energía eólica y solar, estas tomarán sus valores máximos, pues así contribuirán a un abaratamiento de los costes al reducir la producción térmica e hidráulica, es decir, eliminando términos de la función objetivo y por ende, se reduce su coste.

[15] Cabe destacar, que las cotas máximas, dependen del generador (eólico o fotovoltaico), nivel de cargan, subperíodo s y período p. De forma sinóptica, se construye el perfil del viento, de forma lineal, siendo menor las cotas en las primeras horas del día (niveles de carga más bajos) y aumentando hasta llegar a la noche donde alcanza sus cotas más altas (niveles de carga más elevados), En el caso de la energía solar, el perfil es cóncavo: al principio y al final del día, pues la intensidad lumínica es menor, también lo serán las cotas máximas. Conforme nos acercamos al ecuador del día, la intensidad lumínica aumenta y también sus cotas máximas. Por simplicidad, se establecerá un perfil lineal con los niveles de carga n.

2) La producción neta anual apilada.

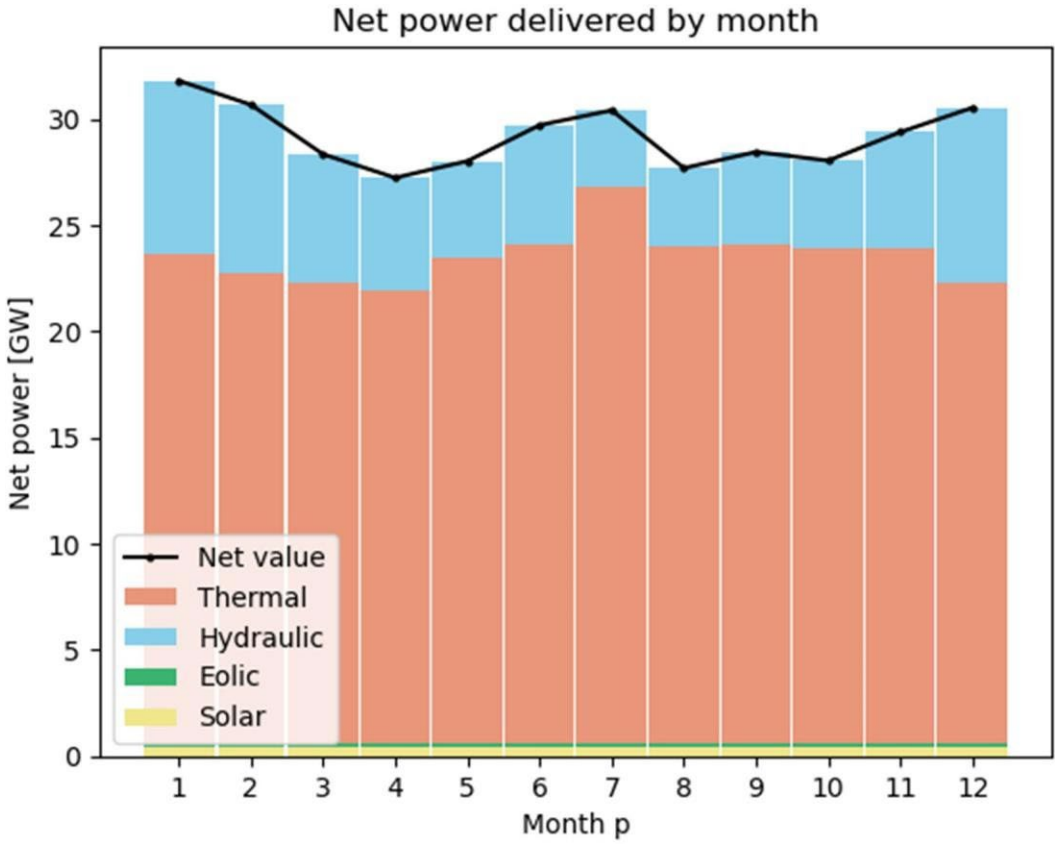


Figura 38. Resultados MMP de la producción anual apilada por tecnologías en Pyomo. Fuente: Elaboración propia

En la figura 38, se observa la producción neta para cada uno de los meses. Como era de esperar, en los meses de invierno y de verano, la producción es mayor que en otoño y primavera, pues la demanda en estos meses aumenta significativamente.

La construcción de este modelo se aleja un poco de la distribución de producción energética actual, pues recientemente, el cupo de producción de energías renovables en España ha superado al de las no renovables. Sin embargo, la sola presencia de las energías renovables manifiesta su importancia pues, el coste de producción al introducirlas en el cupo (sin tener en cuenta el precio de CO2) disminuye significativamente, a pesar de que su relevancia en términos porcentuales es mínima: el coste se ve reducido en 30 millones de euros con una

mínima producción de energías renovables. Esto supone una reducción de aproximadamente un 8% del coste inicial, con una cuota que apenas supera el 2-3 % de la producción total.

Tomando como referencia el perfil establecido inicialmente para las energías renovables y teniendo en cuenta, que las cotas máximas de la eólica doblan a las de la solar para cada nivel de carga, subperíodo s y período p, si cuadruplicamos la cota máxima para la energía eólica en el nivel de carga n1, se reducirá el coste de producción en 16 millones aproximadamente. En definitiva, cuanto mayor presencia tengan las energías renovables en la producción, menor será su coste.

3) *La producción mensual apilada desglosada en subperíodos s y niveles de carga n.*

A continuación, se realiza una visión detallada mensual, de lo descrito en el apartado anterior.

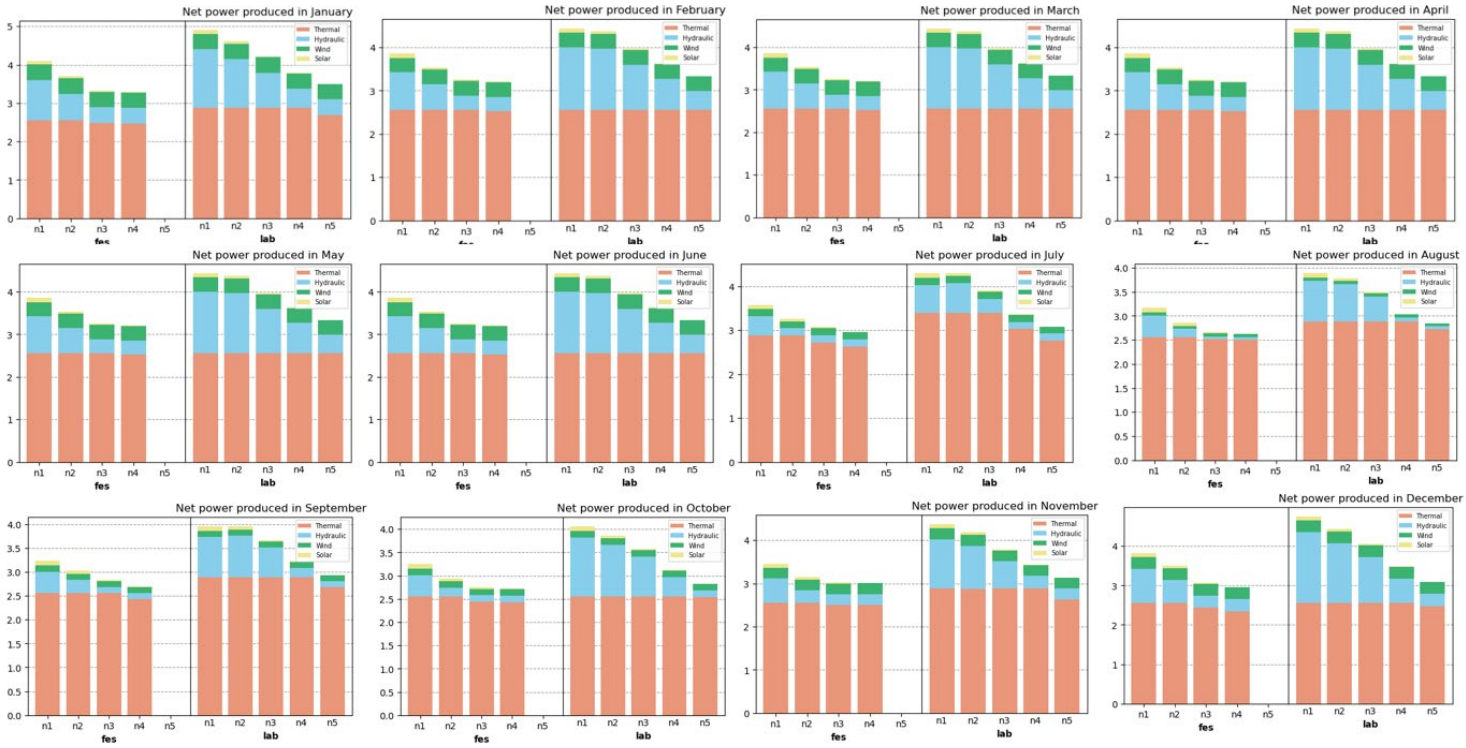


Figura 39. Resultados MMP de la producción mensual apilada y desglosada en Pyomo. Fuente: Elaboración propia

4) *La producción anual apilada desglosada en subperíodos s y niveles de carga n.*

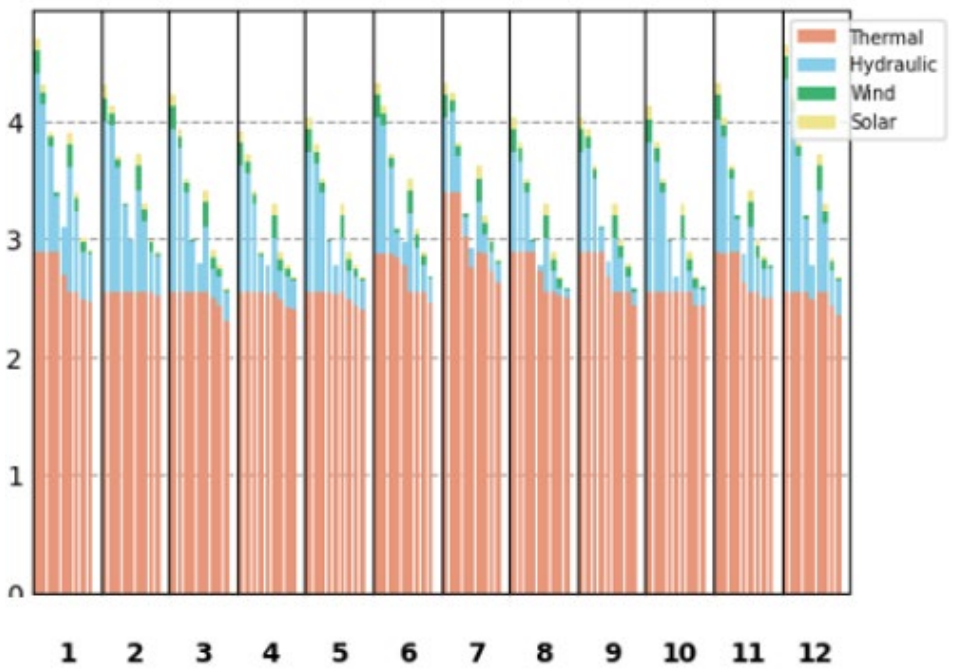


Figura 40. Resultados MMP de la producción anual apilada en Pyomo.

Fuente: Elaboración propia

En la figura 40, se observa la producción del cómputo global de todas las tecnologías. El eje X, se subdivide en 3 niveles: En el nivel superior, se establecen los períodos p o meses. A su vez, para cada período p, se realiza la subdivisión en subperíodos s (días laborales y festivos). Finalmente, cada subperíodo s, se divide en cinco niveles de carga n (super-pico, pico, llano, valle, super-valle). En el eje Y, se describe la producción energética total medida en GW.

En el caso de la producción total, se observa un perfil de producción mayor para el caso de los días laborales, en comparación con los festivos. La explicación, a pesar de que pueda parecer trivial, obedece al funcionamiento de las industrias, mayor volumen de transporte, actividad laboral, en los días de semana.

Cabe destacar, que para no desestabilizar el perfil construido por la producción térmica e hidráulica inicialmente, las tecnologías renovables, tendrán menor relevancia en este estudio.

5) *Estudio de la variable dual de la ecuación de balance de demanda (6): Costes marginales del sistema.*

   a. *Costes marginales anuales*

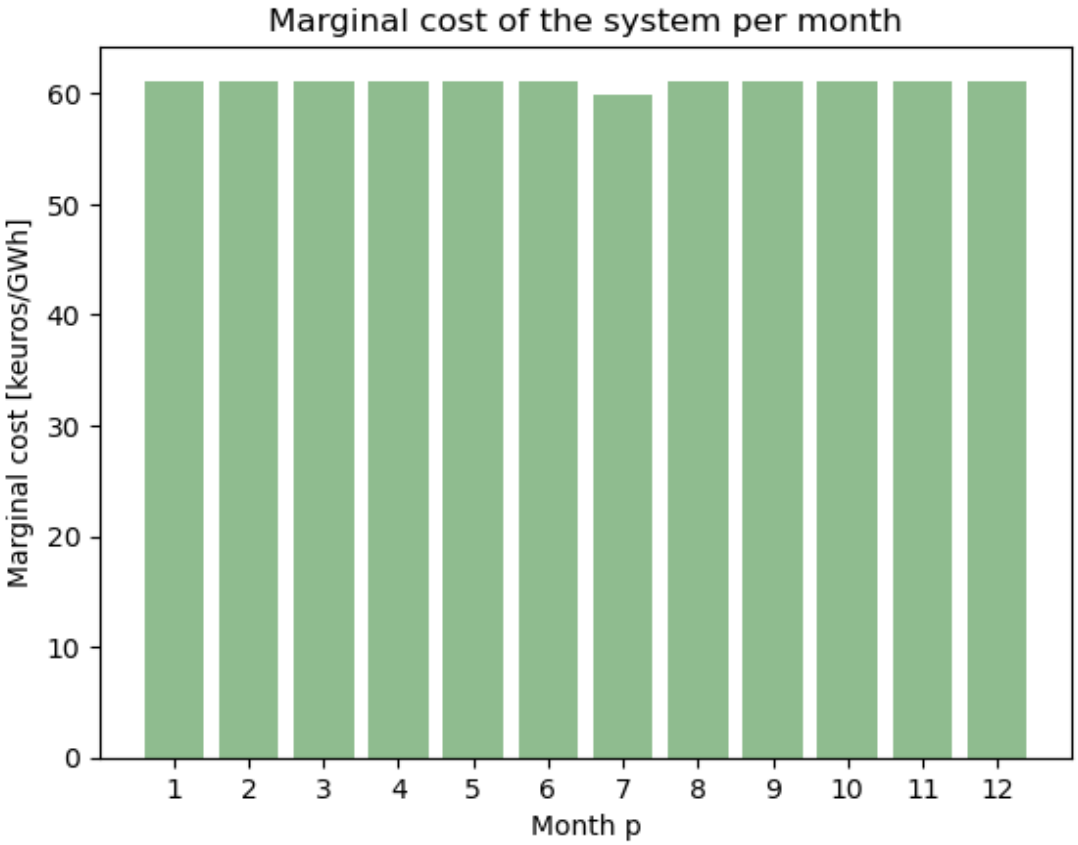


Figura 41. Resultados coste marginal de la producción anual en Pyomo.
Fuente: Elaboración propia

En la figura 41, se observa que el coste marginal es prácticamente homogéneo en todos los meses. Se observa tan solo una disminución en el mes de julio. Para una visión más exhaustiva, se realizará un análisis por mes, desglosando este en subperíodos s (laboral y festivo) y niveles de carga n (n1, n2, n3, n4, n5).

b. Costes marginales mensuales para subperíodos s y niveles de carga n

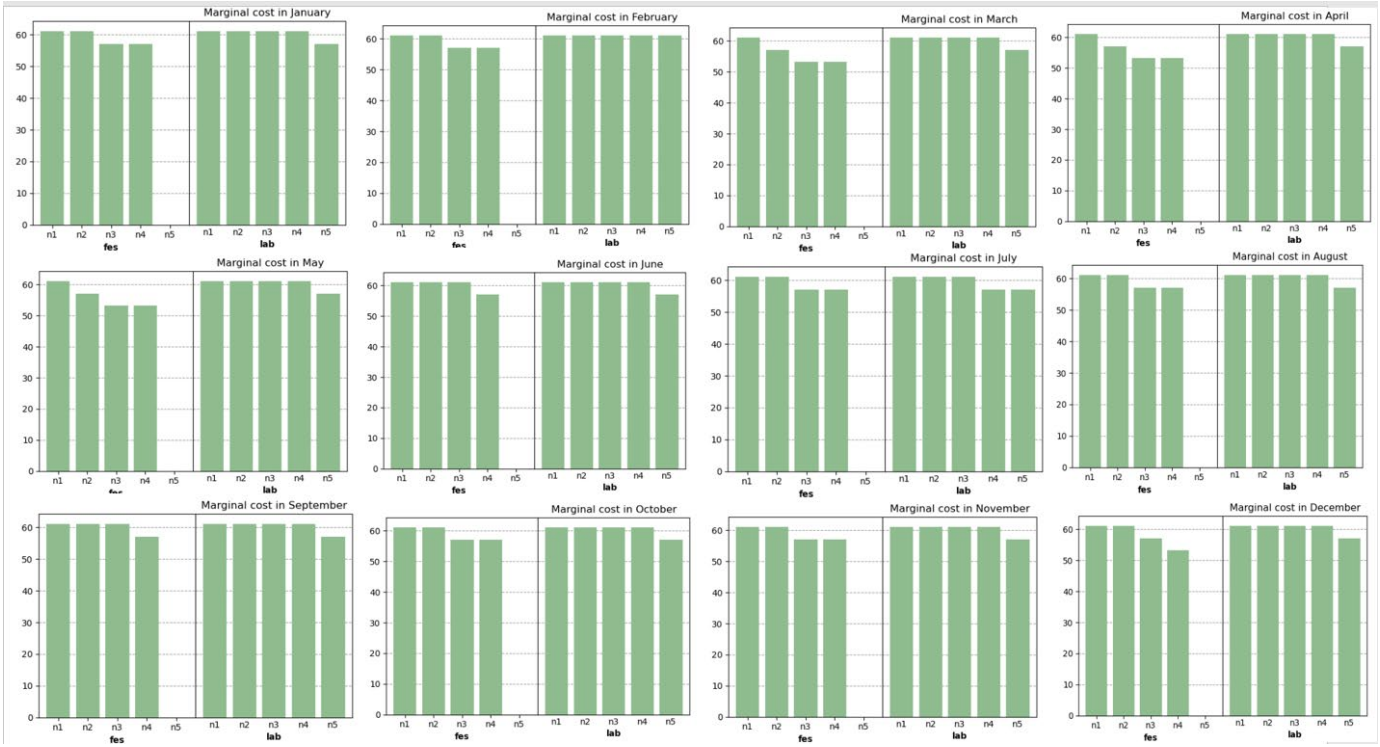


Figura 42. Costes marginales mensuales. Fuente: Elaboración propia

En la figura anterior, se observa la evolución de los costes marginales asociados a la ecuación de demanda (6) desglosada en períodos p, subniveles s y niveles de carga n. En primera instancia, se observan perfiles prácticamente homogéneos, observándose que en general, el perfil para los días festivos es ligeramente inferior al de los días laborables.

Por otro lado, los perfiles para cada mes apenas varían, llegando a ser casi idénticos para cada uno de los meses. Las unidades en las que se mide el eje Y son k€.

6) *Estudio de la evolución anual de las reservas para las distintas plantas hidráulicas.*

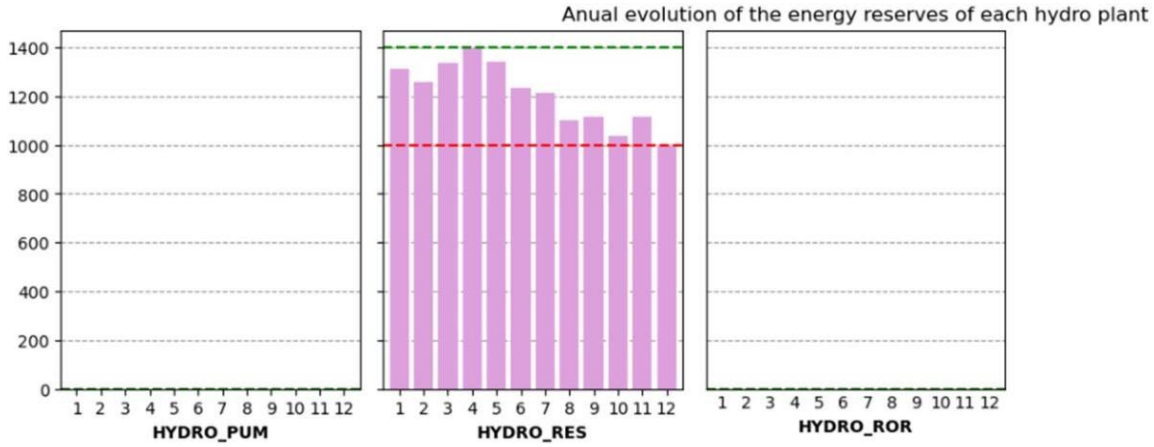


Figura 43. Reservas anuales. Fuente: Elaboración propia

En la figura 43, tan solo se observa evolución anual de las reservas en el segundo caso, pues 'HYDRO_PUM' representa el bombeo y 'HYDRO_ROR' alude a 'Run Of the River' lo cual implica que no existen reservas. Las líneas verdes y rojas aluden a la capacidad máxima y mínima de las reservas respectivamente.

7) *Estudio de la evolución de la variable dual de la ecuación de balance de energía de la reserva hidráulica (13): El valor marginal del agua.*

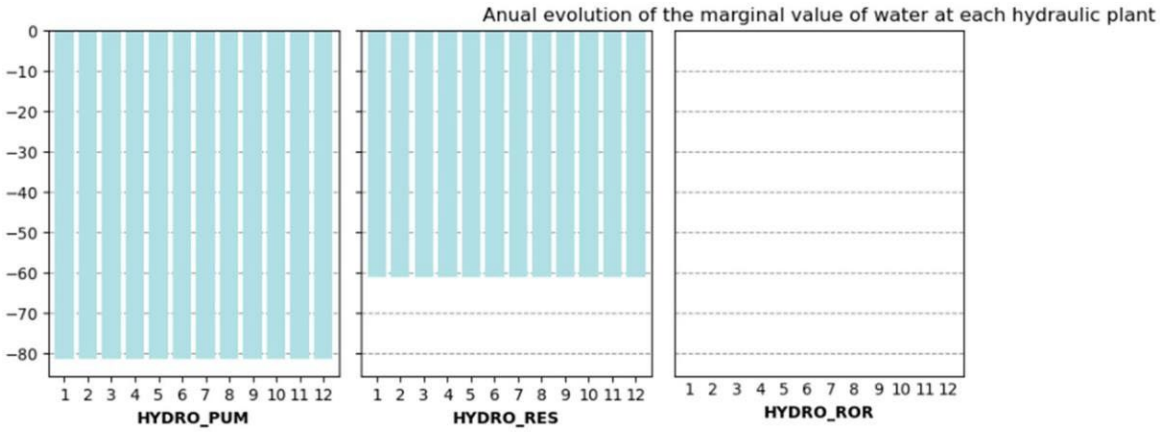


Figura 44. Evolución del valor marginal del agua anual. Fuente: Elaboración propia

En la figura anterior, se indica el coste marginal asociado al valor del agua, es decir, en cuanto se reduce el coste de producción en k€ al añadir al sistema un m^3 de agua.

8) Análisis de la producción de CO2.

Finalmente, es preciso analizar el efecto del precio del CO2 en el coste de producción. Para ello, tomaremos como referencia el precio de compraventa estimado por la empresa de compras de derechos de emisión SendeCO2.

Precios CO2

2023 | 2022 | 2021 | 2020 | 2019 | 2018 | 2017 | 2016 | 2015 | 2014 | 2013
2012 | 2011 | 2010 | 2009 | 2008

Precios CO2	EUA	CER
Media anual	**86,43 €**	**0,00 €**
Enero	80,29 €	0,00 €
Febrero	91,82 €	0,00 €
Marzo	89,23 €	0,00 €
Abril	90,52 €	0,00 €
Mayo	83,89 €	0,00 €
Junio	85,62 €	0,00 €
Julio	86,36 €	0,00 €
Agosto	84,88 €	0,00 €

Figura 45. Precios de CO2. Fuente: [42]

Teniendo en cuenta los precios de la figura 45, para evitar que el coste se dispare, se utilizará un precio de aproximadamente la mitad del establecido por el mercado. Aun así, se observará que el coste evoluciona de los 459 millones de euros hasta uno ligeramente superior a los 800 millones.

Capítulo 6. CONCLUSIONES Y TRABAJOS FUTUROS

6.1 RESUMEN DE LOS LOGROS ALCANZADOS

En este proyecto se ha propuesto un modelo de medio plazo, para la toma de decisiones para los sistemas de potencia eléctrica, mediante su implementación en Python Pyomo. El modelo de optimización tiene en cuenta factores técnicos legales y medioambientales a la hora de elaborar los costes de producción de la electricidad

De forma anidada, se podría concluir que los logros alcanzados son los siguientes:

1) Familiarizarse con el funcionamiento de Python, sus entornos, sintaxis y construcción de funciones, clases, etc.
2) Aprendizaje y utilización de la herramienta Pyomo para el desarrollo de modelos de optimización. Además, hemos aprendido los distintos tipos de solucionadores disponibles que se pueden integrar con la herramienta. Se ha realizado también un estudio de otras posibles opciones análogas a la trabajada en este proyecto.
3) Aprendizaje y teorización para la creación de una interfaz web. Se ha estudiado y descrito algunos de los campos de estudio requeridos para la creación de esta: conceptos de backend y frontend, bases de datos, protocolo de ciberseguridad, diseño web, etc.
4) Desde un enfoque más ingenieril, se ha realizado un estudio exhaustivo sobre el funcionamiento de la producción eléctrica, en particular, en el marco temporal del medio plazo (desde unos meses hasta un año aproximadamente). En líneas generales, se ha aprendido sobre las distintas tomas de decisiones involucradas, las restricciones existentes y la presencia, cada vez mayor, de las energías renovables (las cuales no se tratan en este proyecto, pero deben de estar presentes en los razonamientos).
5) Finalmente, desde el punto de vista matemático, se ha aprendido *grosso modo* algunas de las principales técnicas de optimización lineal, utilizadas para simplificar el planteamiento del problema objeto de estudio.

6.2 LIMITACIONES Y FUTURAS LÍNEAS DE INVESTIGACIÓN

Como parte final de este proyecto, se elabora un breve informe en el que se describe algunas de las principales limitaciones y se invita a futuros alumnos, a continuar esta investigación en otra línea de trabajo, pero complementaria a este estudio.

En primer lugar, como en todo proyecto, la principal limitación es la asociada al factor tiempo, pues este, junto con el ritmo de trabajo del investigador, definen los límites de la investigación.

En este estudio, se ha realizado la programación de un modelo de optimización sencillo en Pyomo. De realizar un proceso de escalabilidad (aumento del número de generadores, restricciones, variables del modelo…) como se ha mencionado a lo largo de este proyecto, existiría también una limitación asociada al tiempo de ejecución del código, por lo que es un aspecto a tener en cuenta para futuros proyectos. Por tanto, se recomendaría la evaluación de otros lenguajes como posibles alternativas (i.e AMPL, GAMS, etc).

Finalmente, como futuras líneas de investigación, (y relacionado con las limitaciones de este proyecto) se propone la profundización en lo descrito en la sección 2.4, relativa a la creación de una interfaz web. Una de las principales limitaciones de este proyecto, está relacionada con la portabilidad. Esto quiere decir, que para que algún tercero pueda ejecutar este código, es necesario enviarles, no solo el bloque de código, sino también todos los archivos de datos asociados (CSV, JSON, YAML, etc.). Esto, supone una ocupación de espacio que es innecesario. Por tanto, se propone la creación de una interfaz-web, en la cual se permita la modificación de datos de generación, restricciones… de forma dinámica y, estando estos ya cargados en la nube (lo que supone un ahorro de espacio, tiempo de ejecución y de carga de los archivos asociados al programa). Por tanto, se recomienda la realización de un proyecto que, como continuación a este, implemente la idea presentada (teniendo en cuenta la motivación expuesta).

Referencias bibliográficas

[1] González, J. G. (2014) *Decision support models in electric power systems*.

[2] Navarro, R. (2009) «Short and medium term operation planning in electric power systems», en *2009 IEEE/PES Power Systems Conference and Exposition*. IEEE.

[3] dos Santos, M. L. L. *et al.* (2009) «Practical aspects in solving the medium-term operation planning problem of hydrothermal power systems by using the progressive hedging method», *International journal of electrical power & energy systems*, 31(9), pp. 546-552. doi: 10.1016/j.ijepes.2009.03.032.

[4] Gómez, A. *et al.* (sin fecha) *Analisis y operacion de sistemas de energa electrica*, *Departamento.us.es*. Disponible en: https://departamento.us.es/ielectrica/wp-content/uploads/2021/08/Analisis-y-operacion-de-SEE_2010.pdf (Accedido: 23 de junio de 2023).

[5] Sun, X. A. y Conejo, A. J. (2021) «Medium-Term Planning Models», en *International Series in Operations Research & Management Science*. Cham: Springer International Publishing, pp. 281-302.

[6] Aizenberg, N. y Palamarchuk, S. (2020) "Medium-term scheduling of electric power system states under a wholesale electricity market", *E3S web of conferences*, 209, p. 02025. doi: 10.1051/e3sconf/202020902025.

[7] *Proceso de planificación* (sin fecha) *Ree.es*. Disponible en: https://www.ree.es/es/actividades/planificacion-electrica/proceso-planificacion-de-la-red (Consultado: el 28 de junio de 2023).

[8] *IBM Documentation* (2021) *Ibm.com*. Disponible en: https://www.ibm.com/docs/es/spss-modeler/saas?topic=language-object-oriented-programming (Consultado: el 30 de junio de 2023).

[9] "¿Qué es Python? Definición, características y sus ventajas" (2018) *Pythones*. Pyromaniac, 11 septiembre. Disponible en: https://pythones.net/que-es-python-y-sus-caracteristicas/ (Consultado: el 30 de junio de 2023).

[10] Lopez, J. (2017) *Programación Funcional - Jose Lopez*, *Medium*. Disponible en: https://medium.com/@gollodev/programaci%C3%B3n-funcional-5da872a080fe (Consultado: el 30 de junio de 2023).

[11] Wikipedia contributors (sin fecha) *Tipado dinámico*, *Wikipedia, The Free Encyclopedia*. Disponible en: https://es.wikipedia.org/w/index.php?title=Tipado_din%C3%A1mico&oldid=141832130. (Consultado: el 30 de junio de 2023)

[12] Browning, J. B. y Alchin, M. (2014) "The Zen of Python", en *Pro Python*. Berkeley, CA: Apress, pp. 333–334.

[13] "Python: qué es, para qué sirve y cómo se programa" (2020) *aula21 | Formación para la Industria*. aula21, 8 octubre. Disponible en: https://www.cursosaula21.com/que-es-python/ (Consultado: el 30 de junio de 2023).

[14] *Pyomo* (sin fecha) *Pyomo*. Disponible en: https://www.pyomo.org/ (Consultado: el 2 de julio de 2023).

[15] pyomo: An object-oriented algebraic modeling language in Python for structured optimization problems. Disponible en: https://github.com/Pyomo/pyomo (sin fecha).

[16] Optimization Expert (sin fecha) Pyomo: Pyomo respository provides a comprehensive library of solved models in Supply chain management. Disponible en: https://github.com/OptimizationExpert/Pyomo (sin fecha)

[17] OpenTEPES: Open Generation, Storage, and Transmission Operation and Expansion Planning Model with RES and ESS (openTEPES) (sin fecha). Disponible en: https://github.com/IIT-EnergySystemModels/openTEPES (sin fecha)

[18] PYOMO: Pyomo personal projects (sin fecha). Disponible en: https://github.com/juanraposo2000/PYOMO (sin fecha)

[19] Hart, W. E. *et al.* (2018) *Pyomo - Optimization Modeling in Python*. 2nd Edition.Cham, Suiza: Springer International Publishing.

[20] Hart, W. E. *et al.* (2014) *Pyomo - Optimization Modeling in Python*. Cham, Suiza: Springer International Publishing.

[21] Bynum, M. L. *et al.* (2022) *Pyomo - Optimization Modeling in Python*. 3a ed. Cham, Suiza: Springer Nature.

[22] *AMPL – optimizing the world's most complex tasks* (sin fecha) *Ampl.com*. Disponible en: https://ampl.com/ (Consultado: el 3 de julio de 2023).

[23] GAMS Software GmbH (sin fecha) *GAMS - cutting edge modeling*, *Gams.com*. Disponible en: https://www.gams.com/ (Consultado: el 3 de julio de 2023).

[24] Tomlab Optimization Inc (sin fecha) *MATLAB® Optimization Software*, *Tomopt.com*. Disponible en: https://tomopt.com/ (Consultado: el 3 de julio de 2023).

[25] Jusevičius, V., Oberdieck, R. y Paulavičius, R. (2021) "Experimental analysis of algebraic modelling languages for mathematical optimization", *Lithuanian Academy of Sciences. Informatica (Vilnius)*, pp. 283–304. doi: 10.15388/21-infor447.

[26] (Sin fecha c) *Udemy.com*. Disponible en: https://www.udemy.com/ (Consultado: el 3 de julio de 2023).

[27] Ramos, A. (sin fecha) Good optimization modeling practices with pyomo all you wanted to know about practical optimization but were afraid to ask, Comillas.edu. Disponible en: https://pascua.iit.comillas.edu/aramos/simio/transpa/s_GoodOptimizationModelingPracticesPyomo.pdf (Consultado: el 4 de julio de 2023).

[28] (Sin fecha) *Comillas.edu*. Disponible en: https://www.comillas.edu/images/Biblioteca/Formas_de_citar_TFG_15-16.pdf (Consultado: el 30 de junio de 2023).

[29] *8. Errores y excepciones* (sin fecha) *Python documentation*. Disponible en: https://docs.python.org/es/3/tutorial/errors.html (Consultado: el 30 de junio de 2023).

[30] *Anaconda Python y Conda* (2020) *El Pythonista*. Disponible en: https://elpythonista.com/anaconda (Consultado: el 2 de julio de 2023).

[31] Nahian, I. (2020) *8 Reasons you won't use Python*, *DEV Community*. Disponible en: https://dev.to/evilprince2009/8-reasons-you-won-t-use-python-3b46 (Consultado: el 2 de julio de 2023).

[32] "Ventajas y Desventajas de Python" (2021) *KeepCoding Bootcamps*. KeepCoding Tech School, 22 septiembre. Disponible en: https://keepcoding.io/blog/ventajas-y-desventajas-de-python/ (Consultado: el 2 de julio de 2023).

[33] *El tutorial de Python* (sin fecha) *Python documentation*. Disponible en: https://docs.python.org/es/3/tutorial/ (Consultado: el 2 de julio de 2023).

[34] Wikipedia contributors (sin fecha a) *Python*, *Wikipedia, The Free Encyclopedia*. Disponible en: https://es.wikipedia.org/w/index.php?title=Python&oldid=152110416. (Consultado: el 2 de julio de 2023).

[35] (Sin fecha b) *Amazon.com*. Disponible en: https://aws.amazon.com/es/what-is/ide/ (Consultado: el 2 de julio de 2023).

[36] *PyCharm: el IDE de Python para desarrolladores profesionales, por* (sin fecha) *JetBrains*. Disponible en: https://www.jetbrains.com/es-es/pycharm/ (Consultado: el 2 de julio de 2023).

[37] Extensions, L. M. A. (sin fecha) *Visual Studio Code - code editing. Redefined*, *Visualstudio.com*. Disponible en: https://code.visualstudio.com/ (Consultado: el 2 de julio de 2023).

[38] Guindon, C. (sin fecha) *Eclipse desktop & Web IDEs*, *Eclipse Foundation*. Disponible en: https://www.eclipse.org/ide/ (Consultado: el 2 de julio de 2023).

[39] *Anaconda* (2023) *Anaconda*. Disponible en: https://www.anaconda.com/ (Consultado: el 2 de julio de 2023).

[40] *Project jupyter* (sin fecha) *Jupyter.org*. Disponible en: https://jupyter.org/ (Consultado: el 2 de julio de 2023).

[41] *Stack Overflow - where developers learn, share, & build careers* (sin fecha) *Stack Overflow*. Disponible en: https://stackoverflow.com/ (Consultado: el 3 de julio de 2023).

[42] Sendeco2 (sin fecha) Sendeco2.com. Disponible en: https://www.sendeco2.com/es/ (Consultado: el 27 de agosto de 2023).

Anexos

ANEXO I

	PRINCIPALES EXCEPCIONES DEFINIDAS EN PYTHON
TypeError:	Operación con tipo de dato inapropiado.
ZeroDivisionError:	Error al intentar dividir por cero.
OverflowError:	Error debido a que el cálculo excede el límite para un tipo de dato numérico.
IndexError:	Error tras intentar acceder a una secuencia con un índice inexistente.
KeyError:	Acceso a un diccionario con una clave que no existe.
FileNotFoundError:	Acceso a un fichero que no existe en la ruta indicada.
ImportError:	Fallo de la importación de un módulo.

Tabla 2. Principales errores o excepciones definidas en Python.

ANEXO II

In [1]:
```
from pyomo.environ import*
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
model = AbstractModel()
model.N = Param(initialize = 60, within= NonNegativeIntegers, mutable= True)
model.i = RangeSet(model.N)
model.j = Set(initialize = model.i)
model.r = Var(initialize = 0, bounds= (0.001, 2), within= NonNegativeReals)
model.x = Var(model.i, initialize= 0, bounds= (0.001, 2), within= NonNegativeReals)
model.y = Var(model.i, initialize= 0, bounds=(0.001, 2), within= NonNegativeReals)
def c1_eq_rule(model, i, j):
    if i > j:
        return (model.x[i] - model.x[j])**2 + (model.y[i] -model.y[j])**2 >= (model.r + model.r)**2
    else:
        return Constraint.Skip
model.c1 = Constraint(model.i, model.j, rule= c1_eq_rule)

def c2_eq_rule(model, i):
    return (model.x[i] - 1)**2 + (model.y[i] - 1)**2 <= (1 - model.r)**2

model.c2 = Constraint(model.i, rule= c2_eq_rule)

model.obj = Objective(expr = model.r, sense= maximize)
```

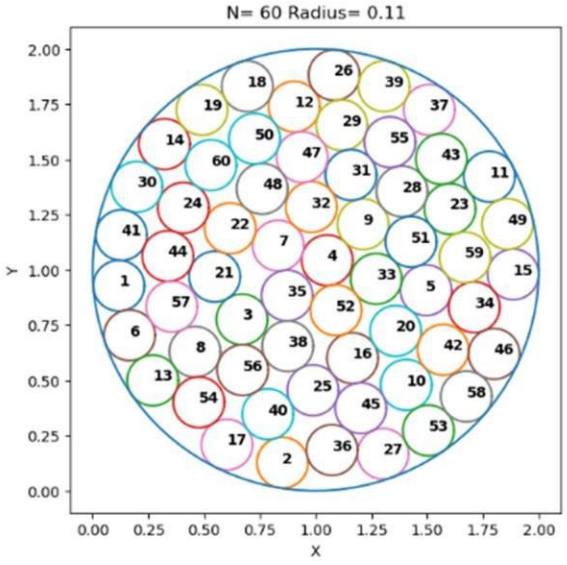
In [3]:
```
opt = SolverFactory('ipopt')
instance = model.create_instance()
results = opt.solve(instance)
```

```
WARNING: Constructing RangeSet 'i' from non-constant data (e.g., Var or
    mutable Param).  The linkage between this RangeSet and the original source
    data will be broken, so updating the data value in the future will not be
    reflected in this RangeSet.  To suppress this warning, explicitly convert
    the source data to a constant type (e.g., float, int, or immutable Param)
WARNING (W1002): Setting Var 'r' to a numeric value `0` outside the bounds
    (0.001, 2).
    See also https://pyomo.readthedocs.io/en/stable/errors.html#w1002
WARNING (W1002): Setting Var 'x[1]' to a numeric value `0` outside the bounds
    (0.001, 2).
    See also https://pyomo.readthedocs.io/en/stable/errors.html#w1002
WARNING (W1002): Setting Var 'y[1]' to a numeric value `0` outside the bounds
    (0.001, 2).
    See also https://pyomo.readthedocs.io/en/stable/errors.html#w1002
```

In [4]:
```
instance.pprint()
```

```
2 Set Declarations
    c1_index : Size=1, Index=None, Ordered=True
        Key  : Dimen : Domain : Size : Members
        None :     2 :    i*j : 3600 : {(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (1, 9), (1, 10), (1,
11), (1, 12), (1, 13), (1, 14), (1, 15), (1, 16), (1, 17), (1, 18), (1, 19), (1, 20), (1, 21), (1, 22), (1, 23), (1, 24),
(1, 25), (1, 26), (1, 27), (1, 28), (1, 29), (1, 30), (1, 31), (1, 32), (1, 33), (1, 34), (1, 35), (1, 36), (1, 37), (1, 3
8), (1, 39), (1, 40), (1, 41), (1, 42), (1, 43), (1, 44), (1, 45), (1, 46), (1, 47), (1, 48), (1, 49), (1, 50), (1, 51), (1,
52), (1, 53), (1, 54), (1, 55), (1, 56), (1, 57), (1, 58), (1, 59), (1, 60), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 6),
(2, 7), (2, 8), (2, 9), (2, 10), (2, 11), (2, 12), (2, 13), (2, 14), (2, 15), (2, 16), (2, 17), (2, 18), (2, 19), (2, 20),
(2, 21), (2, 22), (2, 23), (2, 24), (2, 25), (2, 26), (2, 27), (2, 28), (2, 29), (2, 30), (2, 31), (2, 32), (2, 33), (2, 3
4), (2, 35), (2, 36), (2, 37), (2, 38), (2, 39), (2, 40), (2, 41), (2, 42), (2, 43), (2, 44), (2, 45), (2, 46), (2, 47), (2,
48), (2, 49), (2, 50), (2, 51), (2, 52), (2, 53), (2, 54), (2, 55), (2, 56), (2, 57), (2, 58), (2, 59), (2, 60), (3, 1), (3,
2), (3, 3), (3, 4), (3, 5), (3, 6), (3, 7), (3, 8), (3, 9), (3, 10), (3, 11), (3, 12), (3, 13), (3, 14), (3, 15), (3, 16),
(3, 17), (3, 18), (3, 19), (3, 20), (3, 21), (3, 22), (3, 23), (3, 24), (3, 25), (3, 26), (3, 27), (3, 28), (3, 29), (3, 3
0), (3, 31), (3, 32), (3, 33), (3, 34), (3, 35), (3, 36), (3, 37), (3, 38), (3, 39), (3, 40), (3, 41), (3, 42), (3, 43), (3,
44), (3, 45), (3, 46), (3, 47), (3, 48), (3, 49), (3, 50), (3, 51), (3, 52), (3, 53), (3, 54), (3, 55), (3, 56), (3, 57),
(3, 58), (3, 59), (3, 60), (4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (4, 6), (4, 7), (4, 8), (4, 9), (4, 10), (4, 11), (4, 1
2), (4, 13), (4, 14), (4, 15), (4, 16), (4, 17), (4, 18), (4, 19), (4, 20), (4, 21), (4, 22), (4, 23), (4, 24), (4, 25), (4,
26), (4, 27), (4, 28), (4, 29), (4, 30), (4, 31), (4, 32), (4, 33), (4, 34), (4, 35), (4, 36), (4, 37), (4, 38), (4, 39),
```

In [6]:
```
fig = plt.figure(figsize=(6,6)) #Opening a frame
theta = np.linspace(0, 2*np.pi, 100)
for i in instance.i:
    Xc = value(instance.x[i]) + value(instance.r)*np.cos(theta) #drawing a circle
    Yc = value(instance.y[i]) + value(instance.r)*np.sin(theta)
    plt.text(value(instance.x[i]), value(instance.y[i]), str(i), fontweight= 'bold')
    plt.plot(Xc,Yc)

Xc = 1 + np.cos(theta)
Yc = 1 + np.sin(theta)
plt.plot(Xc, Yc)#plot big circle
#plt.plot(1 + 1*np.cos(theta), 1 + 0.5*np.sin(theta))
plt.title('N= ' + str(value(instance.N)) + ' Radius= ' + str(round(value(instance.r), 2)))
plt.xlabel('X')
plt.ylabel('Y')
plt.show() #Show means the end of the plotting (this means if you plt anything after it will be in another frame)
```



ANEXO III

Reals = floating point values
PositiveReals = strictly positive floating point values
NonPositiveReals = non-positive floating point values
NegativeReals = strictly negative floating point values
NonNegativeReals = non-negative floating point values
PercentFraction = floating point values in the interval [0,1]
UnitInterval = alias for PercentFraction
Integers = integer values
PositiveIntegers = positive integer values
NonPositiveIntegers = non-positive integer values
NegativeIntegers = negative integer values
NonNegativeIntegers = non-negative integer values
Binary = binary values

ANEXO IV

```
from __future__import division
from pyomo.environ import *
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

m = AbstractModel('TFG_MMP_v2.0_July_2023')

#SET DECLARATION
m.G = Set()
m.p = Set()
m.s = Set()
m.n = Set()
m.t = Set()
m.h = Set()
m.hf = Set()
m.eo = Set()
m.pv = Set()

data  =  DataPortal()
data.load(filename= 'g.csv', set= m.G)
data.load(filename= 'p.csv', set= m.p)
data.load(filename= 's.csv', set= m.s)
data.load(filename= 'n.csv', set= m.n)
data.load(filename= 't.csv', set= m.t)
data.load(filename= 'h.csv', set= m.h)
data.load(filename= 'hf.csv', set= m.hf)
data.load(filename= 'e.csv', set= m.eo)
data.load(filename= 'pv.csv', set= m.pv)

#SCALAR DECLARATION
m.e = 480 #Minimum equivalent hours that a generator must run in the year [h]

#PARAMETER DECLARATION
m.CO2rate = Param(m.G, within = NonNegativeReals) #CO2 emission rate [tones CO2 per MWh]
m.alfa = Param(m.G, within = NonNegativeReals) #Incremental term of fuel consumption of generator
g [MTh per GWh]
m.beta = Param(m.G, within = NonNegativeReals) #Fixed term of fuel consumption of generator g
[MTh per h]
m.gamma = Param(m.G, within = NonNegativeReals) #Start up fuel consumption of generator g
[MTh]
m.f = Param(m.G, within = NonNegativeReals) #Price of fuel consumed by generator g [kEuros per
MTh]
m.o = Param(m.G, within = NonNegativeReals) #Marginal operation and maintenance cost of
```

```
generator g [kEuros per GWh]
m.qmax = Param(m.G, within = NonNegativeReals) #Maximum gross power of generator g [GW]
m.qmin = Param(m.G, within = NonNegativeReals) #Minimum gross power of generator g [GW]
m.bmax = Param(m.G, within = NonNegativeReals) #Maximum gross pump power of generator g [GW]
m.wmax = Param(m.G, within = NonNegativeReals) #Maximum reservoir energy of hydro plant g [GWh]
m.w0 = Param(m.G, within = NonNegativeReals) #Initial reserve level of hydro plant g [GWh]
m.wmin = Param(m.G,within = NonNegativeReals) #Minimum reserve level hydro plant g [GWh]
m.k = Param(m.G, within = NonNegativeReals) #Gross to net power conversion factor [p.u.]
m.rend = Param(m.G, within = NonNegativeReals) #Efficiency of the pumping-storrage cycle [p.u.]



# Read CSV file into pandas DataFrame and convert to Pyomo parameter
param_data = pd.read_csv('Data_duration_demand1.csv', header=0, index_col=[0, 1, 2])
m.a = Param(m.n, m.s, m.p, initialize=param_data['a'].to_dict(), within = NonNegativeReals)
#Duration of level n subperiod s period p [h]
m.d = Param(m.n, m.s, m.p, initialize=param_data['d'].to_dict(), within = NonNegativeReals)
#Demand of level n subperiod s period p [GW]

param_data2 = pd.read_csv('Data_inflows1.csv', header=0, index_col=[0, 1])
m.i = Param(m.h, m.p, initialize=param_data2['i'].to_dict(), within = NonNegativeReals) #Inflows received by catchment basin of generator g during period p [GWh]


param_data3 = pd.read_csv('Data_qred1.csv', header=0, index_col=[0, 1, 2, 3])
m.qred = Param(m.t, m.n, m.s, m.p, initialize=param_data3['qred'].to_dict(), within=NonNegativeReals) #Minimum power due to network constraints of generator g [GW]

param_data4 = pd.read_csv('CO2euros.csv', header=0, index_col=[0])
m.CO2euros = Param(m.p, initialize=param_data4['CO2euros'].to_dict(), within = NonNegativeReals)

param_data5 = pd.read_csv('qpvmax.csv', header=0, index_col=[0, 1, 2, 3])
m.qpvmax = Param(m.pv, m.n, m.s, m.p, initialize=param_data5['qpvmax'].to_dict(), within = NonNegativeReals) #Maximum gross power of generator eo [GW]

param_data6 = pd.read_csv('qemax.csv', header=0, index_col=[0, 1, 2, 3])
m.qemax = Param(m.eo, m.n, m.s, m.p, initialize=param_data6['qemax'].to_dict(), within = NonNegativeReals)

#m.qemax = Param(m.G, within = NonNegativeReals)
#m.qpvmax = Param(m.G, within = NonNegativeReals) #Maximum gross power of generator pv [GW]

data.load(filename="Data_generators1.csv", param=(m.CO2rate, m.alfa , m.beta , m.gamma , m.f , m.o , m.qmax , m.qmin , m.bmax , m.wmax , m.w0 , m.wmin , m.k , m.rend))
instance = m.create_instance(data)
```

```
instance.pprint()

#VARIABLE DECLARATION
m.u = Var(m.t, m.s, m.p,within= Binary) #u(t,s,p) Decision to connect generator g
m.y = Var(m.t,m.s, m.p, within= Binary)#y(t,s,p) Start up decision for generator g
m.z = Var(m.t, m.s, m.p, within= Binary)#z(t,s,p) Shut down decision for generator g

m.qt = Var(m.t, m.n, m.s, m.p, within= NonNegativeReals) #q(t,n,s,p) Net power delivered by thermal generator t [GW]
m.qh = Var(m.h, m.n, m.s, m.p, within= NonNegativeReals) #q(h,n,s,p) Net power delivered by hydraulic generator h [GW]
m.b = Var(m.h, m.n, m.s, m.p, within= NonNegativeReals) #b(h,n,s,p)Gross power consumed by pumping-storage plant [GW]
m.w = Var(m.h, m.p, within= NonNegativeReals) #w(h,p) Hydroelectric energy in the reservoir of the plant [GWh]
m.qe = Var(m.eo, m.n, m.s, m.p, within= NonNegativeReals) #qe(eo,n,s,p) Net power delivered by eolic generator eo [GW]
m.qpv = Var(m.pv, m.n,m.s,m.p, within= NonNegativeReals) #qpv(pv,n,s,p) Net power delivered by photovoltaic generator pv [GW]
m.CO2out = Var(m.t, m.n,m.s,m.p, within= NonNegativeReals) #Tons of CO2 generated by thermal generator t [tons]

#OBJECTIVE FUNCTION DECLARATION
def obj_rule (m):
   return sum((m.f[t]*(m.gamma[t]*m.y[t,s,p] + sum((m.a[n, s, p] * (m.alfa[t] * m.qt[t,n,s,p] / m.k [t] + m.beta[t] * m.u[t, s, p])) for n in m.n)) +
      sum((m.a [n, s, p]* m.o[t] * m.qt[t, n, s, p] / m.k[t]) for n in m.n )) for t in m.t for p in m.p for s in m.s) + sum((m.CO2out[t,n,s,p]*m.CO2euros[p]*0.001) for t in m.t for n in m.n for s in m.s for p in m.p)

m.obj = Objective(rule= obj_rule, sense= minimize)


#DECLARATION OF CONSTRAINTS (meaning of warnings????) ## Very important the order
   #Demand balance and marginal cost

def Demand_Bal_rule(m,n,s,p):
   if m.d[n,s,p] == 0:
      return Constraint.Skip
   else:
      return sum((m.qt[t,n,s,p]) for t in m.t) + sum((m.qh[h,n,s,p] - m.b[h,n,s,p]) for h in m.h) + sum((m.qe[eo,n,s,p]) for eo in m.eo) + sum((m.qpv[pv,n,s,p]) for pv in m.pv)  ==  m.d[n,s,p]

m.Demand_Bal = Constraint(m.n, m.s, m.p, rule= Demand_Bal_rule)

   #Upper thermal limit
def Upper_Thermal_rule(m,t,n,s,p):
   if m.d[n,s,p] == 0:
      return Constraint.Skip
   else:
      return m.qt[t,n,s,p] <= m.u[t,s,p] * m.k[t] * m.qmax[t]
```

```
m.Upper_Thermal = Constraint(m.t, m.n, m.s, m.p, rule= Upper_Thermal_rule)
   #Upper hydro constraints
def Upper_Hydro_rule(m,h,n,s,p):
   return m.qh[h,n,s,p] <= m.k[h] * m.qmax[h]

m.Upper_Hydro = Constraint(m.h, m.n, m.s, m.p, rule= Upper_Hydro_rule)

   #Maximum Capacity limits for pumped storage
def Upper_Pumping_rule(m,h,n,s,p):
   return m.b[h,n,s,p] <= m.k[h] * m.bmax[h]

m.Upper_Pumping_Hydro = Constraint(m.h, m.n, m.s, m.p, rule= Upper_Pumping_rule)

   #Lower thermal limit
def Lower_Thermal_rule(m,t,n,s,p):
   if m.d[n,s,p] == 0:
      return Constraint.Skip
   else:
      return m.qt[t,n,s,p] >= m.u[t,s,p] * m.k[t] * m.qmin[t]

m.Lower_Thermal = Constraint(m.t, m.n, m.s, m.p, rule= Lower_Thermal_rule)

   #Energy balance in an equivalent reservoir
def E_Bal_rule(m,h,p):
   if p == m.p.first() and h == 'HYDRO_RES':
       return m.w[h,p] + sum(m.a[n,s,p] * (m.qh[h,n,s,p]/m.k[h] - m.rend[h] * m.b[h,n,s,p]) for n in m.n
for s in m.s) <= m.w0['HYDRO_RES'] + m.i[h,p]
   elif p == m.p.first():
       return m.w[h,p] + sum(m.a[n,s,p] * (m.qh[h,n,s,p]/m.k[h] - m.rend[h] * m.b[h,n,s,p]) for n in m.n
for s in m.s) <= m.i[h,p]
   else:
      return m.w[h,p] + sum(m.a[n,s,p] * (m.qh[h,n,s,p]/m.k[h] - m.rend[h] * m.b[h,n,s,p]) for n in m.n
for s in m.s) <= m.w[h,int(p) - 1] + m.i[h,p]

m.E_Bal = Constraint(m.h, m.p, rule= E_Bal_rule)

   #Upper limit equivalent energy stored in reservoir
def Upper_W_rule(m,h,p):
   return m.w[h,p] <= m.wmax[h]

m.Upper_W = Constraint(m.h, m.p, rule = Upper_W_rule)

   #Lower limit equivalent energy stored in basin
def Lower_W_rule(m,h,p):
   return m.w[h,p] >= m.wmin[h]

m.Lower_W = Constraint(m.h, m.p, rule = Lower_W_rule)

   #Minimum operating hours for thermal group during the year
def Min_H_t_rule(m,t):
```

```
    return sum((m.a[n,s,p] * m.qt[t,n,s,p]) for n in m.n for s in m.s for p in m.p) >= m.k[t] * m.qmax[t] *
m.e

m.Min_H_t = Constraint(m.t, rule = Min_H_t_rule)

 #Minimum operating hours for hidraulic group during the year
def Min_H_h_rule(m,h):
    return sum((m.a[n,s,p] * m.qh[h,n,s,p]) for n in m.n for s in m.s for p in m.p) >= m.k[h] *
m.qmax[h] * m.e

m.Min_H_h = Constraint(m.h, rule = Min_H_h_rule)

    #Minimum production to accommodate grid constraints
def Min_Grid_rule(m,t,n,s,p):
    return m.qt[t,n,s,p] >= m.qred[t,n,s,p]

m.Min_Grid = Constraint(m.t, m.n, m.s, m.p, rule = Min_Grid_rule)

    #Constraint for intra-period changes
def Intrap_changes_rule(m,t,p):
    return m.u[t,'fes',p] - m.u[t,'lab',p] == m.y[t,'fes',p] - m.z[t,'fes',p]

m.Intrap_changes = Constraint(m.t, m.p, rule= Intrap_changes_rule)

    #Constraint for inter-period transition
def Intrap_transitions_rule(m,t,p):
    if p == m.p.first():
        return Constraint.Skip
    else:
        return m.u[t,'lab',p] - m.u[t,'fes',int(p) - 1] == m.y[t,'lab',p] - m.z[t,'lab',p]

m.Intrap_transitions = Constraint(m.t, m.p, rule= Intrap_transitions_rule)

def Fix_run_river_rule (m,h,n,s,p):
    if h != 'HYDRO_ROR':
        return Constraint.Skip
    else:
        return m.qh[h,n,s,p] == m.i[h,p]/sum(m.a[n,s,p] for n in m.n for s in m.s)

m.Fix_run_river_rule = Constraint (m.h,m.n, m.s, m.p, rule= Fix_run_river_rule)

    #Constraint maximum eolic power
def Max_eolic_rule(m,eo,n,s,p):
    return m.qe[eo,n,s,p] <= m.qemax[eo,n,s,p]

m.Max_eolic = Constraint(m.eo, m.n, m.s, m.p, rule= Max_eolic_rule)

    #Constraint for maximum photovoltaic power
def Max_photovoltaic_rule(m, pv,n,s,p):
    return m.qpv[pv,n,s,p] <= m.qpvmax[pv,n,s,p]
```

```
m.Max_photovoltaic = Constraint(m.pv, m.n, m.s, m.p, rule= Max_photovoltaic_rule)


def CO2_emissions_rule(m,t,n,s,p):
    return m.CO2out[t,n,s,p] == m.qt[t,n,s,p]*m.a[n,s,p]*m.CO2rate[t] * 1 / 1e-3

m.CO2_emissions = Constraint(m.t, m.n, m.s, m.p, rule = CO2_emissions_rule)

#DUAL DECLARATION
m.dual = Suffix(direction=Suffix.IMPORT, datatype= None)

#INSTANCE & SOLVER DECLARATION
instance = m.create_instance(data)
#instance.pprint()

solver = SolverFactory('gurobi')

solver.options['Method'          ] =    1                    # Solution method: dual-simplex {1} o
barrier {2}
solver.options['MIPGap'          ] =    0.00                    # Optcr
solver.options['Threads'         ] =    7                    # Number of cores dedicated to the
optimization
solver.options['IntFeasTol'      ] = 1e-9
solver.options['OptimalityTol'   ] = 1e-9
solver.options['FeasibilityTol' ] = 1e-9
solver.options['RINS'            ] = 100
solver.options['DisplayInterval' ] = 30
solver.options['NumericFocus'    ] =    3
solver.options['MarkowitzTol'    ] =    0.999
solver.options['ScaleFlag'       ] =    2

results = solver.solve(instance)
instance.pprint() #PYOMO WORKS IN ALPHABETICAL ORDER
results.write()

#RESULT STATUS
if (results.solver.status == SolverStatus.ok) and (results.solver.termination_condition ==
TerminationCondition.optimal):
    print('feasible')
    instance.write('MMP.lp',io_options={'symbolic_solver_labels': True})
elif (results.solver.termination_condition == TerminationCondition.infeasible):
    print('infeasible')
else:
    print ('Solver Status:',  result.solver.status)

#SOLUTION
print('OBJ=',round(value(instance.obj),2), "keuros")
```

```
#FIXING BINARY VARIABLES TO DISPLAY DUAL VARIABLES

if results.solver.termination_condition == TerminationCondition.optimal:
    # Fix binary variable to its optimal value
    for t in instance.t:
        for s in instance.s:
            for p in instance.p:
                instance.y[t,s, p].fix(instance.y[t, s,p].value)
    for t in instance.t:
        for n in instance.n:
            for s in instance.s:
                for p in instance.p:
                    instance.u[t,s,p].fix(instance.u[t,s,p].value)
    for t in instance.t:
        for n in instance.n:
            for s in instance.s:
                for p in instance.p:
                    instance.z[t,s,p].fix(instance.z[t,s,p].value)



    # Solve the model again
    result = solver.solve(instance)  # Solve the model with fixed binary variables

    # Check the new solution
    if result.solver.termination_condition == TerminationCondition.optimal:
        # Access the updated objective value
        updated_objective_value = instance.obj.expr()  # Updated objective value
        print("Updated objective value:", updated_objective_value)
    else:
        print("The model did not reach an optimal solution after fixing the binary variables.")
else:
    print("The model did not reach an optimal solution.")

#instance.dual.display()

#STACKED PRODUCTION BY TECHNOLOGY

#THERMAL
fig, ax = plt.subplots()
x_values = range(1, len(instance.p) + 1)
for p in instance.p:
    [ax.bar(p, value(sum(instance.qt[t, n, s, p] for s in instance.s for n in instance.n for t in instance.t)),
color = 'darksalmon')]
ax.set_xticks(x_values)
ax.set_xlabel("Month p")
ax.set_ylabel("Net power[GW]")
ax.set_title("Thermal Net power delivered by month")
plt.show()
```

```
#HYDRAULIC

fig2, ax2 = plt.subplots()
x_values = range(1, len(instance.p) + 1)
for p in instance.p:
    [ax2.bar(p, value(sum(instance.qh[h, n, s, p] for s in instance.s for n in instance.n for h in
instance.h)), color = 'skyblue')]
ax2.set_xticks(x_values)
ax2.set_xlabel("Month p")
ax2.set_ylabel("Net power[GW]")
ax2.set_title("Hydraulic Net power delivered by month")
plt.tight_layout()
plt.show()

#WIND
fig3, ax3 = plt.subplots()
x_values = range(1, len(instance.p) + 1)
for p in instance.p:
    [ax3.bar(p, value(sum(instance.qe[eo, n, s, p] for s in instance.s for n in instance.n for eo in
instance.eo)), color = 'mediumseagreen')]
ax3.set_xticks(x_values)
ax3.set_xlabel("Month p")
ax3.set_ylabel("Net power[GW]")
ax3.set_title("Wind Net power delivered by month")
plt.show()

#SOLAR
fig4, ax4 = plt.subplots()
x_values = range(1, len(instance.p) + 1)
for p in instance.p:
    [ax4.bar(p, value(sum(instance.qpv[pv, n, s, p] for s in instance.s for n in instance.n for pv in
instance.pv)), color = 'khaki')]
ax4.set_xticks(x_values)
ax4.set_xlabel("Month p")
ax4.set_ylabel("Net power[GW]")
ax4.set_title("Solar Net power delivered by month")
plt.show()


#STACKED ANNUAL PRODUCTION
#month_dict =('January', 'February','March','April','May','June','July','August','September'
            #'October', 'November', 'December')

values1 = []
values2 = []
values3 = []
values4 = []
awidth = []
fig3,ax3 = plt.subplots()
x_values = range(1, len(instance.p) + 1)
```

```
for p in instance.p:
    values1.append(value(sum(instance.qt[t, n, s, p] for s in instance.s for n in instance.n for t in
instance.t)))
    values2.append(value(sum(instance.qh[h, n, s, p] for s in instance.s for n in instance.n for h in
instance.h)))
    values3.append(value(sum(instance.qe[eo, n, s, p] for s in instance.s for n in instance.n for eo in
instance.eo)))
    values4.append(value(sum(instance.qpv[pv, n, s, p] for s in instance.s for n in instance.n for pv in
instance.pv)))
    awidth.append(value(sum(instance.a[n, s, p] for s in instance.s for n in instance.n)))
    ax3.bar(p, values1[p - 1], width = awidth[p - 1] / 750,color='darksalmon')
    if values2 [p - 1] >= 0:
        ax3.bar(p, values2[p - 1], bottom = values1[p - 1], width = awidth[p - 1] / 750, color='skyblue')
    else: #Because we have negative values, the bottom needs to be set to zero to avoid overlapping
        ax3.bar(p, values2[p - 1],width = awidth[p - 1] / 750, color = 'skyblue')
    ax3.bar(p,values3[p - 1], width = awidth[p - 1] / 750, color = 'mediumseagreen')
    ax3.bar(p,values4[p -1],width = awidth[p - 1] / 750, color = 'khaki')
    #print("value thermal", p, ":", values1[p - 1])
    #print("value hydraulic", p, ":", values2[p - 1])
    #print("hours of work in month", p,":", awidth[p - 1])

#values3 = []
#values3 = [(values1[p - 1] + values2[p - 1]) for p in instance.p]
plt.plot(instance.p, list(map(lambda p: values1[p - 1] + values2[p - 1], instance.p))
, color = 'black', marker= 'o', markersize= '2')
#print(map(lambda p: values1[p - 1] + values2[p - 1], instance.p))

# Customizing the chart
ax3.set_xticks(x_values)
ax3.set_xlabel("Month p")
ax3.set_ylabel("Net power [GW]")
ax3.set_title("Net power delivered by month")
ax3.legend(['Net value', 'Thermal', 'Hydraulic','Eolic', 'Solar'])

# Displaying the chart
plt.show()


#STACKED MONTHLY PRODUCTION IN S & N:
#Create multindex dataframe
arrays = []
aux = []
for i in range(1,len(instance.s) + 1):
    if i == 1:
        aux = [instance.s.at(i) for j in range(1,len(instance.n) + 1)]
        arrays.append(aux)
    else:
        aux = [instance.s.at(i) for j in range(1,len(instance.n) + 1)]
        arrays[i - 2].extend(aux)
for k in range(1, len(instance.s) + 1):
    aux = [instance.n.at(l) for l in range(1,len(instance.n) + 1)]
```

```
    if k == 1:
        arrays.append(aux)
    else:
        arrays[len(instance.s) - 1].extend(aux)
#print(arrays)

index = pd.MultiIndex.from_tuples(list(zip(*arrays)))
#print(index)
#print(list(zip(*arrays)))

data_array= [10.0]
#t,h -> n-> s -> p
for s in instance.s:
    for n in instance.n:
        values1 = value(sum(instance.qt[t, n, s, 1] for t in instance.t))
        data_array.append(values1)
        #print("values thermal ",values1, " at level ",n,"in week ",s)
for s in instance.s:
    for n in instance.n:
        values2 = value(sum(instance.qh[h, n, s, 1] for h in instance.h))
        data_array.append(values2)
        #print("values hydraulic ",values2, " at level ",n,"in week ",s)
for s in instance.s:
    for n in instance.n:
        values3 = value(sum(instance.qe[eo, n, s, 1] for eo in instance.eo))
        data_array.append(values2)
        #print("values eolic ",values2, " at level ",n,"in week ",s)
for s in instance.s:
    for n in instance.n:
        values4 = value(sum(instance.qpv[pv, n, s, 1] for pv in instance.pv))
        data_array.append(values4)
        #print("values photovoltaic ",values2, " at level ",n,"in week ",s)


data_array.remove(10.0)
#print(data_array)
my_array = np.array(data_array)

resized_array = my_array.reshape(4,10)
#print(resized_array)

df = pd.DataFrame(data=resized_array ,columns= index)
#print(df)
#print(df.columns.levels[0])
#print(df.columns.levels[1])
#Plotting
fig, axes = plt.subplots(nrows=1, ncols=2, sharey= True, figsize= (14 / 2.54, 10 / 2.54))
for i, col in enumerate(df.columns.levels[0]):
    #print(col)
    #print(i)
    ax = axes[i]
```

```
    df[col].T.plot(ax=ax, kind='bar',stacked= True, width=.8,
color=['darksalmon','skyblue','mediumseagreen','khaki'])


    ax.legend_.remove()
    ax.set_xlabel(col, weight='bold')
    ax.yaxis.grid(visible=True, which='major', color='black', linestyle='--', alpha=.4)
    ax.set_axisbelow(True)

    for tick in ax.get_xticklabels():
        tick.set_rotation(0)

#make the ticklines invisible
ax.tick_params(axis=u'both', which=u'both', length=0)
ax.set_title("Net power produced in January")
plt.tight_layout()
# remove spacing in between
fig.subplots_adjust(wspace=0)  # space between plots
ax.legend(['Thermal', 'Hydraulic','Wind','Solar'], fontsize= 'x-small')
plt.show()


#STACKED ANNUAL PRODUCTION IN S & N:

#Create multindex dataframe
arrays = []
aux = []

for p in instance.p:
    if p == 1:
        aux = [instance.p.at(1) for j in range(1, len(instance.p) - 1)]
        arrays.append(aux)
        aux=[]
    else:
        aux = [instance.p.at(p) for j in range(1, len(instance.p) - 1)]
        arrays[0].extend(aux)
for p in instance.p:
    for i in range(1,len(instance.s) + 1):
        if i == 1 and p == 1:
            aux = [instance.s.at(i) for j in range(1,len(instance.n) + 1)]
            arrays.append(aux)
        else:
            aux = [instance.s.at(i) for j in range(1,len(instance.n) + 1)]
            arrays[1].extend(aux)
for p in instance.p:
    for k in range(1, len(instance.s) + 1):
        aux = [instance.n.at(l) for l in range(1,len(instance.n) + 1)]
        if k == 1 and p == 1:
            arrays.append(aux)
        else:
            arrays[2].extend(aux)
```

```
#print(arrays)

index = pd.MultiIndex.from_tuples(list(zip(*arrays)))
#print(index)
#print(list(zip(*arrays)))

data_array= [10.0]
for p in instance.p:
    for s in instance.s:
        for n in instance.n:
            values1 = value(sum(instance.qt[t, n, s, p] for t in instance.t))
            data_array.append(values1)
            #print("values thermal ",values1, " at level ",n,"in week ",s)
for p in instance.p:
    for s in instance.s:
        for n in instance.n:
            values2 = value(sum(instance.qh[h, n, s, p] for h in instance.h))
            data_array.append(values2)
            #print("values hydraulic ",values2, " at level ",n,"in week ",s)
for p in instance.p:
    for s in instance.s:
        for n in instance.n:
            values3 = value(sum(instance.qe[eo, n, s, p] for eo in instance.eo))
            data_array.append(values3)
            #print("values wind ",values3, " at level ",n,"in week ",s)
for p in instance.p:
    for s in instance.s:
        for n in instance.n:
            values4 = value(sum(instance.qpv[pv, n, s, p] for pv in instance.pv))
            data_array.append(values4)
            #print("values hydraulic ",values2, " at level ",n,"in week ",s)

data_array.remove(10.0)
#print(data_array)
my_array = np.array(data_array)

resized_array = my_array.reshape(4,120)
#print(resized_array)

df = pd.DataFrame(data=resized_array ,columns= index)
#print(df)
#print(df.columns.levels[0])
#print(df.columns.levels[1])
#print(df.columns.levels[2])
#Plotting
fig, axes = plt.subplots(nrows=1, ncols=12, sharey= True, figsize= (14 / 2.54, 10 / 2.54))
for i, col in enumerate(df.columns.levels[0]):
    #print(col)
    #print(i)
```

```
    ax = axes[i]
    df[col].T.plot(ax=ax, kind='bar',stacked= True, width=.8,
color=['darksalmon','skyblue','mediumseagreen','khaki'])


    ax.legend_.remove()
    ax.set_xlabel(col, weight='bold')
    ax.yaxis.grid(visible=True, which='major', color='black', linestyle='--', alpha=.4)
    ax.set_axisbelow(True)

    for tick in ax.get_xticklabels():
        tick.set_rotation(0)

    #make the ticklines invisible
    ax.tick_params(axis=u'both', which=u'both', length=0)
#ax.set_title("Net power produced in January")
#plt.tight_layout()
# remove spacing in between
fig.subplots_adjust(wspace=0)  # space between plots
ax.legend(['Thermal', 'Hydraulic','Wind','Solar'], fontsize= 'x-small')
plt.show()


#MARGINAL COSTS OF THE SYSTEM

shadow_price = value(instance.dual[instance.Demand_Bal['n1','lab',1]])
print("Dual variable associated to marginal cost example:",shadow_price)


fig5, ax5 = plt.subplots()
x_values = range(1, len(instance.p) + 1)
for p in instance.p:
    [ax5.bar(p, value(sum((instance.dual[instance.Demand_Bal[n, s, p]]) for s in instance.s for n in
instance.n if n != 'n5' and s != 'fes')), color = 'darkseagreen')]
ax5.set_xticks(x_values)
ax5.set_xlabel("Month p")
ax5.set_ylabel("Marginal cost [euros/GW]")
ax5.set_title("Marginal cost of the system per month")
plt.show()

data_array= [10.0]
for s in instance.s:
    for n in instance.n:
        if n != 'n5' or s != 'fes':
            values3 = value(instance.dual[instance.Demand_Bal[n, s, 12]])
            data_array.append(values3)
        else:
            data_array.append(0.0)

data_array.remove(10.0)
my_array = np.array(data_array)
```

```
#print(my_array)

resized_array = my_array.reshape(1,10)

#Create multindex dataframe
arrays = []
aux = []
for i in range(1,len(instance.s) + 1):
    if i == 1:
        aux = [instance.s.at(i) for j in range(1,len(instance.n) + 1)]
        arrays.append(aux)
    else:
        aux = [instance.s.at(i) for j in range(1,len(instance.n) + 1)]
        arrays[i - 2].extend(aux)
for k in range(1, len(instance.s) + 1):
    aux = [instance.n.at(l) for l in range(1,len(instance.n) + 1)]
    if k == 1:
        arrays.append(aux)
    else:
        arrays[len(instance.s) - 1].extend(aux)

index = pd.MultiIndex.from_tuples(list(zip(*arrays)))

df = pd.DataFrame(data=resized_array ,columns= index)


fig, axes = plt.subplots(nrows=1, ncols=2, sharey=True, figsize=(14 / 2.54, 10 / 2.54))  # width, height
for i, col in enumerate(df.columns.levels[0]):
    ax = axes[i]
    df[col].T.plot(ax=ax, kind='bar', width=.8, color='darkseagreen')

    ax.legend_.remove()
    ax.set_xlabel(col, weight='bold')
    ax.yaxis.grid(visible=True, which='major', color='black', linestyle='--', alpha=.4)
    ax.set_axisbelow(True)

    for tick in ax.get_xticklabels():
        tick.set_rotation(0)
#make the ticklines invisible
ax.tick_params(axis=u'both', which=u'both', length=0)
plt.tight_layout()
# remove spacing in between
fig.subplots_adjust(wspace=0)  # space between plots

ax.set_title("Marginal cost in December")

plt.show()

#EVOLUTION OF THE RESERVES OF THE HYDRO PLANT

data_array= [10.0]
```

```
for h in instance.h:
    for p in instance.p:
        values3 = value(instance.w[h,p]) #+ value(instance.w0[h]) #No olvidarse del valor inicial
        data_array.append(values3)

data_array.remove(10.0)
my_array = np.array(data_array)
#print(my_array)

resized_array = my_array.reshape(1,36)

#Create multindex dataframe
arrays = []
aux = []

for k in range(1, len(instance.h) + 1):
    aux = [instance.h.at(k) for l in range(1,len(instance.p) + 1)]
    if k == 1:
        arrays.append(aux)
        #print(arrays)
    else:
        arrays[0].extend(aux)
for i in range(1,len(instance.h) + 1):
    if i == 1:
        aux = [instance.p.at(j) for j in range(1,len(instance.p) + 1)]
        arrays.append(aux)
        #print(arrays)
    else:
        #print(i)
        aux = [instance.p.at(j) for j in range(1,len(instance.p) + 1)]
        arrays[1].extend(aux)
        #print(arrays)

#print("final arrays",arrays)

index  = pd.MultiIndex.from_tuples(list(zip(*arrays)))

df = pd.DataFrame(data=resized_array,columns= index)


fig, axes = plt.subplots(nrows=1, ncols=3, sharey=True, figsize=(22 / 2.54, 10 / 2.54))  # width, height
for i, col in enumerate(df.columns.levels[0]):
    if i == 0:
        word = 'HYDRO_PUM'
    elif i == 1:
        word = 'HYDRO_RES'
    else:
        word = 'HYDRO_ROR'
    ax = axes[i]
    df[col].T.plot(ax=ax, kind='bar', width=.8, color='plum')
```

```
    ax.legend_.remove()
    ax.set_xlabel(col, weight='bold')
    ax.yaxis.grid(visible=True, which='major', color='black', linestyle='--', alpha=.4)
    ax.set_axisbelow(True)
    ax.axhline(instance.wmin[word], color='r', linestyle='--')
    ax.axhline(instance.wmax[word], color='g', linestyle='--')

    for tick in ax.get_xticklabels():
        tick.set_rotation(0)
#make the ticklines invisible
ax.tick_params(axis=u'both', which=u'both', length=0)
plt.tight_layout()
# remove spacing in between
fig.subplots_adjust(wspace=0)  # space between plots

ax.set_title("Anual evolution of the energy reserves of each hydro plant")
plt.tight_layout()
plt.show()

#EVOLUTION OF THE MARGINAL VALUE OF WATER

data_array= [10.0]
for h in instance.h:
    for p in instance.p:
        values3 = value(instance.dual[instance.E_Bal[h,p]])
        data_array.append(values3)

data_array.remove(10.0)
my_array = np.array(data_array)
#print(my_array)

resized_array = my_array.reshape(1,36)

#Create multindex dataframe
arrays = []
aux = []

for k in range(1, len(instance.h) + 1):
    aux = [instance.h.at(k) for l in range(1,len(instance.p) + 1)]
    if k == 1:
        arrays.append(aux)
        #print(arrays)
    else:
        arrays[0].extend(aux)
for i in range(1,len(instance.h) + 1):
    if i == 1:
        aux = [instance.p.at(j) for j in range(1,len(instance.p) + 1)]
        arrays.append(aux)
        #print(arrays)
    else:
        print(i)
```

```
        aux = [instance.p.at(j) for j in range(1,len(instance.p) + 1)]
        arrays[1].extend(aux)
        #print(arrays)

#print("final arrays",arrays)

index  = pd.MultiIndex.from_tuples(list(zip(*arrays)))

df = pd.DataFrame(data=resized_array,columns= index)


fig, axes = plt.subplots(nrows=1, ncols=3, sharey=True, figsize=(22 / 2.54, 10 / 2.54))  # width, height
for i, col in enumerate(df.columns.levels[0]):
    ax = axes[i]
    df[col].T.plot(ax=ax, kind='bar', width=.8, color='powderblue')

    ax.legend_.remove()
    ax.set_xlabel(col, weight='bold')
    ax.yaxis.grid(visible=True, which='major', color='black', linestyle='--', alpha=.4)
    ax.set_axisbelow(True)

    for tick in ax.get_xticklabels():
        tick.set_rotation(0)
#make the ticklines invisible
ax.tick_params(axis=u'both', which=u'both', length=0)
plt.tight_layout()
# remove spacing in between
fig.subplots_adjust(wspace=0)  # space between plots

ax.set_title("Anual evolution of the marginal value of water at each hydraulic plant")
plt.tight_layout()
plt.show()
```