MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

DESARROLLO DE UN MODELO DE UNIT COMMITMENT EN PYTHON-PYOMO PARA SU EJECUCIÓN MEDIANTE INTERFAZ WEB

Autor: Francisco Labora Gómez

Director: Javier García González

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título Desarrollo de un modelo de Unit Commitment en Python-Pyomo para su ejecución mediante interfaz web

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2022-2023 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.: Francisco Labora Gómez Fecha: 19/ 07/ 2023

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

25685421X JAVIER GARCÍA

Firmado digitalmente por 25685421X JAVIER GARCÍA
Fecha: 2023.07.19 15:47:22 +02'00'

Fdo.: Javier García González Fecha: 19/ 07/ 2023

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

DESARROLLO DE UN MODELO DE UNIT COMMITMENT EN PYTHON-PYOMO PARA SU EJECUCIÓN MEDIANTE INTERFAZ WEB

Autor: Francisco Labora Gómez

Director: Javier García González

Madrid

DESARROLLO DE UN MODELO DE UNIT COMMITMENT EN PYTHON-PYOMO PARA SU EJECUCIÓN MEDIANTE INTERFAZ WEB

Autor: Labora Gómez, Francisco
Director: García González, Javier.

RESUMEN DEL PROYECTO

El proyecto trata sobre el desarrollo de una herramienta de software ejecutable mediante interfaz web para resolver problemas de Unit Commitment. A lo largo de este documento se trata la formulación matemática del problema de optimización y la implementación del mismo en la librería Pyomo de Python. Además se explora la implementación del modelo en la nube para su ejecución mediante interfaz web y se realiza una demostración del uso de la herramienta aplicada a un modelo del sistema eléctrico español.

Palabras clave: Unit Commitment, MILP, Pyomo, Binder.

1 Introducción

El problema de Unit Commitment es uno de los pilares fundamentales para la operación eficiente de sistemas eléctricos. Consiste en realizar la asignación óptima de producción de energía eléctrica entre todos los generadores disponibles en un sistema pudiendo variar su estado de conexión. Dependiendo de la formulación concreta del problema el sistema puede estar sujeto a diferentes restricciones que modelan comportamientos del mismo. Es posible simplificar el modelo del sistema y modificar la formulación exacta para conseguir tiempos de computación menores sin perder precisión [1][2].

Dada la naturaleza discreta del problema al incorporar variables de tipo binario para modelar la conexión de los grupos generadores, el problema pertenece a la familia de la programación entera mixta o MILP (Mixed Integer Linear Programming) por sus siglas en inglés. Los problemas de este tipo destacan por la complejidad computacional de su resolución, ya que al incorporar variables enteras, se pierden propiedades matemáticas en el espacio de soluciones que dificultan su análisis[3].

Con el objetivo de hacer este problema de Unit Commitment más accesible, se ha buscado diseñar una herramienta que permita analizar una formulación del problema mediante interfaz web. Esta implementación en la nube hace posible para los usuarios analizar un problema de Unit Commitment sin necesidad de instalar ningún software, simplemente accediendo a la herramienta mediante un navegador web.

2 Definición del proyecto.

El proyecto tiene tres etapas: Creación de la herramienta, despliegue en la nube y demostración de la herramienta. Durante la primera etapa se diseñan e implementan el código y el proceso para la resolución del Unit Commitment de manera local. En la segunda se adapta el software existente y se añaden elementos nuevos a la herramienta para poder desplegarla en la nube. En la tercera y última se hace una demostración de las capacidades de la herramienta para el análisis de sistemas.

3 Descripción de la herramienta

Hay que distinguir dos herramientas diferentes: la versión local y la versión para la nube.

La versión local consta de un código de Python-Pyomo que contiene la formulación matemática, la importación de datos desde archivos CSV y la exportación y representación de los resultados. Además son necesarios archivos que contienen los datos sobre el modelo y un solver que resuelva la formulación planteada por Python.

En cuanto a la versión preparada para ejecución web hay dos variantes: una diseñada para la plataforma Colab de Google y otra diseñada para Binder. La versión para Colab consiste de un Jupyter Notebook con el código necesario para configurar una sesión en la plataforma además de una interfaz gráfica interactiva para manejar el software. La versión de Binder utiliza un repositorio de código público que contiene los archivos de configuración necesarios para crear un contenedor que permita ejecutar el código, además utiliza la misma interfaz que la versión para Colab, ya que ambas plataformas están basadas en Jupyter Notebook.

La interfaz diseñada junto a la herramienta de resolución del Unit Commitment permiten la interacción en tiempo real con el problema, su resolución y la selección de datos que se pretenden representar.

En primer lugar, accediendo a mybinder.org y creando un contenedor con la herramienta, esta puede ser ejecutada llevando al usuario al menú principal como se muestra en la figura 10.

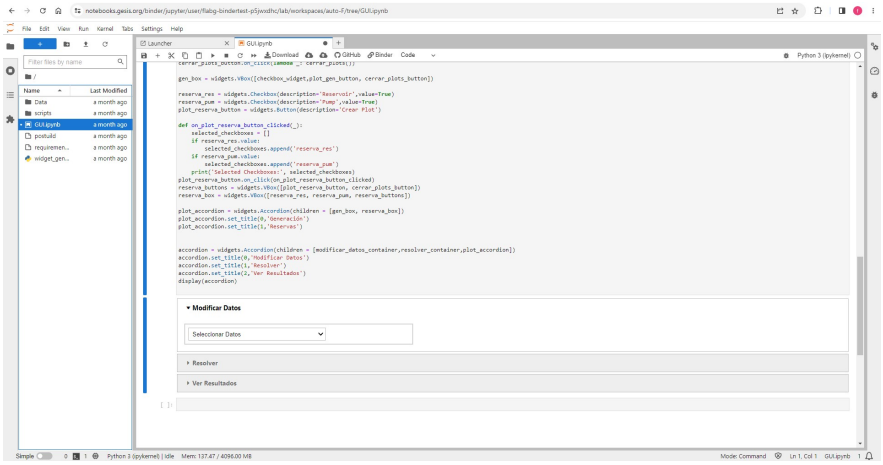


Figura 1. Menú principal de la interfaz

Utilizando la interfaz gráfica es posible interactuar con los datos modificándolos en tiempo real para así poder editar los parámetros del modelo. Esto se muestra en la figura 2.

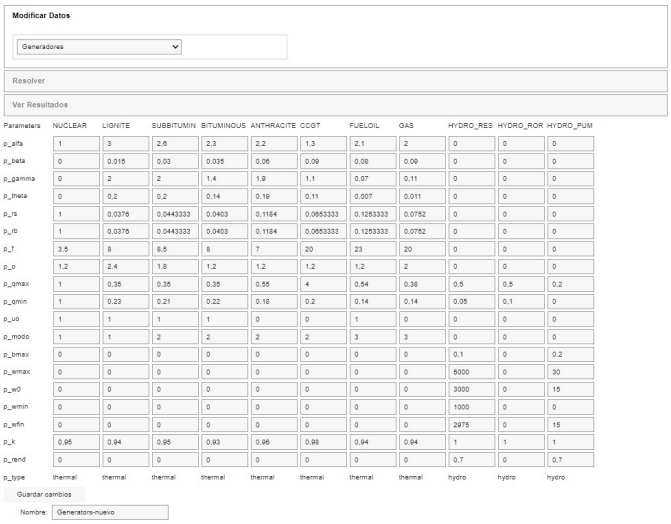


Figura 2. Menú de edición de generadores.

Una vez los datos han sido editados es posible ejecutar el código que crea la formulación matemática e invoca al solver para su resolución. El solver devuelve los datos de la solución a Python y desde la interfaz es posible personalizar la representación de los resultados tal y como se muestra en la figura 3.

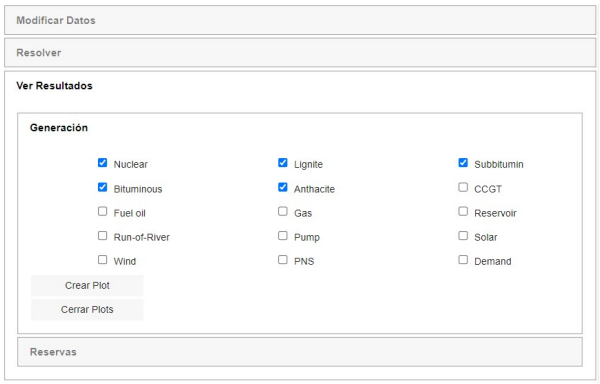


Figura 3. Menú de edición de representación.

Una vez se ha decidido que datos representar basta con utilizar el botón de la interfaz para que estos aparezcan en la gráfica correspondiente como se muestra en la figura 4

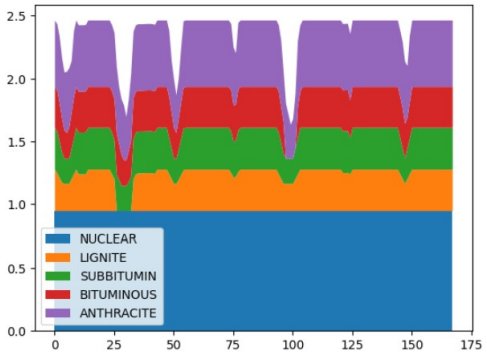


Figura 4. Representación de datos elegidos.

4 Resultados

Ha sido posible crear la herramienta para ejecución local y utilizarla para realizar un análisis que ejemplifica sus capacidades. En cuanto a la versión web, no se dispone de un software comercial compatible con la nube que tenga la capacidad para resolver el problema de optimización. Se ha recurrido a un software de código abierto que no es capaz de resolver el problema de optimización en un tiempo aceptable.

Para la verificación del funcionamiento del software se ha utilizado un modelo de referencia en GAMS con el que se ha comparado el modelo desarrollado. Estos modelos son equivalentes y dan lugar al mismo resultado, por tanto la formulación implementada durante el proyecto es correcta.

Además se ha creado un modelo basado basado en el sistema eléctrico español que permite estudiar el efecto de baterías, emisiones y restricciones adicionales a las planteadas en el modelo de referencia. Este segundo modelo se ha utilizado como demostración del potencial de la herramienta y permite comprobar efectos como la influencia de los derechos de emisiones sobre la producción basada en carbón o el aumento de aprovechamiento de energías renovables por medio de baterías. Una comparativa de la producción semanal desglosada por tecnología se muestra a continuación. En la figura5 se muestra un caso base en el que no se tienen en cuenta derechos de emisiones, en la figura 6 se muestra un caso en el que las emisiones está presentes.

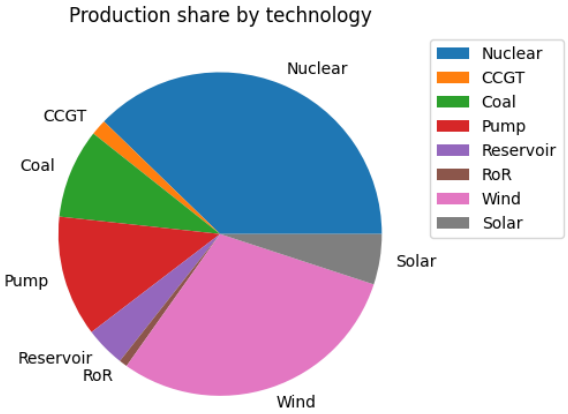


Figura 5. Generación por tecnología. Caso base.

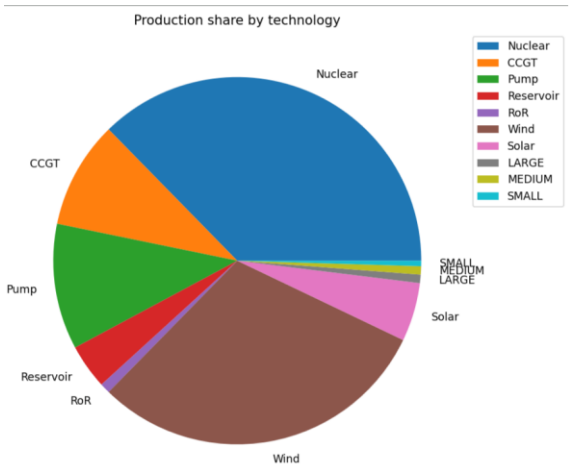


Figura 6. Generación por tecnología. Caso con emisiones.

Se aprecia como la inclusión de derechos de emisiones en el modelo elimina por completo la generación por medio de carbón que antes estaba presente. Esto se debe al bajo precio del carbón como combustible y a lo altamente contaminante que es.

Gracias a la herramienta desarrollada es posible realizar análisis sobre comportamientos del sistema como este utilizado a modo de demostración.

5 Conclusiones

- Python-Pyomo representa una opción muy atractiva para el análisis de modelos de optimización dada su versatilidad y compatibilidad con el ecosistema de Python.
- Binder no es una buena opción para la implementación de herramientas que requieran gran cantidad de recursos computacionales. No obstante ofrece muchas cualidades atractivas respecto a la capacidad de demostrar y comprobar el funcionamiento de aplicaciones de manera inmediata y remota
- El tiempo de resolución de problemas complejos como los de tipo MILP está fuertemente condicionado por el algoritmo y las heurísticas concretas que se usan para resolverlos, lo que vuelve solvers comerciales de alto rendimiento una necesidad a la hora de enfrentar problemas grandes.
- La implementación web permite una alta carga computacional sin la necesidad de realizar inversiones en infraestructura. Esto lo vuelve una muy buena opción para empresas que requieran solucionar problemas de gran tamaño.

  Debido a estos factores la implementación web resulta muy atractiva en ambientes empresariales para compañías que deban gestionar recursos energéticos de manera eficiente. La formulación es fácil de actualizar para adaptarse a la naturaleza dinámica del sector energético y sus avances, se alcanzan soluciones precisas en márgenes de tiempo aceptables y no es necesaria la inversión en infraestrucutra. A cambio si es necesaria la contratación de servicios de compuitación en la nube y de licencias corporativas para solvers de alto rendimiento.

  En el ámbito académico la ejecución por interfaz web, aunque resulte atractiva por su comodidad de uso y facilidad para el alumnado, es difícil de implementar sin recurrir a servicios de computación y licencias de tipo profesional.

Referencias

[1] Morales-Espana, Latorre, J. M., & Ramos, A. (2013). Tight and Compact MILP Formulation for the Thermal Unit Commitment Problem. IEEE Transactions on Power Systems, 28(4), 4897–4908.

[2] Li, T., & Shahidehpour, M. (2005). Price-Based Unit Commitment: A Case of Lagrangian Relaxation Versus Mixed Integer Programming. IEEE Transactions on Power Systems, 20(4), 2015–2025.

[3] Bixby, R., & Rothberg, E. (2007). Progress in computational mixed integer programming—A look back from the other side of the tipping point. Annals of Operations Research, 149(1)

DEVELOPMENT OF A UNIT COMMITMENT MODEL IN PYTHON-PYOMO FOR EXECUTION THROUGH A WEB INTERFACE

Author: Labora Gómez, Francisco
Supervisor: García González, Javier.

PROJECT SUMMARY

The project is about the development of a software tool, executable through a web interface, to solve Unit Commitment problems. Throughout this document, the mathematical formulation of the optimization problem and its implementation in the Pyomo library of Python are discussed. Furthermore, the implementation of the model in the cloud for its execution via a web interface is explored, and a demonstration of the tool's usage applied to a model of the Spanish electrical system is provided.

Keywords: Unit Commitment, MILP, Pyomo, Binder.

1 Introduction

The Unit Commitment problem is one of the fundamental pillars for the efficient operation of electrical systems. It involves making the optimal allocation of electricity production among all available generators in a system, considering the possibility of changing their connection status. Depending on the specific formulation of the problem, the system may be subject to different constraints that model its behavior. It is possible to simplify the system model and modify the exact formulation to achieve shorter computation times without sacrificing precision [1][2].

Due to the discrete nature of the problem, as it incorporates binary variables to model the generator groups' connection, the problem belongs to the family of Mixed Integer Linear Programming (MILP). Problems of this type are known for their computational complexity in solving them, as the inclusion of integer variables leads to the loss of mathematical properties in the solution space, making their analysis challenging[3].

To make this Unit Commitment problem more accessible, the aim has been to design a tool that allows analyzing a problem formulation through a web interface. This cloud-based implementation enables users to analyze a Unit Commitment problem without the need to install any software, simply by accessing the tool through a web browser.

2 Project definition

The project consists of three stages: Tool creation, cloud deployment, and tool demonstration. During the first stage, the code and process for solving the Unit Commitment problem locally are designed and implemented. In the second stage, the existing software is adapted, and new elements are added to enable the deployment of the tool in the cloud. In the third and final stage, a demonstration is conducted to showcase the capabilities of the tool for system analysis.

3 Tool description

There are two distinct tools to consider: the local version and the cloud version.

The local version comprises Python-Pyomo code that includes the mathematical formulation, importing data from CSV files, and exporting and representing the results. Additionally, data files containing the model information and a solver capable of solving the problem formulation are required.

Regarding the version prepared for web execution, there are two variants: one designed for the Google Colab platform and another designed for Binder. The Colab version consists of a Jupyter Notebook with the necessary code to set up a session in the platform, along with an interactive graphical interface to handle the software. The Binder version utilizes a public code repository containing the necessary configuration files to create a container for running the code. It also uses the same interface as the Colab version since both platforms are based on Jupyter Notebook.

The interface designed along with the Unit Commitment resolution tool allows real-time interaction with the problem, its resolution, and the selection of data to be represented.

Firstly, by accessing mybinder.org and creating a container with the tool, it can be executed, leading the user to the main menu as shown in Figure 1.

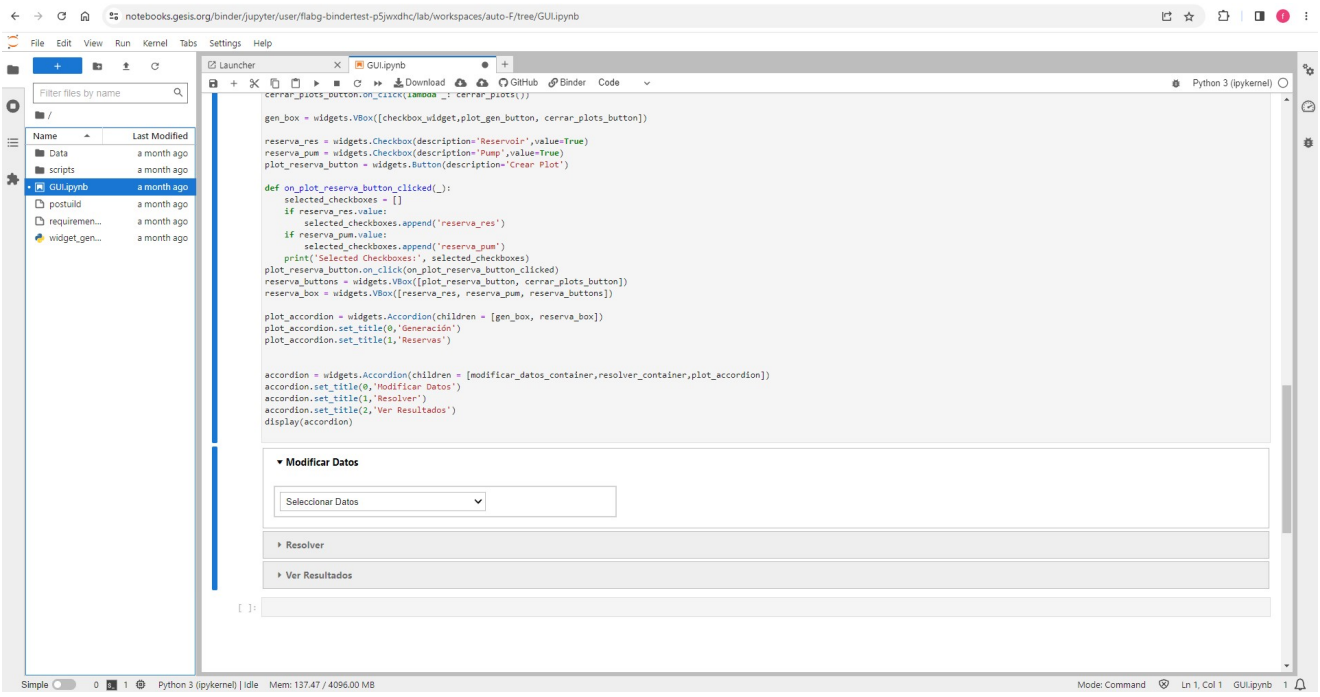
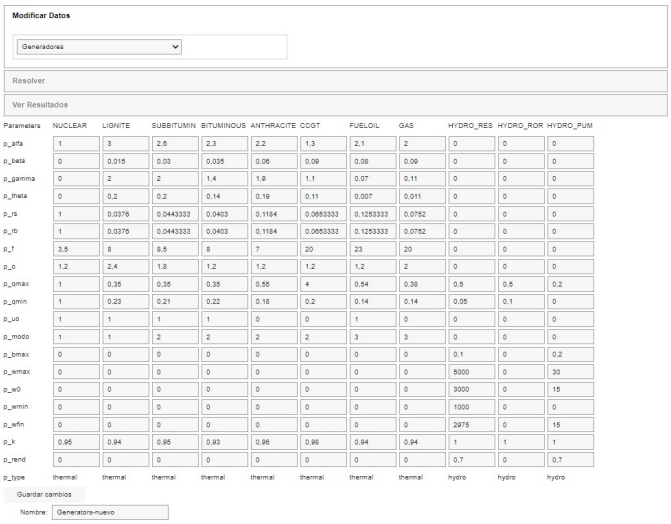


Figure 1. Interface main menu

Using the graphical interface, it is possible to interact with the data, modifying them in real-time to edit the model parameters. This is shown in Figure2.



Parameters	NUCLEAR	LIGNITE	SUBBITUMIN	BITUMINOUS	ANTHRACITE	CCGT	FUELOIL	GAS	HYDRO_RES	HYDRO_ROR	HYDRO_PUM
p_alfa	1	3	2,6	2,3	2,2	1,3	2,1	2	0	0	0
p_beta	0	0,015	0,03	0,035	0,06	0,09	0,08	0,09	0	0	0
p_gamma	0	2	2	1,4	1,9	1,1	0,07	0,11	0	0	0
p_theta	0	0,2	0,2	0,14	0,19	0,11	0,007	0,011	0	0	0
p_rs	1	0,0376	0,0443333	0,0403	0,1184	0,0653333	0,1253333	0,0752	0	0	0
p_rb	1	0,0376	0,0443333	0,0403	0,1184	0,0653333	0,1253333	0,0752	0	0	0
p_f	3,5	8	8,5	8	7	20	23	20	0	0	0
p_o	1,2	2,4	1,8	1,2	1,2	1,2	1,2	2	0	0	0
p_qmax	1	0,35	0,35	0,35	0,55	4	0,54	0,38	0,5	0,5	0,2
p_qmin	1	0,23	0,21	0,22	0,18	0,2	0,14	0,14	0,05	0,1	0
p_uo	1	1	1	1	0	0	1	0	0	0	0
p_modo	1	1	2	2	2	2	3	3	0	0	0
p_bmax	0	0	0	0	0	0	0	0	0,1	0	0,2
p_wmax	0	0	0	0	0	0	0	0	5000	0	30
p_w0	0	0	0	0	0	0	0	0	3000	0	15
p_wmin	0	0	0	0	0	0	0	0	1000	0	0
p_wfin	0	0	0	0	0	0	0	0	2975	0	15
p_k	0,95	0,94	0,95	0,93	0,96	0,98	0,94	0,94	1	1	1
p_rend	0	0	0	0	0	0	0	0	0,7	0	0,7
p_type	thermal	thermal	thermal	thermal	thermal	thermal	thermal	thermal	hydro	hydro	hydro

Figure 2. Generators data editing menu.

Once the data has been edited, it is possible to execute the code that creates the mathematical formulation and calls the solver for its resolution. The solver returns the solution data to Python, and from the interface, it is possible to customize the representation of the results, as shown in Figure 3.

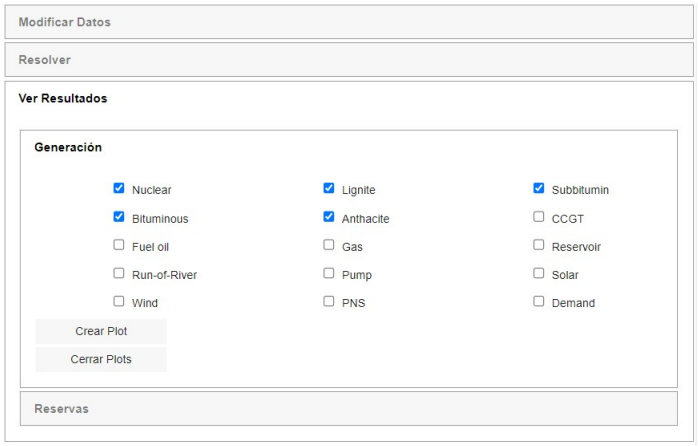


Figure 3. Plot editing menu.

Once the decision on which data to represent has been made, simply use the button in the interface to display them in the corresponding graph, as shown in Figure 4.

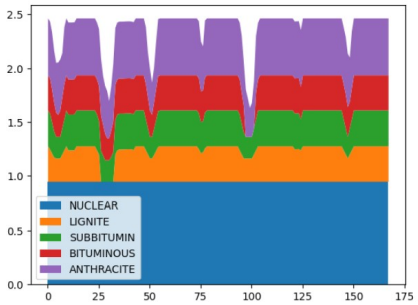


Figure 4. Plot of selected data.

4 Results

It has been possible to create the tool for local execution and use it to perform an analysis that exemplifies its capabilities. As for the web version, there is no commercially available cloud-compatible software with the capability to solve the optimization problem. An open-source software has been used, but it is not capable of solving the optimization problem within an acceptable time frame.

To verify the functioning of the software, a reference model in GAMS has been used and compared with the developed model. These models are equivalent and yield the same result, confirming that the formulation implemented during the project is correct.

Furthermore, a model based on the Spanish electrical system has been created, allowing the study of the effects of batteries, emissions, and additional constraints beyond those proposed in the reference model. This second model has been used as a demonstration of the tool's potential and enables the examination of effects such as the influence of emission rights on coal-based production or the increased utilization of renewable energies through batteries. A comparison of weekly production, broken down by technology, is shown below. Figure 5 depicts a base case where emissions are not considered, while Figure 6 shows a case where emissions are present.

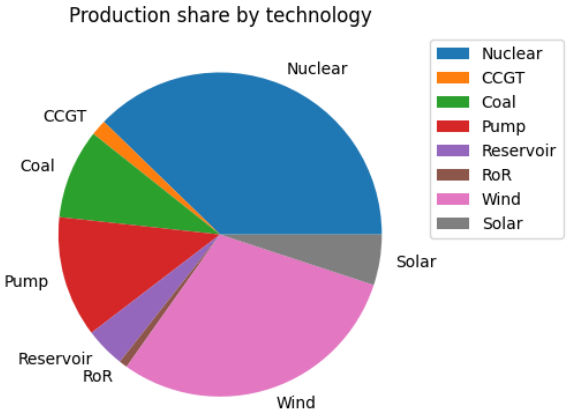


Figure 5. Generation by technology. Base model.

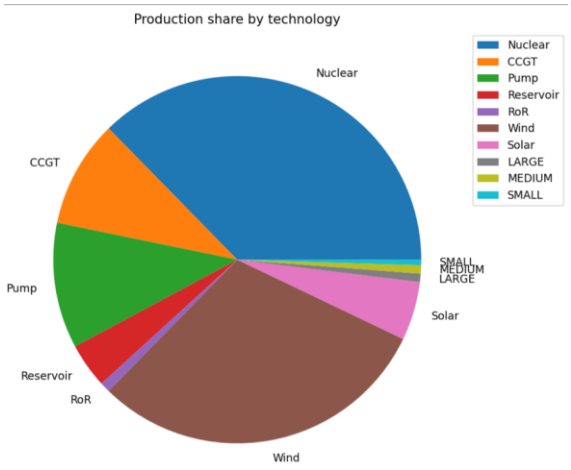


Figure 6. Generation by technology. Model with emissions.

It can be observed that the inclusion of emission rights in the model completely eliminates coal-based generation that was previously present. This is due to the low price of coal as a fuel and its high level of pollution.

Thanks to the developed tool, it is possible to conduct analyses on system behaviors like this one, used here as a demonstration.

5 Conclusions

- Python-Pyomo represents a highly attractive option for analyzing optimization models due to its versatility and compatibility with the Python ecosystem.
- Binder is not a suitable choice for implementing tools that require a large amount of computational resources. However, it offers many appealing qualities in terms of the ability to demonstrate and verify the functionality of applications immediately and remotely.
- The resolution time of complex problems such as MILP is strongly influenced by the algorithm and specific heuristics used to solve them, making high-performance commercial solvers a necessity when dealing with large problems.
- Web implementation allows for a high computational load without the need for investments in infrastructure. This makes it an excellent option for companies that need to solve large-scale problems.

Due to these factors, web implementation becomes highly attractive in corporate environments for companies that need to efficiently manage energy resources. The formulation is easy to update, adapting to the dynamic nature of the energy sector and its advancements, resulting in precise solutions within acceptable timeframes without the need for infrastructure investment. However, it does require the hiring of cloud computing services and corporate licenses for high-performance solvers.

In the academic field, web-based execution, while appealing for its user-friendliness and ease for students, is challenging to implement without resorting to professional computing services and licenses.

References

[1] Morales-Espana, Latorre, J. M., & Ramos, A. (2013). Tight and Compact MILP Formulation for the Thermal Unit Commitment Problem. IEEE Transactions on Power Systems, 28(4), 4897–4908.

[2] Li, T., & Shahidehpour, M. (2005). Price-Based Unit Commitment: A Case of Lagrangian Relaxation Versus Mixed Integer Programming. IEEE Transactions on Power Systems, 20(4), 2015–2025.

[3] Bixby, R., & Rothberg, E. (2007). Progress in computational mixed integer programming—A look back from the other side of the tipping point. Annals of Operations Research, 149(1)

Índice

Índice de Figuras

1 Introducción

1.1 Motivación

El problema del Unit Commitment [1] es un problema de gran relevancia en la planificación y operación de los sistemas eléctricos. El objetivo es determinar la combinación óptima de las unidades de generación que deben estar encendidas en cada hora para satisfacer la demanda eléctrica prevista en un horizonte de corto plazo respetando todas las restricciones técnicas y medioambientales que afectan al equipo de generación y del sistema. El criterio para tomar esa decisión suele ser minimizar el coste total de explotación, aunque es posible definir otras funciones objetivo como por ejemplo minimizar las emisiones totales de dióxido de carbono, maximizar el beneficio esperado por una empresa de generación que participa en el mercado eléctrico, etc. La importancia de este problema radica en que su correcta resolución garantiza un suministro eléctrico eficiente y fiable que minimiza el coste de producción y que aseguran el abastecimiento de la demanda en todo momento así como otros servicios complementarios. Además, el Unit Commitment juega un papel crucial en la integración óptima de fuentes de energía renovable no despachables, como por ejemplo la generación eólica o solar, contribuyendo así a la transición hacia sistemas eléctricos más sostenibles y respetuosos con el medio ambiente.

El Unit Commitment es uno de los problemas clásicos en la gestión de los sistemas de energía eléctrica y ha sido objeto de numerosos estudios e investigaciones para mejorar el tiempo de resolución requerido e intentando respetar una representación del sistema que sea lo más exacta posible. Por ello, la resolución del Unit Commitment es una de las piedras angulares en la gestión eficiente de cualquier sistema eléctrico y por tanto cualquier ingeniero con cierto grado de especialización en sistemas eléctricos lo estudia durante su etapa de formación.

Uno de los problemas a la hora de resolver problemas de optimización matemática como el Unit Commitment es que debido a su tamaño y complejidad, suele ser necesario el uso de ordenadores para resolverlos en un tiempo razonable. Esto obliga a que para resolverlos, sea necesario el acceso a cierta capacidad de computación y a instalar y configurar diversos paquetes de software.

Para evitar estos dos requerimientos previos y facilitar el planteamiento, resolución y análisis de problemas de Unit Commitment, se ha decidido en este proyecto desarrollar un modelo de Unit Commitment ejecutable mediante interfaz web utilizando Python-Pyomo.

1.2 Objetivos

El objetivo de este proyecto es crear una herramienta para resolver problemas de optimización que sea fácil de usar, intuitiva y no requiera la instalación y configuración de múltiples paquetes de software complejos. Para ello, se desarrollará la herramienta utilizando la librería Pyomo de Python que se utilizará para codificar el problema del Unit Commitment. Se ha elegido esta formulación concreta ya que el Unit Commitment se trata de uno de los problemas más relevantes, conocidos y estudiados en el ámbito de la operación de sistemas eléctricos y además, cuenta con ciertas propiedades como la existencia de variables binarias que lo vuelven computacionalmente interesante. Para poder realizar la implementación en la nube, es necesario diseñar primero una herramienta que funcione de manera local, para luego extenderla y configurarla de tal forma que sea posible ejecutarla mediante interfaz web. Es decir, se pretende que al usuario le baste disponer de una conexión a internet y una dirección URL para conectarse y configurar el caso

[1] Término en inglés que en español se conoce también como problema de *asignación de unidades* o *programación horaria*.

de estudio, lanzar la ejecución y poder acceder a los resultados sin necesidad de instalarse en el ordenador ningún tipo de paquete o software especializado.

La herramienta se diseñará en Python para así poder aprovechar la versatilidad de la librería Pyomo y su compatibilidad con todo el ecosistema de Python, especialmente con librerías de tipo científico y con las posibilidades de implementación web mediante Jupyter Notebooks. Por estos mismos motivos de compatibilidad, se han elegido Binder y Colab como plataformas para la implementación web. Cada una de ellas ofrece ventajas e inconvenientes únicos que las pueden hacer atractivas en diferentes circunstancias. Además de la herramienta en sí, serán necesarios la creación y el diseño de las diferentes configuraciones y procesos que permitan el despliegue de la herramienta en la nube

Por último y para comprobar la eficacia de la herramienta diseñada, se pretende realizar un pequeño caso de estudio que permita mostrar el potencial del software.

Se establecen por tanto tres metas a alcanzar a lo largo del proyecto:

- Desarrollo y creación del software para su ejecución local.
- Despliegue y ejecución mediante interfaz web.
- Elaboración y análisis de casos de estudio que ilustren las capacidades de la herramienta desarrollada.

1.3 Metodología

Debido a la naturaleza del proyecto, este tiene tres partes claramente diferenciadas con su correspondiente metodología específica.

En primer lugar para diseñar la formulación del problema de Unit Commitment, se desarrollará una serie de códigos que permitan el planteamiento y resolución del problema de optimización de manera local. La validez de este modelo y los resultados obtenidos se verificará utilizando un modelo de referencia escrito en GAMS.

Una vez obtenidos resultados satisfactorios en la primera parte del proyecto, se procederá al despliegue de la aplicación en servicios de computación en la nube para facilitar su accesibilidad y uso. Además se añadirán y programarán las diferentes configuraciones e interfaces requeridas.

Por último se creará un caso de estudio basado en una versión simplificada del sistema eléctrico español para medir la efectividad del software diseñado como herramienta de análisis.

2 Estado del arte

El estado del arte es una sección esencial de cualquier trabajo de investigación, ya que proporciona una visión general y actualizada de los avances más relevantes en el campo de estudio. En este apartado, se explorará el estado actual del desarrollo de modelos de Unit Commitment , centrándonos en la implementación en Python utilizando la biblioteca Pyomo y la innovadora posibilidad de ejecución a través de la nube.

Python y Pyomo proporcionan herramientas versátiles y poderosas para el modelado y la solución de problemas de optimización. La elección de Pyomo como framework para desarrollar modelos de Unit Commitment permite una implementación que aproveche las capacidades del lenguaje Python y sus otras librerías. Aportando flexibilidad mediante la capacidad de combinar la resolución del problema de Unit Commitment con tratamiento de datos, representación de resultados, interfaces gráficas, etc.

La verdadera innovación radica en la posibilidad de ejecutar estos modelos a través de una interfaz web. Esto brinda a los usuarios la conveniencia de interactuar con los modelos de Unit Commitment de manera accesible y amigable, sin la necesidad de conocimientos especializados en programación o en el campo de la planificación energética. La ejecución web permite realizar análisis en tiempo real, realizar ajustes y explorar diferentes escenarios de manera interactiva. Esto es especialmente atractivo en ambientes académicos en los que se pretende formar a alumnos en el uso y resolución tanto de problemas de Unit Commitment como de optimización en general.

En este apartado, exploraremos los enfoques y las técnicas más relevantes utilizadas en el desarrollo de modelos de Unit Commitment en Pyomo. Discutiremos las características clave de Pyomo y cómo se integra en el modelado y la solución de problemas de Unit Commitment. Además, destacaremos las ventajas y los desafíos de la implementación a través de una interfaz web, y examinaremos ejemplos y aplicaciones prácticas que demuestran su utilidad en la planificación energética.

2.1 Despacho económico

El despacho económico o Economic Dispatch es un proceso que busca asignar la generación de los grupos disponibles en el sistema de manera óptima para minimizar el coste total de operación. En su formulación inicial, el despacho económico puede ser abordado utilizando métodos de optimización sencillos cuando se consideran restricciones básicas.

Sin embargo, gracias a décadas de investigación académica en este campo, se ha avanzado en la incorporación de restricciones más complejas en los modelos de despacho económico. Estas restricciones pueden abarcar aspectos como los efectos de la red eléctrica y el comportamiento estocástico de parámetros sujetos a incertidumbre. Es tal la relevancia del modelo de despacho económico en la gestión del sistema eléctrico que casos complejos como estos ya empezaban a estudiarse en las décadas de 1970 y 1980 [1].

El modelo de despacho económico es la formulación más básica de problemas de operación a corto plazo de sistemas eléctricos y su ampliación permitiendo la conexión y desconexión de grupos da lugar al problema de Unit Commitment, que es el objeto de estudio de este documento.

2.2 Unit Commitment

Como ya se ha indicado, el Unit Commitment amplía el Economic Dispatch con la posibilidad de modificar el estado de conexión de los grupos generadores, esto incorpora variables de tipo binario para modelar esta nueva capacidad del sistema. Añadir dichas variables binarias para modelar el estado, la conexión y la desconexión de grupos supone la incapacidad de utilizar

algoritmos de programación lineal clásicos como el método Simplex o algoritmos no lineales como el método de multiplicadores de Lagrange, ya que este tipo de métodos solo consideran variables de tipo continuo. Para poder resolver el Unit Commitment se ha de recurrir a métodos como el Branch and Bound, que plantea un problema de optimización de variables continuas para cada combinación de variables discretas y elige la solución óptima de entre las posibles. Métodos así, que necesitan trabajar con una inmensa cantidad de diferentes posibilidades debido a la combinatoria que introducen las variables discretas, utilizan técnicas para eliminar parte del problema y no tener que explorar todas las posibles soluciones. Por ejemplo, al utilizar Branch and Bound se comienza resolviendo un problema "relajado" en el que las variables discretas se vuelven continuas y se impone que sean discretas gradualmente mientras se eliminan ciertas soluciones que se comprueba que no pueden ser la óptima.

Al igual que con los problemas de despacho, en los problemas de Unit Commitment es posible añadir restricciones más allá de las formulaciones básicas de ajuste de demanda. Permitiendo así modelar comportamientos de los sistemas estudiados de manera más exacta. Durante el desarrollo de este proyecto se ha realizado el análisis de un problema de Unit Commitment con coordinación hidrotérmica y fuentes de energía renovables.

2.3 Coordinación hidrotérmica

La coordinación hidrotérmica es un proceso fundamental en la operación de sistemas eléctricos que involucra la optimización conjunta de la generación hidráulica y térmica para satisfacer la demanda de energía eléctrica de la manera más eficiente posible.

En este contexto, la generación hidráulica se refiere a la producción de energía eléctrica a partir de la energía potencial del agua en embalses y ríos, mientras que la generación térmica se basa en la conversión de calor a energía eléctrica mediante el accionado de turbinas con fluidos de trabajo.

En el marco del Unit Commitment la coordinación hidrotérmica permite utilizar un recurso finito como el agua para explotar sus propiedades energéticas aportando al sistema eléctrico robustez y flexibilidad. La generación hidroeléctrica suministra al sistema una energía limitada pero virtualmente gratuita que puede ser utilizada en los momentos de mayor demanda para reducir notablemente el coste de operación del sistema.

Además, dada la naturaleza volátil de la energía eléctrica, que no puede ser almacenada a gran escala de manera económica y eficiente, la generación hidroeléctrica plantea un método de almacenamiento de energía en forma de agua gracias a los embalses y las centrales con capacidad de bombeo. Esto permite suavizar las curvas de demanda, que presentan de manera natural una configuración de picos y valles, almacenando agua en momentos donde la demanda es baja y la producción de energía se vuelve muy barata y utilizándola en los picos de demanda. Esto permite que grupos generadores más eficientes y baratos de operar puedan proporcionar electricidad a una mayor proporción de la demanda, abaratando los costes del sistema.

La disponibilidad del agua como recurso, al igual que la disponibilidad de otras fuentes renovables de energía sobre las que no existe un control directo, es y ha sido motivo de estudio en ambientes académicos y empresariales durante décadas.

2.4 Métodos de Resolución

Una vez el problema de Unit Commitment ha sido planteado, debe formularse de tal forma que pueda ser resuelto por algún algoritmo de optimización. Dadas las dimensiones de este tipo de problemas y la inmensa cantidad de parámetros y variables asociados, en cualquier caso práctico es necesario recurrir a métodos basados en computación.

Existen varios métodos efectivos para resolver problemas de Unit Commitment como relajación Lagrangiana [8] o algoritmos genéticos [7] y no es raro encontrar en la literatura estudios centrados en determinar su efectividad y eficiencia [3][9][4]. En las últimas décadas con los avances en capacidad de procesamiento y la disponibilidad de paquetes de software comercial especializados, los algoritmos basados en programación lineal entera mixta o MILP por sus siglas en inglés (Mixed Integer Linear Programming) se han posicionado como uno de los métodos predilectos para la resolución de problemas de Unit Commitment [9]. Desarrollándose formulaciones específicas para ser resueltas de manera rápida y efectiva por este tipo de algoritmos desde hace años[2].

Este tipo de algoritmos busca mediante diferente métodos heurísticos la exploración del espacio de posibles soluciones para encontrar la solución óptima, o en algunos casos, una muy cercana a la óptima. Existen empresas dedicadas a la investigación y desarrollo de algoritmos de resolución de problemas de optimización además de existir algunos algoritmos públicos de código abierto. Un software que contiene y ejecuta estos algoritmos para encontrar la solución a problemas de optimización se conoce como solver.

2.5 Solvers

Como ya se ha explicado, un solver es un software utilizado para resolver problemas de optimización matemática. Los solvers normalmente son capaces de resolver diferentes tipos de problemas como MILP o programación cuadrática (QP) y se comunican mediante interfaces de programación (API) a algún otro software que les comunica la formulación del modelo concreto a resolver.

Existen multitud de solvers, tanto comerciales (Gurobi, CPLEX, MOSEK,etc) como de código abierto (GLPK, CBC, SCIP, etc...). Cada uno de estos solver admite diferentes tipos de formulaciones y utiliza diferentes algoritmos y heurísticas para encontrar la solución de los problemas de optimización. En este proyecto se han utilizado los solvers Gurobi y GLPK.

Gurobi es un solver comercial desarrollado por Gurobi Optimization LLC. pero cuenta con licencias gratuitas tanto para uso académico como para uso académico en la nube. Es además un solver rápido y eficiente compatible con Python-Pyomo que ha sido el método utilizado para crear la formulación del problema.

GLPK es un solver de código abierto desarrollado por el proyecto GNU, uno de los mayores referentes en software de código abierto del mundo. Capaz de resolver problemas de MILP y compatible con Python-Pyomo.

A la hora de utilizar un solver, es necesario comunicarle a través de su API la formulación concreta del problema en cuestión. Para ello existen diferentes métodos, tanto a través de lenguajes específicos como de librerías en lenguajes de propósito general.

2.6 Formulación matemática del problema

Es necesario plasmar el modelo que se desea analizar de manera metódica y unívoca para así poder comunicarle sus características a un solver y que este pueda resolver el problema de optimización. Igual que en el caso de los solvers, existen múltiples opciones para definir el modelo, tanto de código abierto como comerciales. Se destacan dos de especial relevancia para este proyecto: GAMS y Pyomo.

GAMS (General Algebraic Modeling System) es un lenguaje de modelado y un sistema de software utilizado para la formulación y resolución de problemas de optimización matemática. GAMS es ampliamente utilizado en investigación operativa, planificación de la producción, economía y otros campos donde se requiere tomar decisiones óptimas.

GAMS proporciona una forma flexible y eficiente de describir problemas de optimización en términos algebraicos. Además del lenguaje de modelado, GAMS también incluye un motor de resolución que utiliza solvers compatibles para encontrar la solución óptima.

El lenguaje de modelado de GAMS utiliza una sintaxis similar a la notación matemática estándar, por lo que resulta fácil de aprender una vez se tienen conocimientos sobre optimización matemática. Permite describir problemas de programación lineal, programación entera mixta, programación no lineal y programación cuadrática entre otros. GAMS ofrece la capcidad de modelar y resolver problemas complejos de manera concisa y eficiente.

Pyomo es una librería del lenguaje Python creada de manera específica para la resolución de problemas de optimización. Permite crear, mediante las reglas sintácticas de Python, modelos de optimización complejos y resolverlos mediante la utilización de un solver externo.

Una de las principales ventajas de Pyomo es su integración con Python, un lenguaje de programación de alta popularidad y potencia. Esto significa que los usuarios pueden aprovechar todas las capacidades de Python para el preprocesamiento de datos, la manipulación de modelos y la visualización de resultados. Pyomo también es compatible con otras bibliotecas científicas de Python, lo que permite una integración perfecta con herramientas y técnicas adicionales para análisis y optimización.

Esta integración con Python y su compatibilidad con otras librerías ha llevado a que Pyomo sea el lenguaje de modelado elegido para este proyecto. GAMS ha sido utilizado mediante un modelo preexistente para verificar el correcto funcionamiento del modelo desarrollado en Python-Pyomo.

Pyomo, al formar parte del ecosistema de Python cuenta con una característica muy importante: su fácil integración con métodos de computación en la nube o Cloud Computing.

2.7 Cloud Computing

El término cloud computing se refiere al modelo de entrega de servicios de computación a través de Internet. En lugar de ejecutar aplicaciones o almacenar datos en servidores locales o en dispositivos individuales, el cloud computing permite acceder a recursos informáticos compartidos, como servidores, almacenamiento, bases de datos y software, a través de la red.

Los proveedores de servicios en la nube mantienen y administran la infraestructura y los recursos informáticos, mientras que los usuarios pueden acceder a ellos bajo demanda y pagar solo por el uso que hacen de estos recursos, en lugar de invertir en infraestructura propia.

Existen además modelos de cloud computing completamente gratuitos basados en el concepto de servicio freemium, en el cual se ofrece a los usuarios una versión limitada del producto de manera gratuita y ofreciendo versiones ampliadas en modelos de suscripción o pago único. Un ejemplo de esto sería Colab, de la empresa Google, donde todos los usuarios tienen acceso a recursos de computación en la nube permitiendo ejecutar código en un entorno interactivo.

Los recursos de cloud computing relacionados con Python suelen estar basados en el Jupyter Notebooks, que se tratan de un formato especial de representación de código en Python para crear un entorno de desarrollo interactivo en el que se pueden incorporar texto, imágenes, tablas, etc.

Entre estos proveedores de servicios de computación en la nube basados en Jupyter Notebook destaca el proyecto Binder, que permite la creación de entornos de Jupyter Notebook construidos a partir de un repositorio web.

2.8 Binder Project

El Binder Project, como otros muchos proyectos de código abierto en internet nace desde el deseo de otorgar al público general el acceso a recursos informáticos accesibles y fáciles de utilziar. El objetivo del proyecto es crear un software capaz de crear entornos de ejecución a través de repositorios de código públicos[10].

Cualquier usuario puede conectarse a MyBinder.org y simplemente introduciendo la URL de un repositorio público es posible crear una versión ejecutable del código en dicho repositorio. Esto permite compartir y probar código de manera rápida y cómoda únicamente a través de un navegador web. Se pueden incluir diferentes archivos de configuración en el repositorio para poder modificar la forma en la que se crea el entorno, incluyendo librerías o paquetes de software externos.

Este entorno es creado de manera automática y alojado en los servidores de alguna de las instituciones y organizaciones que ceden sus recursos al Proyecto. Para generar el entorno de ejecución se utiliza un formato estandarizado conocido como contenedor. Estos contenedores son una forma estandarizada de asignar los recursos de un servidor a diferentes aplicaciones independientes, se crean y controlan mediante paquetes de software especializados que actúan como sistema operativo del contenedor e interfaz con el servidor. Uno de los softwares más conocidos para este propósito es Docker. Los contenedores sustituyen a la partición en máquinas virtuales independientes, permitiendo utilizar un sistema operativo común, ahorrando espacio y mejorando los tiempos de inicio del servidor. En la figura 1 se muestra una comparativa entre un servidor basado en máquinas virtuales y uno basado en contenedores controlados por Docker.

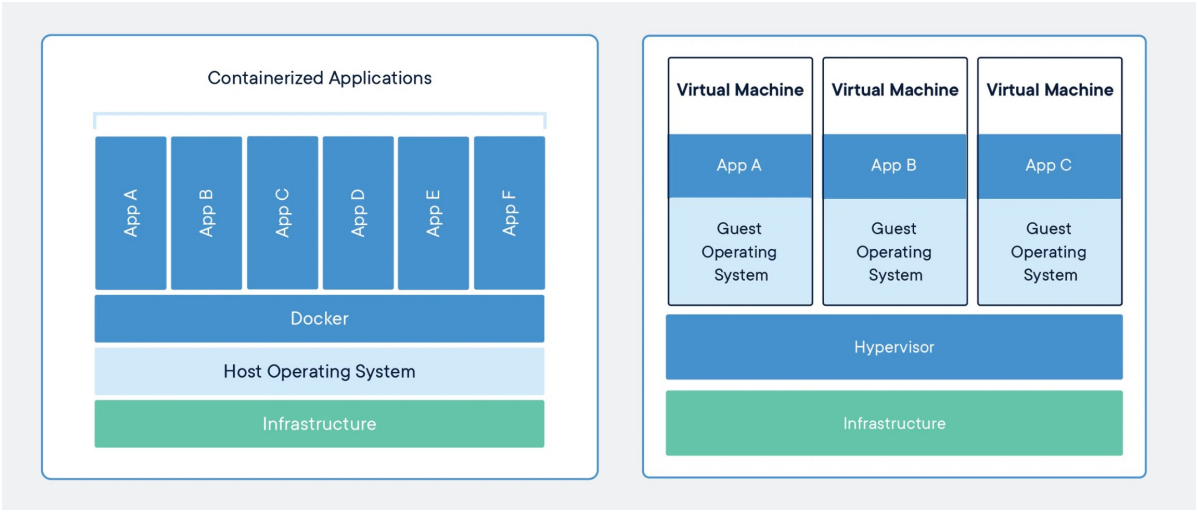


Figura 1. Estructura de un servidor Docker vs. VM. Fuente: Docker.org

3 Modelo Desarrollado

Para poder resolver el problema de Unit Commitment se ha desarrollado un problema de optimización lineal con variables binarias para representar el estado de conexión de los grupos a lo largo del tiempo y la potencia generada por cada fuente. El modelo cuenta con sets para representar generadores, periodos temporales y baterías, parámetros para caracterizar su funcionamiento, restricciones que deben ser cumplidas y una función de costes que se busca minimizar. Los diferentes elementos del modelo junto a las hipótesis asumidas a la hora de elaborarlo se detallan a lo largo de esta sección.

3.1 Hipótesis del modelo

En la elaboración del modelo se han asumido ciertas hipótesis para reducir la complejidad del mismo. Estas hipótesis permiten analizar una versión simplificada de un sistema eléctrico de manera efectiva y sin perder excesiva precisión, posibilitando utilizar ciertas técnicas de optimización que no serían válidas si se considerasen dinámicas no simplificadas e ideales del problema.

- Nudo único: Se excluyen del modelo las redes de transporte y distribución. Esto permite analizar el sistema como un único nodo al que están conectados toda la generación y la demanda, despreciando las pérdidas, restricciones y efectos que impone la red (máximo flujo por líneas, control de tensiones,etc).
- Modos de operación: Se impone mediante una restricción el modo de operación de ciertos generadores para emular de manera más fiel comportamientos en sistemas reales.
- Representación discreta del tiempo: Se expresa el tiempo como periodos discretos de una hora en vez de como un continuo, esto lleva a la representación de ciertos parámetros y variables tanto como valores medios como por valores instantáneos al finalizar un periodo determinado.
- Linearización de parámetros: Para obtener un problema lineal es necesario aproximar ciertos comportamientos por equivalentes lineales. Se busca así simplificar la resolución sin comprometer excesivamente la calidad de la solución.
- Enfoque determinista: En el modelo no se consideran múltiples escenarios ni la influencia de factores aleatorios. Esto da lugar a una solución única y determinada para cada caso de estudio concreto.
- Representación idealizada y simplificada de los elementos generadores de potencia. Esta representación será explicada con detalle más adelante y desglosada por tipo de generación.

3.2 Sets

Los sets de un problema de optimización son conjuntos de elementos relacionados gracias a los cuales es posible organizar de manera sistemática parámetros, variables y restricciones. Como ya se ha enunciado en la introducción a esta sección, los sets que se han utilizado para modelar el comportamiento del sistema eléctrico son:

1. Generadores: El conjunto de unidades generadoras de potencia eléctrica tanto térmicas como hidroeléctricas disponibles para satisfacer la demanda del sistema. Está a su vez dividido en estos dos subconjuntos térmico e hidroeléctrico.

2. Periodos: Representa los 168 periodos de una hora que hay en la semana analizada en el modelo.

3. Baterías: Representa la existencia de baterías conectadas al sistema.

En las siguientes subsecciones se detalla la naturaleza y particularidades de cada uno de estos sets y los subconjuntos que los componen.

Generadores térmicos

Las centrales térmicas son aquellas que generan energía eléctrica mediante la utilización de calor. En estas centrales se hace girar una turbina mediante la utilización de un gas, típicamente vapor de agua o aire, que ha sido calentado por una fuente de energía térmica, la energía cinética de rotación de estas turbinas puede ser convertida en energía eléctrica mediante un alternador. Las fuentes más habituales de energía térmica son combustibles fósiles que contienen energía química, la cual se libera como calor mediante la combustión (carbón, fueloil, gas natural,etc). En centrales que utilizan estos combustibles se utilizan cámaras de combustión para extraer la energía térmica que luego se transmite a un fluido de trabajo para posteriormente ser introducido a las turbinas y extraer energía eléctrica. El esquema de una central de gas natural de ciclo combinado o CCGT (Combined Cycle Gas Turbine) se muestra en la figura 2.

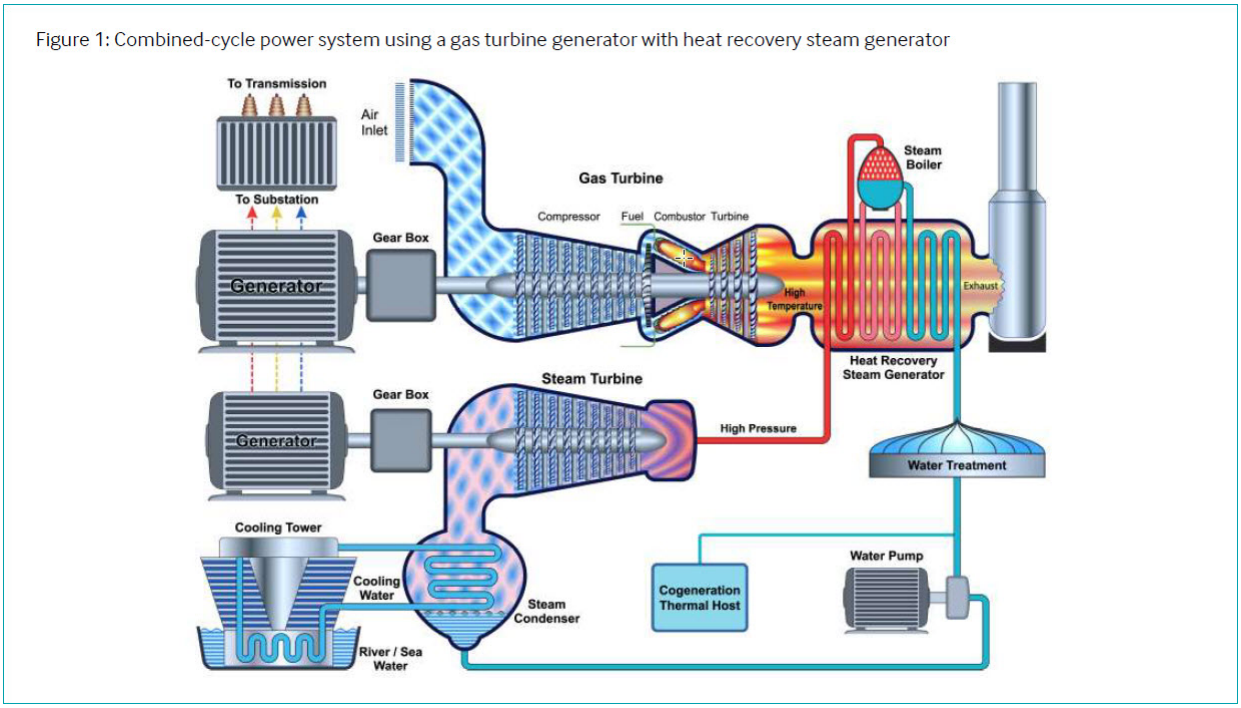


Figura 2. Ciclo combinado de gas natural. Fuente: Ipieca.

Además, existen otro tipo de centrales térmicas que utilizan un combustible y un mecanismo de extracción de calor diferentes: las centrales nucleares. En este tipo de centrales se aprovecha la inestabilidad del núcleo atómico de ciertos elementos pesados, principalmente el uranio, aunque el uso de otros elementos como combustible es fruto de estudio. Estos elementos tienen la capacidad de dividir su núcleo atómico bajo ciertas condiciones, obteniendo como producto dos elementos nuevos, partículas subatómicas y una gran cantidad de energía. En este tipo de centrales se sustituye la cámara de combustión por un reactor en el que se fisionan los átomos de combustible obteniendo inmensas cantidades de calor, el cual es extraído mediante

un refrigerante, habitualmente agua, y utilizado para generar energía eléctrica. Las centrales nucleares tienen la peculiaridad de tener poca flexibilidad en cuanto a la cantidad de calor se puede generar en el reactor y además tienen un coste variable muy bajo, por lo que en la práctica funcionan de manera continua a su capacidad nominal.

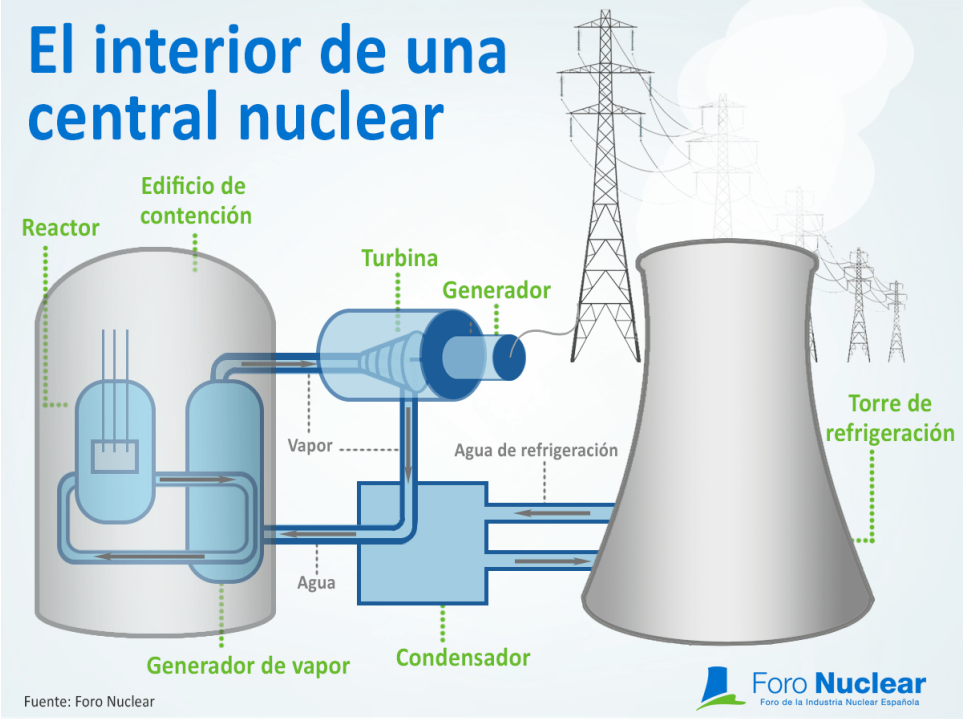


Figura 3. Esquema de una central nuclear. Fuente: Foro Nuclear.

Existen además otros tipos de centrales térmicas no basadas en combustibles fósiles , como centrales solares, de biomasa o geotérmicas, este tipo de centrales no han sido consideradas en este proyecto y no forman parte de los casos de estudio planteados.

Hipótesis asumidas

Para el modelado de las centrales térmicas se han hecho las siguientes aproximaciones:

- Se han considerado un consumo de combustible lineal, caracterizado por una pendiente α y un consumo a mínima carga β.
- Se ha considerado un consumo de combustible constante en el arranque, independientemente del tiempo que llevase el grupo desconectado y por tanto de la temperatura en la caldera.
- Se ha considerado como un factor constante la cantidad de energía destinada a servicios auxiliares dentro de las centrales.
- Se han establecido modos de funcionamiento para los generadores, imponiendo un estado de conexión concreto en función de dicho modo. Estos posibles modos son: funcionamiento normal, must-run y no disponible.

Generadores hidroeléctricos

Las centrales hidroeléctricas funcionan aprovechando la energía cinética de un flujo de agua, este es canalizado a través de una turbina haciéndola girar y permitiendo la extracción de energía eléctrica mediante un alternador. Este flujo de agua puede provenir de un embalse o de un río, teniendo cada una de estas dos fuentes un funcionamiento y características diferentes.

Ríos

Las centrales hidroeléctricas colocadas directamente sobre un río se conocen como running-of-river, en ellas no existe ningún tipo de control sobre el flujo de agua y por tanto sobre la cantidad de potencia disponible en cada momento. Simplemente se aprovecha el agua del río en cada momento en la medida de lo posible.

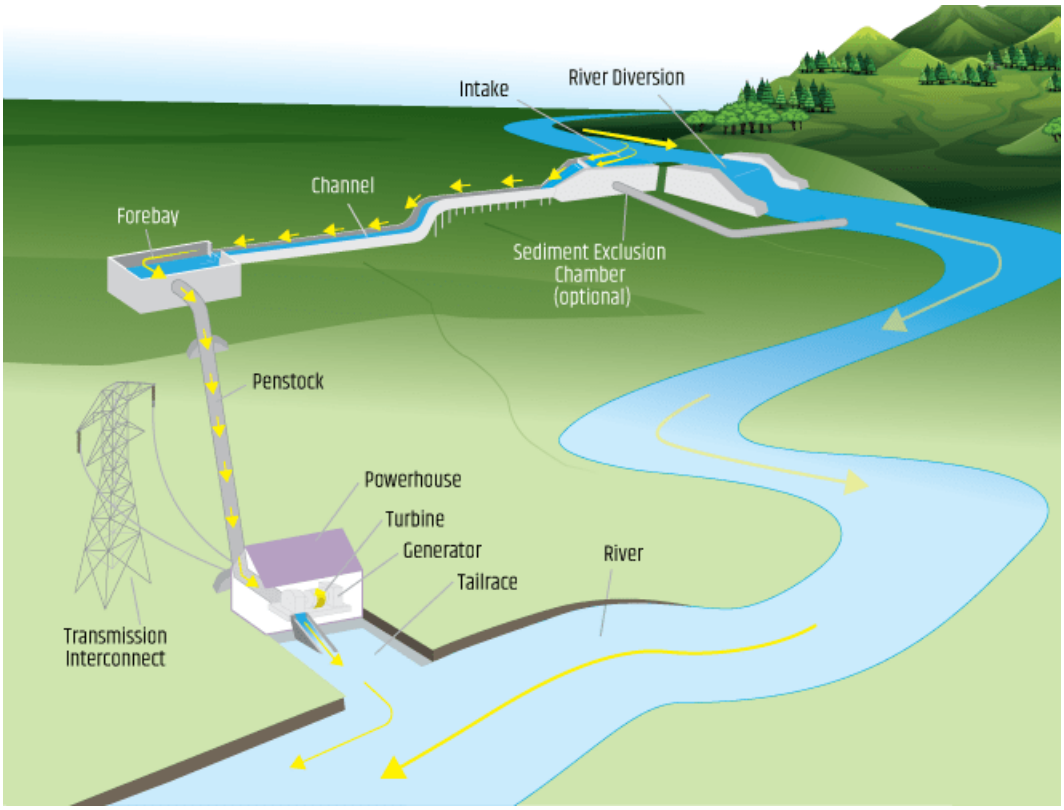


Figura 4. Central de tipo RoR. Fuente: USA Department of Energy.

Embalses

En el caso del agua proveniente de embalses, esta puede ser almacenada y utilizada para extraer energía en cualquier momento. Simplemente se acumula el agua en el embalse mediante una presa y se libera en el momento en el que se considere oportuno. No obstante, los embalses están sujetos a ciertas limitaciones operativas, tanto físicas como legales. Los embalses tienen unas capacidades máximas y mínimas que pueden estar marcadas tanto por límites técnicos como por directrices legales. Por ejemplo existe una cantidad máxima de agua que se puede acumular, por encima de la cual se pone en riesgo la seguridad de la presa y una capacidad mínima, que puede estar marcada por diversos factores, como la altura de las compuertas por las que se canaliza el agua o los diversos planes hidrológicos que rigen como debe utilizarse el agua como recurso no solo eléctrico sino para el consumo humano. Este tipo de restricciones son de vital importancia para minimizar los efectos económicos y ambientales de la generación hidroeléctrica[12][13].

Estos planes pueden imponer además otras restricciones como por ejemplo caudales mínimos para controlar el flujo de agua en una presa ya sea por motivos de consumo humano, regadío, eventos estacionales, etc.

El agua disponible en los embalses es un recurso renovable pero limitado por lo que debe intentar explotarse de la manera más eficiente posible. Además, por la propia naturaleza geográfica de ríos y embalses, el agua disponible en diferentes centrales hidroeléctricas variará de manera no independiente, ya que en muchos casos los embalses se encuentran relacionados geográficamente formando cuencas. Estas características especiales suponen la necesidad de hacer una serie de simplificaciones para poder modelar de manera sencilla la generación hidroeléctrica.

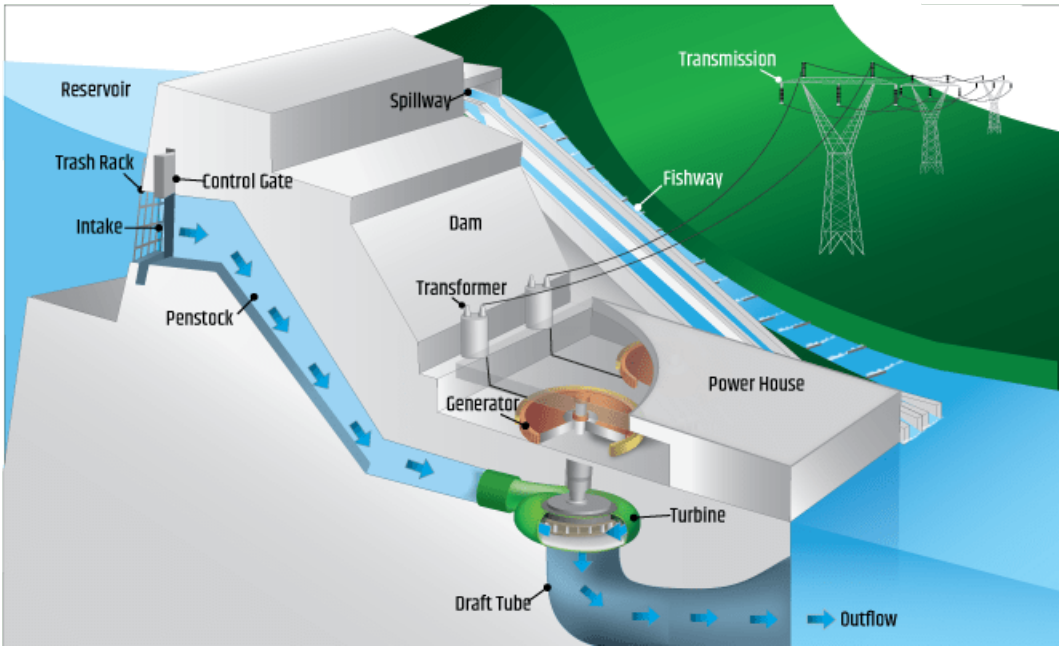


Figura 5. Central de tipo embalse. Fuente: USA Department of Energy.

La cantidad de energía que es posible extraer de una cierta masa de agua depende de la altura recorrida por el agua hasta alcanzar las turbinas, lo que depende a su vez de la altura a la que se encuentre el nivel del embalse en un momento determinado. Este comportamiento es marcadamente no lineal y es dependiente del tiempo, ya que decisiones pasadas tendrán un impacto sobre la cota del embalse. Dicho comportamiento se rige por la ecuación:

$$p = \rho \cdot g \cdot \eta \cdot h \cdot q \tag{1}$$

Además, dependiendo del punto de operación de los generadores, el rendimiento de conversión de energía cinética a eléctrica será diferente, generando una dependencia también con el caudal utilizado. Esta naturaleza dependiente y no lineal hacen que modelar el comportamiento sea extremadamente complicado, ya que la potencia se relaciona con la cota y el caudal generando una superficie curva en tres dimensiones que resulta muy difícil de analizar mediante métodos de optimización clásicos, siendo necesario recurrir a técnicas más avanzadas[11]. Se muestra un ejemplo de una curva de eficiencia de una central hidroeléctrica en la figura 6

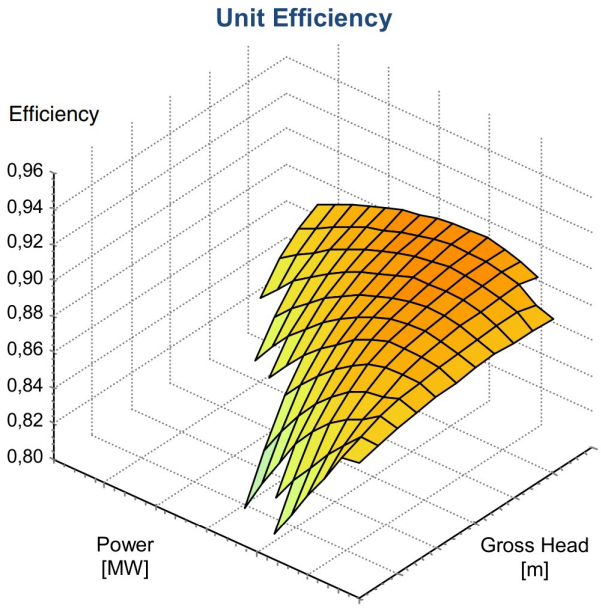


Figura 6. Curva de eficiencia de una central hidroeléctrica. Fuente: Hidalgo, 2014[11].

En modelos dónde se considera que la cota de embalse puede asumirse constante, solo es necesario representar curvas características de cota constante para modelar el comportamiento de la central. Esto sigue dando lugar a comportamientos no lineales y dependientes del caudal pero elimina la dependencia a través del tiempo.

En el caso del modelo diseñado se ha considerado constante tanto la cota de embalse como el rendimiento de los grupos generadores, permitiendo así asignar una equivalencia constante entre una cantidad determinada de agua y la energía potencial que esta representa para el sistema. Al tratarse de un modelo de planificación a corto plazo, se ha asumido que la cota de los embalses puede considerarse esencialmente constante y por tanto no influye significativamente en el modelo.

La naturaleza finita del agua como recurso y el desconocimiento de su valor potencial a largo plazo propician que dicho valor se intente establecer y sea utilizado como una parámetro para influir en la explotación del agua a corto plazo. El agua se vuelve más valiosa en momentos en los que el precio de la energía es más alto, ya que permite el acceso a un recurso prácticamente gratuito, por lo que siempre será rentable emplear el agua en aquellos momentos en los que mayor sea su impacto sobre el coste del sistema. El problema reside en que determinar cuales serán estos momentos es increíblemente complicado debido a toda la incertidumbre asociada, por ello se recurre a diferentes técnicas basadas en datos obtenidos de modelos de medio y largo plazo.

Además, en el caso de ciertos embalses existe la posibilidad de revertir el funcionamiento de las turbinas, convirtiéndolas en bombas que permiten elevar agua de vuelta al embalse para almacenar energía. Esta capacidad permite aprovechar momentos de baja demanda para almacenar energía a bajo coste que posteriormente podrá ser utilizada para sustituir a tecnologías más caras durante picos de demanda.

- Una de las posibles técnicas para gestionar el uso de los recursos hídricos es el establecimiento de consignas tanto de producción como de agua disponible al finalizar el periodo. Este es el método más simple de planificación en el que simplemente se marca una cierta cantidad de agua que se desea que esté disponible al finalizar el periodo incluido en el modelo o una cierta cantidad de agua que debe ser turbinada a lo largo del tiempo

utilizado en el modelo. Tiene la ventaja de ser el método más simple pero a cambio es muy sensible a las incertidumbres asociadas a la cantidad de agua disponible en cada momento, ya que la distribución óptima de recursos variará según se obtenga nueva información y las directrices pueden necesitar ser calculadas de nuevo.

- Otra técnica más sofisticada es utilizar datos obtenidos del problema dual y asignarle al agua un valor basado en su valor marginal en un modelo de medio o largo plazo. Esto implica que el agua será utilizada en momentos en los que su valor esté por encima de un cierto límite, asegurando que es aprovechada de manera eficiente. Esto presenta la ventaja de ser más flexible que la técnica anterior, ya que no requiere recalcular parámetros de un modelo a largo plazo cuando se obtiene más información que antes estaba asociada a incertidumbre, pero a su vez genera comportamientos cortoplacistas e impulsivos de todo o nada según el precio de la energía en cada momento determinado se sitúa por encima o por debajo del valor del agua.

- Por último se puede utilizar una curva que determine el valor del agua en función de su disponibilidad. Esto permite que el valor del agua se determine de manera dinámica obteniendo un valor futuro esperado basado en su utilización, dando así más valor al recurso cuanto más escaso sea, asegurando que no es ni malgastado de manera demasiado impulsiva ni considerado tan valioso que acaba siendo desperdiciado en vertidos. El principal problema de está técnica es que requiere modelar la curva en sí lo cual es muy complicado y está asociado a las incertidumbres del mercado eléctrico.

Hipótesis asumidas

Las hipótesis asumidas para modelar los comportamientos de los grupos hidroeléctricos de manera lineal e independiente son las siguientes:

- Se ha eliminado la dependencia de la potencia generada por un generador hidroeléctrico con el nivel del agua en un embalse.

- Se ha eliminado la relación entre el caudal que atraviesa una turbina y su rendimiento.

- Se ha eliminado la dependencia entre diferentes embalses y grupos generadores.

- Se ha considerado de manera determinista los influjos de agua que llegarán a los diferentes ríos y embalses.

- Se han establecido cantidades de agua que debe haber en cada embalse finalizado el periodo de análisis, fijando así la cantidad de agua disponible para ser utilizada durante la semana.

Baterías

La incorporación de baterías a la red eléctrica como sistema para apoyar a la red eléctrica durante picos y valles de demanda es un importante tema de estudio e investigación, por lo que se ha incorporado en el modelo la capacidad de incluir baterías que almacenen y aporten energía en caso de demandarlo el sistema. Las baterías han sido modeladas como un almacén de energía con capacidades máximas de almacenamiento, carga y descarga y rendimiento a la hora de almacenar y aportar energía eléctrica.

Periodos

El tiempo ha sido modelado como 168 periodos de una hora a lo largo de una semana. Esta representación discreta del tiempo es necesaria para poder resolver el problema sin la necesidad de aplicar técnicas complejas de optimización. La elección de una semana como horizonte temporal se debe a la naturaleza cíclica de la demanda eléctrica, que tiene un periodo de una semana. Entre el lunes y el viernes la industria y las actividades laborales generan una mayor demanda que durante el fin de semana, franja de tiempo durante la cual esta demanda disminuye. Esto lleva a que además las semanas desde el punto de vista de planificación y operación del sistema comiencen la primera hora del sábado y terminen al finalizar el viernes. Dentro de una misma semana existe también un comportamiento cíclico asociado al día y la noche que determina la demanda en cada periodo.

La discretización del tiempo en periodos horarios conlleva la necesidad de representar ciertas magnitudes como una aproximación de su valor medio a lo largo de un periodo mientras que otras son representadas como un valor instantáneo al final de un periodo concreto. Variables del primer tipo serían la demanda en un periodo o la potencia generada por un grupo mientras que variables del segundo serían magnitudes como el agua disponible en un embalse o la carga en una batería.

3.3 Parámetros

Para caracterizar el comportamiento del sistema es necesario recurrir a parámetros que indiquen las cualidades de los diferentes elementos que lo componen. Estos parámetros pueden ser una constante del sistema o variar a lo largo del tiempo o entre diferentes elementos. Por tanto pueden ser descompuestos en 5 grupos:

- Parámetros de los generadores: Definen el comportamiento de un generador concreto. Dependiendo del tipo de generador, térmico o hidroeléctrico los parámetros que lo definen varían.
- Parámetros de las baterías.
- Parámetros del sistema constantes en el tiempo.
- Parámetros del sistema dependientes del tiempo.
- Parámetros de los generadores dependientes del tiempo. En este modelo solo existe un parámetro de este tipo: los influjos de agua a cada generador hidroeléctrico.

Parámetros comunes a todos los generadores

$\overline{q}$	Potencia máxima que puede aportar un generador, [GW].
$\underline{q}$	Potencia mínima que puede aportar un generador, [GW].
k	Fracción de la potencia generada por un grupo que se entrega a la red, [p.u]
u_0	Representa el estado de conexión de cada generador en el periodo anterior al horizonte temporal de estudio, [0,1].

Parámetros asociados a generadores térmicos

α Pendiente en la recta que relaciona el consumo de combustible y la potencia generada por un generador, [MTh/GWh].

β Consumo de combustible en el mínimo técnico, [MTh/h].

γ Consumo de combustible asociado a conectar el grupo generador a la red, llevándolo desde el reposo a su mínimo técnico, [MTh/GWh].

θ Coste de combustible asociado a apagar y desconectar un grupo. Incorpora también la noción de desgaste y estrés por realizar un ciclo de encendido-apagado, [MTh/GWh].

rs Representa la máxima rampa de subida que puede experimentar el generador, [MTh/GWh].

rb Representa la máxima rampa de bajada que puede experimentar el generador, [MTh/GWh].

f Representa el coste del combustible que consume el generador, [k€/GWh].

o Representa el coste asociado a la operación y mantenimiento del generador, [k€/GWh].

TU Representa el número mínimo de periodos que debe pasar conectado antes de poder volver a ser apagado, [h]

TD Representa el número mínimo de periodos que debe pasar desconectado antes de poder volver a ser encendido, [h]

e Cantidad de emisiones generada por el generador, [ton CO_{2eq}/Mth].

Parámetros asociados a generadores hidroeléctricos

$\overline{b}$ Representa la máxima potencia de bombeo de la que es capaz un grupo hidroeléctrico, [GW].

$\overline{w}$ Representa la máxima cantidad de agua que puede ser almacenada en el embalse asociado al grupo hidroeléctrico, [GWh].

$\underline{w}$ Representa la mínima cantidad de agua que puede ser almacenada en el embalse asociado al grupo hidroeléctrico, [GWh].

w_0 Representa la cantidad de agua disponible al inicio del horizonte temporal estudiado en el embalse asociado al grupo hidroeléctrico, [GWh].

w_{fin} Representa la consigna de cantidad de agua que debe haber al final del horizonte temporal en el embalse asociado al grupo hidroeléctrico, [GWh]

η Representa el rendimiento en la conversión de energía cinética a eléctrica en un grupo hidroeléctrico, [p.u.].

Parámetros asociados a las baterías

η^c	Rendimiento en la carga de la batería. [p.u.].
$\eta^d isc$	Rendimiento en la descarga de la batería, [p.u.].
E_0	Carga de la batería al inicio del horizonte temporal, [GWh].
$\overline{E}$	Capacidad máxima de carga de la batería, [GWh].
$\underline{E}$	Capacidad mínima de carga de la batería, [GWh].
$\overline{P}^c$	Potencia máxima de carga de la batería, [GW].
$\overline{P}^{disc}$	Potencia máxima de descarga de la batería, [GW].

Parámetros asociados al tiempo

d	Demanda media del sistema durante un periodo, [GW].
p_{rod}	Reserva rodante media requerida durante el periodo, [GW].
$\overline{solar}$	Potencia solar media disponible durante el periodo, [GW].
$\overline{wind}$	Potencia eólica media disponible durante el periodo, [GW].

Parámetros asociados a generadores variables en el tiempo

Como ya se ha indicado antes, los parámetros que caracterizan los generadores se han considerado constantes en su inmensa mayoría a excepción de las aportaciones de agua en las centrales hidroeléctricas, que son altamente variables a lo largo de una semana. Este es por tanto el único parámetro que depende de tanto el generador al que referencia como de un periodo temporal concreto.

i	Aportaciones de agua a una central hidroeléctrica durante un periodo, [GWh].

3.4 Variables de decisión

En esta sección se detallan todas las variables sobre las que puede decidir el sistema para hallar el uso óptimo de los recursos de generación.Debido a la naturaleza del problema de Unit Commitment existe una combinación de variables reales que representan cuantitativamente la gestión de los recursos con variables binarias que modelan las decisiones sobre el estado de conexión de los grupos. Las variables al igual que en el caso de los generadores pueden estar asociadas a ciertos sets, concretamente en este modelo hay dos tipos de variable:

- Asociadas a generadores y periodos. Dentro de está categoría hay variables propias de grupos térmicos, hidroeléctricos y variables comunes propias de cualquier generador.
- Asociadas a periodos.

El hecho de que una variable esté asociada a uno o varios sets implica que puede tomar un valor diferente en cada una de las instancias de esa variable. Entiéndase por instancia una combinación única de elementos que formen parte de los sets correspondientes. Por ejemplo la variable u_{gp} podrá tomar un valor diferente para combinación de generador g y periodo p.

Variables asociadas a generadores y periodos

Se tratan de las diferentes variables sobre las que existe capacidad de decisión y marcan como serán operados los generadores del sistema.

Variables comunes a todos los generadores

q_{gp} Potencia bruta media generada durante el periodo, [GW].

Variables asociadas a generadores térmicos

q'_{tp} Potencia media por encima del mínimo técnico. Estrechamente relacionada con la variable q_{tp}. El uso de esta variable se detalla en la sección 3.5, [GW].

u_{tp} Estado de conexión del generador. Vale 0 cuando el generador está desconectado y 1 cuando está conectado.

y_{tp} Decisión de arranque. Vale 1 en los periodos en los que se decide arrancar el generador y 0 en cualquier otro caso.

z_{tp} Decisión de parada. Vale 1 en los periodos en los que se decide desconectar el generador y 0 en cualquier otro caso.

Variables asociadas a generadores hidroeléctricos

w_{hp} Cantidad de agua almacenada en la reserva asociada al generador al final del periodo, [GWh].

w_{bp} Cantidad de agua bombeada durante el periodo, [GW].

S_{bp} Vertidos realizados en el embalse asociado al generador, [GW].

Variables asociadas a periodos.

Estas son variables de decisión del sistema que no afectan a ningún generador concreto.

$solar_p$ Cantidad de potencia solar aportada al sistema, [GW].

$wind_p$ Cantidad de potencia eólica aportada al sistema, [GW].

pns_p Cantidad de potencia no suministrada a la demanda del sistema, [GW].

3.5 Restricciones consideradas en el modelo

Restricciones asociadas a generadores térmicos

Factor de carga auxiliar

En los grupos térmicos, parte de la potencia generada por el alternador se emplea en alimentar las instalaciones de la planta. Esta consumo de energía se utiliza para alimentar bombas, compresores, sistemas de control, iluminación,etc. Por tanto, no toda la potencia generada en el alternador llega a ser accesible por el sistema y es necesario representar esta fracción de la potencia destinada a servicios auxiliares. Se expresa mediante la siguiente ecuación:

$$q^{\text{neta}} = q \cdot k \tag{2}$$

donde el parámetro k es conocido como factor de carga auxiliar y representa la fracción de potencia generada por las turbinas accesible en barras de la central.

Límites mínimos y máximos de generación

Los generadores térmicos tienen una potencia nominal la cual no pueden superar ya que causarían daños a la instalación por lo que siempre deben mantenerse por debajo de ellos. Además las calderas de las centrales térmicas tienen un requerimiento de potencia mínima por debajo del cual la combustión de vuelve inestable. Estas consideraciones solo deben tenerse en cuenta en caso de que el generador esté conectado a la red aportando potencia.

$$q_{tp} \leq u_{tp} \cdot \overline{q}_t^{neta} \tag{3}$$

$$q_{tp} \geq u_{tp} \cdot \overline{q}_t^{neta} \tag{4}$$

Nótese que en caso de estar el generador desconectado y por tanto la variable u valer 0, ambas restricciones estarían activas igualando la potencia suministrada a 0.

Existe otra forma común de representar estas ecuaciones utilizando la potencia por encima del mínimo técnico , q', como medio para expresarlas.

$$q_{tp} = u_{tp} \cdot k_t \cdot \underline{q}_t + q'_{t,p} \tag{5}$$

$$q'_{tp} \leq u_{t,p} \cdot k_t \cdot \left(\overline{q}_t - \underline{q}_t\right) \tag{6}$$

Este segundo método es el que se ha utilizado para elaborar el modelo ya que su carácter incremental permite incorporar restricciones de rampa de subida y bajada que no generen conflicto con el modelado de los procesos de arranque y parada.

Rampa de subida y bajada

Estas restricciones limitan la variación en la potencia aportada por un grupo generador entre dos periodos consecutivos. Sirven para evitar que se requiera a los generadores dar una respuesta más rápida a las variaciones de demanda de la que son realmente capaces de dar.

$$q'_{tp} - q'_{t(p-1)} \leq rs_t \tag{7}$$

$$q'_{t(p-1)} - q'_{tp} \leq rb_t \tag{8}$$

Como ya se ha indicado anteriormente, estas ecuaciones utilizan la formulación incremental de la potencia producida por los generadores.

Consistencia en el estado de conexión, arranque y parada

Esta ecuación asegura la consistencia lógica de las operaciones de arranque y parada y su efecto sobre el estado de conexión de los grupos. Impide a un grupo encenderse cuando no está apagado, apagarse cuando no está encendido y cambiar su estado de conexión sin realizar ningún arranque ni parada.

$$y_{tp} - z_{tp} = u_{tp} - u_{t(p-1)} \tag{9}$$

Esta restricción debe cumplirse en todos los periodos salvo en el primero, donde se sustituye por una versión modificada.

$$y_{tp} - z_{tp} = u_{tp} - u_0 \tag{10}$$

Durante el primer periodo se utiliza el parámetro u_0 que indica el estado de conexión en el que se encontraba el grupo antes de empezar la semana, y sobre el cual no se tiene ningún tipo de capacidad de decisión.

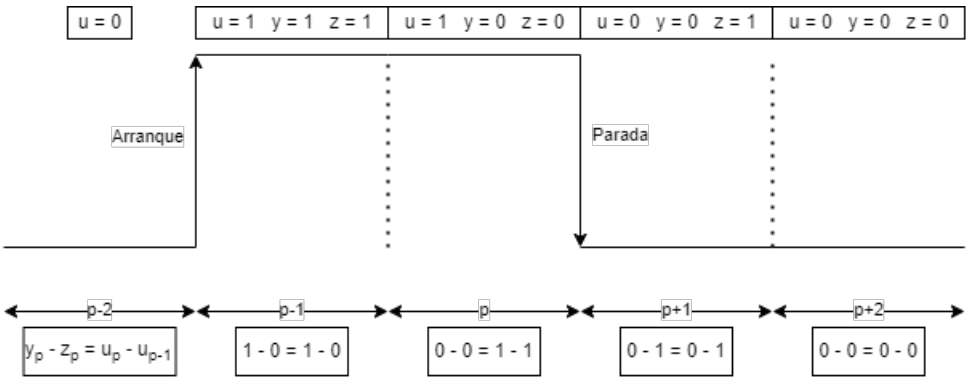


Figura 7. Consistencia entre conexión, arranque y parada

Tiempo mínimo de conexión y desconexión

Para evitar un excesivo desgaste por estrés térmico en los procesos de encendido y apagado, se impide a los generadores tener varios cambios en su estado de conexión en un intervalo pequeño de tiempo. Esto impide que la caldera de una central esté siendo enfriada y calentada rápidamente de manera repetida, ahorrando así el coste asociado al desgaste y las operaciones de mantenimiento que esto genera.

$$\sum_{i=p-TU_t+1}^{p} y_{ti} \leq u_{t,i} \forall t, p \in [p_{TU_t}, p_{168}] \tag{11}$$

$$\sum_{i=p-TD_t+1}^{p} z_{ti} \leq 1 - u_{t,i} \forall t, p \in [p_{TU_t}, p_{168}] \tag{12}$$

La ecuación (11) representa la restricción de tiempo mínimo entre conexión y desconexión y la 11 representa el tiempo mínimo entre desconexión y conexión.

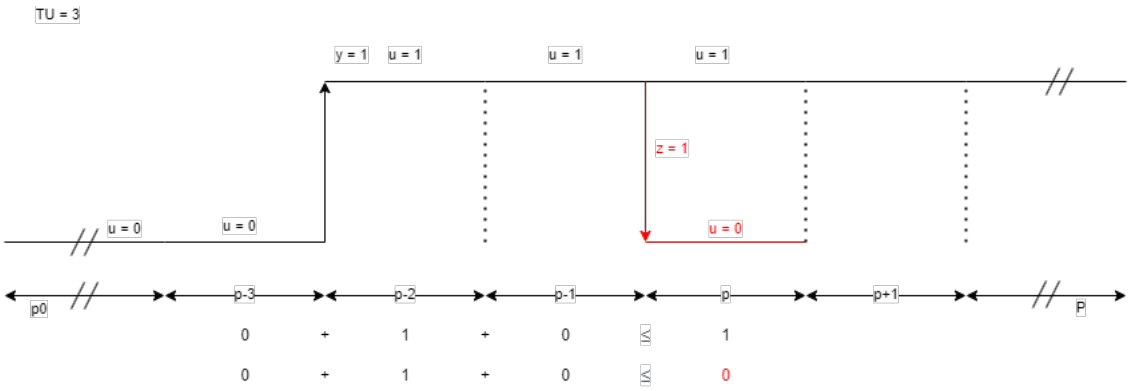


Figura 8. Restricción de mínimo tiempo de conexión

Restricciones asociadas a generadores hidroeléctricos

Potencia máxima

Al igual que en los generadores térmicos, las turbinas de centrales hidroeléctricas tienen una potencia nominal la cual no debe superarse para no dañar la central, generando la siguiente restricción:

$$q_{hp} \leq u_{tp} \cdot \overline{q}_h^{neta} \tag{13}$$

Una vez más el factor de carga auxiliar k representa la fracción de potencia sobrante tras abastecer de energía las instalaciones de la propia planta.

Caudal mínimo

Debido a la necesidad en ciertas centrales hidroeléctricas de compaginar el uso del agua como un recurso energético con su uso como bien de consumo para la población, es necesario imponer en ciertos casos un mínimo flujo para asegurar que la demanda de agua se satisface aunque no sea óptimo desde el punto de vista de operación del sistema. Esto esencialmente impone la misma restricción de potencia mínima que en los grupos térmicos debido al método de modelado del agua como su equivalente en términos de energía.

$$q_{hp} \geq flu_{hp} \tag{14}$$

Balance de energía en reservas

Dada la representación del agua como una cantidad de energía mecánica almacenada, la cantidad de agua disponible en un embalse al final de cada periodo queda representada como la cantidad de energía contenida en ese agua, teniendo en cuenta todas las aproximaciones y simplificaciones hechas en la sección 3.2. Esta reserva de energía debe ser consistente con el consumo para producción de energía y los aportes de precipitaciones, resultando en la siguiente ecuación:

$$w_{hp} = w_{h(p-1)} - [q_{hp} + s_{hp} - \eta_h b_{hp}] + i_{hp} \tag{15}$$

Esta ecuación asegura que la cantidad de agua almacenada aumenta al recibir aportaciones de bombeo e influjos y disminuye al extraer agua del embalse. El agua puede ser retirada del embalse para la producción de energía o en forma de vertidos en caso de necesitarse retirar más agua de la que pueden aprovechar los grupos generadores al pie del embalse. Debido a la forma de modelado del agua, la energía almacenada por bombeo debe multiplicarse por el rendimiento de bombeo de la central hidroeléctrica.

Esta restricción debe modificarse ligeramente durante el primer periodo para representar la cantidad de agua disponible al finalizar la semana anterior, que es un parámetro de entrada al modelo sobre el cual no hay ningún tipo de decisión. Tras esta modificación la se obtiene la ecuación:

$$w_{h,1} = w_{h,0} - [q_{h,1} + s_{h,1} - \eta_h b_{h,1}] + i_{h,1} \tag{16}$$

Además, de igual manera que las reservas de agua disponibles en el primer periodo son dictadas por modelos de medio y largo plazo e impuestas al problema de Unit Commitment, la cantidad de agua que debe haber en cada embalse al finalizar la semana es una también una directriz fijada de antemano y representada por el parámetro w_{fin}. Debiéndose modificar también el balance de agua disponible en el último periodo de la siguiente manera:

$$w_h^{fin} = w_{h,167} - [q_{h,168} + s_{h,168} - \eta_h b_{h,168}] + i_{h,168} \tag{17}$$

En este caso se ha utilizado el índice 168 para representar el último periodo analizado, pero en otros modelos con diferente tratamiento del tiempo este periodo simplemente será el último del horizonte analizado, normalmente representado por una P mayúscula.

Capacidad máxima de bombeo

En el caso de las centrales con la capacidad de revertir el funcionamiento de sus turbinas convirtiéndolas en bombas, estas tienen una potencia nominal de bombeo que no puede ser superada. Además no es posible bombear cantidades negativas de agua. Esto implicaría matemáticamente que es posible generar energía a través del agua con el rendimiento de una bomba en vez de con el de una turbina, lo cual no es cierto. La restricción resultante es:

$$0 \leq b_{hp} \leq \overline{b}_{hp} \tag{18}$$

Es importante tener en cuenta que al no haber asociado a la generación hidroeléctrica ninguna forma de monitorizar su modo de funcionamiento, es posible para una central estar trabajando como bomba y turbina de manera simultánea, lo cual es imposible. El efecto de no modelar este comportamiento se considera despreciable ya que la propia naturaleza de las centrales de bombeo lleva a que en una hora determinada se elija si consumir agua, bombearla o no hacer nada en función del precio actual de la energía. Debido a que para bombear agua y luego convertirla en electricidad otra vez hay que pagar el precio de las ineficiencias en estos procesos de conversión, por norma general no tiene sentido que una central quiera acumular y consumir agua a la vez. Quedando reservada la acción de bombear para momentos donde hay poca demanda que cubrir y el precio de la electricidad es muy bajo pudiendo aprovecharse este agua almacenada en momentos donde el precio sea muy alto y se puedan sustituir tecnologías más caras.

Existen excepciones a esta regla, ya que en ciertos momentos la central puede estar obligada a dejar pasar una cierta cantidad de agua aunque esto vaya en contra de sus intereses económicos, resultando en que se utilice el grupo generador como bomba y turbina simultáneamente. Se considera que excepciones como esta no representan un impacto significativo en el modelo y por tanto no se han considerado.

Límites de almacenamiento

Como se ha explicado en 3.2 tanto por motivos físicos como por motivos estructurales, la cantidad de agua máxima y mínima que debe haber disponible en un embalse en un momento determinado debe estar limitada. Para ello se impone la siguiente restricción:

$$q_{hp} \leq k_h \overline{q}_h \tag{19}$$

En el modelo planteado, las restricciones de almacenamiento mínimo y máximo se consideran constantes a lo largo de la semana ya que se asume que no varían en un intervalo de tiempo tan corto. No obstante, estos límites podrían depender también del tiempo y tomar diferente valor en cada periodo.

Restricciones asociadas a fuentes renovables

Se ha desarrollado un modelo en el cual se plantea la energía eólica y solar en cada periodo como un parámetro de energía máxima disponible en un periodo determinado. Esto es una representación muy simplificada del funcionamiento de dos fuentes de potencia muy complejas y asociadas a mucha incertidumbre. Simplemente se asume que en cada periodo existe una cantidad de potencia máxima de la que el sistema puede hacer uso sin incurrir en coste alguno,

dejando de lado consideraciones sobre si realmente toda esa energía es realmente aprovechable o si por culpa de restricciones no modeladas solo una fracción realmente puede ser incorporada a la red. Por ejemplo, en momentos donde hay una gran disponibilidad de energía solar y eólica puede no ser toda aprovechable ya que las líneas de transporte que conectan esa generación estén saturadas, obligando a incurrir en vertidos.

Además, acorde con el resto de variables asociadas a incertidumbre, se considera que la energía disponible es perfectamente conocida, lo cual no se ajusta a la realidad de estas tecnologías con alto grado de incertidumbre en la generación.

Tras aplicar las simplificaciones mencionadas arriba, resultan dos restricciones esencialmente iguales, una para cada tecnología.

$$0 \leq wind \leq \overline{wind} \tag{20}$$

$$0 \leq solar \leq \overline{solar} \tag{21}$$

Restricciones asociadas a baterías

Por la forma de modelar la cantidad de agua disponible en generadores hidroeléctricos, las baterías se comportan de manera muy similar. Pueden cargarse o descargarse a una velocidad limitada, comienzan y terminan la semana con cantidades de carga concretas y por ultimo es necesario mantener la consistencia entre la carga que contienen y la potencia que han recibido y suministrado

Carga disponible

Esta restricción es esencialmente la misma que la asociada a las reservas de agua en embalses, es una mera representación del principio de conservación de la energía. Debe haber consistencia entre la energía almacenada en cada momento y la energía que se ha aportado y extraído de la batería. Simplemente en este caso no hay que tener en cuenta aportaciones de energía y la capacidad de realizar vertidos no está disponible. Esta expresada por la siguiente ecuación:

$$e_{bP} = e_{b(p-1)} - \frac{p_{bp}^{disc}}{\eta_b^{disc}} + \eta_b^c \cdot p_{bp}^c \tag{22}$$

De igual forma que en los embalses, la energía al inicio y final de la semana requiere una pequeña variación en la ecuación, convirtiéndola en las ecuaciones:

$$e_{bp} = e_0 - \frac{p_{b,1}^{disc}}{\eta_b^{disc}} + \eta_b^c \cdot p_{b,1}^c \tag{23}$$

$$e_{b,168} = e_{b,167} - \frac{p_{b,168}^{disc}}{\eta_b^{disc}} + \eta_b^c \cdot p_{b,168}^c \tag{24}$$

Máxima potencia de carga y descarga

Las baterías tienen una corriente nominal que no debe ser superada, por lo que es necesario limitar tanto la carga como la descarga. En el caso de las baterías no se ha considerado ningún límite inferior.

$$p_{bp}^c \leq \overline{p}_{bp}^c \tag{25}$$

$$p_{bp}^{disc} \leq \overline{p}_{bp}^{disc} \tag{26}$$

Restricciones del sistema

Estas son las restricciones que modelan el comportamiento del sistema, deben cumplirse en cada periodo y relacionan los diferentes tipos de generación con la demanda.

Balance de demanda

Esta es la piedra angular de la gestión y operación del sistema eléctrico, captura la noción de que la energía eléctrica generada en cada instante debe ser igual a la consumida. Para satisfacer la demanda se hace uso de generadores térmicos, generadores hidroeléctricos, energía solar y eólica y, en última instancia, se reserva la posibilidad de no satisfacer una cierta fracción de la demanda en caso de no poderse cumplir todas las restricciones. Bajo estas premisas, el equilibrio generación-demanda puede ser modelado de la siguiente forma:

$$\sum_{t} q_{tp} + \sum_{h} (q_{hp} - b_{hp}) + \sum_{b} \left(p_{bp}^{c} - p_{bp}^{disc}\right) + pns_p = d_p \tag{27}$$

Reserva rodante térmica

Para asegurar la estabilidad del sistema eléctrico es necesario asegurarse de que en todo momento es posible acomodar la generación a cambios repentinos de demanda. Al no poder almacenarse energía a gran escala, el sistema eléctrico es altamente sensible a los cambios inesperados de demanda, los cuales, debido a la naturaleza impredecible de la demanda, suceden constantemente.

Cuando la demanda varía y se desajusta con respecto a la generación, la frecuencia a la que opera el sistema cambia ligeramente, empieza a reducirse o a aumentar, según el incremento en la demanda haya sido positivo o negativo respectivamente. Esto se produce ya que el sistema presenta una inercia natural debido a las propiedades eléctricas y mecánicas de los generadores y la red. Detectando estas variaciones en la frecuencia se puede actuar sobre la generación para devolverla al estado de equilibrio con la demanda y devolver la frecuencia a su valor de referencia.

Es por esto que es necesario hacer que los generadores cuya carga es fácilmente variable tengan ciertos márgenes de maniobra para adaptarse a las necesidades de la red. Tanto a nivel instantáneo como a largo plazo.Estos márgenes entre la producción del generador y la posible producción máxima o mínima se conocen como reservas. Existen diferentes tipos de reservas, que varían en cuanto a objetivo concreto y tiempo de actuación de las mismas.

En el modelo desarrollado se ha tenido en cuenta únicamente la reserva secundaria, que es la más relevante en la operación del sistema eléctrico. Es la reserva encargada de devolver la frecuencia a su valor nominal, 50Hz en España, supliendo las diferencias entre demanda y generación. Se trata de una reserva rápida, que debe actuar en menos de unos pocos minutos y por tanto es importante que solo se consideren como grupos capaces de aportar esta reserva a aquellos grupos que estén conectados en un momento dado. A la reserva aportada por estos grupos se la conoce como reserva rodante, ya que los grupos están ya en marcha o "rodando".

Dependiendo de si la operación del sistema está centralizada o si está liberalizada, las reservas se impondrán a la hora de planificar la producción o se recompensarán mediante el mercado de servicios auxiliares respectivamente. La incorporación de la reserva rodante de grupos térmicos da lugar a la siguiente restricción:

$$\sum_{t} \left(u_{tp} \cdot k_t \cdot \overline{q}_{tp} - q_{tp} \right) \geq rod_p \tag{28}$$

Típicamente la reserva rodante se impone como una fracción de la demanda, en el caso de este modelo se ha considerado una reserva mínima del 10% de la demanda en cada periodo.

3.6 Función objetivo

El objetivo del modelo desarrollado es conseguir la operación óptima del sistema en cuanto a coste total respetando todas las restricciones pertinentes en el proceso. La mayoría del coste incurrido en la operación del sistema proviene de la necesidad de obtener el combustible de las centrales térmicas y de la obtención de los derechos de emisión de CO_2. Considerando despreciables los costes de operación del resto de tecnologías, se busca minimizar estas dos magnitudes para encontrar la operación óptima del sistema. Además hay que considerar otros costes asociados a la operación de las centrales térmicas no directamente relacionados con el consumo de combustible y con la incapacidad del sistema de satisfacer toda la demanda de manera satisfactoria. Esto queda representado en la función objetivo del modelo:

$$min \sum_{p} \left[c^{pns} pns_p + \sum_{t} \left[f_t \theta_t z_{tp} + (f_t + c_e \cdot e_t) \cdot \left[\beta_t u_{tp} + \gamma_t y_{y,p} + \alpha_t \frac{q_{tp}}{k_t} \right] + o_t \cdot \frac{q_{tp}}{k_t} \right] \right] \tag{29}$$

Como ya se ha indicado en 3.2, la relación entre potencia generada y consumo de combustible se modela como una recta afín definida por su pendiente α y su mínimo consumo β. Además se asocia a las acciones de arranque y parada un consumo de combustible, tanto por la necesidad de calentar la caldera durante el arranque como por la pérdida de combustible ya introducido en la cámara durante la parada y el coste asociado al estrés del ciclo encendido-apagado. Estos consumos multiplicados por el factor f_t, que representa el coste del combustible utilizado por cada generador, dan lugar al coste asociado a la compra de combustible para cada generador.

Además, el consumo de combustible durante la producción de energía y el arranque de ciertos grupos tiene asociada a su vez una emisión de gases contaminantes por la cual se incurre en un coste en concepto de derechos de emisión. El coste de estos derechos de emisión viene representado por el producto de c_e, que representa el coste de emisión de una tonelada de CO_{2eq} y la cantidad de emisiones por cada Mth de combustible consumida e_t.

Si a estos dos factores se añade el coste de operación y mantenimiento de cada grupo generador, se obtienen el coste de operar dicho grupo durante un periodo determinado. Combinando el coste de todos los grupos para cada periodo junto al coste incurrido por la incapacidad de satisfacer la demanda, representado por el producto $c^{pns} pns_p$, se obtiene el coste total de operar el sistema durante un periodo. Cabe destacar que este coste es solo una representación matemática de un modelo y no se ajusta necesariamente a conceptos económicos reales. Esto se debe a que decisiones instantáneas de arranque y parada afectan a un único periodo sobre el que recae todo el coste, en vez de quedar distribuidas de manera más acorde a principios de lógica económica.

Nótese también que el término de potencia no suministrada se emplea mayoritariamente para asegurar la resolubilidad del problema y típicamente se utilizan valores de coste para la energía no suministrada muy superiores al coste marginal de operación del sistema, evitando así utilizar la no satisfacción de demanda como un recurso energético.

Formulación compacta

$$min \sum_{p} \left[c^{pns} pns_p + \sum_{t} \left[f_t \theta_t z_{tp} + (f_t + c_e \cdot e_t) \cdot \left[\beta_t u_{tp} + \gamma_t y_{y,p} + \alpha_t \frac{q_{tp}}{k_t} \right] + o_t \cdot \frac{q_{tp}}{k_t} \right] \right]$$

$$q_{tp} \leq u_{tp} \cdot \overline{q}_t^{neta} \qquad \forall t, \forall p$$

$$q_{tp} \geq u_{tp} \cdot \overline{q}_t^{neta} \qquad \forall t, \forall p$$

$$q_{tp} = u_{tp} \cdot k_t \cdot \underline{q}_t + q'_{t,p} \qquad \forall t, \forall p$$

$$q'tp \leq u_{t,p} \cdot k_t \cdot \left(\overline{q}_t - \underline{q}_t \right) q'_{tp} - q'_{t(p-1)} \leq rs_t \qquad \forall t, \forall p$$

$$q'_{t(p-1)} - q'_{tp} \leq rb_t \qquad \forall t, \forall p$$

$$y_{tp} - z_{tp} = u_{tp} - u_{t(p-1)} \qquad \forall t, \forall p$$

$$y_{tp} - z_{tp} = u_{tp} - u_0 \qquad \forall t, \forall p$$

$$\sum_{i=p-TU_t+1}^{p} y_{ti} \leq u_{t,i} \qquad \forall t, \forall p \in [p_{TU_t}, p_{168}]$$

$$\sum_{i=p-TD_t+1}^{p} z_{ti} \leq 1 - u_{t,i} \qquad \forall t, \forall p \in [p_{TU_t}, p_{168}]$$

$$q_{hp} \leq u_{tp} \cdot \overline{q}_h^{neta} \qquad \forall h, \forall p$$

$$q_{hp} \leq k_h \overline{q}_h \qquad \forall h, \forall p$$

$$w_{hp} = w_{h(p-1)} - [q_{hp} + s_{hp} - \eta_h b_h] + i_{hp} \qquad \forall h, \forall p \in [p_2, p_{167}]$$

$$w_{hp} = w_{h0} - [q_{hp} + s_{hp} - \eta_h b_h] + i_{hp} \qquad \forall h, p = p_1$$

$$w_h^{fin} = w_{h(p-1)} - [q_{hp} + s_{hp} - \eta_h b_h] + i_{hp} \qquad \forall h, p = p_{168}$$

$$0 \leq b_{hp} \leq \overline{b}_{hp} \qquad \forall h, \forall p$$

$$q_{hp} \leq k_h \overline{q}_h \qquad \forall h, \forall p$$

$$0 \leq wind \leq \overline{wind} \qquad \forall p$$

$$0 \leq solar \leq \overline{solar} \qquad \forall p$$

$$e_{bp} = e_{b(p-1)} - \frac{p_{bp}^{disc}}{\eta_b^{disc}} + \eta_b^c \cdot p_{bp}^c \qquad \forall b, \forall p \in [p_2, p_{168}]$$

$$e_{bp} = e_0 - \frac{p_{bp}^{disc}}{\eta_b^{disc}} + \eta_b^c \cdot p_{bp}^c \qquad \forall b, \forall p = p_1$$

$$e_{bp} = e_{b(p-1)} - \frac{p_{bp}^{disc}}{\eta_b^{disc}} + \eta_b^c \cdot p_{bp}^c \qquad \forall b, \forall p = P$$

$$p_{bp}^c \leq \overline{p}_{bp}^c \qquad \forall b, \forall p$$

$$p_{bp}^{disc} \leq \overline{p}_{bp}^{disc} \qquad \forall b, \forall p$$

$$\sum_t q_{tp} + \sum_h (q_{hp} - b_{hp}) + \sum_b \left(p_{bp}^c - p_{bp}^{disc}\right) + pns_p = d_p \qquad \forall b, \forall p$$

$$\sum_t \left(u_{tp} \cdot k_t \cdot \overline{q}_{tp} - q_{tp}\right) \geq rod_p \qquad \forall t, \forall p$$

3.7 Parámetros adicionales

Por motivos de comodidad para el usuario se han incluido una serie de parámetros adicionales en el código que no modelan ningún fenómeno físico real, sino que permiten estudiar el comportamiento del modelo bajo ciertos supuestos. Existen tres parámetros de este tipo: factor de energía solar, factor de energía eólica y factor de demanda. Dichos parámetros se encuentran en uno de los archivos de configuración y permiten multiplicar el valor de su parámetro asociado por un factor constante. Simplemente sirven para estudiar el comportamiento del sistema frente a variaciones de demanda y disponibilidad de energías renovables.

4 Implementación del modelo. Pyomo e interfaz web

En esta sección se va a tratar la implementación del modelo para su ejecución mediante interfaz web. Para ello se explicará el modelado de sets, variables, restricciones y función objetivo en Pyomo, la importaciópn y exportación de datos, representación de resultados y por último despliege en la web a través tanto de Binder como de Google Colab y de las interfaces de usuario diseñadas.

4.1 Modelado en Python-Pyomo

Pyomo permite al usuario adaptar las capacidades del lenguaje Python al modelado de problemas de optimización algebraica. Para ello se utiliza una sintaxis similar a la utilizada de manera convencional en matemáticas. Pyomo permite crear objetos personalizados propios de la librería como restricciones, variables, parámetros y función objetivo, estos se combinan para obtener la formulación del problema que posteriormente será resuelto.

La implementación de un modelo de optimización matemática en Pyomo busca emular el planteamiento matemático del mismo. Se comienza creando una estructura de datos conocida como modelo, este es un contenedor ordenado de todos los elementos que conforman el problema de optimización. A continuación, es necesario declarar todos los parámetros y variables que existen, incluyéndolos en el modelo y estableciendo tipo, dominio y los sets mediante los que están indexados. Tras la declaración de parámetros y variables, es posible utilizarlos para establecer ecuaciones que deben cumplirse dentro del modelo, estas ecuaciones se conocen dentro de Pyomo como reglas.

Las reglas son similares a las restricciones pero no son exactamente lo mismo, las reglas son una forma para Python de definir el concepto de igualdad e inecuación, que luego pueden utilizarse para establecer restricciones. Pyomo permite generar restricciones a través de otros métodos como el de simplemente fijar los límites inferior y superior de una variable. Además, las reglas tienen más funciones que la de crear restricciones, en concreto, la función objetivo se constituye a través de una regla y un sentido o sense, el sentido marca si la función objetivo debe buscar minimizar o maximizar su valor.

4.2 Tipos de modelo.

Pyomo distingue entre dos tipos de modelo: abstractos y concretos. Los modelos abstractos son el equivalente a los modelos de optimización algebraica matemáticos, donde se expresan relaciones entre variables y parámetros mediante ecuaciones pero sin utilizar valores numéricos o categóricos específicos. Los modelos concretos son la particularización de un modelo abstracto para un caso específico, donde todos los elementos del problema de optimización toman un valor concreto. En la figura 9 se muestra el proceso para la construcción del modelo y su resolución.

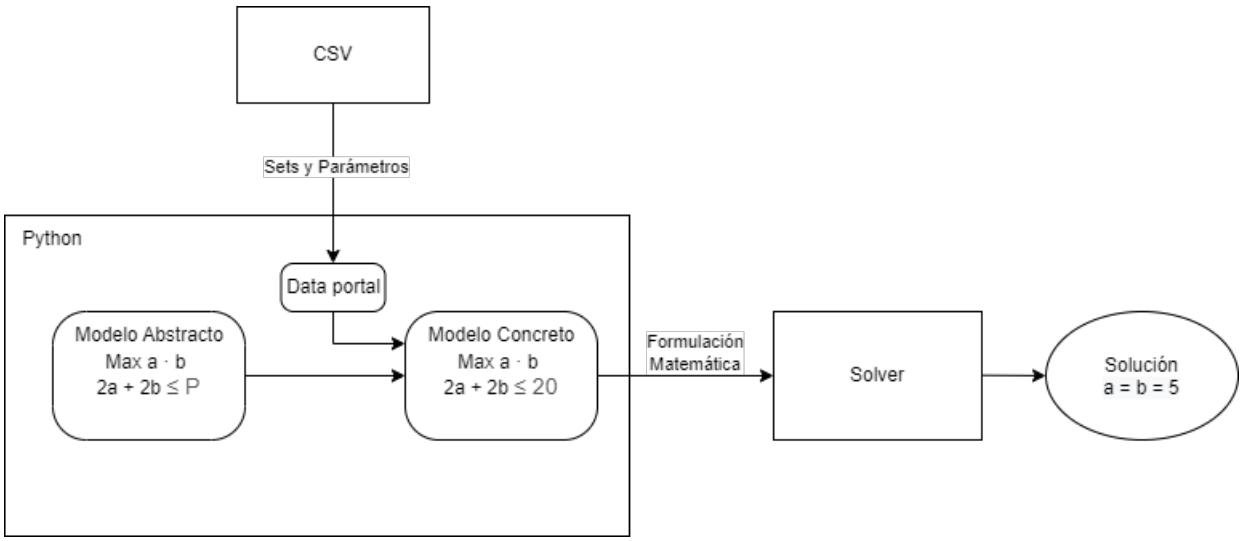


Figura 9. Proceso de formulación y resolución del problema.

Pyomo realmente solo permite resolver y trabajar sobre modelos concretos, utiliza los modelos abstractos como un esqueleto conceptual, del cual genera una instancia con valores

específicos y definidos que puede ser resuelta. Una de las ventajas de Pyomo es que permite realizar pequeñas modificaciones sobre un mismo modelo abstracto generando modelos concretos diferentes. Esto se ha utilizado en este proyecto para determinar el precio marginal por periodo en el sistema, generando una versión relajada del problema de Unit Commitment para así poder obtener un problema dual y analizar la variable asociada a la restricción de demanda.

4.3 Importación de datos. Data Portals.

La forma de obtener un modelo concreto a partir de uno abstracto es generando todos los sets requeridos en el problema y dando valor a todos los parámetros que resultan de estos sets. Pyomo entonces se encargará de generar todas las restricciones, variables y función objetivo de manera acorde. Para introducir los valores necesarios, se utiliza una estructura de datos propia de Pyomo llamada Data Portal, que contiene datos numéricos y categóricos de manera organizada.

Existen diferentes fuentes desde las que importar los datos mediante un Data Portal, generalmente se usan archivos sencillos como JSON o dat para almacenar los datos e importarlos desde Python. El formato usado en este proyecto es el de valores separados por coma o CSV (Comma Separated Values) que es uno de los formatos mas simples y extendidos en el mundo de la ingeniería para la gestión de datos.

4.4 Resolución del problema

La resolución del problema con Pyomo una vez está planteado es sencilla, la propia librería se comunicará de manera completamente automática con la API del solver que haya elegido el usuario. Además, es posible mediante Pyomo modificar diferentes opciones de entre todas las que ofrece el solver, permitiendo personalizar y refinar la resolución del problema de optimización. En el caso de este proyecto se han utilizado diferentes solvers, dependiendo de si el problema se estaba resolviendo de manera local o en la nube

Durante la resolución de manera local, se ha utilizado el solver Gurobi con una licencia de uso académico personal. Gurobi ofrece una gran cantidad de opciones para personalizar la resolución. Estas opciones permiten modificar los algoritmos utilizados durante la resolución y sus parámetros. Las configuración elegida en este proyecto ha sido:

Opción	Valor	Efecto
Method	1	Fija el método de resolución a simplex-dual
MIPGap	0.01	Permite una diferencia o gap del 1% entre el problema de variables enteras y el problema relajado
Threads	7	Número de hilos de computación en paralelo empleados en la resolución
IntFeasTol	1e-9	Desviación máxima aceptable de la factibilidad entera
OptimalityTol	1e-9	Fija el valor máximo que puede tener el dual de una variable para ser considerada óptima
FeasibilityTol	1e-9	Desviación máxima aceptable en la factibilidad de una restricción
RINS	100	Cada cuantos nudos explorados se ejecuta una iteración del método heurístico RINS (Relaxation Induced Neighborhood Search)
DisplayInterval	30	Cada cuantas iteraciones se muestra el progreso del solver
NumericFocus	3	Permite al usuario personalizar la precisión numérica de la ejecución
MarkowitzTol	0.999	Fija el valor de la tolerancia de Markowitz en el proceso de resolución
ScaleFlag	2	Utiliza escalado automático en las restricciones del modelo

Tabla 1. Opciones utilizadas en Gurobi

Para la ejecución en entorno web, sobre la que se hablará más adelante, se intentó utilizar el solver Gurobi con una licencia especial para computación en la nube. Esta licencia está limitada a problemas de 2000 variables como máximo, lo cual es insuficiente para los modelos desarrollados a lo largo de este proyecto. Hay que tener en cuenta que cada variable indexada, como por ejemplo, la potencia generada q_{gp}, es realmente un conjunto de variables, existiendo una variable diferente para cada combinación generador-periodo. Esto hace que salvo en el caso de problemas con sistemas muy simplificados y reducidos, el límite de 2000 variables haga imposible su resolución.

Por esto mismo se decidió utilizar un solver de código abierto que no tenga estas limitaciones en la versión desplegada en la nube. El solver elegido, como ya se ha mencionado en las primeras secciones de este documento, ha sido GLPK. Este solver no dispone de tantas opciones como Gurobi y solo se han utilizado dos. La primera configuración que se ha realizado es fijar el MIP gap al 1% de manera similar a la resolución con Gurobi. La segunda es la activación de la heurística de ramificación por pseudocostes híbrida (hybrid branching pseudocost heuristic), que modifica la forma en la que se eligen las ramas que se van a estudiar en el algoritmo de branch-and-cut. Esta opción es necesaria para conseguir tiempos de ejecución razonables debido al tamaño de los problemas. Aún así, como se explicará más adelante, los tiempos de ejecución mediante cloud computing son excesivamente altos incluso tomando estas medidas.

4.5 Exportación de datos y análisis de resultados

De igual manera que Pyomo permite importar datos de diferentes formas, permite exportarlos de manera cómoda y rápida. Para ello Pyomo aprovecha su lugar dentro del ecosistema de librerías científicas de Python y por tanto su compatibilidad con librerías como pandas, que es quizás de las más conocidas y utilizadas por su utilidad en el análisis y procesado de datos.

Pyomo permite cargar fácilmente los datos obtenidos al solucionar el problema en diferentes estructuras de datos, siendo una de ellas el Data Frame de pandas, que actúa como una matrix de objetos en Python. Para obtener un Data Frame con datos sobre algún elemento del problema basta con exportar a un diccionario dichos datos e importar el diccionario a un Data Frame, obteniendo una estructura organizada con cualquier dato del problema de manera cómoda.

Además es posible exportar las soluciones a archivos dentro del dispositivo local que se esté utilizando para así poder almacenarlas tras la ejecución del código. En el caso de este proyecto las soluciones han sido almacenadas en formato CSV por los mismos motivos por los que los datos fueron importados del mismo, su sencillez, ligereza y compatibilidad con otros programas.

4.6 Análisis de Datos y representación de resultados

Una vez el modelo ha sido formulado y resuelto y las soluciones introducidas en Data Frames, es posible utilizar todas las herramientas de Python para hacer representación y análisis de resultados.

El análisis concreto de los modelos planteados tiene su propia sección en este documento ya que se puede extraer una gran cantidad de información de interés de la resolución de un problema de Unit Commitment. De manera general se ha elegido hacer un pequeño preprocesado de los datos, combinando grupos de variables similares y obteniendo métricas sobre la operación del sistema para luego representarlos mediante el uso de la libreria matplotlib de Python, especializada en representación de datos en formato gráfica.

4.7 Despliegue en la web. Binder y Colab

Para ejecutar el código que formula el problema de Unit Commitment se han utilizado dos enfoques ligeramente diferentes, dependiendo de la plataforma en la que se ha desplegado el código. En primera instancia, se planteó como objetivo del proyecto únicamente el despliegue en Binder, con el objetivo de utilizar el modelo desarrollado para fines didácticos y académicos. Esta opción fue descartada posteriormente dado que Binder no permite a la cantidad suficiente de usuarios crear contenedores basados en el mismo repositorio de manera simultánea. Esto impide que una clase entera de alumnos pueda trabajar sobre el problema de Unit Commitment de manera efectiva.

Aunque esta limitación podría sortearse creando varios repositorios exactamente iguales y consiguiendo que los alumnos en grupos pequeños se distribuyan entre dichos repositorios, decidió explorarse Colab como método para la implementación web.

Al utilizar Colab en vez de Binder, los recursos de computación están asociados a una cuenta de Google en vez de a un repositorio. Esto elimina la limitación asociada al número de usuarios de Binder pero impide que el contenedor sea personalizado y configurado de antemano, requiriendo añadir archivos e instalar tanto librerías necesarias como el solver. Además Colab permite el diseño de interfaces más sofisticadas que Binder, debido a un error de diseño de esta segunda plataforma que se comentará más adelante.

Para poder interactuar de manera cómoda con el software, se ha diseñado una interfaz gráfica de usuario o GUI (Graphic User Interface) a través de la libreria ipywidgets. Esta librería permite incorporar dentro de la estrucutra de un Jupyter Notebook diferentes elementos interactivos como sliders, cuadros de texto, botones, etc. Estos elementos llamados widgets permiten crear una GUI dentro de un Notebook, volviendo la utilización del software mucho más cómoda en intuitiva. Durante este proyecto se ha diseñado una GUI hecha a partir de widgets que permite modificar archivos de entrada, personalizar la representación de resultados y resolver el problema de optimización. Para ello se han creado los siguientes códigos:

- solver.py: Es mayoritariamente el mismo código utilizado en la versión local de la herramienta. En el se formula el problema y se resuelve. Se ha eliminado la representación de resultados para poder volverla interactiva.
- widget_generators.py: Permite la creación del widget encargado de mostrar, modificar y almacenar los parámetros de los generadores.
- widget_demand.py: Permite interactuar con la demanda, es similar a widget_generators.py pero incorporando la capacidad para modificar de manera simultánea la demanda en varios periodos, tanto sumando un valor constante como multiplicando por un factor.
- GUI.ipynb: Es el Jupyter Notebook que ejecuta y controla los demás códigos. Contiene la primera capa de la GUI, la invocación del resto de códigos de Python y la configuración y ejecución de la representación de resultados.

Como se ha mencionado antes, existe una diferencia entre Binder y Colab en lo referente a la librería ipywidgets. Por como está diseñado Binder, este no es capaz de anidar un widget dentro de otro, al intentarlo no se genera el segundo widget sino que se imprime por terminal la estructura de este objeto. Esto, sumado a la limitación de tráfico, llevó a dejar de lado el desarrollo en Binder durante el proyecto y a centrarlo en Colab. No obstante, modificando la estructura de la interfaz para evitar los widgets anidados sería posible mantener las mismas funcionalidades en una versión compatible con Binder.

Aparte de las diferencias en la gestión de widgets, existen diferencias en la propia naturaleza de los entornos, que obligan a acomodar la herramienta de manera específica a las cualidades de

cada uno. Por ello una explicación sobre el uso y configuración de cada entorno en particular se incluye a continuación.

Binder

Binder permite configurar a través de archivos desde el propio respositorio como será el contendedor construido. Esto permite en el arranque instalar y configurar programas como Gurobi o GLPK y añadir las librerías deseadas. En concreto para este proyecto se han utilizado dos archivos diferentes de configuración:

- postbuild: Contiene una secuencia de acciones a realizar en el momento en el que se crea el contenedor, en este caso instala GLPK.

- requirements.txt: Permite especificar dependencias del código en el repositorio, permitiendo instalar librerías y especificar su versión.

Combinando estos archivos de configuración con la capacidad para incluir código en el repositorio desde el que se construye el contenedor, se puede obtener un método para resolver problemas de Unit Commitment de manera cómoda y a través únicamente de un navegador web. No obstante sigue existiendo la limitación del número de usuarios además de la incapacidad para usar una licencia gratuita de Gurobi.

Colab

Como se ha indicado anteriormente,al utilizar Colab en vez de Binder es posible eliminar la restricción de número de usuarios simultáneos a cambio de perder la posibilidad de personalizar y configurar el entorno durante el arranque. Utilizando Colab se elimina también la necesidad de contruir un contenedor a través de un repositorio cada vez que se quiere utilizar el solver, el cual es un proceso automático pero lento.

Para utilizar la herramienta en Colab basta simplemente con incluir dentro de una sesión los archivos necesarios y ejecutar una serie de comandos para configurar el entorno. Dado que Colab no puede importar carpetas de manera directa, el método que se ha seguido durante este proyecto ha sido incluir todos los archivos necesarios en un archivo comprimido tipo zip que más tarde se descomprime dentro de la sesión de Colab. El proceso a seguir al inicio de cada sesión es el siguiente:

1. Abrir una sesión de colab sibre el cuaderno GUI.ipynb.

2. Subir a la sesión de Colab la carpeta comprimida con los archivos del proyecto. En este caso se ha llamado a dicha carpeta Project.zip.

3. Descomprimir la carpeta dentro de Colab. Para ello se ejecuta en una de las celdas el comando "!unzip Project.zip". el símbolo de exclamación al principio del comando indica que debe ser leído como un comando de terminal en vez de por el interpretador de Python. El comando en sí descomprime el archivo Proyect.zip.

4. Instalar GLPK en la sesión. Para ello es necesario ejecutar el comando "!apt install glpk-utils", que descarga e instala GLPK automáticamente.

5. Instalar las diferentes librerías que no se incluyen por defecto en el entorno de Colab. En este caso solo se requiere instalar Pyomo, mediante el comando "!pip install pyomo"

Una vez se han seguido estos pasos, la sesión de Colab queda configurada para permitir el uso de la GUI y la resolución del problema. A continuación se muestra una breve demostración del funcionamiento del software.

En primer lugar, habiendo descomprimido la carpeta con los scripts y los datos, basta con ejecutar la celda de código de la GUI para que aparezca el siguiente menú:

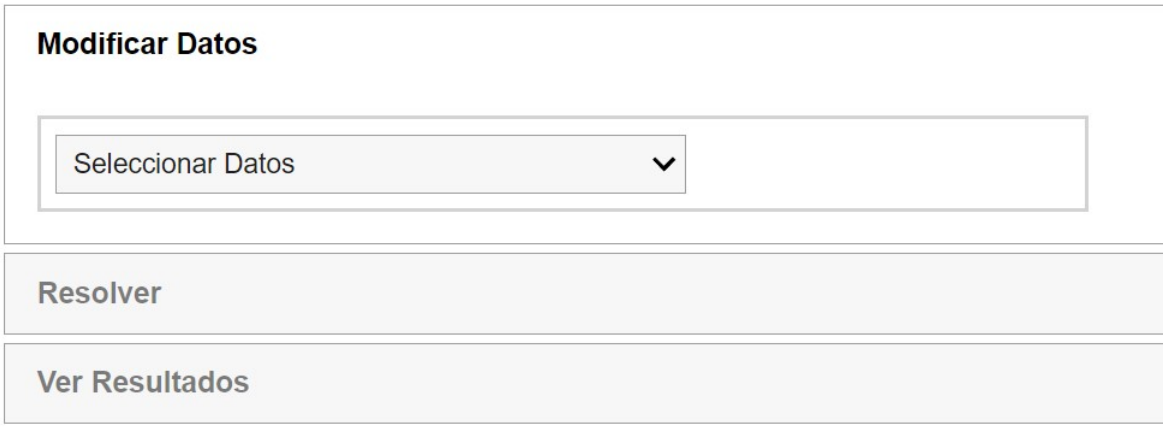
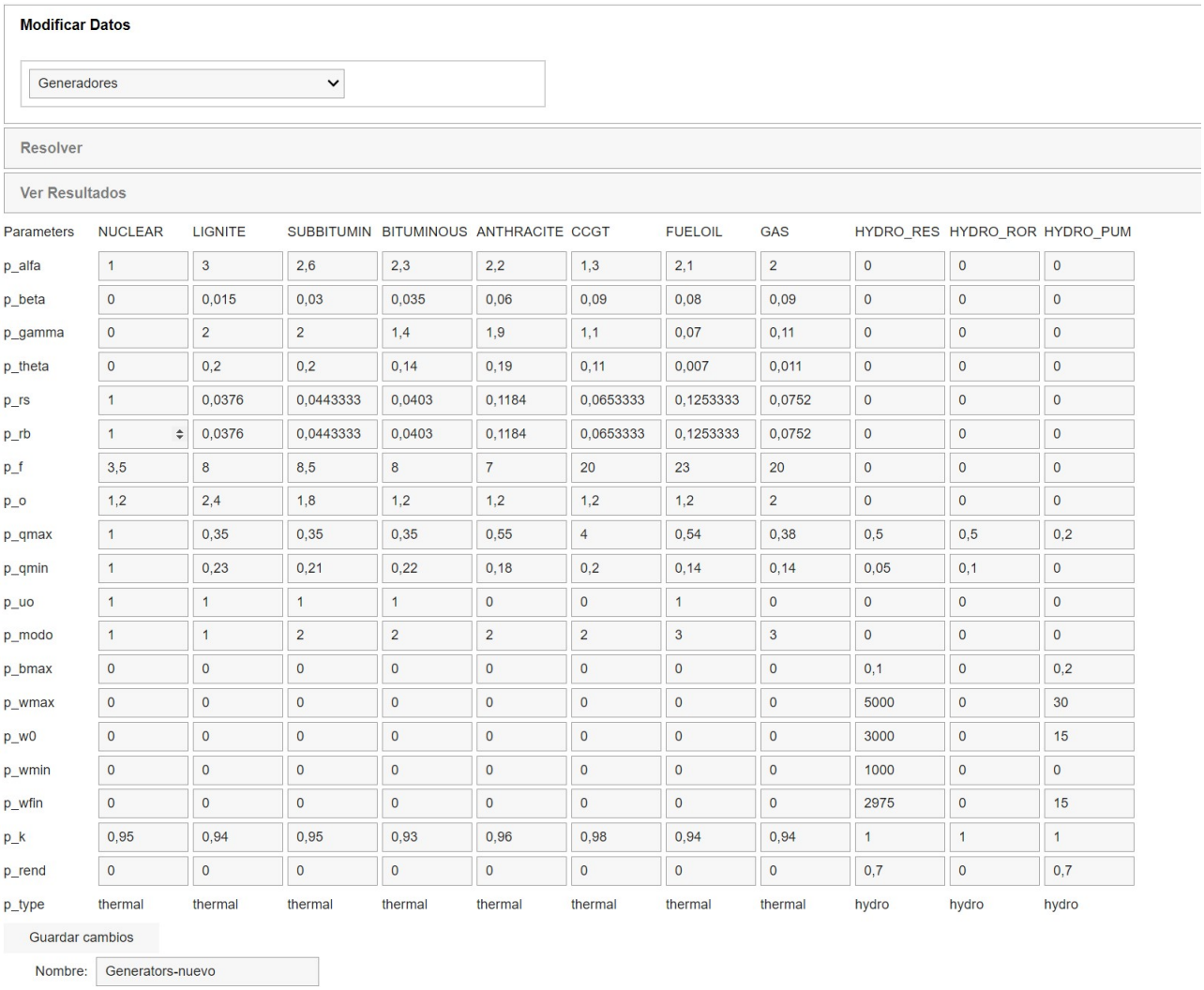


Figura 10. Menú principal del programa

Utilizando el desplegable de seleccionar datos, es posible modificar los datos de los generadores y la demanda.



Parameters	NUCLEAR	LIGNITE	SUBBITUMIN	BITUMINOUS	ANTHRACITE	CCGT	FUELOIL	GAS	HYDRO_RES	HYDRO_ROR	HYDRO_PUM
p_alfa	1	3	2,6	2,3	2,2	1,3	2,1	2	0	0	0
p_beta	0	0,015	0,03	0,035	0,06	0,09	0,08	0,09	0	0	0
p_gamma	0	2	2	1,4	1,9	1,1	0,07	0,11	0	0	0
p_theta	0	0,2	0,2	0,14	0,19	0,11	0,007	0,011	0	0	0
p_rs	1	0,0376	0,0443333	0,0403	0,1184	0,0653333	0,1253333	0,0752	0	0	0
p_rb	1	0,0376	0,0443333	0,0403	0,1184	0,0653333	0,1253333	0,0752	0	0	0
p_f	3,5	8	8,5	8	7	20	23	20	0	0	0
p_o	1,2	2,4	1,8	1,2	1,2	1,2	1,2	2	0	0	0
p_qmax	1	0,35	0,35	0,35	0,55	4	0,54	0,38	0,5	0,5	0,2
p_qmin	1	0,23	0,21	0,22	0,18	0,2	0,14	0,14	0,05	0,1	0
p_uo	1	1	1	1	0	0	1	0	0	0	0
p_modo	1	1	2	2	2	2	3	3	0	0	0
p_bmax	0	0	0	0	0	0	0	0	0,1	0	0,2
p_wmax	0	0	0	0	0	0	0	0	5000	0	30
p_w0	0	0	0	0	0	0	0	0	3000	0	15
p_wmin	0	0	0	0	0	0	0	0	1000	0	0
p_wfin	0	0	0	0	0	0	0	0	2975	0	15
p_k	0,95	0,94	0,95	0,93	0,96	0,98	0,94	0,94	1	1	1
p_rend	0	0	0	0	0	0	0	0	0,7	0	0,7
p_type	thermal	thermal	thermal	thermal	thermal	thermal	thermal	thermal	hydro	hydro	hydro

Figura 11. Menú de modificación de generadores

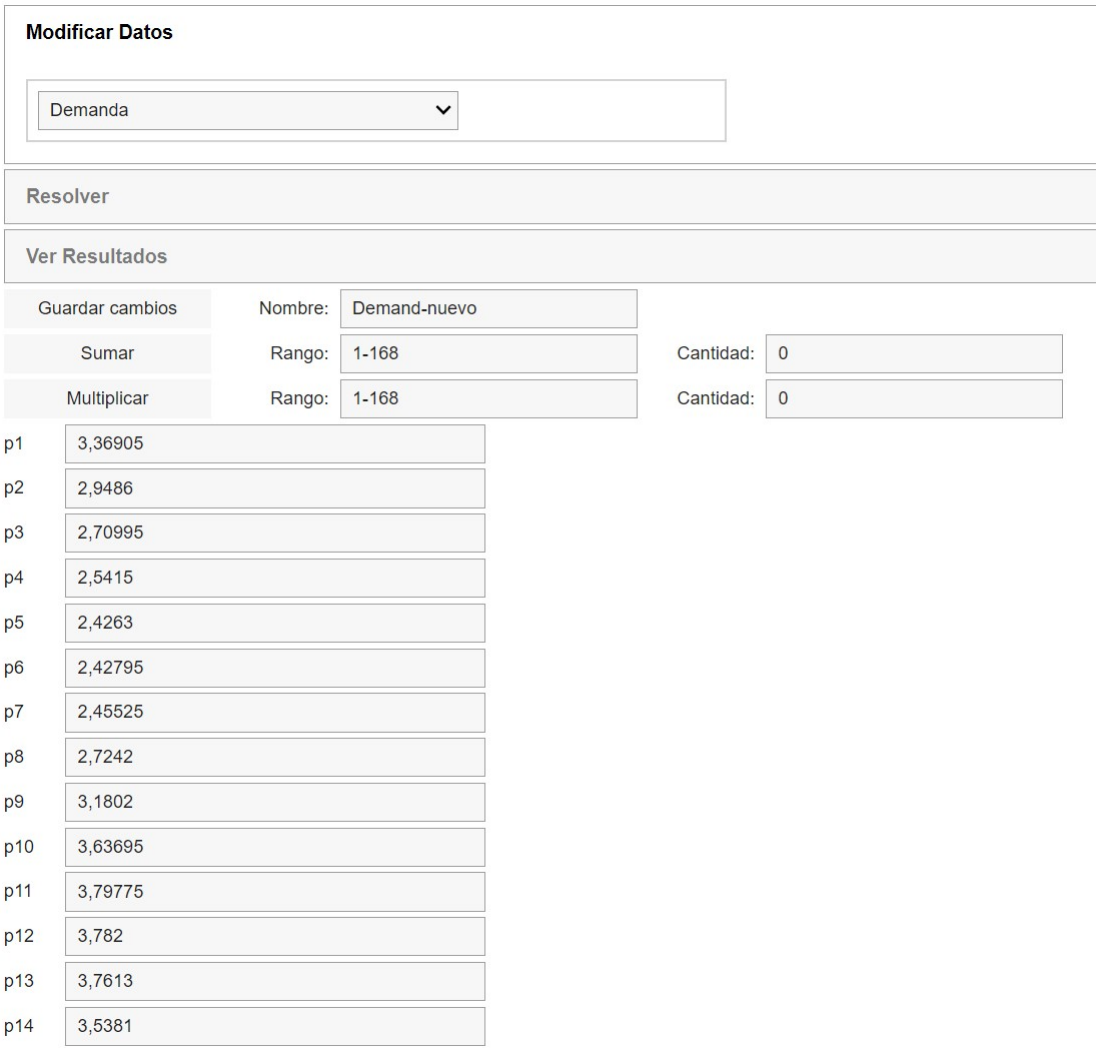


Figura 12. Menú de modificación de demanda

Una vez los datos han sido modificados si así se desea, basta con utilizar el botón de resolver para que el programa encuentre la solución al problema. Tras haber encontrado una solución es posible seleccionar que datos se desea representar en el menú Ver Resultados.

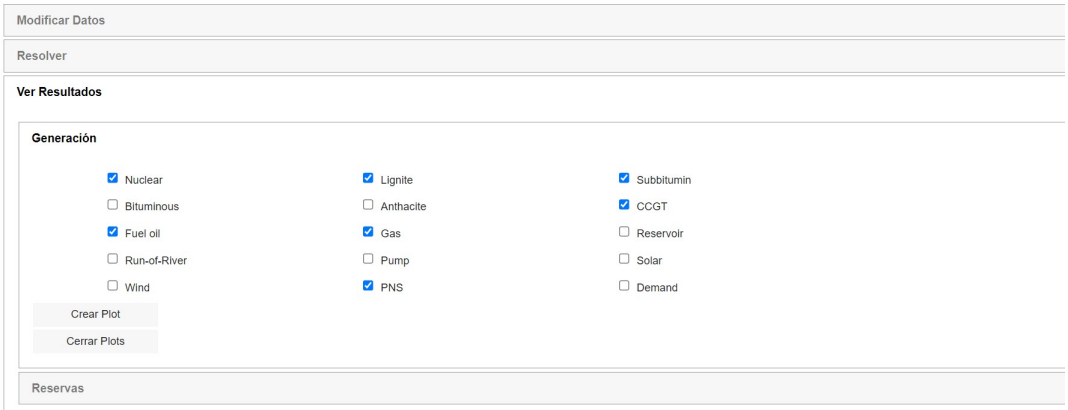


Figure 13. Selector de representación de resultados

5 Análisis de resultados

A lo largo de esta sección se estudiarán dos modelos de versiones simplificadas de sistemas eléctricos. El primer modelo está basado en uno ya existente y con un equivalente escrito en GAMS, y se utilizará como método de verificación para comprobar que el código escrito para este proyecto funciona correctamente. El segundo modelo se utilizará para ilustrar las capacidades y los posibles usos del software diseñado. Por este motivo, el segundo modelo se ha acompañado de un análisis del sistema modelado.

El problema de optimización resuelto y su formulación son prácticamente iguales en ambos casos. La principal diferencia entre ambos modelos es la cantidad de elementos en los sets y los parámetros asociados. Además, en el segundo modelo se han incluido restricciones y sets adicionales. Estas restricciones no forman parte del primer modelo ya que este debe coincidir con el ya disponible en GAMS por motivos de verificación.

El segundo modelo tiene la capacidad añadida de modelar:

- Baterías.
- Emisiones.
- Restricciones adicionales como mínimo tiempo de encendido y apagado o flujo mínimo en centrales hidroeléctricas.

Para poder comprobar si el problema está bien implementado mediante Python-Pyomo se ha comprobado si el valor de la función objetivo en el punto óptimo es el mismo tanto para la implementación en GAMS como en Python. El valor de referencia aportado por la solución de GAMS ha sido utilizado para depurar el código de Python hasta conseguir que los resultados de ambos coincidan.

Durante la resolución del problema, no se encuentra necesariamente el óptimo global sino un óptimo local que cuyo valor de la función objetivo está suficientemente cerca del valor de la función objetivo. Además, no se conoce exactamente ni el modo en que tanto GAMS como Python formulan a bajo nivel el problema ni la forma que tiene el solver de hallar la solución. Por este motivo las soluciones generadas por GAMS y Python no son exactamente iguales, en concreto existe una diferencia de entorno al 0.4% entre ellas en el valor de la función objetivo.

Para realizar el análisis de resultados se ha decidido representar una serie de datos y métricas que se han considerado de interés para un posible usuario del software. Estos datos han sido representados, salvo en el caso del valor de la función objetivo, en formato de gráfica a través de la librería matplotlib.

Las gráficas utilizadas para representar el funcionamiento del sistema han sido:

- Producción por grupo durante cada periodo.
- Producción por tipo de tecnología durante cada periodo.
- Precio marginal en el sistema.
- Producción solar, eólica y sus respectivos factores de carga por periodo.
- Reserva rodante.
- Cantidad de energía total producida por tipo de tecnología.
- Agua almacenada en cada embalse por periodo.

- Carga almacenada en las baterías por periodo.

Al tratarse de un problema con variables binarias, el espacio de posibles soluciones es no convexo. Esto causa que la determinación del precio marginal mediante el problema dual no sea posible. Por este motivo se ha calculado el precio marginal del sistema relajando las variables binarias y convirtiéndolas en reales entre 0 y 1. Esta aproximación permite capturar los conceptos de conexión, parada y arranque sin perder las propiedades de diferenciabilidad y convexidad que permiten la obtención de variables duales.

A continuación se describirán los modelos desarrollados, los datos exactos utilizados se pueden consultar en el Anexo II al final de este documento 6.

5.1 Primer Modelo. MSEM

Este modelo, junto a una gran parte de la formulación del problema de optimización, han sido obtenidos de un documento [14] realizado en el Instituto de Investigación Tecnológica asociado al ICAI. Consta de 8 generadores térmicos, 3 hidroeléctricos, 168 periodos y tanto generación solar como eólica.

Se trata de un sistema sencillo con pocos generadores utilizado para ilustrar de manera didáctica el comportamiento de los problemas de Unit Commitment. En el se pueden apreciar ciertas dinámicas como el uso de generadores térmicos baratos para cubrir la carga base combinado con generadores más caros y rápidos para satisfacerlos picos de demanda. Esto se aprecia claramente en la gráfica de generación desglosada por tecnologías que se muestra en la figura 14.

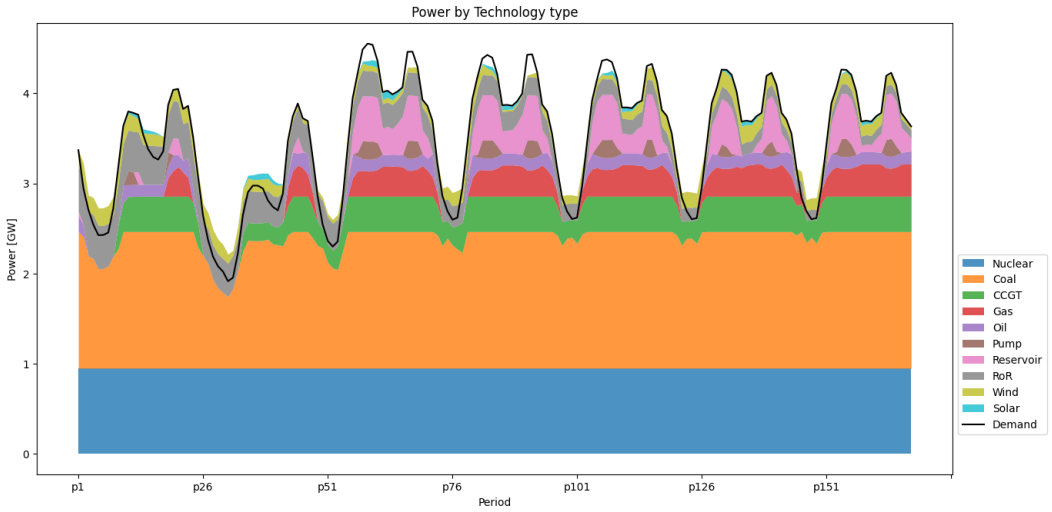


Figura 14. Precio desglosado por tecnología. Primer modelo

Se puede ver como la energía nuclear funciona constantemente a potencia nominal y el carbón funciona a una fracción considerable de su potencia nominal pero alterando ligeramente la producción para acomodarse a los picos y valles.

Ya que este modelo tiene pocos generadores, es posible apreciar un efecto interesante de las curvas, o en este caso rectas, de consumo de combustible. Los generadores son más eficientes cuanto menor es la potencia que proporcionan, por lo que ante una variación de demanda la opción más económica es cambiar el nivel de producción de todos los generadores a la vez para minimizar el coste. Este efecto se aprecia perfectamente en la figura 15.

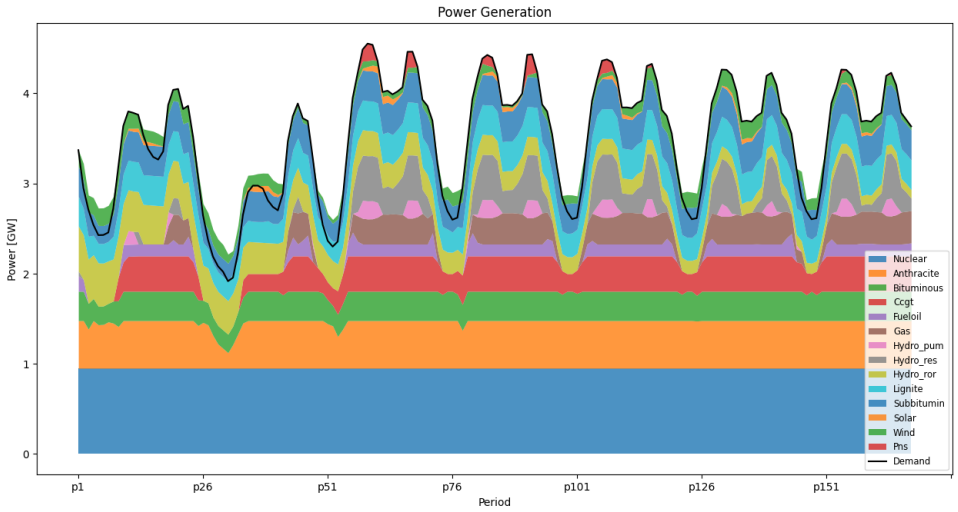


Figura 15. Producción por generador. Primer modelo

Aquí se puede apreciar como en los momentos en los que la demanda cambia entre periodos, los generadores cambian su nivel de generación de manera conjunta. Esto ocurre siempre y cuando no haya un generador sea tan barato de operar que llegue a estar saturado, siendo más rentable seguir aumentando el consumo de otros generadores. No obstante, en ciertos casos un generador que "querría" estar saturado, se verá obligado a variar su producción debido a limites en las rampas de otros generadores o a las necesidades de reserva rodante del sistema.

En este modelo ocurre otro efecto curioso que ilustra el motivo de incluir el término de potencia no suministrada en la formulación del modelo. En este caso en ciertos picos, el sistema no es capaz de satisfacer toda la demanda, por lo que para que el problema se vuelva factible el sistema recurre a no suministrar esa potencia. Esto se ve perfectamente tanto en la figura 15, donde en ciertos picos se ve que la generación no consigue alcanzar a la demanda como en la figura 16 donde se representa el coste marginal del sistema.

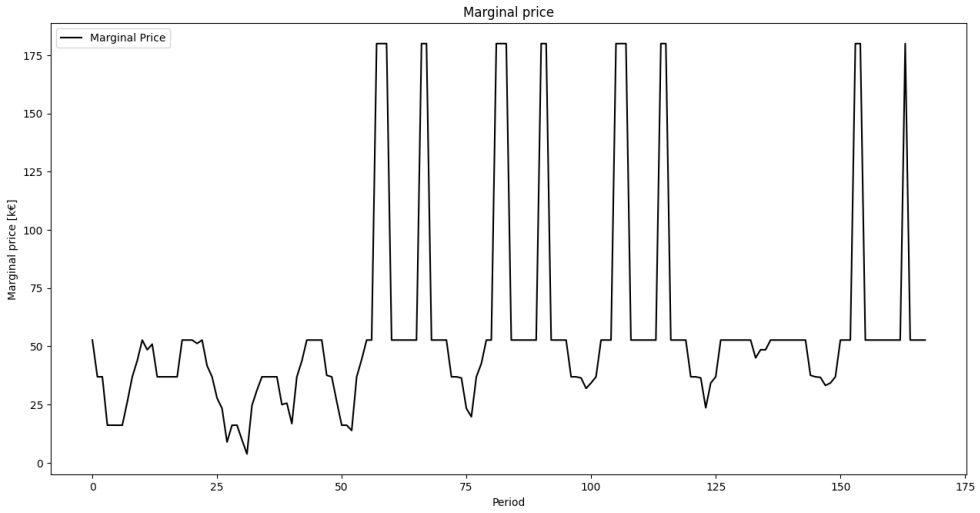


Figura 16. Precio marginal. Primer modelo

Aquí se puede apreciar como el coste marginal en el sistema asciende hasta los 180€/MWh, que es el coste que se ha asignado a la energía no suministrada. Este coste se fija en un valor muy alto ya que no debe ser utilizado por el sistema como un recurwso energético.

5.2 Segundo Modelo. MESP

Este segundo modelo, de elaboración propia, esta inspirado en datos públicos sobre centrales térmicas en España, de las que es posible conocer la tecnología que emplean y su potencia nominal. Estos datos públicos han sido complementados con estimaciones razonables del resto de parámetros. Esta generación térmica se ha suplementado con grupos hidroeléctricos y la disponibilidad de una cierta cantidad de potencia eólica y solar por periodo. En total se han incluido 40 generadores térmicos convencionales de diversas tecnologías, 7 reactores nucleares y 10 grupos hidroeléctricos.

El análisis planteado se trata meramente de una demostración de la capacidad del software y no pretende sacar conclusiones definitivas sobre ningún sistema en específico. Por este motivo se ha decidido limitar dicho análisis a un breve estudio sobre viabilidad y sostenibilidad energética, ya que se ha considerado un tema de especial interés en la actualidad. Se han analizado los efectos de la incorporación de baterías y el mercado de emisiones sobre la sostenibilidad y operación del sistema eléctrico.

Caso base

Se ha partido de un caso base en el que el sistema no tiene acceso a baterías para almacenar energía y no existe ninguna restricción sobre las emisiones de CO_2. En esta situación se aprecia como la demanda base del sistema es producida en su inmensa mayoría por energía nuclear con ayuda de tanto las diferentes fuentes renovables como de una pequeña cantidad de carbón. En los picos de demanda el sistema aprovecha las reservas de agua acumuladas, aumenta la producción del carbón e incorpora ciclos combinados de gas para terminar de ajustarla producción. Esto se aprecia en la figura 17 que muestra la producción por tipo de tecnología y periodo.

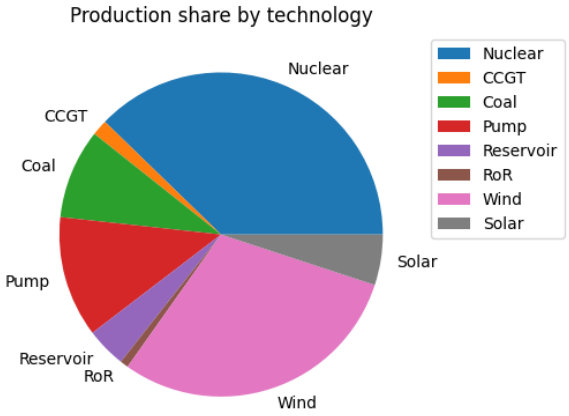


Figura 17. Generación total de energía desglosada por tecnología. Caso base.

En ningún momento de la semana se conecta ninguno de los generadores de fueloil que hay disponibles en el sistema. Esto se debe a que en general su consumo de combustible y el precio del mismo los hace poco económicos incluso sin considerar las emisiones. Como se

aprecia en la figura 18, las dos grandes fuentes de energía en el sistema son la nuclear y las energías renovables, con una fuerte presencia del carbón y el uso de una pequeña cantidad de gas natural. Este comportamiento se da mayoritariamente ya que las emisiones no penalizan la función objetivo y no se tiene en cuenta lo contaminante que es el carbón.

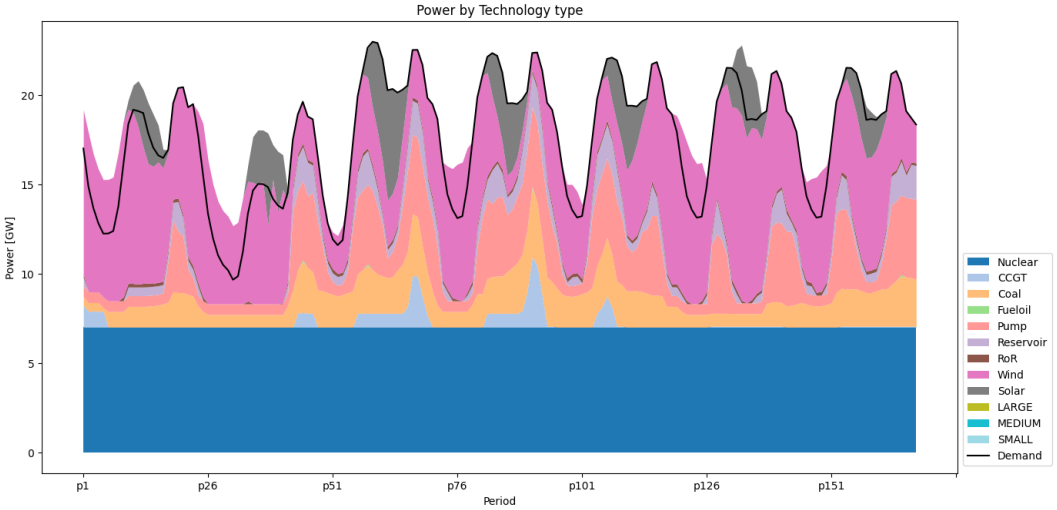


Figura 18. Potencia desglosada por tecnología. Caso Base

El exceso de producción de energía en periodos donde la demanda es baja se está destinando a acumular agua en centrales de bombeo para que esta sea aprovechada en periodos posteriores. El motivo de que sea rentable generar más energía de la realmente necesaria se aprecia especialmente bien cuando se analiza el gráfico de precio marginal mostrado en la figura 19.

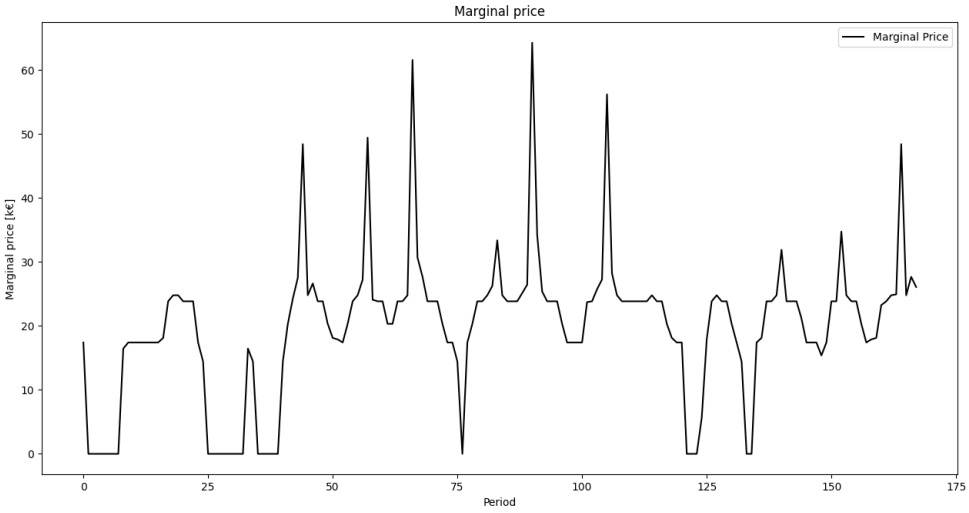


Figura 19. Precio marginal por periodo. Caso base.

Se puede observar que el precio marginal para ciertos periodos es de 0, esto quiere decir que en ese periodo el sistema puede generar más energía de la realmente demandada sin ningún coste. Esto ocurre porque existe un excedente de energías renovables que el modelo considera gratuitas, por lo tanto consumirlas para acumular agua no supone ningún coste adicional.

Para que sea rentable recurrir al bombeo no es necesario que la energía sea totalmente gratuita, basta con que el precio sea suficientemente bajo y se espere en el futuro un pico de demanda suficientemente alto. En este caso, la demanda en el pico necesitará ser cubierta por un generador cuyo coste de operación será caro, puesto que los generadores mas eficientes ya habrán llegado a su límite máximo. Al disponer de agua que se almacenó en el pasado, es posible evitar tener que recurrir a este generador caro, abaratando la operación del sistema.

Esta dinámica de acumulación y consumo periódicos de agua se puede apreciar en la 20, donde se muestra la cantidad de agua disponible para producción eléctrica con respecto al agua inicial.

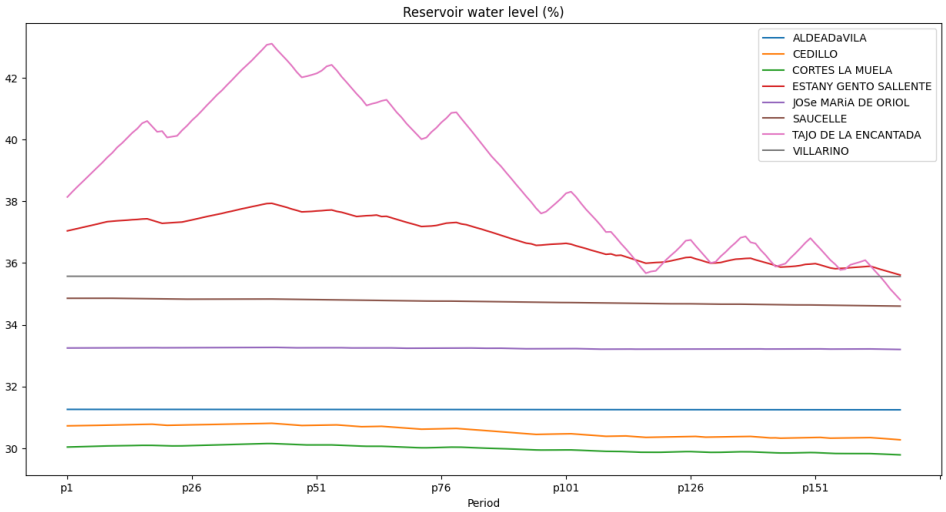


Figura 20. Agua disponible en las reservas. Caso base

Para que esto ocurra, como ya se ha comentado, debe haber un excedente de energía que al sistema no le cueste nada producir, para que al aumentar la demanda, no aumente el coste. En este caso, este fenómeno solo se da en periodos en los que hay un excedente de energías renovables, por lo que se puede apreciar en las curvas de generación de energía eólica y solar y en sus respectivos factores de carga, que dejan de valer 1. Dichas curvas se muestran en las figuras 21 y 22.

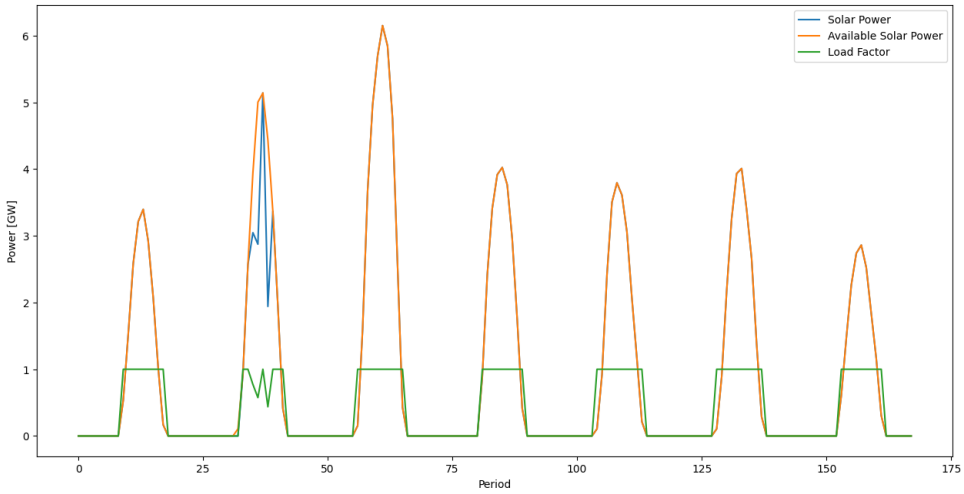


Figura 21. Potencia solar. Caso base

Como la energía solar es producida durante el día, que es cuando más demanda hay, esta se aprovecha al máximo en prácticamente todos los periodos. Se producen vertidos de energía solar únicamente durante uno de los picos de demanda, en el que coincide la demanda más baja de la semana con la segunda producción solar más alta.

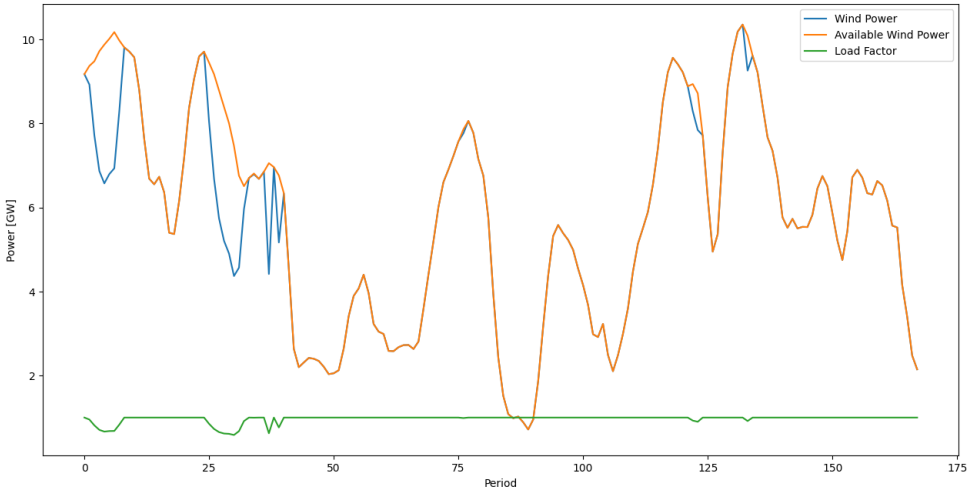


Figura 22. Potencia eólica. Caso base

En el caso del viento, las disponibilidades máximas de potencia se dan en una gran parte durante la noche, donde la demanda es muy baja, incurriendo en vertidos al no poderse aprovechar la energía.

Por último, el valor de la función objetivo es de 15.8 M€. El valor absoluto de la función objetivo no es importante, dado que los valores utilizados en el modelo no son reales. Lo importante es la relación entre el valor en el caso base y el valor en los casos que se muestran a continuación, ya que permite un análisis comparativo entre ellos

Incorporación de baterías

Con el avance tanto en la tecnología de baterías a gran escala como en el nivel de penetración de las energías renovables en el sistema eléctrico, la incorporación de baterías de tamaño industrial a la red son un tema de estudio en auge. Al poner baterías a disposición del sistema, se le da más capacidad para almacenar energía en periodos de baja demanda y utilizarla en los picos. Esto es especialmente útil, como ya se ha visto en el estudio del caso base, para amortiguar los efectos de la naturaleza impredecible e incontrolable de las energías renovables.

Para este pequeño análisis se han incluido 3 baterías con diferente capacidad y velocidad de carga y descarga. Se han utilizado una batería pequeña, de poca capacidad pero mucha potencia de carga y descarga, una batería intermedia y una batería grande pero lenta. Las características específicas de cada batería se muestran en la tabla 2.

Batería	η^c	η^{disc}	E_0	E_{min}	E_{max}	P^c_{max}	P^{disc}_{max}
SMALL	0.9	0.9	0	0	2	1	1
MEDIUM	0.9	0.9	0	0	3	0.75	0.75
LARGE	0.9	0.9	0	0	4	0.5	0.5

Tabla 2. Parámetros de las baterías

El efecto de las baterías se puede apreciar en varios aspectos. En primer lugar, permite que haya un aprovechamiento mucho mayor de la energía eólica y solar, tal y como se muestra en las figuras 23 y 24.

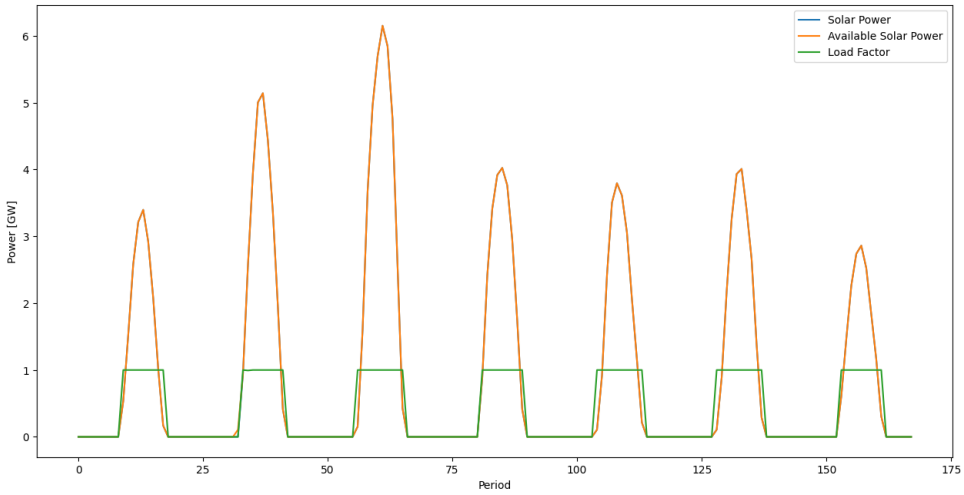


Figura 23. Potencia solar. Caso con baterías

La generación solar es aprovechada al máximo en todos los periodos, ya no existen vertidos durante el primer valle de demanda. Esto en parte se debe a que las baterías empiezan la semana descargadas, por lo que tienen margen para almacenar energía.

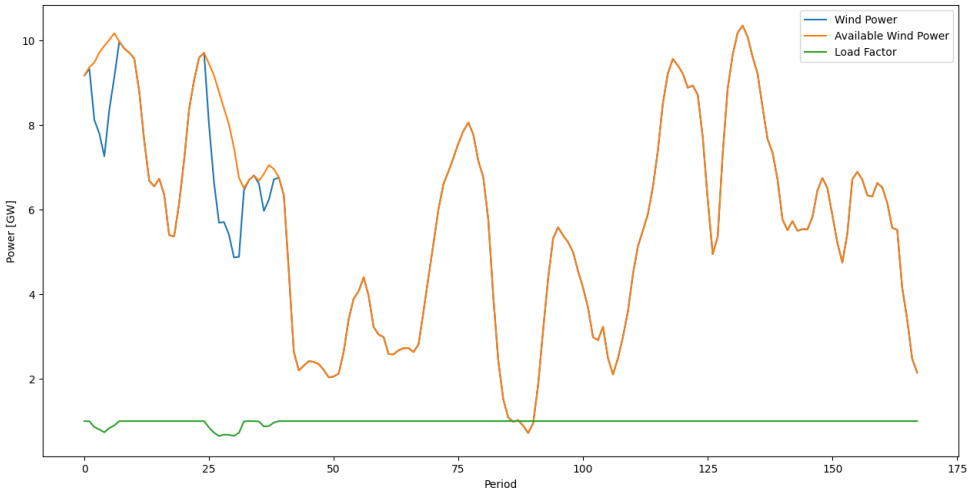


Figura 24. Potencia eólica. Caso con baterías.

En el caso de la producción eólica, aunque se consiga aumentar el factor de carga, no es posible aprovechar toda la potencia disponible ya que las baterías tienen limitaciones en la potencia de carga y descarga. Durante los periodos en los que se están produciendo vertidos las baterías están cargándose al máximo de su capacidad. Además de permitir aprovechar la energía renovable que no se podría aprovechar de otra forma, las baterías tienen otro efecto interesante sobre el sistema. Como se puede apreciar en la figura 25 durante los pequeños valles entre las dos mitades de un pico de demanda, como el que hay entre los periodos 130 y 135 aproximadamente, se almacena energía en las baterías aunque se esté en un pico de demanda.

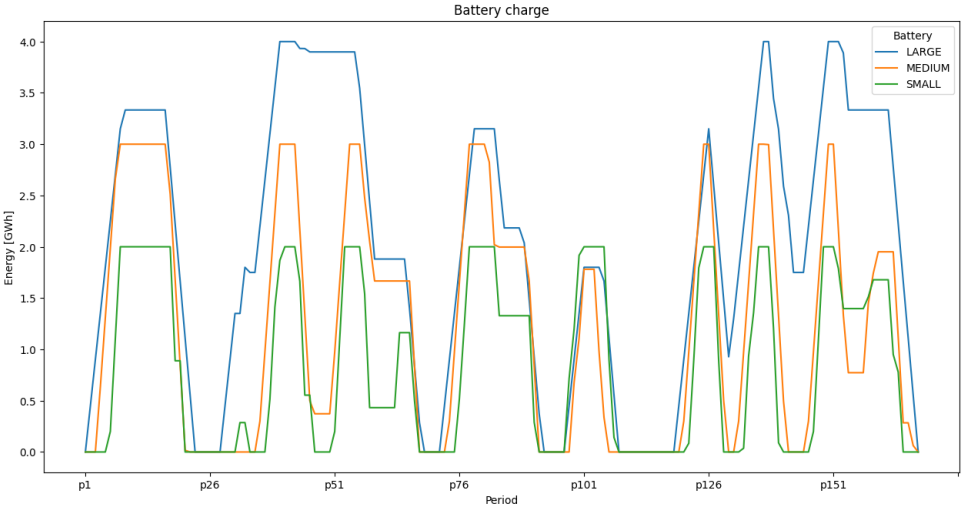


Figura 25. Carga en las baterías. Caso con baterías

Esto se debe a que para satisfacer la demanda en los dos picos que rodean ese pequeño valle es necesario que haya una gran cantidad de grupos conectados. El sistema aprovecha esta situación de unos pocos periodos en la que hay muchos grupos produciendo a una fracción de su potencia nominal para almacenar energía.

Otro de los efectos relacionados con el aprovechamiento de la producción de renovables es que muchos de los periodos que antes tenían precio marginal 0, ahora tienen precios marginales positivos. Como se puede observar en la 26 ya no hay tanta energía siendo desperdiciada y aumentar la demanda supondría un coste para el sistema.

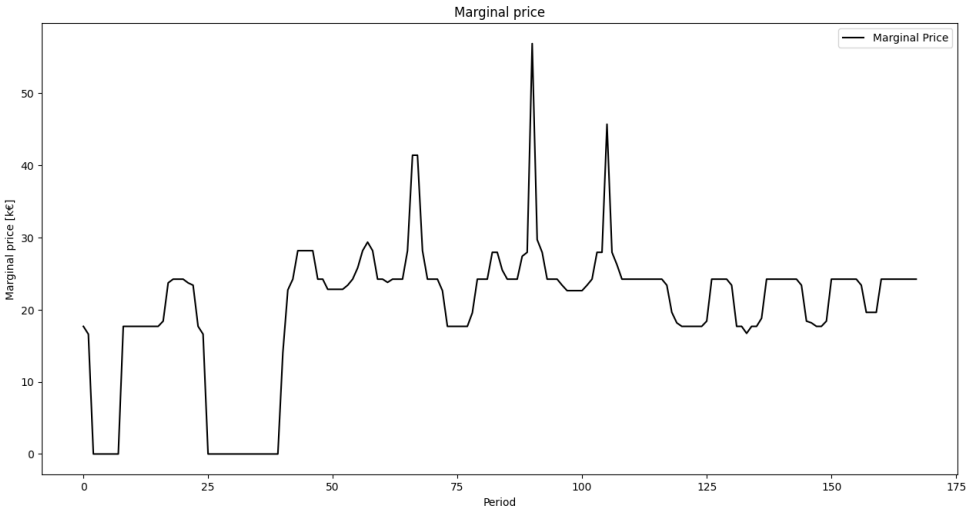


Figura 26. Precio marginal por periodo. Caso con baterías

Nótese que el precio marginal representa el comportamiento diferencial del sistema frente a las variaciones de demanda, aumentarlo no implica necesariamente el aumento del coste de operación. En este caso, al haber dado más libertad al sistema para satisfacer la demanda, el coste precisamente se reduce a 14,82 M€.

Emisiones

Para disminuir la cantidad de contaminación generada por empresas tanto energéticas como industriales, en 2005 se puso en marcha el mercado de emisiones. Este mercado consiste en que cada empresa recibe una cierta cantidad de derechos de emisiones, generar mas CO_2 que la cantidad sobre la que tienes derecho conlleva una sanción. Las empresas ante esta situación tienen dos opciones, o invertir en realizar su actividad producida de forma menos contaminante, o ir a este mercado de emisiones a comprarle sus derechos a empresas que no los necesitan. De esta forma se consigue desincentivar la contaminación e incentivar la inversión en métodos limpios de producción. Contaminar menos permite comprar menos derechos e incluso venderlos. La cantidad de derechos emitida disminuye lentamente cada año, para permitir que las empresas vayan puedan ir invirtiendo para reducir sus emisiones.

Para permitir modelar las emisiones, tal y como se ha mostrado en la formulación del problema, se ha incorporado un parámetro en los generadores que marca las toneladas de CO_2 que emite por cada mega termia que consume. En la función objetivo se suma el coste por derechos de emisión, que es el producto de las emisiones generadas y el precio de los derechos, en este caso se ha fijado en 40€/ton.

Al penalizar las emisiones se consigue que la generación por carbón sea sustituida por ciclos combinados de gas, que son menos contaminantes. Este efecto se aprecia especialmente en el gráfico de generación desglosado por tecnología, que se muestra en la figura 27.

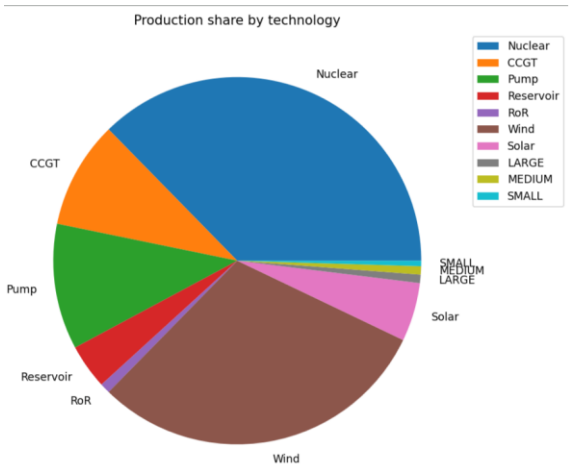


Figura 27. Energía total desglosada por tecnología. Caso con emisiones

El otro efecto notable de incorporar emisiones es el aumento de tanto el precio marginal como del coste total del sistema. La nueva gráfica de coste marginal se muestra en la figura 28, la función objetivo toma un valor de 23,958 M€.

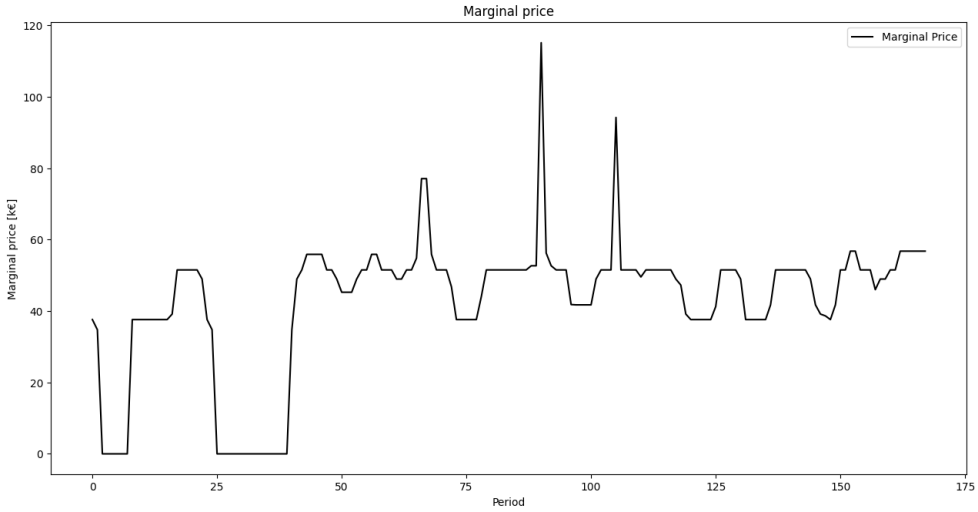


Figura 28. Precio marginal por periodo. Caso con emisiones

6 Conclusiones y futuros desarrollos

En este capítulo final se pretende determinar si los objetivos del proyecto han sido cumplidos satisfactoriamente durante el desarrollo del mismo y que posibles ampliaciones y modificaciones se pueden hacer en un futuro partiendo del trabajo realizado. Además se exponen las conclusiones extraídas a lo largo del desarrollo del proyecto.

En primer lugar, el objetivo de este proyecto era el de crear un software en Python-Pyomo que permitiese la resolución de un problema de Unit Commitment y su ejecución mediante una interfaz web. Para ello se fijaron 3 metas a alcanzar durante el desarrollo: Implemetación de un software de resolución de Unit Commitment, despliegue en la nube y verificación y demostración de la herramienta.

En cuanto a la primera meta, el software creado es capaz de resolver el problema de Unit Commitment de manera eficaz, mostrando de manera intuitiva los resultados al usuario. Permite analizar modelos de sistemas eléctricos y la influencia de diferentes factores sobre los mismos, por lo que en cuanto a la resolución del problema de Unit Commitment, se considera que el objetivo ha sido alcanzado.

Con respecto al segundo y principal objetivo de este proyecto, la implementación en la nube del problema de Unit Commitment, no se considera que haya sido cumplido de manera satisfactoria. Se ha conseguido desarrollar una metodología efectiva para configurar sesiones que permitan el uso de la herramienta en dos plataformas diferentes: Colab y Binder, ambas basadas en Jupyter Notebook. Adicionalmente, se ha diseñado y programado una interfaz de usuario gráfica e interactiva para poder controlar el software de resolución de Unit Commitment. No obstante, debido a las limitaciones en la licencia académica de Gurobi, no ha sido posible resolver problemas de tamaño razonable en un tiempo aceptable. Por tanto no se considera que este objetivo en concreto haya sido cumplido de manera totalmente satisfactoria.

Por último, se ha hecho un pequeño caso de estudio para demostrar las capacidades de la herramienta diseñadam, por lo que esta meta si si considera alcanzada con éxito.

Las conclusiones extraídas a lo largo del transcurso del proyecto son las siguientes:

- Python representa una opción muy atractiva para la implementación de problemas de optimización, gracias a la librería Pyomo y su gran versatilidad como lenguaje de programación.

- La resolución mediante interfaz web es una herramienta efectiva y con gran potencial. Aunque para este proyecto en concreto no haya resultado idónea, representa una opción a considerar tanto para problemas de mayor como de menor tamaño que el estudiado.

  En el caso de problemas a escala real, con mayor número de elementos y restricciones en el sistema, si se recurre a licencias comerciales sin restricciones y a un servicio de cloud computing de alta capacidad computacional, la implementación web permite la resolución de porblemas de optimización sin la necesidad de inversión en infraestructura propia, volviéndola económicamente muy atractiva.

  En el caso de problemas más pequeños, que solo sirvan como modelo ilustrativo para fines no comerciales, la implementación mediante una plataforma como Binder permite eliminar la necesidad de que el usuario final tenga conocimientos sobre la herramienta en sí. Esto la vuelve especialmente atractiva en ambientes académicos, aunque sigue existiendo cierta limitación en cuanto a la capacidad computacional de la plataforma.

En cuanto a posibles futuros proyectos derivados de este, debido a la importancia de la gestión del sistema eléctrico en una sociedad industrial moderna y al aumento constante de

recursos disponibles en el mundo del cloud computing, existen multitud de posibles proyectos que utilicen este como punto de partida. A continuación se da una breve lista sobre algunos de estos posibles proyectos.

- Implementación web de la herramienta mediante contenedores alojados en un servidor. En vez de utilizar una plataforma ya existente como Binder o Colab, diseñar y configurar una plataforma propia y alojarla en un servidor para así tener un mayor control sobre el entorno
- Ampliación de la formulación del problema. Es posible extender el problema incluyendo comportamientos complejos como efectos causados por la red (pérdidas, control de tensiones, etc) modelos estocásticos para ageneraciónde fuentes renovables o representación más exacta de fenómenos no lieneales como el comportamiento real de generadores hidroeléctricos o cuencas hidrológicas.
- Modelado y análisis de sistemas complejos reales por medio de la herramienta. De manera similar pero mas extensa y exhaustiva que el análisis demostrativo que se ha realizado en el proyecto

Bibliografía

[1] Chowdhury, B. H., & Rahman, S. (1990). A review of recent advances in economic dispatch. IEEE Transactions on Power Systems, 5(4), 1248–1259.

[2] Morales-Espana, Latorre, J. M., & Ramos, A. (2013). Tight and Compact MILP Formulation for the Thermal Unit Commitment Problem. IEEE Transactions on Power Systems, 28(4), 4897–4908.

[3] Li, T., & Shahidehpour, M. (2005). Price-Based Unit Commitment: A Case of Lagrangian Relaxation Versus Mixed Integer Programming. IEEE Transactions on Power Systems, 20(4), 2015–2025.

[4] Mallipeddi, R., & Suganthan, P. N. (2014). Unit commitment - a survey and comparison of conventional and nature inspired algorithms. International Journal of Bio-Inspired Computation, 6(2), 71.

[5] Morales, J. M., Conejo, A. J., & Perez-Ruiz, J. (2009). Economic Valuation of Reserves in Power Systems With High Penetration of Wind Power. IEEE Transactions on Power Systems, 24(2), 900–910.

[6] Madraswala, H. S., & Deshpande, A. S. (2016). Genetic algorithm solution to unit commitment problem. 2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES).

[7] Kazarlis, S. A., Bakirtzis, A. G., & Petridis, V. (1996). A genetic algorithm solution to the unit commitment problem. IEEE Transactions on Power Systems, 11(1), 83–92.

[8] Virmani, S., Adrian, E. C., Imhof, K., & Mukherjee, S. (1989). Implementation of a Lagrangian Relaxation Based Unit Commitment Problem. IEEE Power Engineering Review, 9(11), 34–34.

[9] Bixby, R., & Rothberg, E. (2007). Progress in computational mixed integer programming—A look back from the other side of the tipping point. Annals of Operations Research, 149(1)

[10] Binder Project. Recuperado de https://jupyter.org/binder el 17 de julio de 2023.

[11] Hidalgo, I. G., Fontane, D. G., Lopes, J. E. G., Andrade, J. G. P., & de Angelis, A. F. (2014). Efficiency Curves for Hydroelectric Generating Units. Journal of Water Resources Planning and Management

[12] Anderson, D., Moggridge, H., Warren, P., & Shucksmith, J. (2014). The impacts of "run-of-river" hydropower on the physical and ecological condition of rivers. Water and Environment Journal, 29(2), 268–276.

[13] Česonienė, Dapkienė, M., & Punys, P. (2021). Assessment of the Impact of Small Hydropower Plants on the Ecological Status Indicators of Water Bodies: A Case Study in Lithuania. Water, 13(4), 433.

[14] García-González, J (2014). Decision support models in electric power systems. Instituto de Investigación Tecnológica.

Anexo I. Alineación con los objetivos de desarrollo sostenible

El desarrollo de un solver de Unit Commitment desplegado en la nube con fines didácticos y académicos se alinea con varios de los Objetivos de Desarrollo Sostenible (ODS) de las Naciones Unidas, entre ellos:

- ODS 7: Energía asequible y no contaminante: El solver de Unit Commitment tiene como objetivo la optimización de los sistemas eléctricos lo que puede ayudar a reducir los costos de energía y promover el acceso a una energía asequible y no contaminante.
- ODS 4: Educación de calidad: El software permitirá a los alumnos centrarse en el aprendizaje sobre el problema de Unit Commitment en vez de en el uso de herramientas específicas como GAMS lo que contribuye a la mejora de la educación en el campo de la ingeniería eléctrica y la energía renovable.
- ODS 9: Industria, innovación e infraestructura: El desarrollo de un solver de Unit Commitment implica el uso de tecnología de manera eficiente y el fortalecimiento de la infraestructura para la operación de los sistemas eléctricos de potencia.
- ODS 13: Acción por el clima: La implementación de soluciones más eficientes y sostenibles para la operación de los sistemas eléctricos contribuye a la reducción de las emisiones de gases de efecto invernadero y por tanto la mitigación del cambio climático. En general, el desarrollo de un solver de Unit Commitment desplegado en la nube tiene el potencial de contribuir a varios ODS y promover la sostenibilidad y el desarrollo en el campo de la energía eléctrica.

Anexo II. Datos utilizados para los modelos

Módelo MSEM

Parámetros de generadores

g	p_α	p_β	p_γ	p_θ	p_{rs}	p_{rb}	p_f	p_o
NUCLEAR	1	0	0	0	1	1	3.5	1.2
LIGNITE	3	0.015	2	0.2	0.0376	0.0376	8	2.4
SUBBITUMIN	2.6	0.03	2	0.2	0.0443	0.0443	8.5	1.8
BITUMINOUS	2.3	0.035	1.4	0.14	0.0403	0.0403	8	1.2
ANTHRACITE	2.2	0.06	1.9	0.19	0.1184	0.1184	7	1.2
CCGT	1.3	0.09	1.1	0.11	0.0653	0.0653	20	1.2
FUELOIL	2.1	0.08	0.07	0.007	0.1253	0.1253	23	1.2
GAS	2	0.09	0.11	0.011	0.0752	0.0752	20	2
HYDRO_RES	0	0	0	0	0	0	0	0
HYDRO_ROR	0	0	0	0	0	0	0	0
HYDRO_PUM	0	0	0	0	0	0	0	0

Tabla 3. Parámetros 1-8

g	p_{qmax}	p_{qmin}	p_{u0}	p_{mode}	p_{bmax}	p_{wmax}	p_{w0}	p_{wmin}
NUCLEAR	1	1	1	0	0	0	0	0
LIGNITE	0.35	0.23	1	0	0	0	0	0
SUBBITUMIN	0.35	0.21	1	0	0	0	0	0
BITUMINOUS	0.35	0.22	1	0	0	0	0	0
ANTHRACITE	0.55	0.18	0	0	0	0	0	0
CCGT	0.4	0.2	0	0	0	0	0	0
FUELOIL	0.54	0.14	1	0	0	0	0	0
GAS	0.38	0.14	0	0	0	0	0	0
HYDRO_RES	0.5	0.05	0	0	0	0.1	5000	3000
HYDRO_ROR	0.5	0.1	0	0	0	0	0	0
HYDRO_PUM	0.2	0	0	0	0	0.2	30	15

Tabla 4. Parámetros 9-16

g	p_{wfin}	p_k	p_{rend}	p_{type}	p_{α_e}	p_{β_e}	p_{γ_e}	p_{tech}
NUCLEAR	0	0.95	0	thermal	0	0	0	nuclear
LIGNITE	0	0.94	0	thermal	0	0	0	coal
SUBBITUMIN	0	0.95	0	thermal	0	0	0	coal
BITUMINOUS	0	0.93	0	thermal	0	0	0	coal
ANTHRACITE	0	0.96	0	thermal	0	0	0	coal
CCGT	0	0.98	0	thermal	0	0	0	ccgt
FUELOIL	0	0.94	0	thermal	0	0	0	oil
GAS	0	0.94	0	thermal	0	0	0	gas
HYDRO_RES	1000	2975	1	hydro	0	0	0	hydro
HYDRO_ROR	0	0	1	hydro	0	0	0	hydro
HYDRO_PUM	0	15	1	hydro	0	0	0	hydro

Tabla 5. Parámetros 17-24

Parámetros del sistema

$cens$	c_e	$factor_{solar}$	$factor_{wind}$	$factor_{rod}$
180	40	1	1	0.1

Tabla 6. Parámetros del sistema

Demanda

p	d	p	d	p	d	p	d
p1	3.36905	p43	3.46625	p85	4.2113	p127	3.42785
p2	2.9486	p44	3.7463	p86	3.86705	p128	3.88865
p3	2.70995	p45	3.88595	p87	3.87035	p129	4.0538
p4	2.5415	p46	3.72215	p88	3.86075	p130	4.26155
p5	2.4263	p47	3.69215	p89	3.9113	p131	4.2587
p6	2.42795	p48	3.2843	p90	3.9962	p132	4.20425
p7	2.45525	p49	2.84855	p91	4.42565	p133	4.01795
p8	2.7242	p50	2.5412	p92	4.4312	p134	3.68495
p9	3.1802	p51	2.3633	p93	4.2344	p135	3.6956
p10	3.63695	p52	2.29925	p94	3.87515	p136	3.6845
p11	3.79775	p53	2.3549	p95	3.79775	p137	3.7454
p12	3.782	p54	2.82755	p96	3.54725	p138	3.782
p13	3.7613	p55	3.4097	p97	3.16985	p139	4.1927
p14	3.5381	p56	3.94535	p98	2.8427	p140	4.2263
p15	3.37895	p57	4.22225	p99	2.68955	p141	4.08935
p16	3.293	p58	4.4846	p100	2.60435	p142	3.7829
p17	3.26405	p59	4.55	p101	2.62025	p143	3.7079
p18	3.35405	p60	4.5359	p102	2.95985	p144	3.55205
p19	3.8687	p61	4.3568	p103	3.4544	p145	3.1655
p20	4.0367	p62	4.0127	p104	3.9152	p146	2.8421
p21	4.04705	p63	4.0277	p105	4.15175	p147	2.6918
p22	3.82445	p64	3.9869	p106	4.35995	p148	2.6027
p23	3.8597	p65	4.019	p107	4.3745	p149	2.6144
p24	3.5165	p66	4.06475	p108	4.3442	p150	2.9504
p25	3.0743	p67	4.45895	p109	4.1717	p151	3.42785
p26	2.64215	p68	4.46	p110	3.84155	p152	3.88865
p27	2.3738	p69	4.2944	p111	3.84215	p153	4.0538
p28	2.1869	p70	3.9284	p112	3.83375	p154	4.26155
p29	2.0813	p71	3.8594	p113	3.89015	p155	4.2587
p30	2.0216	p72	3.69575	p114	3.9182	p156	4.20425
p31	1.91555	p73	3.21785	p115	4.301	p157	4.01795
p32	1.95485	p74	2.8607	p116	4.32485	p158	3.68495
p33	2.23025	p75	2.69435	p117	4.139	p159	3.6956
p34	2.65145	p76	2.59715	p118	3.815	p160	3.6845
p35	2.90285	p77	2.6198	p119	3.74795	p161	3.7454
p36	2.97545	p78	2.94695	p120	3.55205	p162	3.782
p37	2.97485	p79	3.43625	p121	3.1655	p163	4.1927
p38	2.94215	p80	3.93335	p122	2.8421	p164	4.2263
p39	2.8121	p81	4.17425	p123	2.6918	p165	4.08935
p40	2.73935	p82	4.38275	p124	2.6027	p166	3.782
p41	2.70125	p83	4.424	p125	2.6144	p167	3.7079
p42	2.88635	p84	4.39385	p126	2.9504	p168	3.63305

Tabla 7. Demanda

Energía solar

p	$\overline{solar}$	p	$\overline{solar}$	p	$\overline{solar}$	p	$\overline{solar}$
p1	0	p43	0	p85	0.256	p127	0
p2	0	p44	0	p86	0.263	p128	0
p3	0	p45	0	p87	0.246	p129	0.007
p4	0	p46	0	p88	0.193	p130	0.058
p5	0	p47	0	p89	0.114	p131	0.142
p6	0	p48	0	p90	0.027	p132	0.213
p7	0	p49	0	p91	0	p133	0.257
p8	0	p50	0	p92	0	p134	0.262
p9	0	p51	0	p93	0	p135	0.222
p10	0.035	p52	0	p94	0	p136	0.174
p11	0.1	p53	0	p95	0	p137	0.091
p12	0.169	p54	0	p96	0	p138	0.019
p13	0.21	p55	0	p97	0	p139	0
p14	0.222	p56	0	p98	0	p140	0
p15	0.191	p57	0.01	p99	0	p141	0
p16	0.136	p58	0.105	p100	0	p142	0
p17	0.067	p59	0.24	p101	0	p143	0
p18	0.011	p60	0.324	p102	0	p144	0
p19	0	p61	0.372	p103	0	p145	0
p20	0	p62	0.402	p104	0	p146	0
p21	0	p63	0.382	p105	0.007	p147	0
p22	0	p64	0.311	p106	0.06	p148	0
p23	0	p65	0.168	p107	0.158	p149	0
p24	0	p66	0.028	p108	0.229	p150	0
p25	0	p66	0	p109	0.1954	p151	0.3835
p26	0	p67	0	p110	0.236	p152	0
p27	0	p68	0	p111	0.2	p153	0
p28	0	p69	0	p112	0.135	p154	0.04
p29	0	p70	0	p113	0.076	p155	0.096
p30	0	p71	0	p114	0.014	p156	0.148
p31	0	p72	0	p115	0	p157	0.179
p32	0	p73	0	p116	0	p158	0.187
p33	0.007	p74	0	p117	0	p159	0.165
p34	0.061	p75	0	p118	0	p160	0.12
p35	0.168	p76	0	p119	0	p161	0.076
p36	0.258	p77	0	p120	0	p162	0.02
p37	0.327	p78	0	p121	0	p163	0
p38	0.336	p79	0	p122	0	p164	0
p39	0.29	p80	0	p123	0	p165	0
p40	0.22	p81	0	p124	0	p166	0
p41	0.128	p82	0.058	p125	0	p167	0
p42	0.027	p83	0.158	p126	0	p168	0

Tabla 8. Energía solar disponible

Energía eólica

p	$\overline{wind}$	p	$\overline{wind}$	p	$\overline{wind}$	p	$\overline{wind}$
p1	0.5994	p43	0.1721	p85	0.0991	p127	0.3233
p2	0.6122	p44	0.1437	p86	0.0707	p128	0.3505
p3	0.6194	p45	0.1510	p87	0.0643	p129	0.4778
p4	0.6351	p46	0.1581	p88	0.0669	p130	0.5791
p5	0.6453	p47	0.1568	p89	0.0580	p131	0.6312
p6	0.6544	p48	0.1534	p90	0.0468	p132	0.6654
p7	0.6648	p49	0.1444	p91	0.0622	p133	0.6765
p8	0.6518	p50	0.1329	p92	0.1220	p134	0.6592
p9	0.6410	p51	0.1342	p93	0.2068	p135	0.6282
p10	0.6346	p52	0.1389	p94	0.2858	p136	0.6020
p11	0.6256	p53	0.1725	p95	0.3477	p137	0.5502
p12	0.5747	p54	0.2229	p96	0.3648	p138	0.5011
p13	0.4986	p55	0.2545	p97	0.3521	p139	0.4802
p14	0.4368	p56	0.2661	p98	0.3416	p140	0.4387
p15	0.4281	p57	0.2876	p99	0.3263	p141	0.3767
p16	0.4398	p58	0.2589	p100	0.2971	p142	0.3604
p17	0.4156	p59	0.2109	p101	0.2715	p143	0.3744
p18	0.3528	p60	0.1990	p102	0.2406	p144	0.3595
p19	0.3507	p61	0.1953	p103	0.1949	p145	0.3619
p20	0.4021	p62	0.1690	p104	0.1905	p146	0.3616
p21	0.4688	p63	0.1686	p105	0.2110	p147	0.3807
p22	0.5473	p64	0.1748	p106	0.1627	p148	0.4216
p23	0.5921	p65	0.1780	p107	0.1374	p149	0.4410
p24	0.6269	p66	0.1782	p108	0.1623	p150	0.4252
p25	0.6344	p67	0.1720	p109	0.1954	p151	0.3835
p25	0.6344	p67	0.1720	p109	0.1954	p151	0.3835
p26	0.6173	p68	0.1836	p110	0.2350	p152	0.3408
p27	0.5993	p69	0.2345	p111	0.2922	p153	0.3103
p28	0.5740	p70	0.2879	p112	0.3354	p154	0.3551
p29	0.5489	p71	0.3396	p113	0.3597	p155	0.4390
p30	0.5231	p72	0.3927	p114	0.3850	p156	0.4506
p31	0.4878	p73	0.4314	p115	0.4270	p157	0.4385
p32	0.4414	p74	0.4510	p116	0.4832	p158	0.4143
p33	0.4252	p75	0.4721	p117	0.5558	p159	0.4122
p34	0.4378	p76	0.4946	p118	0.6023	p160	0.4332
p35	0.4449	p77	0.5135	p119	0.6249	p161	0.4265
p36	0.4365	p78	0.5267	p120	0.6149	p162	0.4028
p37	0.4476	p79	0.5081	p121	0.6024	p163	0.3639
p38	0.4610	p80	0.4674	p122	0.5805	p164	0.3609
p39	0.4548	p81	0.4417	p123	0.5840	p165	0.2715
p40	0.4419	p82	0.3762	p124	0.5696	p166	0.2221
p41	0.4141	p83	0.2575	p125	0.5042	p167	0.1618
p42	0.2963	p84	0.1589	p126	0.4079	p168	0.1403

Tabla 9. Energía eólica disponible

Módelo MESP

Parámetros de generadores

g	α	β	γ	θ	rs	rb	f	o
ARCOS	1.01	0.11	1.31	0.01	0.21	0.21	25	1.4
BAHÍA DE ALGECIRAS	1.19	0.1	0.95	0.01	0.12	0.12	25	1.4
CAMPO DE GIBRALTAR	1.49	0.07	1.17	0.01	0.11	0.11	25	1.4
SAN ROQUE	1.53	0.1	1.19	0.01	0.1	0.1	25	1.4
LOS BARRIOS	2.39	0.04	1.62	0.2	0.06	0.06	8	1.44
CRISTOBAL COLÓN	1.45	0.06	0.94	0.01	0.08	0.08	25	1.4
PALOS DE LA FRONTERA	1.04	0.09	1.3	0.01	0.23	0.23	25	1.4
CAMPANILLAS	1.08	0.09	0.82	0.01	0.05	0.05	25	1.4
CASTELNOU ENERGÍA	1.03	0.12	1.38	0.01	0.1	0.1	25	1.4
ESCATRÓN	1.44	0.08	1.2	0.01	0.14	0.14	25	1.4
ESCATRÓN PEAKER	1.34	0.11	1.12	0.01	0.04	0.04	25	1.4
ABOÑO	2.89	0.01	2.0	0.2	0.1	0.1	8	1.01
LA PEREDA	2.4	0.02	1.87	0.19	0.01	0.01	8	1.31
SOTO DE RIBERA	1.26	0.1	1.32	0.01	0.22	0.22	25	1.4
ES MURTERAR	2.15	0.04	1.57	0.2	0.03	0.03	8	1.0
CAS TRESORER	1.26	0.11	1.08	0.01	0.06	0.06	25	1.4
SONR EUS	1.36	0.11	1.08	0.01	0.11	0.11	25	1.4
MAHÓN	1.93	0.09	0.12	0.07	0.1	0.1	23	1.01
IBIZA	2.17	0.09	0.1	0.08	0.11	0.11	23	1.13
ACECA	1.6	0.07	1.19	0.01	0.16	0.16	25	1.4
PUERTO DE BARCELONA	1.09	0.09	1.16	0.01	0.14	0.14	25	1.4
BESÓS	1.07	0.07	1.38	0.01	0.11	0.11	25	1.4
BESÓS V	1.38	0.08	1.29	0.01	0.14	0.14	25	1.4
TARRAGONA POWER	1.49	0.09	0.86	0.01	0.08	0.08	25	1.4
PLANA DEL VENT	1.07	0.1	0.87	0.01	0.11	0.11	25	1.4
CEUTA	2.15	0.11	0.12	0.06	0.03	0.03	23	1.08
PUENTES DE GCÍA. RDGUEZ.	2.7	0.04	1.93	0.17	0.24	0.24	8	1.3
SABÓN	1.25	0.12	1.03	0.01	0.08	0.08	25	1.4
MELILLA	2.24	0.1	0.1	0.09	0.03	0.03	23	1.1
EL FANGAL	1.58	0.07	1.29	0.01	0.23	0.23	25	1.4
ESCOMBRERAS	1.54	0.09	0.94	0.01	0.14	0.14	25	1.4
CC CARTAGENA	1.28	0.09	1.39	0.01	0.22	0.22	25	1.4
CASTEJÓN	1.0	0.11	1.14	0.01	0.15	0.15	25	1.4
CASTEJÓN 2	1.3	0.1	0.83	0.01	0.05	0.05	25	1.4
BAHÍA DE BIZKAIA	1.56	0.07	1.18	0.01	0.15	0.15	25	1.4
BOROA	1.02	0.07	1.29	0.01	0.14	0.14	25	1.4
SANTURCE	1.29	0.1	1.0	0.01	0.08	0.08	25	1.4
ARRÚBAL	1.23	0.11	1.1	0.01	0.16	0.16	25	1.4
CASTELLÓN	1.35	0.11	1.29	0.01	0.27	0.27	25	1.4
SAGUNTO	1.27	0.07	1.19	0.01	0.24	0.24	25	1.4
ALMARAZ I	0.89	0.0	0.0	1.0	1.0	1.0	25	1.09
ALMARAZ II	0.85	0.0	0.0	1.0	1.0	1.0	3.5	1.23
ASCÓ I	1.07	0.0	0.0	1.0	1.0	1.0	3.5	1.22
ASCO II	0.81	0.0	0.0	1.0	1.0	1.0	3.5	1.18
COFRENTES	0.95	0.0	0.0	1.0	1.0	1.0	3.5	1.28
VANDELLÓS II	0.93	0.0	0.0	1.0	1.0	1.0	3.5	1.16
TRILLO	0.89	0.0	0.0	1.0	1.0	1.0	3.5	1.18
ALDEADÁVILA	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
JOSÉ MARÍA DE ORIOL	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
VILLARINO	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
CORTES-LA MUELA	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SAUCELLE	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
CEDILLO	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ESTANY-GENTO SALLENTE	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
TAJO DE LA ENCANTADA	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AGUAYO	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
MEQUINENZA	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tabla 10. Parámetros 1-8

g	q_{max}	q_{min}	u_0	mode	b_{max}	w_{max}	w_0	w_{min}
ARCOS	1.613	0.79	1	0	0.0	0.0	0.0	0.0
BAHÍA DE ALGECIRAS	0.8	0.38	0	0	0.0	0.0	0.0	0.0
CAMPO DE GIBRALTAR	0.789	0.37	0	0	0.0	0.0	0.0	0.0
SAN ROQUE	0.8	0.4	1	0	0.0	0.0	0.0	0.0
LOS BARRIOS	0.589	0.31	1	0	0.0	0.0	0.0	0.0
CRISTOBAL COLÓN	0.4	0.21	1	0	0.0	0.0	0.0	0.0
PALOS DE LA FRONTERA	1.2	0.65	0	0	0.0	0.0	0.0	0.0
CAMPANILLAS	0.4	0.12	1	0	0.0	0.0	0.0	0.0
CASTELNOU ENERGÍA	0.79	0.35	0	0	0.0	0.0	0.0	0.0
ESCATRÓN	0.804	0.43	1	0	0.0	0.0	0.0	0.0
ESCATRÓN PEAKER	0.285	0.1	0	0	0.0	0.0	0.0	0.0
ABOÑO	0.916	0.51	0	0	0.0	0.0	0.0	0.0
LA PEREDA	0.05	0.03	1	0	0.0	0.0	0.0	0.0
SOTO DE RIBERA	1.216	0.7	1	0	0.0	0.0	0.0	0.0
ES MURTERAR	0.335	0.17	1	0	0.0	0.0	0.0	0.0
CAS TRESORER	0.477	0.27	0	0	0.0	0.0	0.0	0.0
SONR EUS	0.611	0.1	0	0	0.0	0.0	0.0	0.0
MAHÓN	0.27	0.1	0	0	0.0	0.0	0.0	0.0
IBIZA	0.27	0.41	0	0	0.0	0.0	0.0	0.0
ACECA	0.8	0.44	0	0	0.0	0.0	0.0	0.0
PUERTO DE BARCELONA	0.85	0.39	1	0	0.0	0.0	0.0	0.0
BESÓS	0.873	0.28	0	0	0.0	0.0	0.0	0.0
BESÓS V	0.8	0.22	1	0	0.0	0.0	0.0	0.0
TARRAGONA POWER	0.424	0.26	1	0	0.0	0.0	0.0	0.0
PLANA DEL VENT	0.8	0.05	0	0	0.0	0.0	0.0	0.0
CEUTA	0.1	0.04	1	0	0.0	0.0	0.0	0.0
PUENTES DE GCÍA. RDGUEZ.	2.338	0.73	0	0	0.0	0.0	0.0	0.0
SABÓN	0.4	0.19	1	0	0.0	0.0	0.0	0.0
MELILLA	0.085	0.05	0	0	0.0	0.0	0.0	0.0
EL FANGAL	1.2	0.7	1	0	0.0	0.0	0.0	0.0
ESCOMBRERAS	0.831	0.33	1	0	0.0	0.0	0.0	0.0
CC CARTAGENA	1.2	0.7	0	0	0.0	0.0	0.0	0.0
CASTEJÓN	0.842	0.26	0	0	0.0	0.0	0.0	0.0
CASTEJÓN 2	0.386	0.14	0	0	0.0	0.0	0.0	0.0
BAHÍA DE BIZKAIA	0.795	0.4	1	0	0.0	0.0	0.0	0.0
BOROA	0.755	0.35	1	0	0.0	0.0	0.0	0.0
SANTURCE	0.402	0.16	0	0	0.0	0.0	0.0	0.0
ARRÚBAL	0.8	0.32	0	0	0.0	0.0	0.0	0.0
CASTELLÓN	1.65	0.69	1	0	0.0	0.0	0.0	0.0
SAGUNTO	1.2	0.44	0	0	0.0	0.0	0.0	0.0
ALMARAZ I	1.0494	0.4	1	1	0.0	0.0	0.0	0.0
ALMARAZ II	1.0445	0.62	1	1	0.0	0.0	0.0	0.0
ASCÓ I	1.0325	0.43	1	1	0.0	0.0	0.0	0.0
ASCO II	1.02721	0.32	1	1	0.0	0.0	0.0	0.0
COFRENTES	1.09202	0.64	1	1	0.0	0.0	0.0	0.0
VANDELLÓS II	1.08714	0.62	1	1	0.0	0.0	0.0	0.0
TRILLO	1.066	0.36	1	1	0.0	0.0	0.0	0.0
ALDEADÁVILA	1.139	0.0	0	0	0.4	616590	184977	246636
JOSÉ MARÍA DE ORIOL	0.916	0.0	0	0	0.0	24003	7200	9
VILLARINO	0.81	0.0	0	0	0.6	802647	240794	1
CORTES-LA MUELA	1.72	0.0	0	0	1.52	24186	7255	8
SAUCELLE	0.485	0.0	0	0	0.0	22732	6819	8625
CEDILLO	0.5	0.0	0	0	0.0	3412	1023	75
ESTANY-GENTO SALLENTE	0.415	0.0	0	0	0.25	867	260	325
TAJO DE LA ENCANTADA	0.36	0.0	0	0	0.25	191	57	33
AGUAYO	0.36	0.0	0	0	0.0	1717	515	25
MEQUINENZA	0.384	0.0	0	0	0.0	137700	41310	55080

Tabla 11. Parámetros 9-16

g	w_{fin}	k	$rend$	$type$	$emissions$	$tech$	TU	TD
ARCOS	0.0	0.98	0	thermal	0.45	CCGT	3.0	6.0
BAHÍA DE ALGECIRAS	0.0	0.99	0	thermal	0.52	CCGT	3.0	6.0
CAMPO DE GIBRALTAR	0.0	0.98	0	thermal	0.52	CCGT	3.0	6.0
SAN ROQUE	0.0	0.97	0	thermal	0.45	CCGT	3.0	6.0
LOS BARRIOS	0.0	0.93	0	thermal	0.96	Coal	3.0	6.0
CRISTOBAL COLÓN	0.0	0.98	0	thermal	0.46	CCGT	3.0	6.0
PALOS DE LA FRONTERA	0.0	0.99	0	thermal	0.54	CCGT	3.0	6.0
CAMPANILLAS	0.0	0.98	0	thermal	0.4	CCGT	3.0	6.0
CASTELNOU ENERGÍA	0.0	0.98	0	thermal	0.46	CCGT	3.0	6.0
ESCATRÓN	0.0	0.99	0	thermal	0.37	CCGT	3.0	6.0
ESCATRÓN PEAKER	0.0	0.98	0	thermal	0.49	CCGT	3.0	6.0
ABOÑO	0.0	0.96	0	thermal	0.99	Coal	3.0	6.0
LA PEREDA	0.0	0.96	0	thermal	0.99	Coal	3.0	6.0
SOTO DE RIBERA	0.0	0.98	0	thermal	0.54	CCGT	3.0	6.0
ES MURTERAR	0.0	0.97	0	thermal	0.89	Coal	3.0	6.0
CAS TRESORER	0.0	0.98	0	thermal	0.38	CCGT	3.0	6.0
SONR EUS	0.0	0.99	0	thermal	0.51	CCGT	3.0	6.0
MAHÓN	0.0	0.93	0	thermal	0.74	Fueloil	3.0	6.0
IBIZA	0.0	0.95	0	thermal	0.72	Fueloil	3.0	6.0
ACECA	0.0	0.97	0	thermal	0.36	CCGT	3.0	6.0
PUERTO DE BARCELONA	0.0	0.97	0	thermal	0.46	CCGT	3.0	6.0
BESÓS	0.0	0.98	0	thermal	0.46	CCGT	3.0	6.0
BESÓS V	0.0	0.99	0	thermal	0.36	CCGT	3.0	6.0
TARRAGONA POWER	0.0	0.97	0	thermal	0.49	CCGT	3.0	6.0
PLANA DEL VENT	0.0	0.98	0	thermal	0.5	CCGT	3.0	6.0
CEUTA	0.0	0.94	0	thermal	0.73	Fueloil	3.0	6.0
PUENTES DE GCÍA. RDGUEZ.	0.0	0.93	0	thermal	0.89	Coal	3.0	6.0
SABÓN	0.0	0.97	0	thermal	0.51	CCGT	3.0	6.0
MELILLA	0.0	0.93	0	thermal	0.78	Fueloil	3.0	6.0
EL FANGAL	0.0	0.98	0	thermal	0.47	CCGT	3.0	6.0
ESCOMBRERAS	0.0	0.98	0	thermal	0.41	CCGT	3.0	6.0
CC CARTAGENA	0.0	0.98	0	thermal	0.41	CCGT	3.0	6.0
CASTEJÓN	0.0	0.98	0	thermal	0.49	CCGT	3.0	6.0
CASTEJÓN 2	0.0	0.97	0	thermal	0.44	CCGT	3.0	6.0
BAHÍA DE BIZKAIA	0.0	0.97	0	thermal	0.5	CCGT	3.0	6.0
BOROA	0.0	0.99	0	thermal	0.51	CCGT	3.0	6.0
SANTURCE	0.0	0.98	0	thermal	0.5	CCGT	3.0	6.0
ARRÚBAL	0.0	0.98	0	thermal	0.55	CCGT	3.0	6.0
CASTELLÓN	0.0	0.98	0	thermal	0.53	CCGT	3.0	6.0
SAGUNTO	0.0	0.98	0	thermal	0.49	CCGT	3.0	6.0
ALMARAZ I	0.0	0.95	0	thermal	0	Nuclear	48.0	48.0
ALMARAZ II	0.0	0.95	0	thermal	0	Nuclear	48.0	48.0
ASCÓ I	0.0	0.95	0	thermal	0	Nuclear	48.0	48.0
ASCO II	0.0	0.94	0	thermal	0	Nuclear	48.0	48.0
COFRENTES	0.0	0.95	0	thermal	0	Nuclear	48.0	48.0
VANDELLÓS II	0.0	0.96	0	thermal	0	Nuclear	48.0	48.0
TRILLO	0.0	0.95	0	thermal	0	Nuclear	48.0	48.0
ALDEADÁVILA	184920	1	0.76	hydro	0	Pump	0	0
JOSÉ MARÍA DE ORIOL	7190	1	0	hydro	0	Reservoir	0	0
VILLARINO	240750	1	0.71	hydro	0	Pump	0	0
CORTES-LA MUELA	7200	1	0.73	hydro	0	Pump	0	0
SAUCELLE	6750	1	0.0	hydro	0	Reservoir	0	0
CEDILLO	1010	1	0	hydro	0	Reservoir	0	0
ESTANY-GENTO SALLENTE	250	1	0.69	hydro	0	Pump	0	0
TAJO DE LA ENCANTADA	55	1	0.75	hydro	0	Pump	0	0
AGUAYO	505	1	0	hydro	0	Reservoir	0	0
MEQUINENZA	41280	1	0	hydro	0	Reservoir	0	0

Tabla 12. Parámetros 17-24

Parámetros de baterías

b	η_c	η_{disc}	E_0	E_{min}	E_{max}	P^c_{max}	P^{discc}_{max}
SMALL	0.9	0.9	0	0	2	1	1
MEDIUM	0.9	0.9	0	0	3	0.75	0.75
LARGE	0.9	0.9	0	0	4	0.5	0.5

Tabla 13. Parámetros de las baterías. Caso con baterías y baterías con emisiones

Parámetros del sistema

$cens$	c_e	$factor_{solar}$	$factor_{wind}$	$factor_{rod}$
180	40	1	1	0.1

Tabla 14. Parámetros del sistema

Demanda

Tabla 15. Datos de demanda

p	$\overline{d}$	p	$\overline{d}$	p	$\overline{d}$	p	$\overline{d}$
p1	23.58335	p43	24.26375	p85	29.4791	p127	23.99495
p2	20.6402	p44	26.2241	p86	27.06935	p128	27.22055
p3	18.96965	p45	27.20165	p87	27.09245	p129	28.3766
p4	17.7905	p46	26.05505	p88	27.02525	p130	29.83085
p5	16.9841	p47	25.84505	p89	27.3791	p131	29.8109
p6	16.99565	p48	22.9901	p90	27.9734	p132	29.42975
p7	17.18675	p49	19.93985	p91	30.97955	p133	28.12565
p8	19.0694	p50	17.7884	p92	31.0184	p134	25.79465
p9	22.2614	p51	16.5431	p93	29.6408	p135	25.8692
p10	25.45865	p52	16.09475	p94	27.12605	p136	25.7915
p11	26.58425	p53	16.4843	p95	26.58425	p137	26.2178
p12	26.474	p54	19.79285	p96	24.83075	p138	26.474
p13	26.3291	p55	23.8679	p97	22.18895	p139	29.3489
p14	24.7667	p56	27.61745	p98	19.8989	p140	29.5841
p15	23.65265	p57	29.55575	p99	18.82685	p141	28.62545
p16	23.051	p58	31.3922	p100	18.23045	p142	26.4803
p17	22.84835	p59	31.85	p101	18.34175	p143	25.9553
p18	23.47835	p60	31.7513	p102	20.71895	p144	24.86435
p19	27.0809	p61	30.4976	p103	24.1808	p145	22.1585
p20	28.2569	p62	28.0889	p104	27.4064	p146	19.8947
p21	28.32935	p63	28.1939	p105	29.06225	p147	18.8426
p22	26.77115	p64	27.9083	p106	30.51965	p148	18.2189
p23	27.0179	p65	28.133	p107	30.6215	p149	18.3008
p24	24.6155	p66	28.45325	p108	30.4094	p150	20.6528
p25	21.5201	p67	31.21265	p109	29.2019	p151	23.99495
p26	18.49505	p68	31.22	p110	26.89085	p152	27.22055
p27	16.6166	p69	30.0608	p111	26.89505	p153	28.3766
p28	15.3083	p70	27.4988	p112	26.83625	p154	29.83085
p29	14.5691	p71	27.0158	p113	27.23105	p155	29.8109
p30	14.1512	p72	25.87025	p114	27.4274	p156	29.42975
p31	13.40885	p73	22.52495	p115	30.107	p157	28.12565
p32	13.68395	p74	20.0249	p116	30.27395	p158	25.79465
p33	15.61175	p75	18.86045	p117	28.973	p159	25.8692
p34	18.56015	p76	18.18005	p118	26.705	p160	25.7915
p35	20.31995	p77	18.3386	p119	26.23565	p161	26.2178
p36	20.82815	p78	20.62865	p120	24.86435	p162	26.474
p37	20.82395	p79	24.05375	p121	22.1585	p163	29.3489
p38	20.59505	p80	27.53345	p122	19.8947	p164	29.5841
p39	19.6847	p81	29.21975	p123	18.8426	p165	28.62545
p40	19.17545	p82	30.67925	p124	18.2189	p166	26.4803
p41	18.90875	p83	30.968	p125	18.3008	p167	25.9553
p42	20.20445	p84	30.75695	p126	20.6528	p168	25.43135

Energía solar

p	$\overline{solar}$	p	$\overline{solar}$	p	$\overline{solar}$	p	$\overline{solar}$
p1	0.0	p43	0.0	p85	2.304	p127	0.0
p2	0.0	p44	0.0	p86	2.367	p128	0.0
p3	0.0	p45	0.0	p87	2.214	p129	0.063
p4	0.0	p46	0.0	p88	1.737	p130	0.522
p5	0.0	p47	0.0	p89	1.026	p131	1.278
p6	0.0	p48	0.0	p90	0.243	p132	1.917
p7	0.0	p49	0.0	p91	0.0	p133	2.313
p8	0.0	p50	0.0	p92	0.0	p134	2.358
p9	0.0	p51	0.0	p93	0.0	p135	1.998
p10	0.315	p52	0.0	p94	0.0	p136	1.566
p11	0.9	p53	0.0	p95	0.0	p137	0.819
p12	1.521	p54	0.0	p96	0.0	p138	0.171
p13	1.89	p55	0.0	p97	0.0	p139	0.0
p14	1.998	p56	0.0	p98	0.0	p140	0.0
p15	1.719	p57	0.09	p99	0.0	p141	0.0
p16	1.224	p58	0.945	p100	0.0	p142	0.0
p17	0.603	p59	2.16	p101	0.0	p143	0.0
p18	0.099	p60	2.916	p102	0.0	p144	0.0
p19	0.0	p61	3.348	p103	0.0	p145	0.0
p20	0.0	p62	3.618	p104	0.0	p146	0.0
p21	0.0	p63	3.438	p105	0.063	p147	0.0
p22	0.0	p64	2.799	p106	0.54	p148	0.0
p23	0.0	p65	1.512	p107	1.422	p149	0.0
p24	0.0	p66	0.252	p108	2.061	p150	0.0
p25	0.0	p67	0.0	p109	2.232	p151	0.0
p26	0.0	p68	0.0	p110	2.124	p152	0.0
p27	0.0	p69	0.0	p111	1.8	p153	0.0
p28	0.0	p70	0.0	p112	1.215	p154	0.36
p29	0.0	p71	0.0	p113	0.684	p155	0.864
p30	0.0	p72	0.0	p114	0.126	p156	1.332
p31	0.0	p73	0.0	p115	0.0	p157	1.611
p32	0.0	p74	0.0	p116	0.0	p158	1.683
p33	0.063	p75	0.0	p117	0.0	p159	1.485
p34	0.549	p76	0.0	p118	0.0	p160	1.08
p35	1.512	p77	0.0	p119	0.0	p161	0.684
p36	2.322	p78	0.0	p120	0.0	p162	0.18
p37	2.943	p79	0.0	p121	0.0	p163	0.0
p38	3.024	p80	0.0	p122	0.0	p164	0.0
p39	2.61	p81	0.0	p123	0.0	p165	0.0
p40	1.98	p82	0.522	p124	0.0	p166	0.0
p41	1.152	p83	1.422	p125	0.0	p167	0.0
p42	0.243	p84	2.007	p126	0.0	p168	0.0

Tabla 16. Energía solar disponible

Energía eólica

p	$\overline{wind}$	p	$\overline{wind}$	p	$\overline{wind}$	p	$\overline{wind}$
p1	5.3946	p43	1.5489	p85	0.8919	p127	2.9097
p2	5.5098	p44	1.2933	p86	0.6363	p128	3.1545
p3	5.5746	p45	1.359	p87	0.5787	p129	4.3002
p4	5.7159	p46	1.4229	p88	0.6021	p130	5.2119
p5	5.8077	p47	1.4112	p89	0.522	p131	5.6808
p6	5.8896	p48	1.3806	p90	0.4212	p132	5.9886
p7	5.9832	p49	1.2996	p91	0.5598	p133	6.0885
p8	5.8662	p50	1.1961	p92	1.098	p134	5.9328
p9	5.769	p51	1.2078	p93	1.8612	p135	5.6538
p10	5.7114	p52	1.2501	p94	2.5722	p136	5.418
p11	5.6304	p53	1.5525	p95	3.1293	p137	4.9518
p12	5.1723	p54	2.0061	p96	3.2832	p138	4.5099
p13	4.4874	p55	2.2905	p97	3.1689	p139	4.3218
p14	3.9312	p56	2.3949	p98	3.0744	p140	3.9483
p15	3.8529	p57	2.5884	p99	2.9367	p141	3.3903
p16	3.9582	p58	2.3301	p100	2.6739	p142	3.2436
p17	3.7404	p59	1.8981	p101	2.4435	p143	3.3696
p18	3.1752	p60	1.791	p102	2.1654	p144	3.2355
p19	3.1563	p61	1.7577	p103	1.7541	p145	3.2571
p20	3.6189	p62	1.521	p104	1.7145	p146	3.2544
p21	4.2192	p63	1.5174	p105	1.899	p147	3.4263
p22	4.9257	p64	1.5732	p106	1.4643	p148	3.7944
p23	5.3289	p65	1.602	p107	1.2366	p149	3.969
p24	5.6421	p66	1.6038	p108	1.4607	p150	3.8268
p25	5.7096	p67	1.548	p109	1.7586	p151	3.4515
p26	5.5557	p68	1.6524	p110	2.115	p152	3.0672
p27	5.3937	p67	1.548	p107	1.2366	p147	3.4263
p28	5.166	p70	2.5911	p112	3.0186	p154	3.1959
p29	4.9401	p71	3.0564	p113	3.2373	p155	3.951
p30	4.7079	p72	3.5343	p114	3.465	p156	4.0554
p31	4.3902	p73	3.8826	p115	3.843	p157	3.9465
p32	3.9726	p74	4.059	p116	4.3488	p158	3.7287
p33	3.8268	p75	4.2489	p117	5.0022	p159	3.7098
p34	3.9402	p76	4.4514	p118	5.4207	p160	3.8988
p35	4.0041	p77	4.6215	p119	5.6241	p161	3.8385
p36	3.9285	p78	4.7403	p120	5.5341	p162	3.6252
p37	4.0284	p79	4.5729	p121	5.4216	p163	3.2751
p38	4.149	p80	4.2066	p122	5.2245	p164	3.2481
p39	4.0932	p81	3.9753	p123	5.256	p165	2.4435
p40	3.9771	p82	3.3858	p124	5.1264	p166	1.9989
p41	3.7269	p83	2.3175	p124	5.1264	p166	1.9989
p42	2.6667	p84	1.4301	p125	4.5378	p167	1.4562

Tabla 17. Energía eólica disponible

Anexo III. Código fuente

main.py

```
import logging
from pyomo.environ import *
from pathlib import Path
from pyomo.opt import SolverFactory
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from pyomo.environ import Reals, PositiveReals,
    NonPositiveReals, NegativeReals, NonNegativeReals,
    PercentFraction, \
     UnitInterval, Integers, PositiveIntegers,
        NonPositiveIntegers, NegativeIntegers,
        NonNegativeIntegers, Binary
from pyomo.core import TransformationFactory
# import time
# Data management and treatment
import pandas as pd
logging.getLogger('pyomo.core').setLevel(logging.ERROR)
glpk = 0
binder = 0
mesp = 1
if binder == 0:
    if mesp == 1:
        path_model_in_set = Path('D:/TFM/TFM-PyomoWeb/DataMESP/
            Input/Sets/')
        path_model_in_par = Path('D:/TFM/TFM-PyomoWeb/DataMESP/
            Input/Parameters/')
        path_model_out = Path('D:/TFM/TFM-PyomoWeb/DataMESP/
            Output')
        path_results = Path('D:/TFM/TFM-PyomoWeb/DataMESP/
            Output')
        gen_order_df = pd.read_csv('D:/TFM/TFM-PyomoWeb/
            DataMESP/Input/Parameters/GeneratorOrder.csv',
            header=None, nrows=1)
        tech_order_df =pd.read_csv('D:/TFM/TFM-PyomoWeb/
            DataMESP/Input/Parameters/TechnologyOrder.csv',
            header=None, nrows=1)
        gens_data = pd.read_csv('D:/TFM/TFM-PyomoWeb/DataMESP/
            Input/Parameters/GeneratorsMESP.csv')
    else:
        path_model_in_set = Path('D:/TFM/TFM-PyomoWeb/Data/
            Input/Sets/')
        path_model_in_par = Path('D:/TFM/TFM-PyomoWeb/Data/
            Input/Parameters/')
```

```
        path_model_out = Path('D:/TFM/TFM-PyomoWeb/Data/Output'
            )
        path_results = Path('D:/TFM/TFM-PyomoWeb/Data/Output')
        gen_order_df = pd.read_csv('D:/TFM/TFM-PyomoWeb/Data/
            Input/Parameters/GeneratorOrder.csv', header=None,
            nrows=1)
        tech_order_df = pd.read_csv('D:/TFM/TFM-PyomoWeb/Data/
            Input/Parameters/TechnologyOrder.csv', header=None,
                                    nrows=1)
        gens_data = pd.read_csv('D:/TFM/TFM-PyomoWeb/Data/Input
            /Parameters/Generators.csv')

else:
    path_model_in_set = Path('Data/Input/Sets/')
    path_model_in_par = Path('Data/Input/Parameters/')
    path_model_out = Path('Data/Output/')
    path_results = Path('Data/Output/')

if (glpk == 0):
    optimizador = 'gurobi'  # Solver selection
    solver = SolverFactory(optimizador)
    # https://www.gurobi.com/documentation/8.1/refman/
        parameter_descriptions.html
    solver.options['Method'] = 1  # Solution method: dual-
        simplex {1} o barrier {2}
    solver.options['MIPGap'] = 0.016  # 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
else:
    import sys

    solverpath_folder = 'C:/glpk-4.65/w64'
    solverpath_exe = 'C:/glpk-4.65/w64/glpsol'
    sys.path.append(solverpath_folder)
    solvername = 'glpk'
    solver = SolverFactory(solvername, executable=
        solverpath_exe)
    solver.options['mipgap'] = 0.999

m = AbstractModel()
```

```
m.g = Set(ordered=True, doc="Generators")
m.b = Set(ordered=True, doc="Batteries")
m.p = Set(ordered=True, doc="periods")

# Definici n de par metros
m.p_alfa = Param(m.g, within=NonNegativeReals, doc="[MTh/GWh]  
    Variable fuel consumption of generator g")
m.p_qmax = Param(m.g, within=NonNegativeReals, doc="[GW]  
    Maximum gross power of generator g")
m.p_qmin = Param(m.g, within=NonNegativeReals, doc="[GW]  
    Minimum gross power of generator g")
m.p_bmax = Param(m.g, within=NonNegativeReals, doc="[GW]  
    Maximum gross pumping power of generatior g")
m.p_k = Param(m.g, within=NonNegativeReals, doc="[GW]  Gross to
    net power conversion factor of generator g")
m.p_rend = Param(m.g, within=NonNegativeReals, doc="[p.u.]
    Efficiency of the pumping-turbine cycle of generator g")
m.p_f = Param(m.g, within=Reals, doc="[kEuros/MTh]  Price of 
    fuel used by generator g ")
m.p_o = Param(m.g, within=Reals, doc="[kEuros/GWh]  Operation 
    and maintenaince variable cost of the generator g")
# m.p_c= Param(m.g, within = Reals, doc = "[kEuros/GWh]  Water
    value (turbined or pumped) by generator g")
m.p_type = Param(m.g, within=Any, doc="Type of generator , 
    thermal or hydro")
m.p_d = Param(m.p, within=Reals, doc="[GW]  Demand of real 
    power at period p")
m.p_rs = Param(m.g, within=NonNegativeReals, doc="Ramp-up limit
    of generator g")
m.p_rb = Param(m.g, within=NonNegativeReals, doc="Ramp-down 
    limit of generator g")
m.p_beta = Param(m.g, within=NonNegativeReals, doc="[MTh/GWh]  
    Fixed fuel consumption of generator g")
m.p_gamma = Param(m.g, within=NonNegativeReals, doc="[MTh/GWh] 
    Start-up fuel consumption of generator g")
m.p_theta = Param(m.g, within=NonNegativeReals, doc="[MTh/GWh] 
    Shot-down fuel consumption of generator g")
m.p_i = Param(m.p, m.g, within=NonNegativeReals, doc="Water 
    inflows at generator's g reservoir during period p")
m.p_w0 = Param(m.g, within=NonNegativeReals, doc="Initial 
    amount of water in generator's g reservoir")
m.p_wfin = Param(m.g, within=NonNegativeReals, doc="Final 
    amount of water in generator's g reservoir")
m.p_wmax = Param(m.g, within=NonNegativeReals, doc="Maximum 
    water storage capacity of generator's g reservoir")
m.p_wmin = Param(m.g, within=NonNegativeReals, doc="Minimum 
    water storage capacity of generator's g reservoir")
m.p_rod = Param(m.p, within=NonNegativeReals, mutable=True,
    default=0, doc="Spinning reserve at period p", initialize=0)
```

```
m.p_cens = Param(within=NonNegativeReals, doc="Cost␣of␣Energy␣
    not␣Supplied")
m.p_solar = Param(m.p, within=NonNegativeReals, doc="Available␣
    solar␣power␣at␣period␣p")
m.p_solar_factor = Param(within=NonNegativeReals)
m.p_wind = Param(m.p, within=NonNegativeReals, doc="Available␣
    wind␣power␣at␣period␣p")
m.p_wind_factor = Param(within=NonNegativeReals)
m.p_u0 = Param(m.g, within=Binary, doc="Initial␣commitment␣
    state␣of␣generator␣g")
m.p_flu = Param(m.g, within=NonNegativeReals, doc="Minimum␣
    output␣flow␣for␣hydro␣generator␣g")
m.p_ce = Param(within=NonNegativeReals, doc="Cost␣of␣CO2␣
    emissions")
m.p_mode = Param(m.g, within=Integers, initialize=2, doc="Mode:
    ␣0:␣Normal,␣1:␣Must␣Connect,␣must␣disconnect", )

m.p_alfa_e = Param(m.g, within=NonNegativeReals, doc="[MTh/GWh]
    ␣␣Variable␣fuel␣consumption␣of␣generator␣g")
m.p_beta_e = Param(m.g, within=NonNegativeReals, doc="[MTh/GWh]
    ␣␣Fixed␣fuel␣consumption␣of␣generator␣g")
m.p_gamma_e = Param(m.g, within=NonNegativeReals, doc="[MTh/GWh
    ]␣Start-up␣fuel␣consumption␣of␣generator␣g")
m.p_theta_e = Param(m.g, within=NonNegativeReals, doc="[MTh/GWh
    ]␣Shot-down␣fuel␣consumption␣of␣generator␣g")

m.p_eta_c = Param(m.b, within=NonNegativeReals, doc="Charging␣
    efficiency␣of␣batterty␣b")
m.p_eta_disc = Param(m.b, within=NonNegativeReals, doc="
    Discharging␣efficiency␣of␣battery␣b")
m.p_e0 = Param(m.b, within=NonNegativeReals, doc="Initial␣
    Charge␣of␣battery␣b")
m.p_emin = Param(m.b, within=NonNegativeReals, doc="Minimum␣
    charge␣of␣battery␣b")
m.p_emax = Param(m.b, within=NonNegativeReals, doc="Maximum␣
    charge␣of␣battery␣b")
m.p_maxpchar = Param(m.b, within = NonNegativeReals, doc="
    Maximum␣power␣that␣can␣be␣injected␣tobattery␣b")
m.p_maxpdischar = Param(m.b, within = NonNegativeReals, doc="
    Maximum␣power␣that␣can␣be␣extracted␣from␣battery␣b")

m.p_TU = Param(m.g, within=NonNegativeIntegers, initialize=3)
m.p_TD = Param(m.g, within=NonNegativeIntegers, initialize=3)

m.p_spinning_reserve_factor = Param(within= NonNegativeReals)

m.p_emissions = Param(m.g, within = NonNegativeReals,
    initialize = 0)
# Definici n de variables
```

```
m.v_e = Var(m.b, m.p, within=NonNegativeReals, doc="Charge␣of␣
    battery␣b␣in␣period␣p")
m.v_pchar = Var(m.b, m.p, within=NonNegativeReals, doc="Gross␣
    power␣used␣to␣charge␣battery␣b␣in␣period␣p")
m.v_pdischar = Var(m.b, m.p, within=NonNegativeReals, doc="
    Gross␣power␣extracted␣from␣battery␣b␣in␣period␣p")

# Variable de arranque de un grupo

m.v_y = Var(m.g, m.p, within=Binary, doc="Binary␣variable␣for␣
    start-up␣of␣generator␣g␣at␣period␣p")

# Variable de parada de un grupo
m.v_z = Var(m.g, m.p, within=Binary, doc="Binary␣variable␣for␣
    shut-down␣of␣generator␣g␣at␣period␣p")

# Variable de unit commitment
m.v_u = Var(m.g, m.p, within=Binary, doc="Unit␣commitment␣
    variable")

# Potencia generada por grupo en un periodo
m.v_q = Var(m.g, m.p, within=NonNegativeReals, doc="Real␣power␣
    produced␣by␣generator␣g␣at␣period␣p")

# Exceso de potencia generada sobre minimo tecnico
m.v_deltaq = Var(m.g, m.p, within=NonNegativeReals,
                 doc="Real␣power␣over␣technical␣minimum␣
                    produced␣by␣generator␣g␣at␣period␣p")
m.v_resup = Var(m.g, m.p, within=NonNegativeReals)
# Energy stored in water
m.v_w = Var(m.g, m.p, within=NonNegativeReals, doc="Amount␣of␣
    water␣stored␣at␣generator's␣g␣reservoir␣atr␣period␣p")

# Spills de una reserva hydro
m.v_s = Var(m.g, m.p, within=NonNegativeReals, doc="")

# Bombeo en una reserva hydro
m.v_b = Var(m.g, m.p, within=NonNegativeReals)

# Cantidad de potencia solar utilizada en el periodo p
m.v_solar = Var(m.p, within=NonNegativeReals)

# Cantidad de potencia e lica utilizada en el periodo p
m.v_wind = Var(m.p, within=NonNegativeReals)

# Potencia no servida
m.v_pns = Var(m.p, within=NonNegativeReals)
```

```
m.v_cost = Var(m.p, within=NonNegativeReals)

m.v_fuel_consumption =Var(m.g, m.p, within =NonNegativeReals)
m.v_q_bat = Var(m.b,m.p,within=NonNegativeReals)

data = DataPortal()
# cosa =repr(open('D:/TFM/TFM-PyomoWeb/DataMESP/Input/Sets/
   gMESP.csv','rb').read(7290)[272:])

if mesp == 1:
    data.load(filename=str(path_model_in_set.joinpath('gMESP.
       csv')),format='set', set='g')
    data.load(filename=str(path_model_in_set.joinpath('bMESP.
       csv')), format='set', set='b')
    data.load(filename=str(path_model_in_set.joinpath('pMESP.
       csv')), format='set', set='p')

    # Load of parameters
    data.load(filename=str(path_model_in_par.joinpath('
       GeneratorsMESP.csv')), index=['g'],
              param=['p_alfa', 'p_beta', 'p_gamma', 'p_theta',
                 'p_rs', 'p_rb', 'p_f', 'p_o', 'p_qmax', '
                 p_qmin', 'p_u0',
                     'p_mode', 'p_bmax', 'p_wmax', 'p_w0', '
                        p_wmin', 'p_wfin', 'p_flu', 'p_k', '
                        p_rend', 'p_type', 'p_emissions',
                     'p_tech', 'p_TU', 'p_TD'])
    data.load(filename=str(path_model_in_par.joinpath('
       InflowsMESP.csv')), index=['p', 'g'], param=['p_i'],
       format='array')
    data.load(filename=str(path_model_in_par.joinpath('
       BatteriesMESP.csv')), index=['b'],
              param=['p_eta_c', 'p_eta_disc', 'p_e0', 'p_emin',
                  'p_emax', 'p_maxpchar','p_maxpdischar'])
    data.load(filename=str(path_model_in_par.joinpath('
       DemandMESP.csv')), index=['p'], param='p_d')
    data.load(filename=str(path_model_in_par.joinpath('WindMESP
       .csv')), param=['p_wind'])
    data.load(filename=str(path_model_in_par.joinpath('
       SolarMESP.csv')), param=['p_solar'])
    # data.load(filename=str(path_model_in_par.joinpath('
       SpinningReserveMESP.csv')), index=['p'], param=['p_rod
       '])
    data.load(filename=str(path_model_in_par.joinpath('
       ScalarsMESP.csv')), param=['p_cens', 'p_ce', '
       p_solar_factor','p_wind_factor','
       p_spinning_reserve_factor'])
else:
```

```
    data.load(filename=str(path_model_in_set.joinpath('g.csv'))
        , format='set', set='g')
    # data.load(filename=str(path_model_in_set.joinpath('b.csv
        ')), format='set', set='b')
    data.load(filename=str(path_model_in_set.joinpath('p.csv'))
        , format='set', set='p')

    # Load of parameters
    data.load(filename=str(path_model_in_par.joinpath('
        Generators.csv')), index=['g'],
              param=['p_alfa', 'p_beta', 'p_gamma', 'p_theta',
                  'p_rs', 'p_rb', 'p_f', 'p_o', 'p_qmax', '
                  p_qmin', 'p_u0',
                       'p_mode', 'p_bmax', 'p_wmax', 'p_w0', '
                           p_wmin', 'p_wfin', 'p_flu','p_k', '
                           p_rend', 'p_type', 'p_emissions',
                       'p_tech', 'p_TU', 'p_TD'])
    data.load(filename=str(path_model_in_par.joinpath('Inflows.
        csv')), index=['p', 'g'], param=['p_i'],
              format='array')
    # data.load(filename=str(path_model_in_par.joinpath('
        Batteries.csv')), index=['b'],
    #           param=['p_eta_c', 'p_eta_disc', 'p_e0', 'p_emin
        ', 'p_emax', 'p_maxpchar', 'p_maxpdischar'])
    data.load(filename=str(path_model_in_par.joinpath('Demand.
        csv')), index=['p'], param='p_d')
    data.load(filename=str(path_model_in_par.joinpath('Wind.csv
        ')), param=['p_wind'])
    data.load(filename=str(path_model_in_par.joinpath('Solar.
        csv')), param=['p_solar'])
    # data.load(filename=str(path_model_in_par.joinpath('
        SpinningReserve.csv')), index=['p'], param=['p_rod'])
    data.load(filename=str(path_model_in_par.joinpath('Scalars.
        csv')),
              param=['p_cens', 'p_ce', 'p_solar_factor', '
                  p_wind_factor', 'p_spinning_reserve_factor'])


# %% Declaration of dynamic sets
def setdin_thermal_generators(m, g):
    return m.p_type[g] == 'thermal'


m.t = Set(ordered=True, initialize=m.g, filter=
    setdin_thermal_generators, doc="Thermal␣generators")


def setdin_hydro_generators(m, g):
    return m.p_type[g] == 'hydro'
```

```
m.h = Set(ordered=True, initialize=m.g, filter=
    setdin_hydro_generators, doc="Hydro generators")

for technology in gens_data['p_tech'].unique():
    setattr(m, str(technology), Set(initialize=gens_data['
        p_tech'] == technology))

def FuelConsumption(m,t,p):
    return(m.p_beta[t] * m.v_u[t, p] + \
                m.p_gamma[t] * m.v_y[t, p] + \
                m.p_theta[t] * m.v_z[t, p] + \
                m.p_alfa[t] * m.v_q[t, p] / m.p_k[t] == m.
                    v_fuel_consumption[t,p])

def CostPerPeriod(m, p):
    return (m.p_cens * m.v_pns[p] + \
            sum((m.p_f[t] + m.p_emissions[t] * m.p_ce) * m.
                v_fuel_consumption[t,p] - m.p_theta[t] * m.v_z[t
                , p] \
                 for t in m.t) == m.v_cost[p])
m.CostDefinition = Constraint(m.p, rule=CostPerPeriod)



def e_Fobj(m, p):
    return (sum(m.v_cost[p] for p in m.p))



def DischargeChargeBounds(m,b,p):
    return m.p_maxpdischar[b] >= m.v_pdischar[b,p]


def ChargeBounds(m,b,p):
    return m.p_maxpchar[b] >= m.v_pchar[b,p]

def BatteryGeneration(m,b,p):
    return (m.v_q_bat[b,p] == m.v_pdischar[b, p] / m.p_eta_disc
        [b])

def e_BatteryCharge(m, b, p):
    if p == m.p.first():
        return (m.v_e[b, p] == m.p_e0[b] + m.v_pchar[b, p] * m.
            p_eta_c[b] - m.v_pdischar[b, p] / m.p_eta_disc[b])
    else:
        return (m.v_e[b, p] == m.v_e[b, m.p.prev(p, 1)] + m.
            v_pchar[b, p] * m.p_eta_c[b] - m.v_pdischar[b, p] /
                m.p_eta_disc[b])
```

```
def e_BatteryMaximumCharge(m, b, p):
    return (m.v_e[b, p] <= m.p_emax[b])


def e_BatteryMinimumCharge(m, b, p):
    return (m.v_e[b, p] >= m.p_emin[b])


def e_DeltaQDefinition(m, t, p):  # 6
    return (m.v_q[t, p] == m.v_u[t, p] * m.p_k[t] * m.p_qmin[t]
        + m.v_deltaq[t, p])


def e_DeltaQBounds(m, t, p):  # 7
    return (m.v_deltaq[t, p] <= m.v_u[t, p] * m.p_k[t] * (m.
       p_qmax[t] - m.p_qmin[t]))


def e_RampingUpConstraint(m, t, p):  # 9
    if p != m.p.first():
        return (m.v_deltaq[t, p] - m.v_deltaq[t, m.p.prev(p, 1)
           ] <= m.p_rs[t])
    else:
        return Constraint.Skip


def e_RampingDownConstraint(m, t, p):  # 10
    if p != m.p.first():
        return (m.v_deltaq[t, m.p.prev(p, 1)] - m.v_deltaq[t, p
           ] <= m.p_rb[t])
    else:
        return Constraint.Skip


def e_CommitmentModeConstraint(m, g, p):
    if m.p_mode[g] == 1:
        return (m.v_u[g, p] == 1)
    elif m.p_mode[g] == 2:
        return (m.v_u[g, p] == 0)
    else:  # m.p_mode[g] == 0:
```

```
        return Constraint.Skip


def e_UnitCommitmentConsistency(m, g, p):  # 11
    if p != m.p.first():
        return (m.v_y[g, p] - m.v_z[g, p] == m.v_u[g, p] - m.
           v_u[g, m.p.prev(p, 1)])
    else:
        return (m.v_y[g, p] - m.v_z[g, p] == m.v_u[g, p] - m.
           p_u0[g])



def e_WaterReserves(m, h, p):  # 16
    if p == m.p.first():
        return (m.v_w[h, p] == m.p_w0[h] - (m.v_q[h, p] + m.v_s
           [h, p] - m.p_rend[h] * m.v_b[h, p]) + m.p_i[p, h])
    else:
        return (m.v_w[h, p] == m.v_w[h, m.p.prev(p, 1)] - (m.
           v_q[h, p] + m.v_s[h, p] - m.p_rend[h] * m.v_b[h, p])
            +
                m.p_i[p, h])



def e_FinalWaterReserves(m, h, p):
    if p == m.p.last():
        return (m.v_w[h, p] == m.p_wfin[h])
    else:
        return Constraint.Skip



#
def e_HydroOutputUpperLimit(m, h, p):  # 17
    return (m.v_q[h, p] <= m.p_k[h] * m.p_qmax[h])



def e_PumpingUpperLimit(m, h, p):  # 19
    return (m.v_b[h, p] <= m.p_bmax[h])


def e_WaterReserveLowerVolumeLimit(m, h, p):  # 20
    return (m.p_wmin[h] <= m.v_w[h, p])


def e_WaterReserveUpperVolumeLimit(m, h, p):  # 20
```

```
    return (m.v_w[h, p] <= m.p_wmax[h])


def e_SolarPowerUpperLimit(m, p):  # 0.2
    return (m.v_solar[p] <= m.p_solar[p] * m.p_solar_factor)


def e_WindPowerUpperLimit(m, p):
    return (m.v_wind[p] <= m.p_wind[p] * m.p_wind_factor)  #
       0.3


def e_DemandBalance(m, p):
    return (sum(m.v_q[t, p] for t in m.t) + sum(m.v_q[h, p] - m
       .v_b[h, p] for h in m.h) + \
            sum(m.v_pdischar[b, p] - m.v_pchar[b, p] for b in m
               .b) + \
            m.v_wind[p] + m.v_solar[p] + m.v_pns[p] == m.p_d[p
               ])


def ReserveUp(m, t, p):
    return m.v_resup[t, p] == m.v_u[t, p] * m.p_k[t] * m.p_qmax
       [t] - m.v_q[t, p]


def e_SpinningReserve(m, p):
    return (sum(m.v_resup[t,p] for t in m.t) >= m.p_d[p] * m.
       p_spinning_reserve_factor)


def e_MinimumUpTime(m, t, p):
    if m.p_TU[t] > 0:
        period_list = list(m.p)
        period_number = period_list.index(p) + 1
        if period_number >= m.p_TU[t] and m.p_TU[t] > 1:
            period_range = range(int(period_number - m.p_TU[t]
               + 1),period_number)
            t_up = sum(m.v_y[t, m.p.prev(p, period)] for period
                in period_range)
            return t_up <= m.v_u[t, p]
        else:
            return Constraint.Skip
    else:
        return Constraint.Skip

def MinimumFlow(m,h,p):
    return(m.v_q[h,p]>=m.p_flu[h])
```

```
def e_MinimumDownTime(m, t, p):
    if m.p_TD[t] > 0:
        period_list = list(m.p)
        period_number = period_list.index(p) + 1
        if period_number >= m.p_TD[t] and m.p_TD[t] > 1:
            period_range = range(int(period_number - m.p_TD[t]
               + 1),period_number)
            t_down = sum(m.v_z[t, m.p.prev(p, period)] for
               period in period_range)
            return t_down <= 1 - m.v_u[t, p]
        else:
            return Constraint.Skip
    else:
        return Constraint.Skip


m.FinalWaterReserves = Constraint(m.h, m.p, rule=
    e_FinalWaterReserves)
m.HydroOutputUpperLimit = Constraint(m.h, m.p, rule=
    e_HydroOutputUpperLimit)
m.WaterReserves = Constraint(m.h, m.p, rule=e_WaterReserves)
m.FuelConsumptionDefinition = Constraint(m.t,m.p, rule=
    FuelConsumption)
m.ObjectiveFunction = Objective(rule=e_Fobj, sense=minimize)
m.DischargeChargeBounds = Constraint(m.b,m.p,rule =
    DischargeChargeBounds)
m.ChargeBounds = Constraint(m.b,m.p,rule = ChargeBounds)
m.BatteryConstraint = Constraint(m.b, m.p, rule=e_BatteryCharge
    )
m.DeltaQDefinition = Constraint(m.t, m.p, rule=
    e_DeltaQDefinition)
m.MinimumChargeConstraint = Constraint(m.b, m.p, rule=
    e_BatteryMinimumCharge)
m.MaximumChargeConstraint = Constraint(m.b, m.p, rule=
    e_BatteryMaximumCharge)
m.DeltaQBounds = Constraint(m.t, m.p, rule=e_DeltaQBounds)
m.RampingUpConstraint = Constraint(m.t, m.p, rule=
    e_RampingUpConstraint)
m.RampingDownConstraint = Constraint(m.t, m.p, rule=
    e_RampingDownConstraint)
m.CommitmentModeConstraint = Constraint(m.g, m.p, rule=
    e_CommitmentModeConstraint)
m.UnnitCommitmentConsistency = Constraint(m.t, m.p, rule=
    e_UnitCommitmentConsistency)
m.PumpingUpperLimit = Constraint(m.h, m.p, rule=
    e_PumpingUpperLimit)
m.WaterReserveLowerVolumeLimit = Constraint(m.h, m.p, rule=
    e_WaterReserveLowerVolumeLimit)
```

```
m.WaterReserveUpperVolumeLimit = Constraint(m.h, m.p, rule=
    e_WaterReserveUpperVolumeLimit)
m.SolarPowerUpperLimit = Constraint(m.p, rule=
    e_SolarPowerUpperLimit)
m.DemandBalance = Constraint(m.p, rule=e_DemandBalance)
m.WindPowerUpperLimit = Constraint(m.p, rule=
    e_WindPowerUpperLimit)
m.SpinningReserve = Constraint(m.p, rule=e_SpinningReserve)
m.MinimumDownTime = Constraint(m.t, m.p, rule=e_MinimumDownTime
    )
m.MinimumUpTime = Constraint(m.t, m.p, rule=e_MinimumUpTime)
m.ReserveUp = Constraint(m.t, m.p, rule=ReserveUp)
m.MinimumFlow = Constraint(m.h,m.p, rule = MinimumFlow)
m.BatteryGeneration = Constraint(m.b,m.p,rule =
    BatteryGeneration)
def CreateInflowsCSV():
    p_list = []
    list1 = []
    list2 = []
    list3 = []
    for p in range(1, 169):
        p_list.append('p' + str(p))
        list1.append(0.01)
        list2.append(0.5 * math.exp(-(p - 1) / 100))
        list3.append(0)

    dict = {'p': p_list, 'HYDRO_RES': list1, 'HYDRO_ROR': list2
       , 'HYDRO_PUM': list3}
    df = pd.DataFrame(dict)
    df.to_csv('Inflows.csv', index=False)


def CreateSpinningReserveCSV():
    df = pd.read_csv(str(str(path_model_in_par.joinpath('
       Data_Demands.csv'))))
    df['p_d'] = df['p_d'].map(lambda p_d: p_d * 0.01)
    df.rename(columns={'p_d': 'p_rod'}, inplace=True)
    df.to_csv(str(path_model_in_par.joinpath('SpinningReserve.
       csv')), index=False)


instance = m.create_instance(data)
solver_results = solver.solve(instance, tee=True)

# for constraint in instance.component_objects(Constraint,
    active=True):
#     for index in constraint:
#         con = constraint[index]
#         print(f"Constraint:{con}")
```

```
#         for var in con.body.variables:
#             print(f"Variable:{var.name}, lb:{var.lb}, ub={var
    .ub}")


instance_relaxed = m.create_instance(data)
for v in instance_relaxed.component_data_objects(Var,
    descend_into = True):
     if v.domain == Binary:
         v.domain = Reals
         v.setlb(0)
         v.setub(1)
instance_relaxed.dual = Suffix(direction = Suffix.IMPORT)
results_relaxed = solver.solve(instance_relaxed,tee=True)
marginal_df = pd.DataFrame(columns=['Period', 'Marginal␣Price'
    ])
for i in instance_relaxed.p:
     new_row = {'Period':i, 'Marginal␣Price' : instance_relaxed.
        dual[instance_relaxed.DemandBalance[i]]}
     marginal_df = marginal_df.append(new_row, ignore_index =
        True)

N = 24
labels = np.arange(0, 168, N)
labels = np.append(labels, 167)
# df_pum.plot(kind='line',xticks=labels)

def set_to_zero(value):
     if -10e-6 <= value <=0:
         return 0
     return value

generators_data = pd.DataFrame.from_dict(instance.v_q.
    extract_values(), orient='index', columns=[str(instance.v_q)
    ])
generators_data.index.names = ['Generator-Period']
generators_data.reset_index(inplace=True)
generators_data[['Generator', 'Period']] = pd.DataFrame(
    generators_data['Generator-Period'].tolist())
generators_data['Period'] = pd.Categorical(generators_data['
    Period'], categories=['p' + str(i) for i in range(1, 169)],
                                               ordered=True)
generators_data = generators_data.sort_values('Period')
generators_data = generators_data.pivot(index='Period', columns
    ='Generator', values='v_q')

solar_data = pd.DataFrame.from_dict(instance.v_solar.
    extract_values(), orient='index', columns=[str(instance.
    v_solar)])
```

```
solar_data.index.names = ['Period']
solar_data.reset_index(inplace=True)
solar_data = solar_data.rename(columns={'v_solar': 'Solar'})
merge_solar = pd.merge(generators_data, solar_data, left_index=
   True, right_on='Period')
merge_solar.set_index('Period', inplace=True)
solar_data.set_index('Period', inplace=True)

wind_data = pd.DataFrame.from_dict(instance.v_wind.
   extract_values(), orient='index', columns=[str(instance.
   v_wind)])
wind_data.index.names = ['Period']
wind_data.reset_index(inplace=True)
wind_data = wind_data.rename(columns={'v_wind': 'Wind'})
merge_wind = pd.merge(merge_solar, wind_data, left_index=True,
   right_on='Period')
merge_wind.set_index('Period', inplace=True)
wind_data.set_index('Period', inplace=True)

if mesp == 1:
    battery_data = pd.DataFrame.from_dict(instance.v_q_bat.
       extract_values(), orient='index', columns=[str(instance.
       v_q_bat)])
    battery_data.index.names = ['Generator-Period']
    battery_data.reset_index(inplace=True)
    battery_data[['Generator', 'Period']] = pd.DataFrame(
       battery_data['Generator-Period'].tolist())
    battery_data['Period'] = pd.Categorical(battery_data['
       Period'], categories=['p' + str(i) for i in range(1,
       169)],
                                            ordered=True)
    battery_data = battery_data.sort_values('Period')
    battery_data = battery_data.pivot(index='Period', columns='
       Generator', values='v_q_bat')
    merge_wind = pd.merge(merge_solar, battery_data, left_index
       =True, right_on='Period')
    battery_charge = pd.DataFrame.from_dict(instance.v_e.
       extract_values(), orient='index', columns=[str(instance.
       v_e)])
    battery_charge = battery_charge.dropna()
    battery_charge.index.names = ['Battery-Period']
    battery_charge.reset_index(inplace=True)
    battery_charge[['Battery', 'Period']] = pd.DataFrame(
       battery_charge['Battery-Period'].tolist())
    battery_charge['Period'] = pd.Categorical(battery_charge['
       Period'],
                                              categories=['p' +
                                                  str(i) for i
```

```
                                                    in range(1,
                                                    169)],
                                                ordered=True)
    battery_charge = battery_charge.sort_values('Period')
    battery_charge = battery_charge.pivot(index='Period',
       columns='Battery', values='v_e')


pns_data = pd.DataFrame.from_dict(instance.v_pns.extract_values
   (), orient='index', columns=[str(instance.v_pns)])
pns_data.index.names = ['Period']
pns_data.reset_index(inplace=True)
pns_data = pns_data.rename(columns={'v_pns': 'PNS'})
merged_data = pd.merge(merge_wind, pns_data, left_index=True,
   right_on='Period')
merged_data.set_index('Period', inplace=True)
merged_data.columns = merged_data.columns.str.capitalize()

available_wind_data = pd.DataFrame.from_dict(instance.p_wind.
   extract_values(), orient='index', columns=[str(instance.
   p_wind)])
wind_factor = instance.p_wind_factor.extract_values()[None]
available_wind_data = available_wind_data*wind_factor
available_wind_data.index.names = ['Period']
available_wind_data.reset_index(inplace=True)
available_wind_data = available_wind_data.rename(columns={ '
   p_wind': 'Available␣Wind␣Power'})
wind_lf_data= pd.merge(wind_data, available_wind_data, on = '
   Period')
wind_lf_data['Load␣Factor'] = wind_lf_data['Wind']/wind_lf_data
   ['Available␣Wind␣Power']
wind_lf_data = wind_lf_data.rename(columns={'Wind' : 'Wind␣
   Power'})
wind_lf_data['Load␣Factor'] = wind_lf_data['Load␣Factor'].
   fillna(0)

available_solar_data = pd.DataFrame.from_dict(instance.p_solar.
   extract_values(), orient='index', columns=[str(instance.
   p_solar)])
solar_factor = instance.p_solar_factor.extract_values()[None]
available_solar_data = available_solar_data*solar_factor
available_solar_data.index.names = ['Period']
available_solar_data.reset_index(inplace=True)
available_solar_data = available_solar_data.rename(columns={'
   Solar' : 'Solar␣Power','p_solar': 'Available␣Solar␣Power'})
solar_lf_data= pd.merge(solar_data, available_solar_data, on =
   'Period')
solar_lf_data['Load␣Factor'] = solar_lf_data['Solar']/
   solar_lf_data['Available␣Solar␣Power']
```

```
solar_lf_data = solar_lf_data.rename(columns={'Solar' : 'Solar␣
   Power'})
solar_lf_data['Load␣Factor'] = solar_lf_data['Load␣Factor'].
   fillna(0)


# generator_order = gen_order_df.values.tolist()[0]
# columns = merged_data.columns
# new_order = [columns[g] for g in generator_order]
# for column in columns:
#     if column not in new_order:
#         new_order = new_order + [column]
# merged_data= merged_data.reindex(columns = new_order)

# merged_data = merged_data[new_order]
tech_dict = dict(zip(gens_data['g'], gens_data['p_tech']))
mapped_techs = generators_data.copy(deep=True)
mapped_techs.columns = mapped_techs.columns.map(tech_dict)
power_by_tech = mapped_techs.groupby(axis=1, level=0).sum()
power_by_tech = pd.merge(power_by_tech, wind_data, left_index=
   True, right_on='Period')
power_by_tech = pd.merge(power_by_tech, solar_data, left_index=
   True, right_on='Period')

if mesp == 1:
    power_by_tech = pd.merge(power_by_tech, battery_data,
       left_index=True, right_on='Period')

tech_order = tech_order_df.values.tolist()[0]
columns = power_by_tech.columns
new_order = tech_order
for column in columns:
    if column not in new_order:
        new_order = new_order + [column]
power_by_tech=power_by_tech.reindex(columns = new_order)



reserves_data = pd.DataFrame.from_dict(instance.v_w.
   extract_values(), orient='index', columns=[str(instance.v_w)
   ])
reserves_data = reserves_data.dropna()
reserves_data.index.names = ['Generator-Period']
reserves_data.reset_index(inplace=True)
reserves_data[['Generator', 'Period']] = pd.DataFrame(
   reserves_data['Generator-Period'].tolist())
reserves_data['Period'] = pd.Categorical(reserves_data['Period'
   ], categories=['p' + str(i) for i in range(1, 169)],
                                         ordered=True)
```

```
reserves_data = reserves_data.sort_values('Period')
reserves_data = reserves_data.pivot(index='Period', columns='
   Generator', values='v_w')
max_reserve = pd.DataFrame.from_dict(instance.p_wmax.
   extract_values(), orient='index', columns=[str(instance.
   p_wmax)])
max_reserve.index.names = ['Reservoir']
max_reserve = max_reserve[max_reserve['p_wmax'] != 0]

min_reserve = pd.DataFrame.from_dict(instance.p_wmin.
   extract_values(), orient='index', columns=[str(instance.
   p_wmin)])
min_reserve.index.names = ['Reservoir']
min_reserve = min_reserve[min_reserve['p_wmin'] != 0]

minmax=max_reserve
minmax['p_wmin']= min_reserve['p_wmin']
minmax['p_diff']=minmax['p_wmax']-minmax['p_wmin']

diff =minmax.drop(['p_wmin','p_wmax'],axis =1)
diff.reset_index(inplace=True)
sorted_columns = diff.sort_values(by='Reservoir')['Reservoir'].
   values
reserves_data = reserves_data[sorted_columns]
reserves_data = reserves_data.div(diff.set_index('Reservoir')['
   p_diff'],axis=1)
reserves_data = reserves_data * 100

reserves_data.plot( title='Reservoir␣water␣level␣(%)')

available_water = pd.DataFrame.from_dict(instance.v_w.
   extract_values(), orient='index', columns=[str(instance.v_w)
   ])
available_water = available_water.dropna()
available_water.index.names = ['Generator-Period']
available_water.reset_index(inplace=True)
available_water[['Generator', 'Period']] = pd.DataFrame(
   available_water['Generator-Period'].tolist())
available_water['Period'] = pd.Categorical(available_water['
   Period'], categories=['p' + str(i) for i in range(1, 169)],
                                              ordered=True)
available_water = available_water.sort_values('Period')
available_water = available_water.pivot(index='Period', columns
   ='Generator', values='v_w')
zero_columns = (available_water == 0).all()
available_water = available_water.drop(zero_columns[
   zero_columns].index, axis=1)
```

```
final_water = pd.DataFrame.from_dict(instance.p_wfin.
    extract_values(), orient='index', columns=[str(instance.
    p_wfin)])
final_water = final_water[final_water['p_wfin'] != 0]
final_water = final_water.sort_index()
final_water.index.names=['Reservoir']
final_water.reset_index(inplace=True)
initial_water = pd.DataFrame.from_dict(instance.p_w0.
    extract_values(), orient='index', columns=[str(instance.p_w0
    )])
initial_water = initial_water[initial_water['p_w0'] != 0]
initial_water = initial_water.sort_index()
initial_water.index.names=['Reservoir']
initial_water.reset_index(inplace=True)
init_final =pd.DataFrame()
# init_final = pd.merge(initial_water, final_water, left_index=
    True, right_on='Reservoir')
init_final['diff'] = initial_water['p_w0']-final_water['p_wfin'
    ]
init_final['Reservoir']=final_water['Reservoir']
print(init_final)
reserves_data = reserves_data.div(diff.set_index('Reservoir')['
    p_diff'],axis=1)
available_water = available_water.sub(final_water.set_index('
    Reservoir')['p_wfin'],axis=1)
available_water = available_water.div(init_final.set_index('
    Reservoir')['diff'],axis=1)#/available_water * 100
available_water = available_water*100
print(available_water)
print(final_water)
print(init_final)

available_water.plot(title = 'Available␣water␣(%)')
```

```
reservas_subir_data = pd.DataFrame.from_dict(instance.v_resup.
    extract_values(), orient='index',
```

```
                                                    columns=[str(
                                                        instance.
                                                        v_resup)])
reservas_subir_data = reservas_subir_data.dropna()
reservas_subir_data.index.names = ['Generator-Period']
reservas_subir_data.reset_index(inplace=True)
reservas_subir_data[['Generator', 'Period']] = pd.DataFrame(
    reservas_subir_data['Generator-Period'].tolist())
reservas_subir_data['Period'] = pd.Categorical(
    reservas_subir_data['Period'],
                                                   categories=['p'
                                                       + str(i) for
                                                       i in range(1,
                                                        169)],
                                                       ordered=True)
reservas_subir_data = reservas_subir_data.sort_values('Period')
reservas_subir_data = reservas_subir_data.pivot(index='Period',
     columns='Generator', values='v_resup')

reservas_subir_data = pd.concat([reservas_subir_data])


deltaq_data = pd.DataFrame.from_dict(instance.v_deltaq.
    extract_values(), orient='index',
                                         columns=[str(instance.
                                             v_deltaq)])
deltaq_data = deltaq_data.dropna()
deltaq_data.index.names = ['Generator-Period']
deltaq_data.reset_index(inplace=True)
deltaq_data[['Generator', 'Period']] = pd.DataFrame(deltaq_data
    ['Generator-Period'].tolist())
deltaq_data['Period'] = pd.Categorical(deltaq_data['Period'],
    categories=['p' + str(i) for i in range(1, 169)],
                                           ordered=True)
deltaq_data = deltaq_data.sort_values('Period')
deltaq_data = deltaq_data.pivot(index='Period', columns='
    Generator', values='v_deltaq')

merged_data.to_csv(str(path_results.joinpath('Merged.csv')))
wind_lf_data.to_csv(str(path_results.joinpath('Wind.csv')))
solar_lf_data.to_csv(str(path_results.joinpath('Solar.csv')))
generators_data.to_csv(str(path_results.joinpath('Generation.
    csv')))
pns_data.to_csv(str(path_results.joinpath('pns.csv')))


pbt_pie = power_by_tech.sum()
pbt_pie = pbt_pie.to_frame()
```

```
pbt_pie = pbt_pie.rename(columns={pbt_pie.columns[0]: 'Total␣
    Energy'})
pbt_pie = pbt_pie[pbt_pie['Total␣Energy'] != 0]
print(pbt_pie)
commitment = pd.DataFrame.from_dict(instance_relaxed.v_u.
    extract_values(), orient='index', columns=[str(
    instance_relaxed.v_u)])
commitment.index.names = ['Generator-Period']
commitment.reset_index(inplace=True)
commitment[['Generator', 'Period']] = pd.DataFrame(commitment['
    Generator-Period'].tolist())
commitment['Period'] = pd.Categorical(commitment['Period'],
    categories=['p' + str(i) for i in range(1, 169)],
                                          ordered=True)
commitment = commitment.sort_values('Period')
commitment = commitment.pivot(index='Period', columns='
    Generator', values='v_u')


# shadow_price_data = pd.DataFrame(index = m.DemandBalance.keys
    (), columns = ['Shadow Price'])
# print(shadow_price_data)
rod = pd.DataFrame.from_dict(instance.p_rod.extract_values(),
    orient='index', columns=[str(instance.p_rod)])
rod.index.names = ['Period']
rod.reset_index(inplace=True)
rod = rod.rename(columns={'p_rod': 'Rod'})

demand_data = pd.DataFrame.from_dict(instance.p_d.
    extract_values(), orient='index', columns=[str(instance.p_d)
    ])
demand_data.index.names = ['Period']
demand_data.reset_index(inplace=True)
demand_data = demand_data.rename(columns={'p_d': 'Demand'})

price_data = pd.DataFrame.from_dict(instance.v_cost.
    extract_values(), orient='index', columns=[str(instance.
    v_cost)])
price_data.index.names = ['Period']
price_data.reset_index(inplace=True)
price_data = price_data.rename(columns={'v_cost': 'Price'})


fig_pbt,ax_pbt = plt.subplots()
power_by_tech.plot(kind='area', title='Power␣by␣Technology␣type
    ',ax=ax_pbt, linewidth = 0,ylabel='Power␣[GW]',
                       xlabel='Period',alpha=1,colormap='tab20')
demand_data.plot.line(y='Demand',ax=ax_pbt,color='black')
ax_pbt.autoscale()
```

```
ax_pbt.legend(loc='lower␣right',fontsize='small')
ax_pbt.legend(bbox_to_anchor=(1,0.5))

fig_mer,ax_mer = plt.subplots()
merged_data = merged_data.applymap(set_to_zero)
merged_data.plot(kind='area', title='Power␣Generation', ax =
   ax_mer, linewidth = 0,ylabel='Power␣[GW]',xlabel='Period',
                  alpha=0.8)
ax_mer.legend(bbox_to_anchor=(1,0.5))
demand_data.plot.line(y='Demand',ax=ax_mer,color='black')
# plt.legend(fontsize='small')
ax_mer.autoscale()

solar_lf_data.plot(ylabel='Power␣[GW]',xlabel='Period')
wind_lf_data.plot(ylabel='Power␣[GW]',xlabel='Period')
# price_data.plot()

fig_resu,ax_resu = plt.subplots()
reservas_subir_data = reservas_subir_data.applymap(set_to_zero)

ax_resu = reservas_subir_data.plot(kind='area', title='Sppining
   ␣Reserves', linewidth = 0, label =''
                                   ,ylabel = 'Power␣[GW]',
                                      colormap='tab20',alpha
                                      =0.8)
handles, labels= ax_resu.get_legend_handles_labels()
# new_labels = [label for label,column in zip(labels[1:],
   reservas_subir_data[1:]) if not (reservas_subir_data[column
   == 0].all())]
# ax_resu.legend(handles[1:len(new_labels)+1], new_labels)
required_rod = demand_data
ax_resu.legend(bbox_to_anchor=(1,0.5))
required_rod['Demand'] = required_rod['Demand']*0.1
required_rod = required_rod.rename(columns={'Demand':'Required␣
   reserves'})
required_rod.plot.line(y='Required␣reserves',ax=ax_resu,color='
   black')
rod.plot.line(y='Rod',ax=ax_resu,color='black')
ax_resu.legend(loc='lower␣right',fontsize='small')
ax_resu.autoscale()

# fig_reba,ax_reba = plt.subplots()
# deltaq_data = deltaq_data.applymap(set_to_zero)
# deltaq_data.plot(kind='area', title='Reserva a bajar de los
   generadores', ax=ax_reba, linewidth = 0)
# demand_data.plot.line(y='Demand',ax=ax_reba,color='black')
# ax_reba.autoscale()

if mesp==1:
```

```
    fig_bat,ax_bat = plt.subplots()
    battery_charge = battery_charge.applymap(set_to_zero)
    battery_charge.plot(title='Battery charge', ax=ax_bat,
       ylabel='Energy [GWh]',xlabel='Period')
    # demand_data.plot.line(y='Demand',ax=ax_bat,color='black')
    ax_bat.autoscale()

# commitment.plot(title='Commitment state')
pbt_ax =pbt_pie.plot(kind='pie', y='Total Energy', title='
   Production share by technology',ylabel=''
                        )
                       #explode=[0.03]*len(pbt_pie['Total Energy
                          ']))
pbt_ax.legend(bbox_to_anchor=(1,1))
marginal_df.plot(kind='line', color='black',title = 'Marginal
   price', ylabel='Marginal price [ k  ]', xlabel='Period')

plt.show()
```

widget_generators.py

```
import ipywidgets as widgets
import pandas as pd

def updateCSV(df, filename):
    for column in range(len(df.columns)):
        for row in range(1, len(df)):
            widget = grid_list[column][row]
            new_value = widget.value
            df.iloc[row, column] = new_value
    df = df.transpose()
    df = df.set_index(df.columns[0])
    df.columns = df.iloc[0]
    df = df[1:]
    df.index.name = 'g'
    filename = filename + '.csv'
    path = 'Data/Input/Parameters/' + filename
    df.to_csv(path)

global data
data = pd.read_csv('Data/Input/Parameters/Generators.csv')
data = data.transpose()
data = data.reset_index()

container = widgets.GridBox(layout=widgets.Layout(
   grid_template_columns="repeat(auto-fit, minmax(60px, 90px))"
   ))
```

```
g_names = data.iloc[0].tolist()
p_names = data.iloc[:, 0].tolist()
p_labels = [widgets.Label(value=name) for name in p_names]

container.children += tuple(p_labels)

grid_list = []

for column in range(len(data.columns)):
    if column == 0:
        g_name = 'Parameters'
    else:
        g_name = g_names[column]

    g_widget = [widgets.Label(value=g_name)]

    for row in range(1, len(data)):
        p_value = data.iloc[row, column]

        try:
            value = float(p_value)
            widget = widgets.FloatText(value=value, layout=
               widgets.Layout(width='auto'))
        except ValueError:
            widget = widgets.Label(value=p_value)

        g_widget.append(widget)

    grid_list.append(g_widget)

container.children = [widgets.VBox(children=row) for row in
   grid_list]

filename_input = widgets.Text(value='Generators-nuevo',
   description='Nombre:')
button = widgets.Button(description='Guardar␣cambios')

def save_changes(_):
    filename = filename_input.value.strip()
    updateCSV(data, filename)

button.on_click(save_changes)

display(widgets.VBox([container, button, filename_input]))
```

widget_demand.py

```
 from IPython.display import display, clear_output
import ipywidgets as widgets
import pandas as pd

demand_data = pd.read_csv('Data/Input/Parameters/Demand.csv')

d_dict = {}


filename_input = widgets.Text(value='Demand-nuevo', description
   ='Nombre:')
button = widgets.Button(description='Guardar cambios')

def save_changes(_):
    filename = filename_input.value.strip()
    updateCSV(filename)

button.on_click(save_changes)

for index, row in demand_data.iterrows():
    p_widget = widgets.Label(value=row['p'], layout=widgets.
       Layout(width='40px'))
    value_widget = widgets.FloatText(value=row['p_d'])
    d_dict[row['p']] = (p_widget, value_widget)

demand_widgets = widgets.VBox([widgets.HBox([p_widget,
   value_widget]) for p_widget, value_widget in d_dict.values()
   ])

def updateCSV(filename):
    for period, (p_widget, value_widget) in d_dict.items():
        demand_data.loc[demand_data['p'] == period, 'p_d'] =
           value_widget.value
    filename = filename + '.csv'
    path = 'Data/Input/Parameters/' + filename
    df.to_csv(path,index=false)

update_button = widgets.Button(description='Guardar Cambios')
update_button.on_click(save_changes)

def sumar_button_clicked(_):
    rango_sumar_str = rango_sumar.value.strip()
    sumando = float(cantidad_sumar.value)
    print(rango_sumar_str)
    if rango_sumar_str:
        rango_sumar_parts = rango_sumar_str.split('-')
        period0_sumar = int(rango_sumar_parts[0].strip())
        end_period = int(rango_sumar_parts[1].strip())
        for period in range(168):
```

```
            if period0_sumar <= period <= end_period:
                d_dict['p'+str(period)][1].value += sumando
    demand_widgets = widgets.VBox([widgets.HBox([p_widget,
        value_widget]) for p_widget, value_widget in d_dict.
        values()])
rango_sumar = widgets.Text(description='Rango:', value='1-168')
cantidad_sumar = widgets.Text(description='Cantidad:', value='0
    ')
sumar_button = widgets.Button(description='Sumar')
sumar_button.on_click(sumar_button_clicked)
sumar_widgets = widgets.HBox([sumar_button, rango_sumar,
    cantidad_sumar])


def multiplicar_button_clicked(_):
    rango_multiplicar_str = rango_multiplicar.value.strip()
    factor = float(cantidad_multiplicar.value)
    print(rango_multiplicar_str)
    if rango_multiplicar_str:
        rango_multiplicar_parts = rango_multiplicar_str.split('
            -')
        period0_multiplicar = int(rango_multiplicar_parts[0].
            strip())
        periodf_multiplicar = int(rango_multiplicar_parts[1].
            strip())
        for period in range(168):
            if period0_multiplicar <= period <=
                periodf_multiplicar:
                d_dict['p'+str(period)][1].value *= factor
    demand_widgets = widgets.VBox([widgets.HBox([p_widget,
        value_widget]) for p_widget, value_widget in d_dict.
        values()])
rango_multiplicar = widgets.Text(description='Rango:', value='
    1-168')
cantidad_multiplicar = widgets.Text(description='Cantidad:',
    value='0')
multiplicar_button = widgets.Button(description='Multiplicar')
multiplicar_button.on_click(multiplicar_button_clicked)
multiplicar_widgets = widgets.HBox([multiplicar_button,
    rango_multiplicar, cantidad_multiplicar])




filename_input = widgets.Text(value='Demand-nuevo', description
    ='Nombre:')
```

```
guardar_widgets = widgets.HBox([button, filename_input])



display(widgets.VBox([guardar_widgets,sumar_widgets,
    multiplicar_widgets, demand_widgets,guardar_widgets,
    sumar_widgets,multiplicar_widgets]))
# display(sumar_widgets)
```

solver.py

```
    from pyomo.environ import *
from pathlib import Path
# import gurobipy
# Output figures
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from pyomo.environ import Reals, PositiveReals,
    NonPositiveReals, NegativeReals, NonNegativeReals,
    PercentFraction, \
     UnitInterval, Integers, PositiveIntegers,
        NonPositiveIntegers, NegativeIntegers,
        NonNegativeIntegers, Binary
# import time
# Data management and treatment
import pandas as pd

path_model_in_set = Path('Data/Input/Sets/')
path_model_in_par = Path('Data/Input/Parameters/')
path_model_out    = Path('Data/Output/')
path_results      = Path('Data/Output/')

m = AbstractModel()

#generadores 1,2,3
m.g = Set()
m.d = Set()
m.p = Set(ordered=True)

#Definici n de par metros

#auxiliary load factor
m.p_k = Param(m.g, within = NonNegativeReals)

#Consumo variable de combustible
m.p_alfa = Param(m.g, within = NonNegativeReals)
```

```
#Potencia M xima del generador
m.p_qmax = Param(m.g, within = NonNegativeReals)

#Potencia M nima del generador
m.p_qmin = Param(m.g, within = NonNegativeReals)

#Demanda del sistema
m.p_d = Param(m.p,within = NonNegativeReals)

#Coste del combustible     /MW
m.p_f = Param(m.g, within = NonNegativeReals)

#Coste de arranque
m.p_ca = Param(m.g, within = NonNegativeReals)

#Emisiones
m.p_e = Param(m.g, within = NonNegativeReals)

#Coste emisiones
m.p_ce = Param(m.g, within = NonNegativeReals)

#rampa subida
m.p_rs = Param(m.g, within = NonNegativeReals)

#rampa bajada
m.p_rb = Param(m.g, within = NonNegativeReals)

#Beta
m.p_beta = Param(m.g, within = NonNegativeReals)

#Gamma
m.p_gamma = Param(m.g, within = NonNegativeReals)

#Theta
m.p_theta = Param(m.g, within = NonNegativeReals)

#Rendimiento de bombeo de una hydro
m.p_rend = Param(m.g, within  = NonNegativeReals)

#Inflows en el embalse h durante el periodo p
m.p_i = Param (m.p, m.g, within = NonNegativeReals)

#Agua Inicial en una reserva hydro
m.p_w0 = Param (m.g, within = NonNegativeReals)

#Agua final en una reserva
m.p_wfin = Param(m.g, within = NonNegativeReals)

#Limite max de bombeo
```

```
m.p_bmax = Param (m.g, within = NonNegativeReals)

#Max water capacity
m.p_wmax = Param (m.g, within = NonNegativeReals)

#Min water amount
m.p_wmin = Param (m.g, within = NonNegativeReals)

#Reserva Rodante
m.p_rod = Param (m.p, within = NonNegativeReals, mutable = True
    , default = 0)

#Coste de demanda no servida
m.p_cens = Param (within = NonNegativeReals,initialize=180.0)

#Tipo de generador: thermal o hydro
m.p_type = Param(m.g, within = Any)

#Potencia Solar disponible durante el periodo p
m.p_solar = Param(m.p,within = NonNegativeReals)

#Potencia e lica disponible durante el periodo p
m.p_wind = Param(m.p, within= NonNegativeReals)

#Estado de acoplamiento del generador en instante inicial
m.p_u0 = Param(m.g, within = Binary)

#Cantidad m nima de flujo de agua en hydro h en periodo p
m.p_flu = Param(m.g,m.p, within = NonNegativeReals)
m.p_o = Param(m.g, within = NonNegativeReals)


#Definici n de variables

#Variable de arranque de un grupo
m.v_y = Var(m.g, m.p, within=Binary)

#Variable de parada de un grupo
m.v_z = Var(m.g,m.p, within=Binary)

#Variable de unit commitment
m.v_u = Var(m.g, m.p, within=Binary)

#Potencia generada por grupo en un periodo
m.v_q = Var(m.g, m.p, within=NonNegativeReals)

#Exceso de potencia generada sobre minimo tecnico
m.v_deltaq = Var(m.g, m.p,within=NonNegativeReals)
```

```
#Energy stored in water
m.v_w = Var(m.g, m.p, within = NonNegativeReals)

#Spills de una reserva hydro
m.v_s = Var(m.g, m.p, within = NonNegativeReals)

#Bombeo en una reserva hydro
m.v_b = Var(m.g, m.p, within = NonNegativeReals)

#Cantidad de potencia solar utilizada en el periodo p
m.v_solar = Var(m.p, within = NonNegativeReals)

#Cantidad de potencia e lica utilizada en el periodo p
m.v_wind = Var(m.p, within = NonNegativeReals)

#Potencia no servida
m.v_pns = Var(m.p, within = NonNegativeReals)

# %% Declaration of dynamic sets
def setdin_thermal_generators(m, g):
    return m.p_type[g] == 'thermal'
m.t = Set(ordered=True, initialize=m.g, filter=
   setdin_thermal_generators, doc="Thermal generators")


def setdin_hydro_generators(m, g):
    return m.p_type[g] == 'hydro'
m.h = Set(ordered=True, initialize=m.g, filter=
   setdin_hydro_generators, doc="Hydro generators")


#Definici n de funci n objetivo

def e_Fobj(m):
    return (sum( m.p_cens * m.v_pns[p] + \
                 sum( m.p_f[t] * \
                      ( m.p_beta[t]  * m.v_u[t,p] + \
                        m.p_gamma[t] * m.v_y[t,p] + \
                        m.p_theta[t] * m.v_z[t,p] + \
                        m.p_alfa[t] * m.v_q[t,p] / m.p_k[t]) +
                            \
                        m.p_o[t] * m.v_q[t,p] / m.p_k[t] for t
                            in m.t) for p in m.p))
m.ObjectiveFunction =Objective(rule = e_Fobj, sense = minimize)


def e_DeltaQDefinition(m,t,p): #6
    return (m.v_q[t,p] == m.v_u[t,p] * m.p_k[t] * m.p_qmin[t] +
        m.v_deltaq[t,p])
```

```
m.DeltaQDefinition = Constraint(m.t, m.p, rule=
    e_DeltaQDefinition)

def e_DeltaQBounds(m,t,p): #7
    return (m.v_deltaq[t,p] <= m.v_u[t,p] * m.p_k[t] * (m.
        p_qmax[t] - m.p_qmin[t]))
m.DeltaQBounds = Constraint(m.t, m.p, rule = e_DeltaQBounds)
#
# ###A adir maintenance rule eq 8 pag 17 de 64
#
def e_RampingUpConstraint(m,t,p): #9
    if p != m.p.first():
        return (m.v_deltaq[t,p] - m.v_deltaq[t,m.p.prev(p,1)]
            <= m.p_rs[t])
    else:
        return Constraint.Skip
m.RampingUpConstraint = Constraint(m.t, m.p, rule =
    e_RampingUpConstraint)

def e_RampingDownConstraint(m,t,p): #10
    if p != m.p.first():
        return (m.v_deltaq[t,m.p.prev(p,1)]-m.v_deltaq[t,p] <=
            m.p_rb[t])
    else:
        return Constraint.Skip
m.RampingDownConstraint = Constraint(m.t, m.p, rule =
    e_RampingDownConstraint)

def e_UnitCommitmentConsistency(m,t,p): #11
    if p != m.p.first():
     return (m.v_y[t,p] - m.v_z[t,p] == m.v_u[t,p] - m.v_u[t,m.
        p.prev(p,1)])
    else:
        return (m.v_y[t,p] - m.v_z[t,p] == m.v_u[t,p] - m.p_u0[
            t])
m.UnnitCommitmentConsistency = Constraint(m.t, m.p, rule =
    e_UnitCommitmentConsistency)

def e_WaterReserves(m,h,p): #16
    if p == m.p.first():
        return (m.v_w[h,p] == m.p_w0[h] - ( m.v_q[h,p] + m.v_s[
            h,p] - m.p_rend[h] * m.v_b[h,p] ) + m.p_i[p,h] )
    else:
        return (m.v_w[h, p] == m.v_w[h, m.p.prev(p, 1)] - (m.
            v_q[h, p] + m.v_s[h, p] - m.p_rend[h] * m.v_b[h, p])
             + m.p_i[p, h])
m.WaterReserves = Constraint(m.h, m.p, rule = e_WaterReserves)

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

```
    if p == m.p.last():
        return (m.v_w[h,p] == m.p_wfin[h])
    else:
        return Constraint.Skip
m.FinalWaterReserves = Constraint(m.h,m.p, rule =
    e_FinalWaterReserves)
#
def e_HydroOutputUpperLimit(m,h,p): #17
    return (m.v_q[h,p] <= m.p_k[h] * m.p_qmax[h])
m.HydroOutputUpperLimit = Constraint (m.h, m.p, rule =
    e_HydroOutputUpperLimit)

#
def e_PumpingUpperLimit(m,h,p): # 19
    return (m.v_b[h,p] <= m.p_bmax[h])
m.PumpingUpperLimit = Constraint(m.h, m.p, rule =
    e_PumpingUpperLimit)
#
def e_WaterReserveLowerVolumeLimit(m,h,p): #20
    return (m.p_wmin[h] <= m.v_w[h,p])
m.WaterReserveLowerVolumeLimit = Constraint(m.h, m.p, rule =
    e_WaterReserveLowerVolumeLimit)
#
def e_WaterReserveUpperVolumeLimit(m, h, p): #20
    return (m.v_w[h,p] <= m.p_wmax[h])
m.WaterReserveUpperVolumeLimit = Constraint(m.h, m.p, rule=
    e_WaterReserveUpperVolumeLimit)

def e_SolarPowerUpperLimit(m,p):#0.2
    return (m.v_solar[p] <= m.p_solar[p] * 0.2)
m.SolarPowerUpperLimit = Constraint(m.p, rule=
    e_SolarPowerUpperLimit)

def e_WindPowerUpperLimit(m,p):
    return (m.v_wind[p] <= m.p_wind[p] * 0.3)#0.3
m.WindPowerUpperLimit = Constraint(m.p, rule=
    e_WindPowerUpperLimit)

def e_DemandBalance(m,p):
    return ( sum(m.v_q[t,p] for t in m.t ) + sum(m.v_q[h,p] - m
        .v_b[h,p] for h in m.h)+ m.v_wind[p] + m.v_solar[p] + m.
        v_pns[p] == m.p_d[p])
m.DemandBalance = Constraint(m.p,rule=e_DemandBalance)

def e_SpinningReserve(m,p):
    return ( sum(m.v_u[t,p] * m.p_k[t] * m.p_qmax[t] - m.v_q[t,
        p] for t in m.t) >= m.p_rod[p])
m.SpinningReserve = Constraint(m.p,rule = e_SpinningReserve)
```

```
data = DataPortal()



# p_list = []
# list1 = []
# list2 = []
# list3 = []
# for p in range(1,169):
#     p_list.append('p'+str(p))
#     list1.append(0.01)
#     list2.append(0.5*math.exp(-(p-1)/100))
#     list3.append(0)
#
# dict = {'p':p_list,'HYDRO_RES':list1,'HYDRO_ROR':list2,'
    HYDRO_PUM':list3}
#
# df = pd.DataFrame(dict)
#
# print(df)
# df.to_csv('Inflows.csv', index = False)

# df = pd.read_csv(str(str(path_model_in_par.joinpath('
    Data_Demands.csv'))))
# df['p_d'] = df['p_d'].map(lambda p_d: p_d * 0.01)
# df.rename(columns = {'p_d':'p_rod'}, inplace = True)
# df.to_csv(str(path_model_in_par.joinpath('SpinningReserve.csv
    ')), index = False)
# print(df)


#data.load(filename='gens.dat',set=m.g)

# Load of sets
data.load(filename=str(path_model_in_set.joinpath('g.csv')),
    format='set',set='g')
data.load(filename=str(path_model_in_set.joinpath('d.csv')),
    format='set',set='d')
data.load(filename=str(path_model_in_set.joinpath('p.csv')),
    format='set',set='p')

# Load of parameters
data.load(filename=str(path_model_in_par.joinpath('Generators.
    csv')), index=['g'], param=['p_alfa','p_beta','p_gamma','
    p_theta','p_rs','p_rb','p_f','p_o','p_qmax','p_qmin','p_u0',
    'p_modo','p_bmax','p_wmax','p_w0','p_wmin','p_wfin','p_k','
    p_rend','p_type'])
```

```
data.load(filename=str(path_model_in_par.joinpath('Demand.csv')
    ), index=['p'], param='p_d')
# data.load(filename=str(path_model_in_par.joinpath('Scalars.
    csv')), param=['p_cens'])
data.load(filename=str(path_model_in_par.joinpath('Wind.csv')),
     param=['p_wind'])
data.load(filename=str(path_model_in_par.joinpath('Solar.csv'))
    , param=['p_solar'])
data.load(filename=str(path_model_in_par.joinpath('Inflows.csv'
    )),index=['p','g'],param = ['p_i'], format = 'array')
data.load(filename=str(path_model_in_par.joinpath('
    SpinningReserve.csv')),index=['p'],param = ['p_rod'])

instance = m.create_instance(data)

solver = SolverFactory('glpk')
solver.options['mipgap'] = 0.1
instance.dual = Suffix(direction=Suffix.IMPORT)
solver_results = solver.solve(instance, tee=False)
solver_results.write()  # Resumen de los resultados del solver
instance.solutions.load_from(solver_results)  # Necesario para
    fijar los valores de las variables binarias


df = pd.DataFrame.from_dict(instance.v_q.extract_values(),
    orient='index', columns=[str(instance.v_q)])
df.index.names = ['Generator-Period']
df.reset_index(inplace=True)
df[['Generator', 'Period']] = pd.DataFrame(df['Generator-Period
    '].tolist())
df['Period'] = pd.Categorical(df['Period'], categories=['p' +
    str(i) for i in range(1, 169)], ordered=True)
df = df.sort_values('Period')
df_out = df.pivot(index='Period', columns='Generator', values='
    v_q')
# generator_order = ['HYDRO_PUM', 'HYDRO_ROR', 'HYDRO_RES', '
    GAS', 'FUELOIL', 'CCGT', 'ANTHRACITE', 'BITUMINOUS', '
    SUBBITUMIN', 'LIGNITE', 'NUCLEAR']
generator_order=['NUCLEAR', 'LIGNITE', 'SUBBITUMIN', '
    BITUMINOUS', 'ANTHRACITE', 'CCGT', 'FUELOIL', 'GAS', '
    HYDRO_RES', 'HYDRO_ROR','HYDRO_PUM']
df_out = df_out.reindex(columns=generator_order)

solar_data = pd.DataFrame.from_dict(instance.v_solar.
    extract_values(), orient='index', columns=[str(instance.
    v_solar)])
solar_data.index.names = ['Period']
solar_data.reset_index(inplace=True)
solar_data = solar_data.rename(columns={'v_solar': 'SOLAR'})
```

```
merge_solar = pd.merge(df_out, solar_data, left_index=True,
    right_on='Period')
merge_solar.set_index('Period', inplace=True)

wind_data = pd.DataFrame.from_dict(instance.v_wind.
    extract_values(), orient='index',columns=[str(instance.
    v_wind)])
wind_data.index.names = ['Period']
wind_data.reset_index(inplace=True)
wind_data = wind_data.rename(columns={'v_wind': 'WIND'})
merge_wind = pd.merge(merge_solar, wind_data, left_index=True,
    right_on='Period')
merge_wind.set_index('Period', inplace=True)

pns_data = pd.DataFrame.from_dict(instance.v_pns.extract_values
    (), orient='index',columns=[str(instance.v_pns)])
pns_data.index.names = ['Period']
pns_data.reset_index(inplace=True)
pns_data = pns_data.rename(columns={'v_pns': 'PNS'})
merged_data = pd.merge(merge_wind, pns_data, left_index=True,
    right_on='Period')
merged_data.set_index('Period', inplace=True)


demand_df = pd.DataFrame.from_dict(instance.p_d.extract_values
    (), orient='index', columns=[str(instance.p_d)])
demand_df.index.names = ['Period']
demand_df.reset_index(inplace=True)
demand_df = demand_df.rename(columns={'p_d': 'Demand'})

generators_df = merged_data

merged_data = pd.merge(merged_data, demand_df, left_index=True,
     right_on='Period')
merged_data.set_index('Period', inplace=True)
df = pd.DataFrame.from_dict(instance.v_w.extract_values(),
    orient='index', columns=[str(instance.v_w)])
df = df.dropna()
df.index.names = ['Generator-Period']
df.reset_index(inplace=True)
df[['Generator', 'Period']] = pd.DataFrame(df['Generator-Period
    '].tolist())
df['Period'] = pd.Categorical(df['Period'], categories=['p' +
    str(i) for i in range(1, 169)], ordered=True)
df = df.sort_values('Period')
df_out = df.pivot(index='Period', columns='Generator', values='
    v_w')
df_out = df_out.drop(columns='HYDRO_ROR')
df_pum = df_out.drop(columns='HYDRO_RES')
```

```
df_res = df_out.drop(columns='HYDRO_PUM')

df_pum.rename(columns={'HYDRO_PUM':'Water_PUM'}, inplace = True
    )
df_res.rename(columns={'HYDRO_RES':'Water_RES'}, inplace = True
    )

reserves_df = pd.merge(df_pum, df_res, left_index=True,
    right_on='Period')

merged_data = pd.merge(merged_data, reserves_df, left_index=
    True, right_on='Period')
from pyomo.environ import *
from pathlib import Path
# import gurobipy
# Output figures
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from pyomo.environ import Reals, PositiveReals,
    NonPositiveReals, NegativeReals, NonNegativeReals,
    PercentFraction, \
      UnitInterval, Integers, PositiveIntegers,
         NonPositiveIntegers, NegativeIntegers,
         NonNegativeIntegers, Binary
# import time
# Data management and treatment
import pandas as pd

path_model_in_set = Path('Data/Input/Sets/')
path_model_in_par = Path('Data/Input/Parameters/')
path_model_out    = Path('Data/Output/')
path_results      = Path('Data/Output/')

m = AbstractModel()

#generadores 1,2,3
m.g = Set()
m.d = Set()
m.p = Set(ordered=True)

#Definici n de par metros

#auxiliary load factor
m.p_k = Param(m.g, within = NonNegativeReals)

#Consumo variable de combustible
m.p_alfa = Param(m.g, within = NonNegativeReals)
```

```
#Potencia M xima del generador
m.p_qmax = Param(m.g, within = NonNegativeReals)

#Potencia M nima del generador
m.p_qmin = Param(m.g, within = NonNegativeReals)

#Demanda del sistema
m.p_d = Param(m.p,within = NonNegativeReals)

#Coste del combustible    /MW
m.p_f = Param(m.g, within = NonNegativeReals)

#Coste de arranque
m.p_ca = Param(m.g, within = NonNegativeReals)

#Emisiones
m.p_e = Param(m.g, within = NonNegativeReals)

#Coste emisiones
m.p_ce = Param(m.g, within = NonNegativeReals)

#rampa subida
m.p_rs = Param(m.g, within = NonNegativeReals)

#rampa bajada
m.p_rb = Param(m.g, within = NonNegativeReals)

#Beta
m.p_beta = Param(m.g, within = NonNegativeReals)

#Gamma
m.p_gamma = Param(m.g, within = NonNegativeReals)

#Theta
m.p_theta = Param(m.g, within = NonNegativeReals)

#Rendimiento de bombeo de una hydro
m.p_rend = Param(m.g, within  = NonNegativeReals)

#Inflows en el embalse h durante el periodo p
m.p_i = Param (m.p, m.g, within = NonNegativeReals)

#Agua Inicial en una reserva hydro
m.p_w0 = Param (m.g, within = NonNegativeReals)

#Agua final en una reserva
m.p_wfin = Param(m.g, within = NonNegativeReals)

#Limite max de bombeo
```

```
m.p_bmax = Param (m.g, within = NonNegativeReals)

#Max water capacity
m.p_wmax = Param (m.g, within = NonNegativeReals)

#Min water amount
m.p_wmin = Param (m.g, within = NonNegativeReals)

#Reserva Rodante
m.p_rod = Param (m.p, within = NonNegativeReals, mutable = True
    , default = 0)

#Coste de demanda no servida
m.p_cens = Param (within = NonNegativeReals,initialize=180.0)

#Tipo de generador: thermal o hydro
m.p_type = Param(m.g, within = Any)

#Potencia Solar disponible durante el periodo p
m.p_solar = Param(m.p,within = NonNegativeReals)

#Potencia e lica disponible durante el periodo p
m.p_wind = Param(m.p, within= NonNegativeReals)

#Estado de acoplamiento del generador en instante inicial
m.p_u0 = Param(m.g, within = Binary)

#Cantidad m nima de flujo de agua en hydro h en periodo p
m.p_flu = Param(m.g,m.p, within = NonNegativeReals)
m.p_o = Param(m.g, within = NonNegativeReals)


#Definici n de variables

#Variable de arranque de un grupo
m.v_y = Var(m.g, m.p, within=Binary)

#Variable de parada de un grupo
m.v_z = Var(m.g,m.p, within=Binary)

#Variable de unit commitment
m.v_u = Var(m.g, m.p, within=Binary)

#Potencia generada por grupo en un periodo
m.v_q = Var(m.g, m.p, within=NonNegativeReals)

#Exceso de potencia generada sobre minimo tecnico
m.v_deltaq = Var(m.g, m.p,within=NonNegativeReals)
```

```
#Energy stored in water
m.v_w = Var(m.g, m.p, within = NonNegativeReals)

#Spills de una reserva hydro
m.v_s = Var(m.g, m.p, within = NonNegativeReals)

#Bombeo en una reserva hydro
m.v_b = Var(m.g, m.p, within = NonNegativeReals)

#Cantidad de potencia solar utilizada en el periodo p
m.v_solar = Var(m.p, within = NonNegativeReals)

#Cantidad de potencia e lica utilizada en el periodo p
m.v_wind = Var(m.p, within = NonNegativeReals)

#Potencia no servida
m.v_pns = Var(m.p, within = NonNegativeReals)

# %% Declaration of dynamic sets
def setdin_thermal_generators(m, g):
    return m.p_type[g] == 'thermal'
m.t = Set(ordered=True, initialize=m.g, filter=
   setdin_thermal_generators, doc="Thermal generators")


def setdin_hydro_generators(m, g):
    return m.p_type[g] == 'hydro'
m.h = Set(ordered=True, initialize=m.g, filter=
   setdin_hydro_generators, doc="Hydro generators")


#Definici n de funci n objetivo

def e_Fobj(m):
    return (sum( m.p_cens * m.v_pns[p] + \
                 sum( m.p_f[t] * \
                      ( m.p_beta[t]  * m.v_u[t,p] + \
                        m.p_gamma[t] * m.v_y[t,p] + \
                        m.p_theta[t] * m.v_z[t,p] + \
                        m.p_alfa[t] * m.v_q[t,p] / m.p_k[t]) +
                            \
                        m.p_o[t] * m.v_q[t,p] / m.p_k[t] for t
                            in m.t) for p in m.p))
m.ObjectiveFunction =Objective(rule = e_Fobj, sense = minimize)


def e_DeltaQDefinition(m,t,p): #6
    return (m.v_q[t,p] == m.v_u[t,p] * m.p_k[t] * m.p_qmin[t] +
        m.v_deltaq[t,p])
```

```
m.DeltaQDefinition = Constraint(m.t, m.p, rule=
    e_DeltaQDefinition)

def e_DeltaQBounds(m,t,p): #7
    return (m.v_deltaq[t,p] <= m.v_u[t,p] * m.p_k[t] * (m.
        p_qmax[t] - m.p_qmin[t]))
m.DeltaQBounds = Constraint(m.t, m.p, rule = e_DeltaQBounds)
#
# ###A adir maintenance rule eq 8 pag 17 de 64
#
def e_RampingUpConstraint(m,t,p): #9
    if p != m.p.first():
        return (m.v_deltaq[t,p] - m.v_deltaq[t,m.p.prev(p,1)]
            <= m.p_rs[t])
    else:
        return Constraint.Skip
m.RampingUpConstraint = Constraint(m.t, m.p, rule =
    e_RampingUpConstraint)

def e_RampingDownConstraint(m,t,p): #10
    if p != m.p.first():
        return (m.v_deltaq[t,m.p.prev(p,1)]-m.v_deltaq[t,p] <=
            m.p_rb[t])
    else:
        return Constraint.Skip
m.RampingDownConstraint = Constraint(m.t, m.p, rule =
    e_RampingDownConstraint)

def e_UnitCommitmentConsistency(m,t,p): #11
    if p != m.p.first():
     return (m.v_y[t,p] - m.v_z[t,p] == m.v_u[t,p] - m.v_u[t,m.
        p.prev(p,1)])
    else:
        return (m.v_y[t,p] - m.v_z[t,p] == m.v_u[t,p] - m.p_u0[
            t])
m.UnnitCommitmentConsistency = Constraint(m.t, m.p, rule =
    e_UnitCommitmentConsistency)

def e_WaterReserves(m,h,p): #16
    if p == m.p.first():
        return (m.v_w[h,p] == m.p_w0[h] - ( m.v_q[h,p] + m.v_s[
            h,p] - m.p_rend[h] * m.v_b[h,p] ) + m.p_i[p,h] )
    else:
        return (m.v_w[h, p] == m.v_w[h, m.p.prev(p, 1)] - (m.
            v_q[h, p] + m.v_s[h, p] - m.p_rend[h] * m.v_b[h, p])
             + m.p_i[p, h])
m.WaterReserves = Constraint(m.h, m.p, rule = e_WaterReserves)

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

```
    if p == m.p.last():
        return (m.v_w[h,p] == m.p_wfin[h])
    else:
        return Constraint.Skip
m.FinalWaterReserves = Constraint(m.h,m.p, rule =
    e_FinalWaterReserves)
#
def e_HydroOutputUpperLimit(m,h,p): #17
    return (m.v_q[h,p] <= m.p_k[h] * m.p_qmax[h])
m.HydroOutputUpperLimit = Constraint (m.h, m.p, rule =
    e_HydroOutputUpperLimit)

#
def e_PumpingUpperLimit(m,h,p): # 19
    return (m.v_b[h,p] <= m.p_bmax[h])
m.PumpingUpperLimit = Constraint(m.h, m.p, rule =
    e_PumpingUpperLimit)
#
def e_WaterReserveLowerVolumeLimit(m,h,p): #20
    return (m.p_wmin[h] <= m.v_w[h,p])
m.WaterReserveLowerVolumeLimit = Constraint(m.h, m.p, rule =
    e_WaterReserveLowerVolumeLimit)
#
def e_WaterReserveUpperVolumeLimit(m, h, p): #20
    return (m.v_w[h,p] <= m.p_wmax[h])
m.WaterReserveUpperVolumeLimit = Constraint(m.h, m.p, rule=
    e_WaterReserveUpperVolumeLimit)

def e_SolarPowerUpperLimit(m,p):#0.2
    return (m.v_solar[p] <= m.p_solar[p] * 0.2)
m.SolarPowerUpperLimit = Constraint(m.p, rule=
    e_SolarPowerUpperLimit)

def e_WindPowerUpperLimit(m,p):
    return (m.v_wind[p] <= m.p_wind[p] * 0.3)#0.3
m.WindPowerUpperLimit = Constraint(m.p, rule=
    e_WindPowerUpperLimit)

def e_DemandBalance(m,p):
    return ( sum(m.v_q[t,p] for t in m.t ) + sum(m.v_q[h,p] - m
        .v_b[h,p] for h in m.h)+ m.v_wind[p] + m.v_solar[p] + m.
        v_pns[p] == m.p_d[p])
m.DemandBalance = Constraint(m.p,rule=e_DemandBalance)

def e_SpinningReserve(m,p):
    return ( sum(m.v_u[t,p] * m.p_k[t] * m.p_qmax[t] - m.v_q[t,
        p] for t in m.t) >= m.p_rod[p])
m.SpinningReserve = Constraint(m.p,rule = e_SpinningReserve)
```

```
data = DataPortal()



# p_list = []
# list1 = []
# list2 = []
# list3 = []
# for p in range(1,169):
#     p_list.append('p'+str(p))
#     list1.append(0.01)
#     list2.append(0.5*math.exp(-(p-1)/100))
#     list3.append(0)
#
# dict = {'p':p_list,'HYDRO_RES':list1,'HYDRO_ROR':list2,'
   HYDRO_PUM':list3}
#
# df = pd.DataFrame(dict)
#
# print(df)
# df.to_csv('Inflows.csv', index = False)

# df = pd.read_csv(str(str(path_model_in_par.joinpath('
   Data_Demands.csv'))))
# df['p_d'] = df['p_d'].map(lambda p_d: p_d * 0.01)
# df.rename(columns = {'p_d':'p_rod'}, inplace = True)
# df.to_csv(str(path_model_in_par.joinpath('SpinningReserve.csv
   ')), index = False)
# print(df)


#data.load(filename='gens.dat',set=m.g)

# Load of sets
data.load(filename=str(path_model_in_set.joinpath('g.csv')),
   format='set',set='g')
data.load(filename=str(path_model_in_set.joinpath('d.csv')),
   format='set',set='d')
data.load(filename=str(path_model_in_set.joinpath('p.csv')),
   format='set',set='p')

# Load of parameters
data.load(filename=str(path_model_in_par.joinpath('Generators.
   csv')), index=['g'], param=['p_alfa','p_beta','p_gamma','
   p_theta','p_rs','p_rb','p_f','p_o','p_qmax','p_qmin','p_u0',
   'p_modo','p_bmax','p_wmax','p_w0','p_wmin','p_wfin','p_k','
   p_rend','p_type'])
```

```
data.load(filename=str(path_model_in_par.joinpath('Demand.csv')
    ), index=['p'], param='p_d')
# data.load(filename=str(path_model_in_par.joinpath('Scalars.
    csv')), param=['p_cens'])
data.load(filename=str(path_model_in_par.joinpath('Wind.csv')),
     param=['p_wind'])
data.load(filename=str(path_model_in_par.joinpath('Solar.csv'))
    , param=['p_solar'])
data.load(filename=str(path_model_in_par.joinpath('Inflows.csv'
    )),index=['p','g'],param = ['p_i'], format = 'array')
data.load(filename=str(path_model_in_par.joinpath('
    SpinningReserve.csv')),index=['p'],param = ['p_rod'])

instance = m.create_instance(data)

solver = SolverFactory('glpk')
solver.options['mipgap'] = 0.1
instance.dual = Suffix(direction=Suffix.IMPORT)
solver_results = solver.solve(instance, tee=True)
solver_results.write()  # Resumen de los resultados del solver
instance.solutions.load_from(solver_results)  # Necesario para
    fijar los valores de las variables binarias

df = pd.DataFrame.from_dict(instance.v_q.extract_values(),
    orient='index', columns=[str(instance.v_q)])
df.index.names = ['Generator-Period']
df.reset_index(inplace=True)
df[['Generator', 'Period']] = pd.DataFrame(df['Generator-Period
    '].tolist())
df['Period'] = pd.Categorical(df['Period'], categories=['p' +
    str(i) for i in range(1, 169)], ordered=True)
df = df.sort_values('Period')
df_out = df.pivot(index='Period', columns='Generator', values='
    v_q')
# generator_order = ['HYDRO_PUM', 'HYDRO_ROR', 'HYDRO_RES', '
    GAS', 'FUELOIL', 'CCGT', 'ANTHRACITE', 'BITUMINOUS', '
    SUBBITUMIN', 'LIGNITE', 'NUCLEAR']
generator_order=['NUCLEAR', 'LIGNITE', 'SUBBITUMIN', '
    BITUMINOUS', 'ANTHRACITE', 'CCGT', 'FUELOIL', 'GAS', '
    HYDRO_RES', 'HYDRO_ROR','HYDRO_PUM']
df_out = df_out.reindex(columns=generator_order)

solar_data = pd.DataFrame.from_dict(instance.v_solar.
    extract_values(), orient='index', columns=[str(instance.
    v_solar)])
solar_data.index.names = ['Period']
solar_data.reset_index(inplace=True)
solar_data = solar_data.rename(columns={'v_solar': 'SOLAR'})
```

```
merge_solar = pd.merge(df_out, solar_data, left_index=True,
    right_on='Period')
merge_solar.set_index('Period', inplace=True)

wind_data = pd.DataFrame.from_dict(instance.v_wind.
    extract_values(), orient='index',columns=[str(instance.
    v_wind)])
wind_data.index.names = ['Period']
wind_data.reset_index(inplace=True)
wind_data = wind_data.rename(columns={'v_wind': 'WIND'})
merge_wind = pd.merge(merge_solar, wind_data, left_index=True,
    right_on='Period')
merge_wind.set_index('Period', inplace=True)

pns_data = pd.DataFrame.from_dict(instance.v_pns.extract_values
    (), orient='index',columns=[str(instance.v_pns)])
pns_data.index.names = ['Period']
pns_data.reset_index(inplace=True)
pns_data = pns_data.rename(columns={'v_pns': 'PNS'})
merged_df = pd.merge(merge_wind, pns_data, left_index=True,
    right_on='Period')
merged_df.set_index('Period', inplace=True)


demand_df = pd.DataFrame.from_dict(instance.p_d.extract_values
    (), orient='index', columns=[str(instance.p_d)])
demand_df.index.names = ['Period']
demand_df.reset_index(inplace=True)
demand_df = demand_df.rename(columns={'p_d': 'Demand'})

generators_df = merged_df

merged_df = pd.merge(merged_df, demand_df, left_index=True,
    right_on='Period')
merged_df.set_index('Period', inplace=True)
df = pd.DataFrame.from_dict(instance.v_w.extract_values(),
    orient='index', columns=[str(instance.v_w)])
df = df.dropna()
df.index.names = ['Generator-Period']
df.reset_index(inplace=True)
df[['Generator', 'Period']] = pd.DataFrame(df['Generator-Period
    '].tolist())
df['Period'] = pd.Categorical(df['Period'], categories=['p' +
    str(i) for i in range(1, 169)], ordered=True)
df = df.sort_values('Period')
df_out = df.pivot(index='Period', columns='Generator', values='
    v_w')
df_out = df_out.drop(columns='HYDRO_ROR')
df_pum = df_out.drop(columns='HYDRO_RES')
```

```
df_res = df_out.drop(columns='HYDRO_PUM')

df_pum.rename(columns={'HYDRO_PUM':'Water_PUM'}, inplace = True
    )
df_res.rename(columns={'HYDRO_RES':'Water_RES'}, inplace = True
    )

reserves_df = pd.merge(df_pum, df_res, left_index=True,
    right_on='Period')

merged_df = pd.merge(merged_df, reserves_df, left_index=True,
    right_on='Period')

merged_df.to_csv('Data/Output/Merged.csv')
reserves_df.to_csv('Data/Output/Reserves.csv')
generators_df.to_csv('Data/Output/Generation.csv')
demand_df.to_csv('Data/Output/Demand.csv')
```

GUI.ipynb

```
  from IPython.display import clear_output,display

clear_output(wait=True)
import ipywidgets as widgets

# modificar_datos_label = widgets.Label(value='Modificar Datos
    ')
global accordion
modificar_datos_opciones = ['Seleccionar␣Datos','Generadores',
    'Demanda']

dropdown = widgets.Dropdown(options=modificar_datos_opciones)

def on_option_selected(change):
    selected_option = change.new
    if selected_option == 'Generadores':
      clear_output()
      display(accordion)
      %run scripts/widget_generators.py
    if selected_option == 'Demanda':
      clear_output()
      display(accordion)
      %run scripts/widget_demand.py

dropdown.observe(on_option_selected, names='value')
```

```
# modificar_datos_container = widgets.VBox([
    modificar_datos_label,dropdown], layout=widgets.Layout(
    border = '2px solid lightgrey', padding='5px'))
# modificar_datos_container.layout.width = '500px'

# dropdown.layout = widgets.Layout(border = '2px solid
    lightgrey', padding='10px')
# dropdown.layout.width = '500px'
modificar_datos_container = widgets.VBox([dropdown],layout=
    widgets.Layout(border = '2px solid lightgrey', padding='5px'
    ) )
modificar_datos_container.layout.width = '500px'

resolver_button = widgets.Button(description = 'Resolver')
resolver_container = widgets.VBox([resolver_button],layout=
    widgets.Layout(border = '2px solid lightgrey', padding='5px'
    ))
resolver_container.layout.width = '500px'

def on_resolver_button_clicked(_):
  %run scripts/solver.py
resolver_button.on_click(on_resolver_button_clicked)

representar_resultados_label = widgets.Label(value='Representar
    Resultados')
representar_resultados_opciones = ['Generaci n','Reservas']


checkbox_nuclear = widgets.Checkbox(description='Nuclear',value
    =True)
checkbox_lignite = widgets.Checkbox(description='Lignite',value
    =True)
checkbox_subbitumin = widgets.Checkbox(description='Subbitumin'
    ,value=True)
checkbox_bituminous = widgets.Checkbox(description='Bituminous'
    ,value=True)
checkbox_anthracite = widgets.Checkbox(description='Anthacite',
    value=True)
checkbox_CCGT = widgets.Checkbox(description='CCGT',value=True)
checkbox_fueloil = widgets.Checkbox(description='Fuel oil',
    value=True)
checkbox_gas = widgets.Checkbox(description='Gas',value=True)
checkbox_hydrores = widgets.Checkbox(description='Reservoir',
    value=True)
checkbox_hydroror = widgets.Checkbox(description='Run-of-River'
    ,value=True)
checkbox_hydropum = widgets.Checkbox(description='Pump',value=
    True)
```

```
checkbox_solar = widgets.Checkbox(description='Solar',value=
    True)
checkbox_wind = widgets.Checkbox(description='Wind',value=True)
checkbox_pns = widgets.Checkbox(description='PNS',value=True)
checkbox_demand = widgets.Checkbox(description='Demand',value=
    True)

checkbox_widget = widgets.VBox([
    widgets.HBox([checkbox_nuclear, checkbox_lignite,
        checkbox_subbitumin]),
    widgets.HBox([checkbox_bituminous, checkbox_anthracite,
        checkbox_CCGT]),
    widgets.HBox([checkbox_fueloil, checkbox_gas,
        checkbox_hydrores]),
    widgets.HBox([checkbox_hydroror, checkbox_hydropum,
        checkbox_solar]),
    widgets.HBox([checkbox_wind, checkbox_pns, checkbox_demand
        ])
])

plot_gen_button = widgets.Button(description='Crear␣Plot')

def on_gen_button_clicked(_):
    checked_indexes = [index for index, checkbox in enumerate([
        checkbox_nuclear, checkbox_lignite, checkbox_subbitumin
            ,
        checkbox_bituminous, checkbox_anthracite, checkbox_CCGT
            ,
        checkbox_fueloil, checkbox_gas, checkbox_hydrores,
        checkbox_hydroror, checkbox_hydropum, checkbox_solar,
        checkbox_wind, checkbox_pns, checkbox_demand
    ]) if checkbox.value]
    # print(checked_indexes)
    generation_df = pd.read_csv('Data/Output/Generation.csv',
        index_col = 0)
    demand_df = pd.read_csv('Data/Output/Demand.csv', index_col
        = 0)
    merged_df = pd.merge(generation_df, demand_df, left_index=
        True, right_on='Period')

    selected_cols = [merged_df.columns[index] for index in
        checked_indexes]
    plot_df = merged_df[selected_cols]
    plot_df.plot(kind='area', stacked = False, linewidth=0)
plot_gen_button.on_click(on_gen_button_clicked)


def cerrar_plots():
  clear_output()
```

```
  display(accordion)
cerrar_plots_button = widgets.Button(description='Cerrar␣Plots'
    )
cerrar_plots_button.on_click(lambda _: cerrar_plots())

gen_box = widgets.VBox([checkbox_widget, plot_gen_button,
    cerrar_plots_button])

reserva_res = widgets.Checkbox(description='Reservoir',value=
    True)
reserva_pum = widgets.Checkbox(description='Pump',value=True)
plot_reserva_button = widgets.Button(description='Crear␣Plot')

def on_plot_reserva_button_clicked(_):
    selected_checkboxes = []
    if reserva_res.value:
        selected_checkboxes.append('reserva_res')
    if reserva_pum.value:
        selected_checkboxes.append('reserva_pum')
    print('Selected␣Checkboxes:', selected_checkboxes)
plot_reserva_button.on_click(on_plot_reserva_button_clicked)
reserva_buttons = widgets.VBox([plot_reserva_button,
    cerrar_plots_button])
reserva_box = widgets.VBox([reserva_res, reserva_pum,
    reserva_buttons])

plot_accordion = widgets.Accordion(children = [gen_box,
    reserva_box])
plot_accordion.set_title(0,'Generaci n')
plot_accordion.set_title(1,'Reservas')


accordion = widgets.Accordion(children = [
    modificar_datos_container,resolver_container,plot_accordion
    ])
accordion.set_title(0,'Modificar␣Datos')
accordion.set_title(1,'Resolver')
accordion.set_title(2,'Ver␣Resultados')
display(accordion)
```