UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DESARROLLO E IMPLEMENTACIÓN DE UN MODELO ESTOCÁSTICO DE ARBITRAJE ENERGÉTICO: REFINANDO EL PROCESO DE OPTIMIZACIÓN EN EL MERCADO ELÉCTRICO

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERA CIVIL EN COMPUTACIÓN

KELLY ANDREA WAGEMANN SIMMONDS

PROFESOR GUÍA:
JAVIER BUSTOS JIMÉNEZ

MIEMBROS DE LA COMISIÓN:
MARIA CECILIA RIVARA ZÚÑIGA
LUCIANO RADRIGÁN FIGUEROA

Este trabajo ha sido parcialmente financiado por NIC Labs Chile

SANTIAGO DE CHILE
2024

Resumen

El arbitraje de energía es una práctica habitual en el rubro energético europeo, que consiste en aprovechar las diferencias de precios de energía del mercado. De esta manera, las empresas energéticas utilizan el arbitraje para comprar energía a un precio determinado, almacenarla en baterías y posteriormente venderla a un precio más alto, obteniendo beneficios económicos.

Desde el 2022, el Centro de Supercomputación de Barcelona se encuentra estudiando una manera de modelar los costos de una cadena de suministro energético, identificando los mejores momentos de compra y venta de energía en el tiempo, con el fin de disminuir los costos totales de ese período. Sin embargo, limitaciones en el modelamiento matemático y en la implementación de software que han desarrollado impiden cumplir con este propósito.

En este contexto, el objetivo general del trabajo realizado consiste en mejorar la representación del modelo de costos de la cadena, incorporando elementos estocásticos y permitiendo su resolución óptima y eficiente.

La solución desarrollada se compone de dos partes: el mejoramiento de la implementación del modelo determinista y la introducción de estocasticidad. Para la primera parte, se repararon los errores de la implementación existente y se desarrolló un paquete en Python, capaz de resolver el problema de optimización para todos los casos de estudio, de manera expedita. De esta manera, para problemas con un horizonte temporal de 24 horas, se logró completar la optimización del modelo en menos de medio segundo.

Para la incorporación de estocasticidad, se modeló la variabilidad del precio de la energía de la red eléctrica generando escenarios aleatorios. Esto se logró mediante la predicción de series temporales utilizando tanto métodos estadísticos tradicionales como redes neuronales. Tras una evaluación de calidad de las predicciones resultantes, se determinó que este método permite representar de manera fiable la variabilidad del parámetro de interés.

De esta forma, se logró cumplir con casi todos los objetivos planteados en el trabajo, desarrollando una solución de gran valor para el BSC. El único objetivo específico que no se cumplió en su totalidad, debido a escasez de tiempo, fue el de la modelación de la variabilidad de todos los parámetros estocásticos del modelo.

Para todxs aquellxs que han aprendido a brillar a través de la adversidad.

Agradecimientos

La finalización de este documento marca consigo el término de una larga y extenuante etapa de aprendizaje y crecimiento.

Todos estos años, llenos de triunfos y fracasos, alegrías y penas, ayudaron a formar la persona que soy hoy en día. Y nada de esto hubiera sido posible de no contar con personas excepcionales que acompañaron mi andar.

En primer lugar, quisiera agradecer a cada uno de los grandes maestros que guiaron mi aprendizaje en todos mis años de existencia. En especial a mi profesor guía, Javier. Gracias por su entrega, apoyo y motivación. Gracias por su paciencia y comprensión. Sepan que ustedes me invitaron a soñar, a creer en mis capacidades y me impulsaron a navegar hacia nuevos horizontes.

En segundo lugar, quisiera agradecer a todas las bellas personas que constituyen mi familia, la que yo escogí. Ustedes saben quienes son. Gracias por estar, por contener y por amar, incluso en los momentos difíciles o mejor dicho, especialmente en aquellos. Ustedes son el mejor regalo que puedo tener. Gracias por dejarme entrar en sus vidas y por ser parte de la mía.

Aquí, se me hace necesario agradecer de manera especial, a dos personas.

A mi tía maravillosa, Jocelyn, por absolutamente todo. La vida no me alcanza para agradecerle y a veces no encuentro las palabras para expresarle lo significa para mi, pero yo sé que ella lo sabe.

Y a mi compañero de vida, Nicolás, por ser una persona sencillamente extraordinaria y una pareja aún mejor. Gracias por infinitas cosas. Gracias por existir.

Tabla de Contenido

1. Introducción **1**

1.1. Contexto 1

1.2. Problema y Relevancia 2

1.3. Objetivos 3

1.3.1. Objetivo General 3

1.3.2. Objetivos Específicos 3

1.4. Descripción general de la solución 3

1.5. Estructura del documento 4

2. Marco Teórico **6**

2.1. Tecnologías relevantes 6

2.1.1. MareNostrum 6

2.1.2. Librerías 7

2.1.3. Solucionadores 7

2.1.4. Redes Neuronales 8

2.2. Predicción de series temporales 9

2.2.1. Series temporales 9

2.2.2. Métodos de predicción 10

3. Análisis **12**

3.1. Situación actual 12

3.1.1. Modelamiento de la cadena de suministro energético 12

3.1.2. Implementación 15
3.2. Requisitos 16
3.3. Limitaciones de situación actual y oportunidades de mejora 17
3.3.1. Aspectos funcionales 17
3.3.2. Aspectos no funcionales 17

4. El modelo determinista **19**
4.1. Diseño 19
4.1.1. Modularización 19
4.1.2. Estándares de código 21
4.2. Implementación 21
4.2.1. Detección y reparación de errores existentes 22
4.2.2. Implementación en Python 23
4.2.3. Experimentación con solucionadores 25
4.3. Validación 27
4.3.1. Diseño 28
4.3.2. Resultados y Análisis 29

5. El modelo con estocasticidad **31**
5.1. Diseño 31
5.1.1. Introducción de estocasticidad 32
5.1.2. Predicción de series temporales 33
5.2. Implementación 34
5.2.1. Modelos estadísticos tradicionales 34
5.2.2. Modelos de redes neuronales 38
5.3. Validación 39
5.3.1. Diseño 40
5.3.2. Resultados 40

5.3.3. Análisis 44

6. Validación **46**

6.1. Análisis de sensibilidad del modelo 46

6.1.1. Diseño 46

6.1.2. Resultados y Análisis 47

6.2. Extensibilidad 49

6.2.1. Añadir un nuevo componente 49

6.2.2. Añadir una nueva restricción 49

6.2.3. Existencia de tests 50

6.2.4. Análisis 50

6.3. Mantenibilidad 50

6.3.1. Nivel de cobertura 50

6.3.2. Flujo de Integración Continua 51

6.3.3. Formateo de código 51

6.3.4. Documentación y ejemplos 51

6.3.5. Análisis 51

7. Conclusiones **52**

7.1. Trabajo realizado 52

7.1.1. Oportunidades de mejora 53

7.2. Trabajo futuro 54

Bibliografía **60**

Índice de Tablas

4.1. Tiempo de resolución de cada solucionador bajo diferentes escenarios. 27

5.1. Modelos de redes neuronales utilizados para realizar las predicciones. 39

5.2. Calidad de la predicción para diferentes modelos de redes neuronales. 44

Índice de Ilustraciones

1.1. Cadena de suministro energético del proyecto *CUCO*. 2

3.1. Diagrama de dependencias de la implementación inicial. 15

4.1. Diagrama de dependencias de la solución propuesta. 20

4.2. Estructura de directorios de la solución propuesta. 21

4.3. Esquema de modificaciones realizadas a la implementación existente. 22

4.4. Estructura final del paquete `cuco_opt`. 25

4.5. Tiempo de ejecución de `Gurobi` en función del tamaño de entrada. 29

5.1. Precio de la energía de la Red Eléctrica Española, por hora. 32

5.2. Descomposición STL para diciembre 2022. 35

5.3. Descomposición STL para julio 2023. 36

5.4. Función de autocorrelación de la serie, con un nivel de significancia del 5 %. . 37

5.5. Función de autocorrelación parcial de la serie, con un nivel de significancia del 5 %. 38

5.6. Predicción realizada por el modelo SARIMAX. 41

5.7. Predicciones realizadas por los modelos FNN y LSTM. 42

5.8. Predicciones realizadas por los modelos GRU y TCN. 43

6.1. Variación del costo total optimizado de la cadena de suministro en función de variaciones en el precio de la energía. 48

Capítulo 1

Introducción

1.1. Contexto

El arbitraje de energía [71, 79] es una práctica habitual en el rubro energético europeo, que consiste en aprovechar las diferencias de precios de energía del mercado. De esta manera, las empresas energéticas utilizan el arbitraje para comprar energía a un precio determinado, almacenarla en baterías y posteriormente venderla a un precio más alto, valiéndose de las variaciones temporales y/o geográficas de los valores de la energía para obtener beneficios económicos.

Una de las formas de realizar esta práctica es considerando el *Day-Ahead Market* [47] (DAM), mercado energético en el cual los participantes compran y venden con un día de anticipación respecto al momento de la entrega real de energía. Para encontrar el punto óptimo de compra y venta de energía, en base a un conjunto de parámetros y restricciones, se puede modelar el problema como uno de optimización matemática. Esto permitiría, en últimos términos, generar un mayor rendimiento económico para las empresas energéticas.

En la actualidad, existen diversos modelos matemáticos de arbitraje energético en el DAM que se han desarrollado con este fin [17, 26, 42, 79]. Sin embargo, nuevas alternativas de modelación continúan siendo de interés, considerando la dificultad de modelar adecuadamente la variabilidad intrínseca de algunas variables involucradas en el problema, como el precio de la energía, la demanda energética y la generación de energías renovables [34], lo que podría significar una toma de decisiones subóptima para las empresas energéticas.

Una de las instituciones que se encuentra estudiando una mejor manera de modelar el problema es el Barcelona Supercomputing Center [7] (BSC), centro nacional de supercomputación de España. A comienzos del año 2022, el BSC dió inicio a *CUCO*, proyecto de optimización orientado a empresas energéticas europeas.

El objetivo principal que se persigue con este proyecto es el de identificar las operaciones a realizar en una cadena de suministro energético, en un día, con el fin de optimizar los costos totales de ese período. De esta manera, se pretende modelar el arbitraje energético en el DAM para la cadena de suministro de energía que se muestra en la figura 1.1, determinando cómo

satisfacer la demanda energética de los consumidores al mínimo coste.

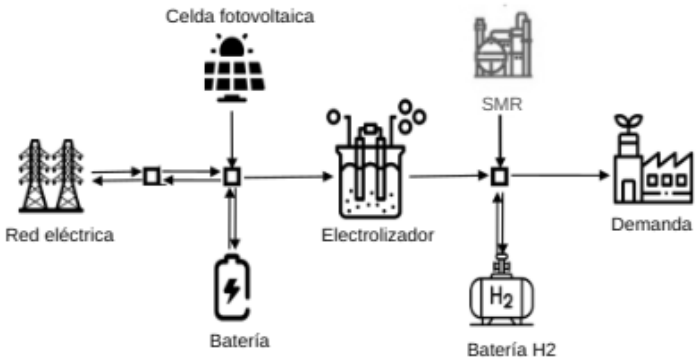


Figura 1.1: Cadena de suministro energético del proyecto *CUCO*.

Como puede apreciarse, la cadena de suministro energético anterior compone un sistema híbrido, considerando fuentes de energía renovables y no renovables. Dentro de las energías no renovables, se encuentra la energía fósil, proveniente de la red eléctrica y la energía nuclear, originada en Reactores Modulares Pequeños (SMR). En cuanto a las energías renovables, se incluye la energía solar, generada en celdas fotovoltaicas y el hidrógeno verde, producido por electrolizadores.

Para determinar el costo mínimo asociado a la cadena anterior, se modeló como un problema determinista de optimización matemática, con su conjunto de parámetros, variables y restricciones.

1.2. Problema y Relevancia

El modelamiento de costos de la cadena de suministro, realizado por el BSC, define el problema de optimización de manera simplificada y sin considerar la variabilidad inherente de algunos de sus parámetros, como el precio de compra de la energía de la red eléctrica, la cantidad de energía captada por las celdas fotovoltaicas o la demanda energética para un momento dado. Esto limita la aplicación del modelo a un análisis retrospectivo de los mejores momentos de compra y venta de energía y no permite la optimización de costos en el futuro.

Además, considerando el volumen de datos, la complejidad de las ecuaciones con las que se trabaja y la necesidad de una respuesta expedita, la resolución del problema de optimización se hace imposible sin soporte computacional. En consecuencia, se vuelve esencial el desarrollo de un software que asista con la optimización del modelo dado. En este contexto, el BSC dispone del superordenador MareNostrum, cuyas notables capacidades de cálculo se pretenden poner a disposición para apoyar este proyecto.

En la actualidad, se cuenta con una implementación preliminar del modelo en `Jupyter Notebook`, que, aunque pretende facilitar la resolución del problema de optimización, se encuentra limitada por errores lógicos significativos. Estos errores impiden resolver el modelo para todos los casos de prueba ejecutados. La falta de un especialista en computación o de

una persona dedicada por completo al desarrollo impide la identificación y corrección de estos problemas.

Por tanto, lo que se tiene hasta ahora es únicamente un esbozo matemático básico, sin avances concretos en la resolución del modelo propuesto ni en la obtención de conclusiones relevantes.

A partir de lo señalado, surgen dos grandes necesidades: mejorar la implementación existente, arreglando los errores que la aquejan y siguiendo buenas prácticas de programación y añadir estocasticidad al modelo para que permita determinar los mejores momentos de compra y venta de energía en el DAM.

Con estos cambios en conjunto, se espera tener un modelo capaz de pronosticar las condiciones del mercado y en base a ellas y a la dinámica de la cadena de suministro, proporcionar directrices para operaciones que minimicen los costos totales y mejoren la rentabilidad de las empresas del sector.

1.3. Objetivos

1.3.1. Objetivo General

El objetivo general de este trabajo consiste en mejorar la representación del modelo de costos de la cadena de suministro energético, incorporando elementos estocásticos y permitiendo su resolución óptima y eficiente.

1.3.2. Objetivos Específicos

A partir del objetivo general, se identifican los siguientes objetivos específicos:

1. Resolver de manera óptima el modelo determinista de costos para el 100 % de los casos de estudio.
2. Reflejar de manera fidedigna el comportamiento variable de los parámetros del problema.
3. Lograr que la resolución del problema de optimización, para un horizonte temporal de 24 horas, se complete en un tiempo inferior a un minuto.

1.4. Descripción general de la solución

La solución desarrollada comprende un trabajo extenso, que se divide en dos etapas principales: el mejoramiento de la implementación del modelo determinista y la introducción de estocasticidad.

Para la implementación del modelo determinista, se inició con la revisión y corrección de errores en la implementación existente, que impedían completamente la optimización del modelo. Posteriormente y para extender el uso de la solución, se migró el código a Python, con un diseño modular y estructura de paquete.

En cuanto a la incorporación de estocasticidad, se identificaron las principales fuentes de variabilidad en el modelo y se investigaron métodos apropiados para abordarlas en este contexto.

La decisión final fue modelar la variabilidad de los precios de la energía de la Red Eléctrica Española mediante la generación de escenarios aleatorios. Estos escenarios se generaron a partir de la predicción de series temporales, utilizando tanto modelos estadísticos tradicionales como modelos de redes neuronales.

Para corroborar la calidad de la solución desarrollada se procedió a validar la implementación modelo determinista y los métodos de incorporación de estocasticidad.

Para la implementación determinista, se evaluó el desempeño del modelo con diferentes solucionadores y bajo una variedad de entradas, con el objetivo de corroborar que se encontrara una solución óptima y eficiente para todos los casos de interés.

De esta manera, se encontró un solucionador adecuado para el problema y que demostró ser capaz de encontrar soluciones factibles para todos los casos de estudio. Notablemente, para los casos con un horizonte temporal de 24 horas, la optimización se completó en menos de medio segundo, y se observó que el tiempo de resolución escala linealmente con el horizonte temporal.

Con respecto a la incorporación de estocasticidad, se evaluaron los escenarios generados por medio de predicciones de series temporales considerando sus errores cuadráticos medios y sus tasas de acierto de cambio direccional. Así, se observó que los modelos de redes neuronales lograban predecir los precios de la energía con un error cuadrático medio menor a 136 y una tasa de acierto mayor a 72 %, considerándose una manera efectiva de reflejar su comportamiento variable.

Todo el desarrollo se llevó a cabo siguiendo buenas prácticas de programación, asegurando así la calidad, extensibilidad y mantenibilidad del software.

1.5. Estructura del documento

El presente documento se estructura en base a capítulos, cuya descripción se detalla a continuación.

Se comienza con el marco teórico, en el capítulo 2. En él, se exponen de manera breve conceptos fundamentales para entender a cabalidad el trabajo realizado.

Posteriormente, en el capítulo 3 se realiza un análisis de la situación inicial del proyecto, abarcando un estudio minucioso de aspectos funcionales, no funcionales y oportunidades de

mejora.

En el capítulo 4 se detalla la primera parte de la solución desarrollada, contemplando el diseño, implementación y validación del modelo determinista.

En el capítulo 5 se detalla la segunda parte de la solución desarrollada, contemplando el diseño, implementación y validación del modelo con estocasticidad.

La validación general de la solución se describe en el capítulo 6, estudiando la sensibilidad del modelo y la calidad del código desarrollado.

Por último, en el capítulo 7 se presentan las conclusiones generales del trabajo, considerando aspectos técnicos y personales y planteando posibles líneas de trabajo futuro.

Capítulo 2

Marco Teórico

En el siguiente capítulo, se expondrán de manera breve conceptos fundamentales del trabajo realizado, cuyo conocimiento básico es necesario para una adecuada comprensión del mismo.

Estos conceptos se clasifican en dos categorías principales: tecnologías relevantes y predicción de series temporales.

2.1. Tecnologías relevantes

2.1.1. MareNostrum

MareNostrum es el nombre del superordenador del Centro de Supercomputación de Barcelona [6]. Es el más potente de España y uno de los siete superordenadores de la infraestructura europea Partnership for Advanced Computing in Europe (PRACE).

Hasta el momento, se han instalado cinco versiones, siendo la más reciente en operación el MareNostrum 5. El supercomputador está disponible para la comunidad científica nacional e internacional y sirve como apoyo para investigaciones en todo tipo de disciplinas.

MareNostrum ejecuta SUSE Linux 11 SP3. Su capacidad de cálculo está repartida en dos partes totalmente diferenciadas: un bloque de propósito general y un bloque de tecnologías emergentes. El bloque de propósito general tiene 48 racks con 3.456 nodos. Cada nodo tiene dos chips Intel Xeon Platinum, con 24 procesadores cada uno, lo que suma un total de 165.888 procesadores y una memoria central de 390 Terabytes. Su potencia máxima es de 11,15 Petaflops.

2.1.2. Librerías

Pyomo

Pyomo es un paquete de software de código abierto de Python, que soporta un conjunto diverso de capacidades de optimización para formular, resolver y analizar modelos de optimización [58].

Puede ser utilizado para definir problemas simbólicos generales, crear instancias específicas de problemas y resolver estas instancias usando solucionadores comerciales y de código abierto.

Una característica relevante es que soporta una amplia gama de tipos de problemas, incluyendo problemas lineales, no lineales, lineales de enteros mixtos, no lineales de enteros mixtos, cuadráticos, estocásticos y programas matemáticos con restricciones de equilibrio.

Statsmodels

Statsmodels es un paquete de software de código abierto de Python que se utiliza para realizar análisis estadísticos y econométricos. Está específicamente diseñado para facilitar diversas tareas de modelado estadístico y análisis de datos [53], siendo ampliamente utilizado en áreas como la economía y las finanzas [64].

El paquete es conocido por su aplicación en el pronóstico de series temporales [21], ofreciendo una amplia gama de modelos de predicción y pruebas estadísticas para realizar análisis rigurosos.

Tensorflow

TensorFlow es un paquete de software de código abierto diseñado para desarrollar y entrenar modelos de aprendizaje automático, con un énfasis particular en las redes neuronales profundas [69]. Cuenta con implementaciones para Python y C++.

Debido a su capacidad para operar a gran escala y en entornos heterogéneos, se ha convertido en una opción popular entre la gran cantidad de bibliotecas de aprendizaje profundo [51].

Para facilitar la prototipación rápida y la experimentación con redes neuronales, integra la API Keras [70].

2.1.3. Solucionadores

Un solucionador es un software matemático, que puede presentarse como un programa o una librería y que resuelve un problema matemático. Un solucionador toma descripciones de

problemas en alguna forma genérica y calcula su solución [43].

Existe una amplia gama de solucionadores especialmente diseñados para problemas de optimización y que son compatibles con Pyomo, como por ejemplo, Gurobi [44], Ipopt [60], Couenne [14] y Bonmin [13].

2.1.4. Redes Neuronales

Las Redes Neuronales (ANNs) son modelos computacionales inspirados en la estructura y funcionamiento del cerebro humano, diseñados para procesar entradas de datos complejos y generar salidas. Constituyen un tipo de modelos de aprendizaje de máquinas, donde nodos interconectados, o neuronas, se organizan en capas. Estas redes son capaces de aprender y adaptarse a través de datos de entrenamiento, lo que les permite reconocer patrones, hacer predicciones y realizar diversas tareas [57].

Las redes neuronales suelen ser entrenadas mediante la minimización del riesgo empírico. Este método se basa en la idea de optimizar los parámetros de la red para minimizar la diferencia, o riesgo empírico, entre la salida predicha y los valores objetivo reales en un conjunto de datos dado [24].

Para el entrenamiento de una red neuronal, es fundamental definir ciertos hiperparámetros que influyen directamente en la eficacia y eficiencia del proceso de aprendizaje. Entre estos hiperparámetros se encuentran los *epochs* y el tamaño de *batch*. Cada *epoch* representa una iteración completa en la que todos los datos de entrenamiento se pasan a través de la red neuronal, mientras que el tamaño de *batch* determina la cantidad de muestras de datos que se procesan en la red antes de actualizar los parámetros del modelo.

Un mayor número de *epochs* puede permitir que la red aprenda patrones más complejos en los datos, pero también corre el riesgo de sobreajustar el modelo si el número es excesivamente alto.

El concepto de redes neuronales ha sido ampliamente estudiado y aplicado en diversos dominios. Se han desarrollado diferentes tipos de modelos de redes neuronales, como Redes neuronales prealimentadas (FNNs), Redes recurrentes (RNNs), Redes neuronales convolucionales (CNNs) y Redes neuronales de grafos (GNNs) [28, 46, 75].

La versatilidad y adaptabilidad de las redes neuronales las convierten en una herramienta poderosa para resolver una amplia gama de problemas en diferentes disciplinas, como el reconocimiento de imágenes, modelación de procesos y pronosticación de series temporales [1, 11, 80].

Redes Neuronales Prealimentadas

Las redes neuronales prealimentadas (FNNs) son un tipo fundamental de red neuronal en la que las conexiones entre los nodos no forman ciclos. Estas redes consisten de una capa de entrada, una o más capas ocultas y una capa de salida, con cada capa completamente

conectada a la siguiente. La información en una FNN se mueve en una sola dirección, desde los nodos de entrada hacia los nodos de salida [15].

La arquitectura de las FNNs les permite aproximar regiones de decisión arbitrarias con alta precisión [15]. Estas redes han sido ampliamente utilizadas en diversas aplicaciones, incluyendo la solución de problemas matemáticos complejos [25].

Redes Neuronales Recurrentes

Las Redes Neuronales Recurrentes (RNNs) son un tipo de red neuronal diseñada para procesar de manera efectiva datos secuenciales o temporales. A diferencia de las FNNs, las RNNs tienen conexiones que forman ciclos, lo que les permite exhibir un comportamiento temporal dinámico y mantener una forma de memoria.

Las RNNs han mostrado éxito en varias aplicaciones que involucran datos secuenciales o temporales, como la planificación y operación de la generación de electricidad [38] y la generación de un pronóstico del impacto de la pandemia de COVID-19, demostrando su versatilidad en el manejo de datos de series temporales [22].

Dentro de los tipos de RNN, se destacan las Redes de Memoria a Corto y Largo Plazo (LSTMs) por su capacidad para manejar dependencias a largo plazo [40] y las Redes de Unidad Recurrente con Compuertas (GRUs), similares a las LSTM pero con menos parámetros, lo que las convierte en opciones computacionalmente menos costosas y más rápidas de entrenar [12].

Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales (CNNs) son un tipo de red neuronal utilizada principalmente en el procesamiento de imágenes [66], aunque también se pueden encontrar reportes de su uso en otros tipos de datos como secuencias temporales, audio y texto [55, 76].

Las CNNs se caracterizan por su arquitectura, que incluye capas convolucionales, capas de agrupamiento y capas completamente conectadas.

Dentro de las CNNs, se encuentran las Redes Convolucionales Temporales (TCNs), diseñadas específicamente para el análisis y la predicción de series temporales [76].

2.2. Predicción de series temporales

2.2.1. Series temporales

Las series temporales son secuencias de puntos de datos medidos en intervalos de tiempo sucesivos. El análisis de series temporales implica estudiar los patrones, tendencias y comportamientos dentro de los datos para hacer pronósticos o derivar percepciones significativas [9].

Este análisis es crucial en varios campos donde se utilizan para la realización de pronósticos [16, 37] y el control y la comprensión de patrones cíclicos [45].

Una de las áreas en que las predicciones de series temporales han contribuido fuertemente es en la investigación de mercados financieros. Aún considerando la complejidad y volatilidad de estos mercados, se han encontrado estrategias efectivas para predecir fluctuaciones económicas en el tiempo [19, 37, 39].

2.2.2. Métodos de predicción

Como se adelantó anteriormente, existen diferentes maneras de realizar predicciones sobre series temporales. Principalmente, se pueden categorizar en métodos que utilizan modelos de redes neuronales y técnicas que emplean modelos estadísticos tradicionales.

Modelos estadísticos tradicionales

La predicción de series temporales utilizando modelos tradicionales, como el Modelo Autorregresivo Integrado de Medias Móviles (ARIMA), ha sido un enfoque prominente en varios dominios [78].

Los modelos Autorregresivos Integrados de Medias Móviles (ARIMA) son una clase de modelos estadísticos utilizados para analizar y predecir datos de series temporales.

Estos modelos se caracterizan por tres componentes principales: autorregresión (AR), diferenciación (I) y media móvil (MA). El componente autorregresivo se refiere al uso de valores pasados de la serie temporal para predecir valores futuros, capturando la relación lineal entre una observación y un número de observaciones retrasadas. El componente de diferenciación implica transformar una serie temporal no estacionaria en una estacionaria, lo que es esencial para estabilizar la media de la serie. El componente de media móvil incorpora la dependencia entre una observación y un error residual de un modelo de media móvil aplicado a observaciones retrasadas [29].

Se han desarrollado extensiones del modelo ARIMA, diseñadas para considerar componentes adicionales, como la estacionalidad de una serie (S) y la existencia de variables exógenas (X).

Métricas de calidad

Las métricas de calidad para las predicciones de series temporales son esenciales para evaluar la precisión y fiabilidad de los modelos predictivos. Dos métricas comúnmente utilizadas para esto son el Error Cuadrático Medio (MSE) y la Tasa de Acierto (Hit Rate) [29].

El MSE es una medida estadística que calcula el promedio de los cuadrados de las diferencias entre los valores reales y los predichos. Proporciona una evaluación integral del

rendimiento del modelo de pronóstico al cuantificar la magnitud de los errores de predicción. Un MSE más bajo indica una mayor precisión y exactitud en la previsión.

Por su parte, el Hit Rate es una métrica categórica que mide la proporción de previsiones o predicciones correctas del total de previsiones. En algunos casos, como en el análisis de series temporales financieras y en la predicción económica, resulta de interés el cálculo del Hit Rate de cambio direccional. Esta métrica se enfoca en medir la precisión con la que un modelo predictivo puede identificar correctamente la dirección del cambio en el precio de una acción, la tasa de interés, o indicadores económicos.

Capítulo 3

Análisis

En el siguiente capítulo, se presentará un análisis minucioso de la situación inicial del proyecto, abarcando una evaluación profunda tanto de aspectos funcionales como no funcionales.

Adicionalmente, se expondrá la identificación de oportunidades de mejora que ayudarán a dar forma, posteriormente, al diseño de la solución.

3.1. Situación actual

Al momento de comenzar a trabajar en el proyecto, se contaba con el modelamiento matemático básico de los costos asociados a la cadena de suministro y su implementación en un archivo de `Jupyter Notebook`[1] [32].

Este modelamiento, determinista, no era capaz de realizar predicciones de sus parámetros. Por este motivo, a partir de una serie de valores dados, entregaba una retrospectiva de las operaciones que se deberían haber realizado para optimizar el costo total de la cadena de un día en el pasado.

3.1.1. Modelamiento de la cadena de suministro energético

La expresión que modela el mínimo costo total de la cadena de suministro está dada por la ecuación 3.1.

$$J_{obj}(T) = min \sum_{t=1}^{T} J_{e,g}(t) + J_{GHG,g}(t) + J_{Gas,SMR}(t) + J_{GHG,SMR}(t) \tag{3.1}$$

[1]Herramienta interactiva para crear, editar y compartir documentos que contienen código ejecutable en tiempo real.

Donde $J_{e,g}$ corresponde al costo energético de la red eléctrica, $J_{GHG,g}$ es el costo asociado al gas invernadero producido por la red eléctrica, $J_{Gas,SMR}$ es el costo del gas asociado a las operaciones del SMR y $J_{GHG,SMR}$ es el costo asociado al gas invernadero producido por el SMR.

Modelamiento de costos parciales

Las ecuaciones que modelan cada uno de los costos parciales de la cadena de suministro, es decir, $J_{e,g}$, $J_{GHG,g}$, $J_{Gas,SMR}$ y $J_{GHG,SMR}$ corresponden a 3.2, 3.3, 3.4 y 3.5 respectivamente.

$$J_{e,g}(t) = u_g(t)\eta_{e,g}(t)W_g^{max}(t) \tag{3.2}$$

$$J_{GHG,g}(t) = u_g(t)\eta_{GHG,g}(t)W_g^{max}(t) \tag{3.3}$$

$$J_{Gas,SMR}(t) = u_{H2,SMR,on}(t)u_{H2,SMR}(t)\eta_{Gas,SMR}(t)W_{SMR}^{max}(t) \tag{3.4}$$

$$J_{GHG,SMR}(t) = u_{H2,SMR,on}(t)u_{H2,SMR}(t)\eta_{GHG,SMR}(t)W_{SMR}^{max}(t) \tag{3.5}$$

Donde $u_g(t)$ es la variable de control de la red eléctrica y $u_{H2,SMR,on}(t)$ y $u_{H2,SMR}(t)$ son variables de control del SMR.

Por otra parte, $\eta_{e,g}(t)$ es el precio de compra de energía de la red eléctrica, $\eta_{GHG,g}(t)$ es el precio del gas invernadero de la red eléctrica, $\eta_{Gas,SMR}(t)$ es el precio del gas del SMR y $\eta_{GHG,SMR}(t)$ es el precio del gas invernadero del SMR.

Por último, $W_g^{max}(t)$ corresponde a la potencia máxima de la red y $W_{SMR}^{max}(t)$ a la producción máxima del SMR.

Restricciones

En cuanto a las restricciones que rigen a la cadena de suministros, se pueden encontrar restricciones de balance energético y restricciones de almacenamiento de energía.

Con respecto a las restricciones de balance energético, se cuenta con las ecuaciones 3.6 de balance de hidrógeno verde y 3.7 de gestión de energía integrada para electrolizadores.

$$u_{H2,b}(t)W_{H2,b}^{max}(t)+u_{H2,SMR,on}(t)u_{H2,SMR}(t)W_{SMR}^{max}(t)+\eta_{H2}^{e}(t)u_{H2,e}(t)W_{H2,e}^{max}(t) = D_{H2}(t) \tag{3.6}$$

Donde $u_{H2,b}(t)$ es la variable de control de la batería de hidrógeno, $u_{H2,SMR,on}(t)$ y $u_{H2,SMR}(t)$ son variables de control del SMR, $u_{H2,e}(t)$ es la variable de control del electrolizador y $\eta_{H2}^{e}(t)$ es el rendimiento de hidrógeno.

Por otra parte, $W_{H2,b}(t)$ es la potencia máxima de carga de la batería de hidrógeno, $W_{H2,SMR}(t)$ es la producción máxima del SMR, $W_{H2,e}^{max}(t)$ es la potencia máxima del electrolizador y $D_{H2}(t)$ es la demanda de hidrógeno.

$$u_g(t)W_g^{max}(t) + u_{e,b}(t)W_{e,b}^{max}(t) + u_{pv}(t)W_{pv}^{max}(t) = u_{H2,e}(t)W_{H2}^{max}(t) \tag{3.7}$$

Donde $u_g(t)$ es la variable de control de la red eléctrica, $u_{e,b}(t)$ es la variable de control de la batería eléctrica, $u_{pv}(t)$ es la variable de control de la celda fotovoltaica y $u_{H2,e}(t)$ es la variable de control del electrolizador.

Por otra parte, $W_g^{max}(t)$ es la potencia máxima de la red eléctrica, $W_{e,b}^{max}(t)$ es la potencia máxima de carga de la batería eléctrica, $W_{pv}^{max}(t)$ es la producción máxima de energía fotovoltaica y $W_{H2}^{max}(t)$ es la potencia máxima del electrolizador.

Para las restricciones de almacenamiento de energía, se consideraron restricciones de almacenamiento de la batería de hidrógeno como se observa en la ecuación 3.8 y de almacenamiento en la batería eléctrica como se muestra en 3.9.

$$u_{H2,b}(t)W_{H2,b}^{max}(t) <= SOC_{H2,b}(t)C_{H2,b} \tag{3.8}$$

Donde $u_{H2,b}(t)$ es la variable de control de la batería de hidrógeno, $W_{H2,b}^{max}(t)$ es la potencia máxima de carga de la batería de hidrógeno, $SOC_{H2,b}(t)$ es el estado de carga de la batería de hidrógeno y $C_{H2,b}$ es la capacidad de carga de la batería de hidrógeno.

$$u_{e,b}(t)W_{e,b}^{max}(t) <= SOC_{e,b}(t)C_{e,b} \tag{3.9}$$

Donde $u_{e,b}(t)$ es la variable de control de la batería eléctrica, $W_{e,b}^{max}(t)$ es la potencia máxima de carga de la batería eléctrica, $SOC_{e,b}(t)$ es el estado de carga de la batería eléctrica y $C_{e,b}$ es la capacidad de carga de la batería eléctrica.

Observaciones

Al examinar el modelamiento anterior, se estudió la complejidad del problema matemático, considerando la cantidad de parámetros, variables y linealidad que poseía.

En este sentido, se observó que el modelo contaba con 18 parámetros y 7 variables. Tomando en cuenta un caso típico de estudio, con un horizonte temporal de 24 horas, se tendrían 24 valores para cada parámetro y variable del modelo.

Estudiando las ecuaciones 3.4 y 3.5 se pudo concluir que el problema a optimizar correspondía a uno de tipo no lineal.

3.1.2. Implementación

Inicialmente, la optimización de costos de la cadena de suministro energético se realizaba mediante un *script* en `Jupyter Notebook`. Esta implementación consistía de un único archivo, basado en el material tutorial de `Pyomo` de la Universidad de Notra Dame de ingeniería química y biomolecular. La implementación era muy similar al ejemplo [72], utilizando la librería `Pyomo` y el solucionador `Ipopt`, pero adaptada a las ecuaciones correspondientes a este caso.

De esta forma, se instanciaba un modelo de `Pyomo`, se le asignaban parámetros, variables, restricciones y función objetivo con los métodos de `Pyomo` y se resolvía utilizando un objeto de tipo `SolverFactory`, con el solucionador `Ipopt`. Finalmente, se entregaba una visualización de los valores asignados a las variables, para todo el horizonte temporal.

En la figura 3.1 se presenta el diagrama de dependencias correspondiente a la implementación inicial.

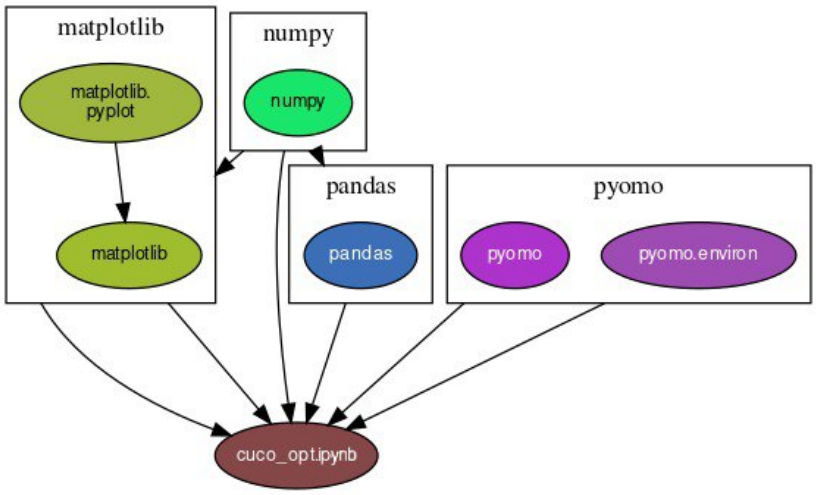


Figura 3.1: Diagrama de dependencias de la implementación inicial.

Es importante mencionar que los valores asignados a los parámetros del modelo, correspondientes a datos de ejemplo suministrados por el BSC[2], se obtenían a partir de un `JSON` *string* directamente incorporado al código.

Observaciones

Examinando la implementación actual y ejecutando el *script*, se detectó un error al momento de intentar resolver el problema de optimización, obteniendo un mensaje del solucionador donde se indicaba que no era posible encontrar una solución factible.

Se plantearon tres causas posibles para la existencia de este error. La primera, que podía tratarse de un error en los datos de entrada suministrados, la segunda, que la fuente del

[2]Basados en datos históricos.

error correspondía a una falla en la definición del modelo y la tercera, que se debía a una incompatibilidad del solucionador con el problema dado.

Analizando más de cerca el modelo implementado y comparándolo con el planteamiento matemático del problema, fue posible ver que ambas representaciones eran inconsistentes entre sí. De esta manera, se notó que en el planteamiento matemático existían variables, parámetros y restricciones que no se encontraban implementadas en el código.

Adicionalmente, algunas variables y parámetros que en el código se agregaban al modelo no se encontraban bien definidos, en cuanto al dominio al que pertenecían y el valor de inicialización que tenían.

El código existente se encontraba alojado en un repositorio privado de *Github*, sin ningún tipo de documentación y con una única rama. Originalmente, no existían convenciones para el código del proyecto.

3.2. Requisitos

Considerando el objetivo del proyecto y su estado inicial de desarrollo, se recibieron una serie de requisitos funcionales y no funcionales que debía cumplir la solución realizada.

Resultó importante considerar, en todo momento, cuáles serían los usuarios finales de la solución desarrollada. En este caso, personal perteneciente a una empresa energética europea, con conocimientos básicos de computación.

De esta manera, dentro de los requisitos funcionales se incluyeron los siguientes.

- Que el software permitiera encontrar una solución óptima al problema original para todos los casos de estudio.
- Que el software permitiera resolver el problema de optimización determinista para distintas configuraciones de la cadena de suministro, quitando y agregando componentes.
- Que la solución incluyera alguna forma de estocasticidad en el modelo, representando de manera fidedigna la variabilidad de sus parámetros.

Con respecto a los requisitos no funcionales, se identificaron los siguientes.

- Que el software se desarrollara en `Python` y utilizara la librería `Pyomo`.
- Que se utilizara *Github* como herramienta de versionamiento.
- Que el software fuera fácil de instalar y de utilizar.
- Que el software desarrollado fuera modular, ordenado y extensible.
- Que la solución permitiera resolver el problema original en menos de un minuto.
- Que el código desarrollado estuviera estandarizado.
- Que el software pudiera ejecutarse en MareNostrum, entorno de producción.

3.3. Limitaciones de situación actual y oportunidades de mejora

Al analizar la situación actual, fue posible identificar una serie de limitaciones que impedían satisfacer los objetivos del proyecto.

3.3.1. Aspectos funcionales

Desde una perspectiva funcional, el aspecto más importante era que la implementación actual, con los datos suministrados, el modelo construido y el solucionador utilizado, no permitía encontrar una solución óptima al problema original. Adicionalmente, la implementación no consideraba un modelo fiel al planteamiento matemático que se tenía de la cadena de suministro energético.

En segundo lugar, el planteamiento matemático que se tenía consideraba un modelo determinista, sin tomar en cuenta la incertidumbre inherente a algunos de sus parámetros. Este era un factor que disminuía la precisión del modelamiento de costos con respecto al comportamiento real que poseía.

Por último y asociado al punto anterior, otra de las grandes limitantes de la situación actual era que, debido a que no se realizaba una predicción de los parámetros con incertidumbre, la optimización que se hacía al modelo siempre se realizaba considerando situaciones pasadas, señalando cuáles habrían sido las medidas a adoptar para disminuir los costo de la cadena de suministro. De esta manera, no era posible disminuir costos de manera efectiva en el presente y futuro.

Todas estas problemáticas abrieron espacios de mejora sustanciales al momento de diseñar y llevar a cabo una nueva solución. Por un lado, se podría mejorar la representación del modelo en `Pyomo` para que fuera fiel a las ecuaciones que describían los costos de la cadena de suministro y se podrían reparar los errores existentes que no permitían la resolución adecuada del problema de optimización.

Por otra parte, se podrían explorar métodos para incorporar la estocasticidad en aquellos parámetros del modelo que lo necesiten, mejorando así la representación matemática mediante un enfoque más realista. Desarrollar técnicas para predecir futuros valores de estos parámetros no solo enriquecería el modelo con estocasticidad, sino que también extendería su aplicabilidad. Esto facilitaría la optimización de operaciones a futuro, impactando positivamente en la reducción del costo total de la cadena a lo largo de este periodo.

3.3.2. Aspectos no funcionales

En cuanto a las limitaciones no funcionales de la situación actual, fue posible observar, por varias razones, que la implementación existente presentaba un diseño que no facilitaba el uso requerido.

Todo el código desarrollado se alojaba en un único archivo de `Jupyter Notebook` sin documentación, por lo que se complicaba su lectura y comprensión. Adicionalmente, la implementación no contaba con ningún tipo de test, dificultando la detección de errores.

El código utilizaba variables globales como entrada del modelo, razón por la cual para trabajar con otro conjunto de datos de entrada se hacía necesario modificar el código directamente. Asimismo, no contemplaba la resolución del modelo bajo diferentes escenarios[3] de la cadena de suministro, por lo que si se querían modificar los componentes que formaban parte de ésta, se debía modificar directamente el código.

Como oportunidad de mejora se identificó, primeramente, la migración del código a `Python`. En cuanto a la estructura de la solución, se podría diseñar considerando varios módulos, separando en cada uno diferentes responsabilidades lógicas de la solución. Esta nueva implementación debía lograr abstraer el problema original, de manera de generalizar lo más posible la solución, para que funcionara bajo distintos escenarios y con diferentes conjuntos de datos de entrada.

Para facilitar el uso y la instalación del software, se decidió tomar en cuenta el conocimiento y manejo tecnológico de los usuarios finales, de manera de desarrollar una herramienta para que resultara accesible y cómoda.

Considerando que la implementación existente utilizaba por defecto un único solucionador para realizar la optimización, se podría estudiar el uso de diferentes solucionadores, para ver cuál de ellos se ajustaba mejor a las necesidades del problema y lo resolvía de manera más eficiente.

Por último, una gran oportunidad de mejora se hallaba en agregar documentación y definir estándares para el código, forzando su cumplimiento con la ayuda de diversas estrategias para este fin.

[3]En este caso, se refiere a diferentes conjuntos de componentes que conforman la cadena de suministro.

Capítulo 4

El modelo determinista

En el siguiente capítulo, se detallará la primera parte de la solución desarrollada, contemplando el diseño, implementación y validación del modelo determinista.

4.1. Diseño

Para determinar el diseño de la implementación del modelo determinista, fue necesario recordar cuáles eran son los principales desafíos a abordar en esta etapa.

En esta primera iteración, las tareas fundamentales de diseño eran las siguientes.

- Diseñar una solución modular en `Python`, que fuera de fácil uso e instalación.
- Definir los estándares de código a adoptar.

Además, el software desarrollado debía ser capaz de ejecutarse localmente en un computador personal y en el superordenador MareNostrum como ambiente de producción.

4.1.1. Modularización

Tomando lo anterior en consideración, el primer paso del diseño fue abstraer el problema lo más posible para poder separar responsabilidades en el código.

Así, se pudo ver que todo se centraba en torno al modelo. Es este modelo al que se añaden sus características y luego se resuelve. Por esta razón, se definió la creación del módulo `model.py` para el modelo.

Para no sobrecargar de responsabilidades a este módulo, se procedió a desmenuzar más el problema. Al observar las ecuaciones que definen el modelo, fue posible notar que, a cada componente de la cadena de suministro, se le asocian parámetros y variables que lo definen.

Es por esto que se determinó la creación de un módulo para cada componente para que, al agregar un componente al modelo, se agregaran automáticamente sus variables y parámetros asociados.

Debido a que la cadena de suministro contaba con muchos componentes, se decidió crear una carpeta llamada `components` donde se alojaría cada módulo de componente.

Si bien cada componente se relacionaba de alguna forma con restricciones del problema, las restricciones podían corresponder a relaciones entre parámetros y variables de uno o más componentes. Es por esto que se optó por crear un módulo `constraints.py` de restricciones.

El último aspecto fundamental del modelo era la función objetivo que optimizaba. Esta función, definida como la sumatoria de varias funciones más pequeñas, relacionaba parámetros y variables de uno o más componentes, por lo que se determinó construir el módulo `objectives.py` de funciones objetivo.

Para finalizar, fue importante recordar que una de las funcionalidades de la implementación inicial era la visualización de los resultados obtenidos. Por comodidad, se determinó la construcción del módulo `plot.py` destinado únicamente para este fin.

La relación simplificada de dependencias entre todos estos módulos se observa en la figura 4.1. Así, el módulo `model.py` invocaría valores y funciones de los otros módulos para poder construir el modelo del problema en `Pyomo`. La resolución del problema de optimización, es decir, la resolución del modelo, es responsabilidad del módulo `model.py`.

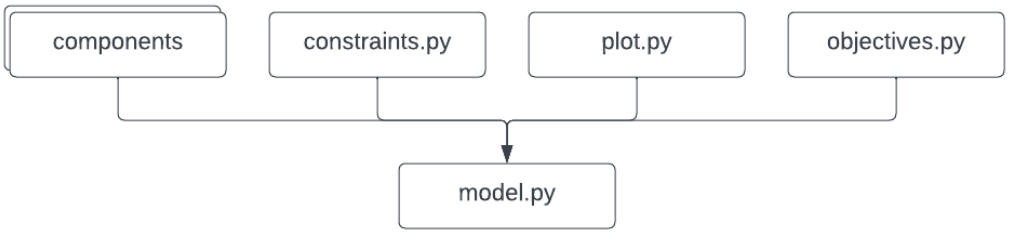


Figura 4.1: Diagrama de dependencias de la solución propuesta.

El diseño propuesto no solamente buscaba ser intuitivo sino que también amigable a la hora de introducir cambios. Existían puntos de extensión claramente identificables para el contexto del proyecto, como por ejemplo, la introducción de un nuevo componente a la cadena de suministro, el cambio o adición de una restricción o la agregación de una nueva variable. La estructura planteada facilitaría identificar los puntos de modificación del código y disminuiría los lugares en los que habría que incorporar cambios.

Otro aspecto relevante en el diseño de la solución era que ésta resultara fácil de usar e instalar. Mediante conversaciones con el BSC, se acordó asumir que los usuarios sabrían ejecutar comandos por consola e instalar paquetes[1] de `Python` usando `pip`[2]. Tomando esto en cuenta, se determinó que la solución desarrollada se diseñaría como un paquete de este

[1]Colección de módulos de Python que pueden ser utilizados por otros programas para realizar una tarea específica.

[2]Sistema de gestión de paquetes de Python, que facilita la instalación de los mismos.

lenguaje.

4.1.2. Estándares de código

Un aspecto fundamental de un buen código es la existencia de tests. De esta manera, el diseño debía contemplar la existencia de una carpeta de tests que probara las funcionalidades del software. Para asegurar un buen grado de cobertura de testo del código, se determinó exigir un mínimo de 80 % de las líneas existentes en todo momento.

De esta forma y a grandes rasgos, la estructura del proyecto sería como la que se señala en la figura 4.2. Todo el código desarrollado se ubicaría dentro de la carpeta `src`. La parte correspondiente al paquete se encontraría dentro de `cuco_opt` y los tests se situarían dentro de la carpeta `tests`.

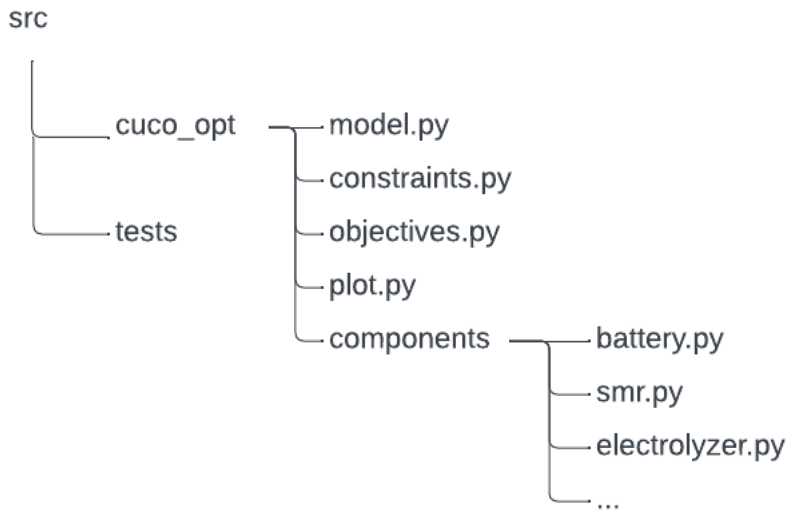


Figura 4.2: Estructura de directorios de la solución propuesta.

Para resguardar que el código que se subiera al repositorio funcionara como se esperaba, se decidió añadir un flujo de integración continua, de manera de que cada cambio subido al repositorio gatillaría la ejecución de tests que corroboraran su calidad.

En cuanto a la escritura de código, se optó por utilizar PEP8 [73] como estándar, asegurando de resguardar su cumplimiento mediante el uso de herramientas de limpieza y formateo de código.

4.2. Implementación

Considerando el estado inicial del proyecto y el listado de requisitos, se determinó que para la implementación del modelo determinista era necesario realizar los siguientes pasos:

- Identificar y arreglar los errores de la implementación existente, que no permiten encontrar una solución al problema de optimización original.
- Aplicar estrategias para resguardar los estándares de código definidos en la sección 4.1.2.
- Migrar el código actual a `Python`, desarrollando software modular según lo definido en 4.1.1 y siguiendo buenas prácticas de programación.
- Explorar diferentes solucionadores para realizar la optimización del problema planteado y elegir el que se adapte mejor a las necesidades que se presentan.

4.2.1. Detección y reparación de errores existentes

Se comenzó realizando cambios sobre la implementación existente, para corroborar que el problema originalmente planteado estuviera bien definido y pudiera ser optimizado.

Contrastando la implementación con el planteamiento matemático del problema, se identificaron y añadieron todas las partes que se encontraban ausentes de la cadena de suministro.

En la figura 4.3 se muestran en verde los componentes que debieron integrarse completamente al código, con sus variables, parámetros y restricciones, en amarillo a los que se les debió agregar variables y en azul aquellos que experimentaron cambios menores.

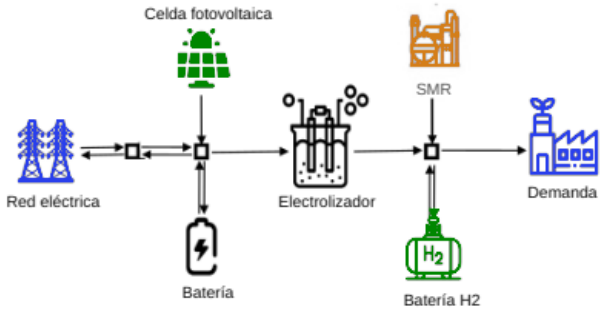


Figura 4.3: Esquema de modificaciones realizadas a la implementación existente.

Dentro de las modificaciones menores que debieron realizarse sobre algunos componentes se encuentra la modificación del dominio de parámetros y variables y el cambio de sus valores de inicialización. Todos estos cambios implicaron modificaciones en el conjunto de restricciones y la función objetivo de la implementación.

Tras realizar estas modificaciones y agregar las nuevas variables a las visualizaciones, se volvió a ejecutar el *script* y se observó que el error que antes existía para encontrar un valor óptimo para el problema se había resuelto. Sin embargo, al examinar la salida del solucionador, se observó que la solución entregada no satisfacía fielmente el problema original, puesto que violaba las restricciones de dominio de las variables discretas. Esta es una limitante conocida para `Ipopt`, el solucionador utilizado hasta ese momento y que requería de la exploración de otros solucionadores pudieran satisfacer las necesidades del problema original.

4.2.2. Implementación en Python

La segunda tarea más importante de esta etapa fue la implementación del modelo determinista como una librería en `Python`. Para ello, se comenzaron aplicando estrategias para resguardar los estándares de código definidos, luego se realizó el desarrollo de la librería solicitada y finalmente, se alojó la solución construida en su entorno de producción.

Estandarización del código

Definir y resguardar estándares de código es una buena práctica al inicio del desarrollo de un proyecto ya que mejora la mantenibilidad y asegura la coherencia y calidad del código desde un principio. Es por esto que la primera tarea realizada al momento de implementar el software en `Python` fue la de introducir diferentes herramientas para resguardar los estándares de código acordados.

En cuanto al formateo de código, se decidió integrar el framework `precommit` [56], por la facilidad de configuración de la herramienta y que es de uso gratuito. Así, se configuraron diferentes *hooks*[3] para forzar la adopción de un buen estilo de escritura de software. Se utilizó `isort` [30] para ordenar las importaciones realizadas en los archivos de `Python`, `black` [8] para reformatear y homogeneizar el código de manera automática y `flake8` [61] identificar y reportar cualquier incumplimiento del estándar PEP8. Para evitar problemas de compatibilidad entre las configuraciones de los *hooks*, se estableció un largo de línea y espaciado común.

Para el testeo del código, se decidió utilizar `pytest` [36] en conjunto con `coverage` [35], librería para obtener reportes de la cobertura de los tests. Con el objetivo de garantizar que el código subido al repositorio cumpliera con el funcionamiento esperado, se añadió un flujo de integración continua utilizando *Github Actions* [23]. Así, cada vez que se introdujera una modificación en el repositorio, automáticamente se activaría la ejecución de la suite completa de tests. Si uno o más de estos tests fallaran, el proceso completo de integración notificaría una falla.

Desarrollo del paquete

Debido a la familiaridad de los futuros usuarios con la instalación y el uso de paquetes de `Python`, el trabajo realizado se estructuró como un paquete en `Python`, llamado `cuco_opt`.

De esta forma, se comenzó a implementar la librería según el diseño modular expuesto en la sección 4.1.1. Se creó una carpeta llamada `components` y dentro de ella, un módulo por cada componente existente de la cadena de suministro. A su vez, en cada uno de estos módulos se crearon funciones que, recibiendo un valor de inicialización y un modelo de `Pyomo`, le añaden variables y parámetros al modelo.

Así, por ejemplo, en el módulo `gen_pv.py` que describe a las celdas fotovoltaicas, se crearon

[3] *Scripts* que se ejecutan automáticamente antes de que suban cambios a un repositorio.

funciones como `var_u_PV(model, U_PV_INIT)` para añadir la variable de control de las celdas y `param_W_PV_max(model, PV_MAX_POTENCY)` para añadir el parámetro de potencia máxima de las celdas.

Luego, se creó el módulo `constraints.py` con todas las restricciones del problema. Cada una de las restricciones se formuló como una función que recibía un modelo y un horizonte temporal y devolvía la ecuación matemática correspondiente. De igual manera, se creó el módulo `objectives.py` con las funciones de costos del problema. Cada función de costo era una función en `Python` que recibía un modelo y devolvía la expresión matemática correspondiente.

Para manejar la visualización de resultados, se creó un módulo llamado `plot.py`.

Posteriormente, se desarrolló el módulo `model.py`, encargado de la creación y resolución de un modelo en `Pyomo`. Teniendo en cuenta que los usos más frecuentes del paquete serían experimentos comparando el desempeño de modelos que responden a un mismo problema, pero con un conjunto diferente de datos de entrada o escenarios de la cadena de suministro, es que se creó la clase `OptModel`.

Un objeto de esta clase consiste de un modelo que, según la entrada que se suministre, crea un modelo de `Pyomo` con los componentes solicitados de la cadena de suministro. La idea principal tras la creación de esta clase era que la generación de experimentos con varios modelos diferentes resultara sencilla.

De esta manera, se ideó la construcción de, al menos, los siguientes elementos en la clase.

- El constructor, que recibe un `JSON` con los valores de los parámetros del modelo y el tipo de escenario a considerar y crea un modelo en Pyomo con todos los parámetros, variables, restricciones y función objetivo.
- El método `solve` que recibe un `JSON` con la configuración del solucionador a utilizar y optimiza la función objetivo del modelo, graficando la asignación de las variables en el horizonte temporal.

De esta manera, para crear un modelo y optimizarlo, solo bastaría con instanciar un objeto `OptModel` y llamar al método `solve`.

La responsabilidad del constructor de la clase es la de crear el modelo de `Pyomo`, agregar los componentes correspondientes, las restricciones y la función objetivo. Es fácil ver que esto podría significar la creación de una función muy extensa y de baja legibilidad. Para evitarlo, se crearon métodos auxiliares que permiten la agregación ordenada y fácil de cada uno de estos elementos al modelo.

De forma adicional se creó el método `solve` que, ayudándose de los métodos de `Pyomo` y las funciones del módulo `plot.py`, optimiza la función objetivo del modelo y entrega una visualización de las asignaciones realizadas a las variables.

La estructura final del paquete desarrollado, `cuco_opt`, se muestra en la figura 4.4. Se puede observar la existencia de otros módulos además de los mencionados anteriormente,

como `constants.py` y `utils.py`. Ambos corresponden a módulos auxiliares que son usados en diferentes partes de la implementación.

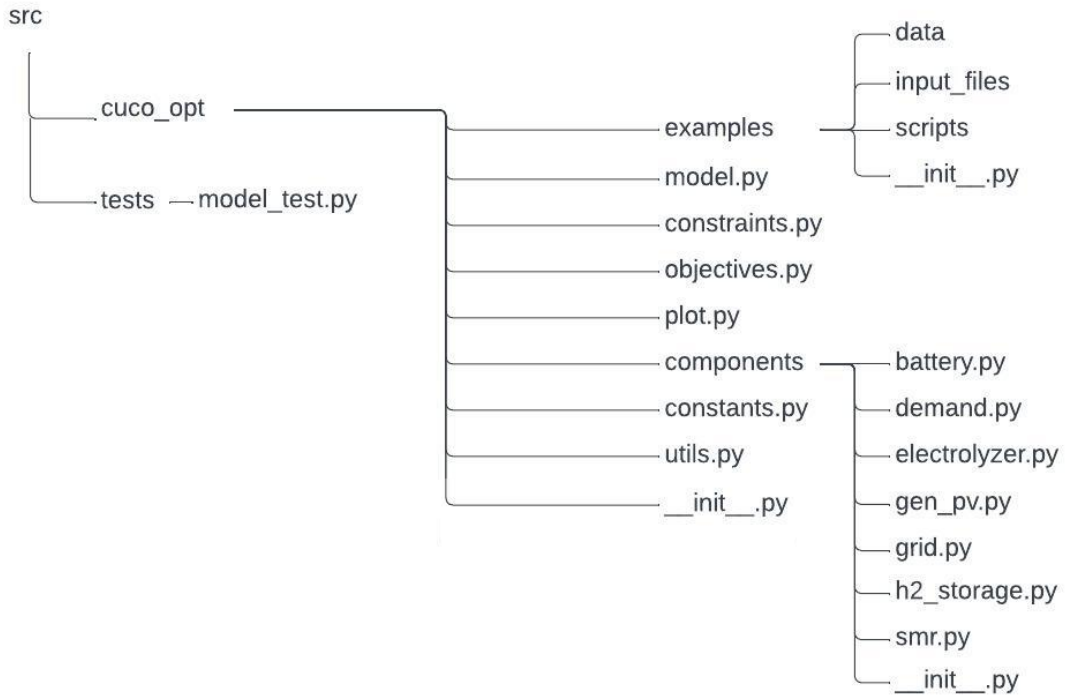


Figura 4.4: Estructura final del paquete `cuco_opt`.

Para facilitar el entendimiento del uso del paquete se creó la carpeta `examples`, con datos de entrada y *scripts* de ejemplo.

Finalmente, se actualizó el `README` del proyecto, generando una guía completa del paquete desarrollado con requerimientos, instrucciones de instalación y ejemplos de uso.

Puesta en producción

Una vez implementado el modelo determinista como paquete de `Python`, otros miembros del equipo se encargaron de encapsular el software en contenedores, utilizando `Singularity` [67]. Tras esto, se cargaron los archivos correspondientes a MareNostrum y se realizaron pruebas de ejecución, confirmando exitosamente que el paquete funcionaba y podía utilizarse en el ambiente de producción.

4.2.3. Experimentación con solucionadores

Una última inquietud a abordar con respecto al modelo determinista consistió en encontrar un solucionador que pudiera resolver el problema de optimización original en todos los

casos solicitados y tardando menos de un minuto de ejecución.

El solucionador que se había probado de manera inicial era `Ipopt`. Sin embargo y como se expuso anteriormente, no era compatible con el problema a resolver debido a que parte de las variables con las cuales se trabaja son discretas. `Pyomo` es compatible con una gran variedad de solucionadores, por lo que se decidió examinar el uso de otras alternativas para resolver el problema.

En este caso, el problema planteado era de tipo no lineal, con variables discretas y continuas. Debido a esto, el BSC solicitó la exploración de la solución del modelo utilizando diversos solucionadores, entre los que se incluyeron opciones comerciales como `Baron` [4] y `Knitro` [3] y software con versiones de uso gratuito como `Bonmin` [13], `Couenne` [14], `Gurobi` [44], `LindoGlobal` [5], `MindtPy` [59] y `Scip` [63].

Para esto, se diseñaron experimentos en los cuales, utilizando los solucionadores de la lista, se intentaría resolvería el problema de optimización y se estudiaría el tiempo empleado en realizarlo.

Debido a que para el BSC era de gran importancia la observación de la solución para modelos en diferentes escenarios, se generaron tres pruebas que se describen a continuación.

Ambiente de prueba

Todos los experimentos se realizaron de manera local en un computador personal. Las especificaciones técnicas del equipo son las siguientes.

- Sistema operativo: Ubuntu 22.04.3 LTS version 64 bit.
- Procesador: Intel® Core™ i7-10510U CPU @ 1.80GHz × 8
- RAM: 16,0 GB.
- Disco: 512 GB

La ejecución de los experimentos se realizó en un entorno virtual creado para este propósito, donde se instaló el paquete `cuco_opt` junto con sus requerimientos.

Datos de prueba

Los valores de los parámetros del modelo fueron datos dados por el BSC en un archivo `JSON`. El archivo contenía 18 campos, cada uno correspondiendo a un parámetro. Cada campo contenía una lista de 24 valores, a excepción del campo de horizonte temporal con un único valor y los campos SOC[4] de la batería y el almacenamiento de hidrógeno, que tenían 3 valores: carga inicial, carga mínima y carga máxima. El horizonte temporal definido fue de 24 horas. Para todos los experimentos, este archivo fue el mismo.

[4]Estado de carga, por sus siglas en inglés.

Los escenarios que se definieron se diseñaron a solicitud del BSC y fueron tres:

- Escenario eléctrico: lo conforman los componentes de red eléctrica, batería, celdas fotovoltaicas y demanda.
- Escenario de hidrógeno verde: lo conforman los componentes de SMR, electrolizador, almacenamiento de hidrógeno y demanda.
- Escenario completo: lo conforman todos los componentes de la cadena de suministro.

Experimentos

Se diseñaron tres experimentos, uno por cada escenario definido anteriormente. Así, con cada solucionador de la lista de candidatos, se intentó resolver el mismo problema de optimización, observando el tiempo estimado de resolución.

Resultados y Análisis

Tras ejecutar los experimentos, se obtuvieron los resultados que se muestran en la tabla 4.1. Puede observarse que en algunos casos, no se registra el tiempo empleado en la optimización, esto se debe a que el solucionador arrojó un código de error durante el proceso.

Tabla 4.1: Tiempo de resolución de cada solucionador bajo diferentes escenarios.

Solucionador	Escenario		
	Hidrógeno verde [s]	Eléctrico [s]	Completo [s]
Baron	0.033	0.025	0.053
Bonmin	3.630	0.050	4.800
Couenne	0.025	0.007	0.005
Gurobi	0.010	0.010	0.010
Knitro	-	0.021	-
LindoGlobal	-	0.038	-
MindtPy	0.396	0.029	0.511
Scip	0.010	0.010	0.010

A partir de los resultados obtenidos, fue posible identificar una gran variedad de solucionadores que permiten resolver el problema de optimización. Del listado de solucionadores candidatos, solo `Knitro` y `LindoGlobal` fueron incompatibles.

Los solucionadores con mejor desempeño general fueron `Baron`, `Couenne`, `Gurobi` y `Scip`. De todos ellos, se acordó descartar el uso de `Baron` por no tener una versión de uso gratuito.

4.3. Validación

Un elemento clave del software desarrollado, era la capacidad para abordar el problema de optimización de manera rápida y eficiente.

Dentro de las inquietudes principales del BSC acerca de este proyecto, destacaba el impacto que tendría la extensión del horizonte temporal del problema sobre el tiempo de ejecución. Así, este análisis resultaba crucial para evaluar la aplicabilidad del modelo en diversas situaciones.

Para responder a esta interrogante se diseñó un experimento que, ocupando diferentes solucionadores, realizaría la optimización del problema considerando un horizonte temporal desde 24 horas hasta 720 horas, registrando el tiempo empleado en su resolución.

4.3.1. Diseño

El objetivo principal del experimento realizado consistió en estimar la curva de eficiencia y la escalabilidad de la solución en base al tamaño de la entrada, para diferentes solucionadores.

Los solucionadores que se utilizaron para el experimento fueron aquellos que tuvieron mejor desempeño en las pruebas realizadas en la sección 4.2.3. En este caso, `Couenne`, `Gurobi` y `Scip`.

Ambiente de prueba

El experimento se realizó de manera local en un computador personal con las mismas especificaciones técnicas que las pruebas realizadas en 4.2.3.

Datos de prueba

Para el experimento, se consideró un escenario con la cadena de suministro completa.

Los valores de los parámetros del modelo fueron datos dados por el BSC en un archivo `JSON`. El archivo contenía 18 campos, cada uno correspondiendo a un parámetro. Cada campo contenía una lista de 24 valores, a excepción del campo de horizonte temporal, con un único valor y los campos SOC de la batería y el almacenamiento de hidrógeno, con 3 valores.

Experimento

Para analizar la variación en el tiempo de ejecución a medida que se extendía el horizonte temporal del problema, se optó por que cada uno de los solucionadores seleccionados resolviera el problema de optimización en intervalos de 24 horas, abarcando desde 24 horas hasta 720 horas, registrando el tiempo empleado.

Con el propósito de mitigar las posibles variaciones aleatorias en el tiempo de ejecución causadas por otros procesos en ejecución en la computadora, por cada solucionador se llevaron a cabo 30 optimizaciones independientes para cada horizonte temporal y se registraron los tiempos de cada una de estas ejecuciones.

Dado que los datos de prueba proporcionados por el BSC estaban originalmente diseñados para un horizonte temporal de 24 horas, a medida que se incrementaba el horizonte temporal, se tomaron medidas para duplicar los datos correspondientes. Así por ejemplo, para un horizonte temporal de 720 horas, se replicaron 30 veces los valores de los parámetros del archivo original.

Al concluir el experimento, se obtuvo el tiempo empleado por los solucionadores para diferentes horizontes temporales y se graficaron los resultados.

4.3.2. Resultados y Análisis

De los tres solucionadores que se utilizaron en las pruebas, dos arrojaron códigos de error durante el experimento.

El solucionador `Couenne` reportó un error en la optimización del problema a partir de un horizonte temporal de 72 horas, estableciendo que la resolución del problema no era factible o era muy costosa.

En cuanto al solucionador `Scip`, se reportó una falla a partir de un horizonte temporal de 144 horas, indicando que la solución óptima hallada no era factible en el problema original.

Para el caso de `Gurobi`, se pudo completar el experimento en su totalidad, obteniéndose los resultados que se observan en la figura 4.5.

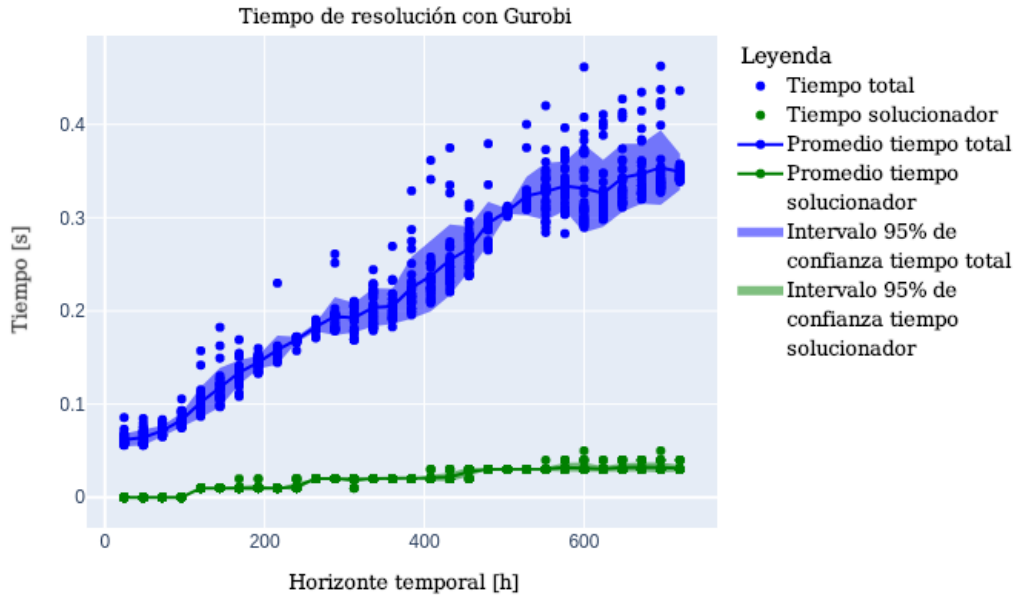


Figura 4.5: Tiempo de ejecución de `Gurobi` en función del tamaño de entrada.

A partir de estos resultados, se puede observar que para `Gurobi`, la curva de eficiencia en base al tamaño de la entrada es lineal.

Resulta importante destacar que el tiempo necesario para completar todo el proceso y

encontrar una solución, incluso en escenarios con horizontes temporales significativamente extensos, como 720 horas, es inferior a medio segundo. Estos resultados superaron las expectativas del BSC, demostrando una escalabilidad que se considera altamente satisfactoria para el proyecto.

El fracaso del experimento para `Couenne` y `Scip` puede deberse a múltiples razones. Una razón sería que estos solucionadores no escalan bien para este tipo de problema. La otra causa podría estar relacionada con limitaciones del entorno de prueba, especialmente con respecto al hardware utilizado para las ejecuciones.

Lamentablemente, por restricciones de acceso, no fue posible realizar este experimento en MareNostrum de manera de corroborar o descartar esta última hipótesis.

Capítulo 5

El modelo con estocasticidad

A partir de la implementación realizada en el capítulo anterior, se obtuvo un paquete capaz de optimizar el modelo de costos de manera eficaz. Sin embargo, aún faltaba resolver el desafío de representar la variabilidad intrínseca de algunos de sus parámetros mediante la integración de estocasticidad.

El siguiente capítulo detallará la segunda parte de la solución desarrollada, contemplando el diseño, implementación y validación del modelo con estocasticidad.

5.1. Diseño

Tal como se adelantó anteriormente, el modelamiento de costos de la cadena de suministro energético consideraba dentro de sus componentes algunos parámetros que, de manera natural, presentan variabilidad.

Este fenómeno es fácil de observar en aspectos como la cantidad de energía solar que se capta, dependiente entre otras cosas, de la luminosidad que se dé en un día determinado. Asimismo, puede notarse en el precio de compra de la energía de la red eléctrica - de ahora en adelante, llamado simplemente precio de la energía - o la cantidad de demanda que se tendrá de la misma en un momento del tiempo.

Debido a que en esta etapa del proyecto se deseaba explorar por primera vez una manera de integrar estocasticidad en el modelo y por el escaso tiempo con que se contaba, se optó por representar una sola de estas fuentes de incertidumbre.

Considerando la cantidad de datos históricos disponibles y el grado de relevancia del parámetro en el modelo, es que se decidió en conjunto con el equipo del BSC que se estudiaría la estocasticidad del precio de la energía.

5.1.1. Introducción de estocasticidad

En la literatura, se pueden encontrar muchas maneras de incorporar elementos de aleatoriedad a un problema [2, 10, 31, 54, 74, 77, 81]. Por su simplicidad, se estudiaron las tres alternativas de introducción de estocasticidad al precio de la energía que se presentan a continuación:

1. Generar la entrada como una variable aleatoria con cierta distribución de probabilidad.
2. Añadir términos de perturbación aleatoria, como sumar términos de ruido, que acompañen al precio en las ecuaciones del modelo.
3. Generar escenarios aleatorios para la entrada, que representen diferentes posibilidades futuras.

El criterio para elegir la alternativa a utilizar, fue decantarse por la opción que permitiera representar el comportamiento del precio de manera más realista. Para determinar cuál de las opciones anteriores cumplía con esto de mejor manera, se procedió a estudiar el comportamiento histórico del precio a lo largo de un año.

De esta manera, se obtuvieron datos del valor del precio de la energía española, por hora. Estos datos son públicos y pueden descargarse en formato `JSON` desde el sitio web de la Red Eléctrica de España [62].

Los datos históricos que se analizaron contemplaron el precio de la energía entre el 14 de octubre del 2022 a las 5:00 hasta el 15 de octubre del 2023 a las 4:00. La representación gráfica de estos valores se presenta en la figura 5.1.

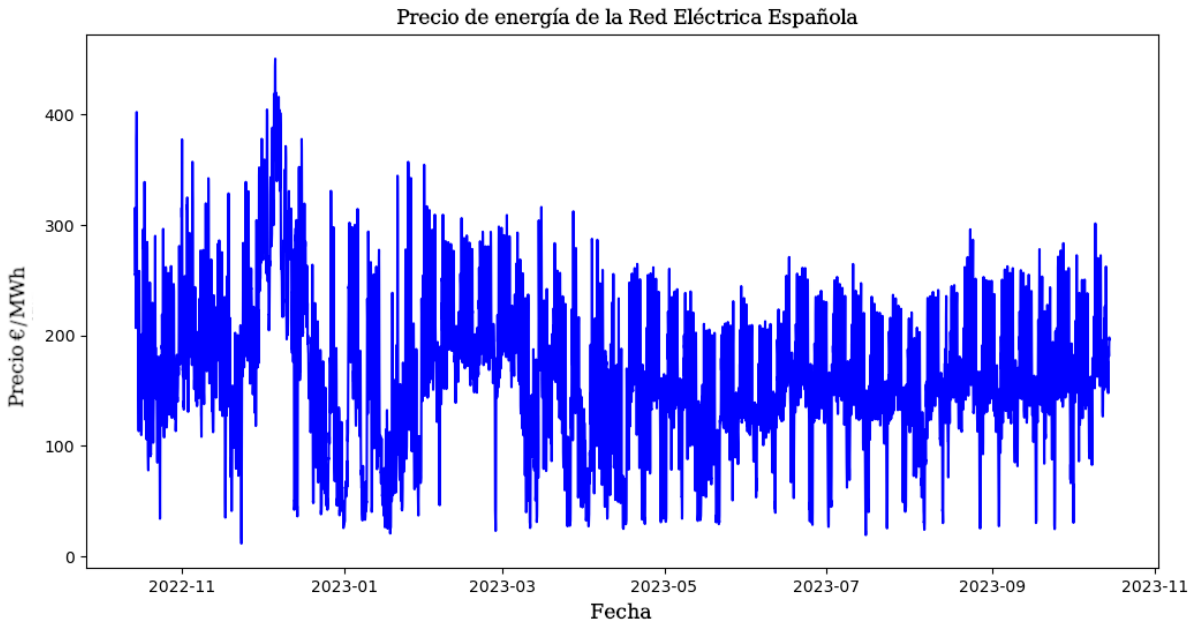


Figura 5.1: Precio de la energía de la Red Eléctrica Española, por hora.

Observando la figura anterior, se pudo notar que el precio de la energía correspondía a una serie temporal con mucha variabilidad y volatilidad, siguiendo un patrón que a simple vista, era difícil de determinar.

Para poder caracterizar a la serie temporal, se exportaron los datos históricos de las fechas mencionadas y se analizaron para detectar si seguían algún tipo de distribución conocida. Para ello, se tomó la totalidad de valores de precio por hora y se les aplicaron los tests de Anderson-Darling [48] y de Kolmogorov-Smirnov [49]. Los resultados obtenidos de los tests indicaron de manera concluyente que la serie temporal no se ajustaba a las distribuciones que fueron examinadas, incluyendo gamma, normal, exponencial y logarítmica.

A raíz de lo anterior, se decidió descartar la opción de añadir estocasticidad mediante el modelamiento del precio como variable aleatoria, con una distribución de probabilidad determinada.

Teniendo en cuenta que para el BSC lo primordial era añadir estocasticidad para representar de la forma más realista posible el comportamiento del precio, se optó por descartar también la segunda opción, que consistía en representar la incertidumbre mediante la introducción de términos de perturbación aleatoria.

De esta manera, la alternativa escogida para introducir estocasticidad fue la de generar escenarios aleatorios como datos de entrada del precio de la energía, que representaran diferentes posibilidades futuras.

5.1.2. Predicción de series temporales

Una manera de generar escenarios aleatorios es realizando una predicción de la serie temporal estudiada. Aplicando este método para el caso estudiado, se obtendrían posibles valores de precio de la energía que se ajustarían al comportamiento real de la serie y al mismo tiempo, se tendría una manera de predecir este parámetro del modelo.

Este último aspecto resultó muy llamativo, ya que implicaba un primer acercamiento para optimizar el costo de la cadena de suministro en el futuro. Así, se decidió generar los escenarios aleatorios con esta técnica.

Mediante el estudio de literatura relacionada, se identificaron dos acercamientos principales para la realización de predicciones de una serie temporal: la utilización de modelos estadísticos tradicionales [78] y el uso de modelos de redes neuronales [22, 40, 76, 80].

Como se expuso anteriormente en 2.1.2, para `Python` existen librerías de código abierto para ambos casos, por lo que se decidió explorar las dos alternativas y observar cuál se ajustaba de mejor manera al problema.

5.2. Implementación

Tal como se comentó en la sección anterior, se decidió introducir estocasticidad al modelo mediante la generación de escenarios del precio de la energía.

Estos escenarios fueron creados a partir de predicciones de la serie temporal mediante métodos estadísticos tradicionales y modelos de redes neuronales.

5.2.1. Modelos estadísticos tradicionales

Dentro de los modelos estadísticos de predicción de series temporales se encuentran los modelos ARIMA. En `Python` existe la librería de código abierto `statsmodels`, que implementa una versión de estos modelos.

Suposiciones de los modelos

Al momento de utilizar un modelo ARIMA para realizar una predicción de una serie temporal, es importante tomar en cuenta que el modelo asume ciertas características de la serie temporal. De no contar con los siguientes atributos, la predicción podría ser poco fiable.

1. Cantidad suficiente de datos: existen al menos 50 observaciones.
2. Estacionariedad: las propiedades estadísticas de la serie, como la media y la varianza, se mantienen constantes en el tiempo.
3. Ausencia de valores atípicos.
4. Linealidad: existe una relación lineal entre los valores actuales y los anteriores de la serie.
5. Normalidad de los errores: los residuos están normalmente distribuidos.

De esta manera, para realizar predicciones utilizando un modelo ARIMA, se llevó a cabo primero un estudio de los atributos de la serie.

Características de la serie

Analizando los datos de la serie temporal de precios de la energía, se observó que se contaba con 8784 registros, correspondientes a cada valor del precio de la energía por hora.

Para estudiar su estacionariedad, se aplicó el test de aumentado de Dickey-Fuller [27]. El resultado obtenido mediante este test indicó que existía evidencia suficiente para concluir que la serie era estacionaria.

Con el objetivo de determinar la estacionalidad[1], tendencia[2] y residuos o errores de la serie, se procedió a utilizar la técnica de Descomposición Estacional y de Tendencia utilizando Loess [20] (STL) para examinar el comportamiento de la serie en ciertos momentos del tiempo. En este caso, se decidió tomar de manera aleatoria la serie con los registros del mes de diciembre de 2022 y del mes de julio de 2023. Las visualizaciones obtenidas se observan en las figuras 5.2 y 5.3 correspondientemente.

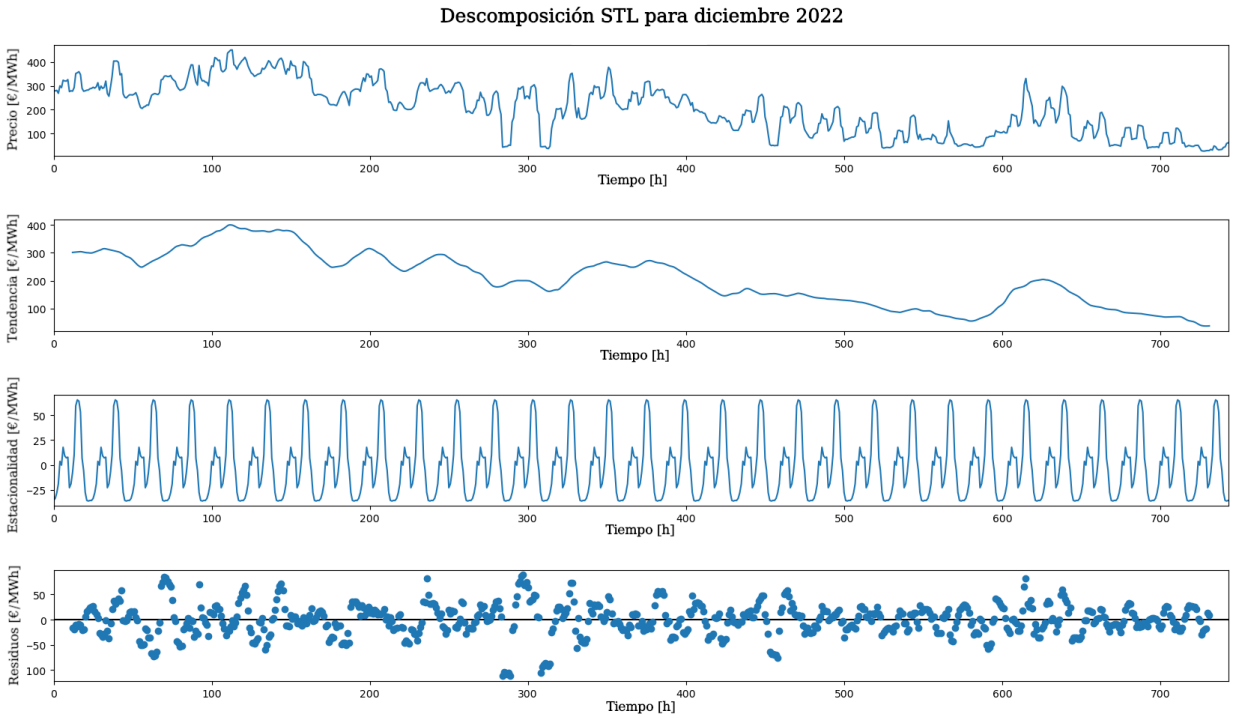


Figura 5.2: Descomposición STL para diciembre 2022.

[1]Existencia de patrones que se repiten a intervalos regulares de tiempo.
[2]Dirección general en la que se mueven los datos en el tiempo.

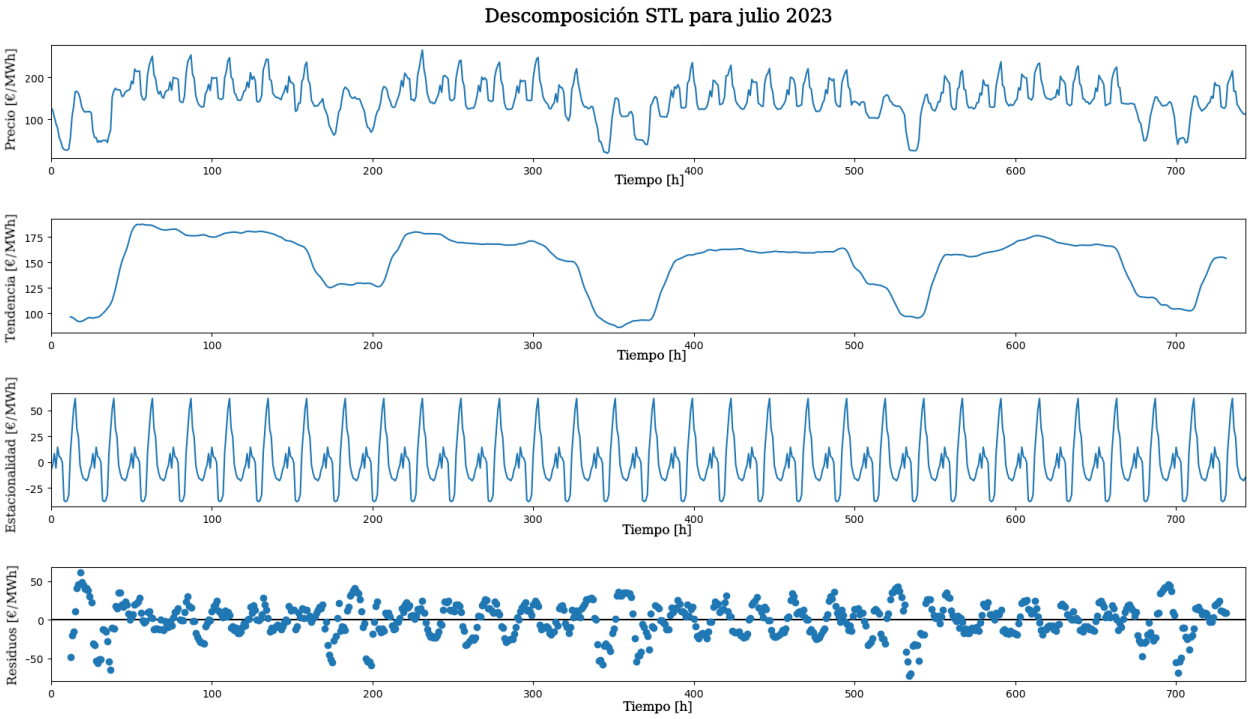


Figura 5.3: Descomposición STL para julio 2023.

A partir de los resultados obtenidos de la descomposición, se pudo observar que la tendencia de los precios en diciembre de 2022 se muestra como una curva suave, sin fluctuaciones drásticas. Al examinar el mes de julio de 2023 se pudieron apreciar claros períodos de alza y baja que se corresponden con las semanas del mes.

El comportamiento más interesante se encontró al observar los gráficos de estacionalidad, donde se pudo notar que en ambos casos existía un comportamiento estacional muy claro, con un patrón que se repetía de manera diaria.

Mediante el estudio del gráfico de residuos, se observó en ambos casos que éstos se distribuyeron en torno a la línea central, sin mostrar una estructura aparente. Sin embargo, fue posible observar cierta evidencia de volatilidad en los residuos, con algunos puntos alejándose significativamente de cero.

De esta forma, se pudo resumir la caracterización de la serie estudiada de la siguiente manera:

- Es una serie estacionaria.
- Tiene una gran cantidad de observaciones.
- Presenta un componente estacional.
- Tiene una alta volatilidad, con presencia de valores atípicos.

Implementación

Al evaluar las premisas de los modelos ARIMA y examinar las características específicas de la serie temporal, se observó que no se cumplían todas las condiciones asumidas por dichos modelos. No obstante, se procedió a realizar predicciones sobre la serie temporal a pesar de esta discrepancia, dado que no estaba claro en qué medida esta divergencia podría impactar en la calidad del resultado.

Estudiando las propiedades de la serie y considerando su estacionalidad y volatilidad, se identificó que el modelo ARIMA más adecuado era uno de tipo SARIMAX. De esta forma, utilizando la librería de `Python statsmodels` y tomando como inspiración los ejemplos de su tutorial oficial [52], se creó un *script* para instanciar un modelo SARIMAX.

Para determinar los valores de los coeficientes del modelo SARIMAX *(P, D, Q, s)* se tomó en cuenta la estacionalidad de la serie y las funciones de autocorrelación y autocorrelación parcial.

Por la estacionalidad diaria de la serie, se definió que $D = 1$ y $s = 24$.

Para obtener *(Q, P)* se graficaron las funciones de autocorrelación y de autocorrelación parcial considerando 24 rezagos, como se observa en las figuras 5.4 y 5.5 respectivamente. En ambos casos, el área sombreada establece un límite para determinar la significancia estadística con un nivel de significancia del 5 %. Todas las barras de rezago que no superan el área sombreada se consideran como estadísticamente insignificantes.

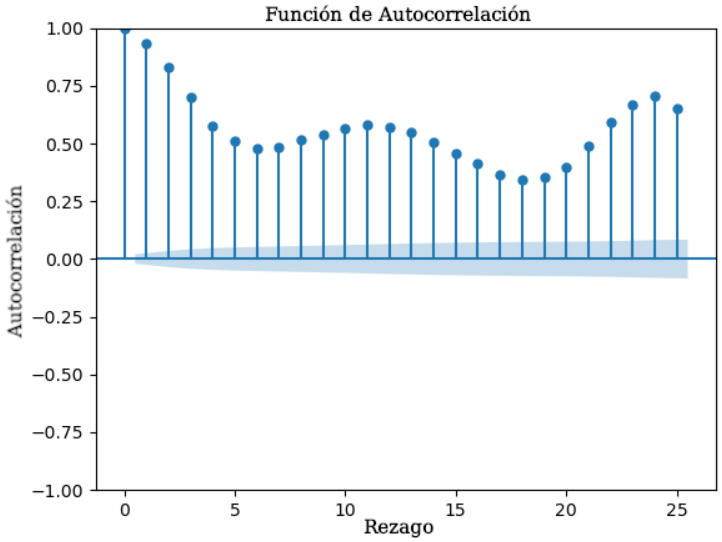


Figura 5.4: Función de autocorrelación de la serie, con un nivel de significancia del 5 %.

De esta manera, en el gráfico de autocorrelación no se observó un decaimiento significativo ni un punto de corte luego de los primeros 24 rezagos, lo que puede explicarse por la estacionalidad diaria de la serie. Por esta razón, se determinó el orden de la media móvil como $Q = 1$, correspondiente a un ciclo estacional.

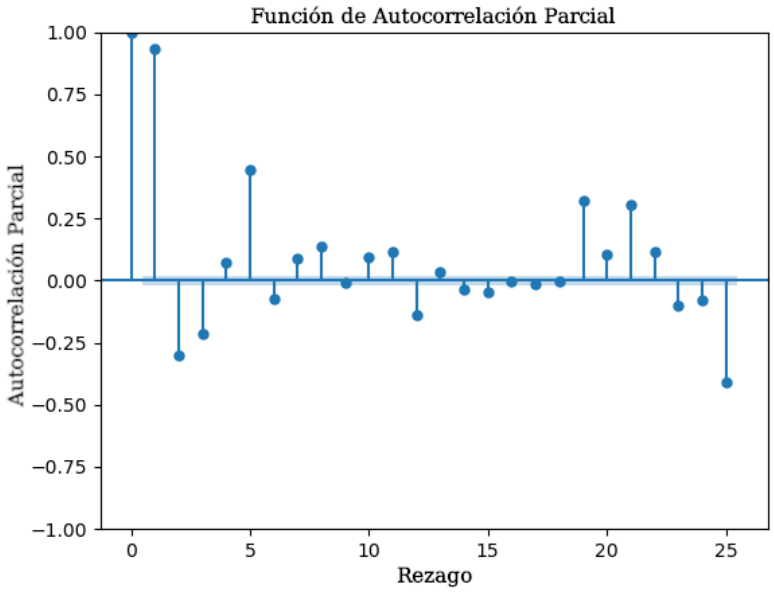


Figura 5.5: Función de autocorrelación parcial de la serie, con un nivel de significancia del 5 %.

A su vez, en el gráfico de autocorrelación parcial se observó significancia estadística hasta el octavo rezago, por lo que se estableció el orden del componente autorregresivo como $P = 8$.

5.2.2. Modelos de redes neuronales

Otra manera de realizar predicciones de series temporales es mediante el uso de modelos de redes neuronales. Esto cuenta con una amplia utilización para la predicción de series financieras [65, 80], por lo que podría adecuarse a la predicción de precios de la energía.

Una de las grandes ventajas de los modelos de redes neuronales sobre los modelos estadísticos, es que permiten realizar predicciones sobre cualquier tipo de serie temporal, independiente de sus propiedades. El único requisito es que cuenten con una gran cantidad de datos.

Implementación

Se estudió el desempeño en la generación de predicciones con cuatro tipos de redes neuronales diferentes: FNN, LSTM, GRU y TCN. La elección se este tipo de arquitecturas se basó en la existencia de documentación de su uso para la predicción de series temporales [12, 33, 40, 76].

La implementación realizada de los modelos de redes neuronales se inspiró en otras implementaciones existentes en `Python` [18, 41, 50, 68], que hacían uso de `Keras` con `tensorflow`.

Para cada tipo de red neuronal se creó una clase y un módulo de `Python`. De esta manera, al instanciar la clase indicando los datos de entrenamiento y de prueba, el *look back*[3] y la descripción de capas, se entregaba un modelo de red neuronal listo para ser entrenado.

Con el objetivo de facilitar este proceso, se creó el módulo `general_nn.py` con funciones comunes a todos los tipos de redes neuronales. Dentro de las funciones principales de este módulo, se encuentran las siguientes:

- `create_dataset`: recibe una serie de datos y los transforma en un formato adecuado para entrenar las redes neuronales.
- `train`: a partir de una serie de datos normalizados, el valor de *epoch* y tamaño de *batch*, entrena a una red neuronal utilizando el método `fit` de `Keras`.
- `forecast`: a partir de una serie de datos y un intervalo de tiempo determinado, genera una predicción de la serie temporal utilizando el método `predict` de `Keras`.
- `compute_mse`: recibe dos series temporales, la serie original y la predicción y entrega el error medio cuadrado de la predicción.
- `compute_hit_rate`: recibe dos series temporales, la serie original y la predicción y entrega el Hit Rate de cambio direccional de la predicción.

En base a lo observado en implementaciones existentes y considerando el contexto del problema, se instanciaron modelos siguiendo la arquitectura de capas que se ilustra en la tabla 5.1. Para todos los modelos, se estableció un valor común de *look back* = 24, debido a la estacionalidad de la serie y una función de pérdida de tipo MSE.

Tabla 5.1: Modelos de redes neuronales utilizados para realizar las predicciones.

Tipo	Capa de entrada	Capa oculta	Capa de salida
FNN	24 nodos	3 de 16 nodos	1 nodo
LSTM	24 nodos	3 de 16 nodos	1 nodo
GRU	24 nodos	3 de 50 nodos	1 nodo
TCN	24 nodos	4 capas de 25 nodos	1 nodo

5.3. Validación

Para evaluar qué tan útiles eran los modelos anteriores para generar escenarios que representaran de manera confiable la estocasticidad del precio de la energía, se procedió a realizar predicciones con los modelos y a evaluar su calidad, en términos de MSE y Hit Rate de cambio direccional.

Para esto, se siguió la metodología que se describe a continuación.

[3]Cantidad de pasos de tiempo previos que se utilizan para predecir el siguiente paso de la secuencia futura.

5.3.1. Diseño

Ambiente de prueba

Todos las predicciones se realizaron de manera local en un computador personal. Las especificaciones técnicas del equipo son las siguientes.

- Sistema operativo: Ubuntu 22.04.3 LTS version 64 bit.
- Procesador: Intel® Core™ i7-10510U CPU @ 1.80GHz × 8
- RAM: 16,0 GB.
- Disco: 512 GB

Datos de prueba

La serie temporal original se obtuvo a partir del sitio web de la Red Eléctrica Española [62], en formato `JSON`. El archivo contenía 8.784 pares de valores {`precio, fecha`}, correspondientes al precio de la energía, por hora, entre el 14 de octubre del 2022 a las 5:00 y el 15 de octubre del 2023 a las 4:00.

A estos datos se les debió realizar un preprocesamiento, eliminando información irrelevante y duplicados y completando registros faltantes.

Generación de predicciones

A partir de los datos procesados, se destinó el primer 80 % de los valores para el ajuste del modelo SARIMAX y el entrenamiento los modelos de redes neuronales. El 20 % restante se utilizó como conjunto de datos de prueba, con los que contrastar las predicciones. Este último porcentaje correspondió a 1.756 datos, comprendiendo precios de energía desde el 3 de agosto del 2023 a las 0:00 hasta el 15 de octubre del 2023 a las 4:00.

Todas las redes neuronales fueron entrenadas utilizando un tamaño de $batch = 96$ y $epoch = 700$, valores determinados de forma empírica tras probar diferentes configuraciones.

Con estas especificaciones, se procedieron a realizar las predicciones para cada modelo. En cada caso, se calculó el MSE y el Hit Rate de cambio direccional como medida de calidad de los pronósticos.

5.3.2. Resultados

La predicción realizada por el modelo SARIMAX se muestra en la figura 5.6. Por otra parte, las predicciones generadas por los modelos de redes neuronales FNN, LSTM, GRU y TCN se pueden observar en las figuras 5.7a, 5.7b, 5.8a y 5.8b respectivamente.

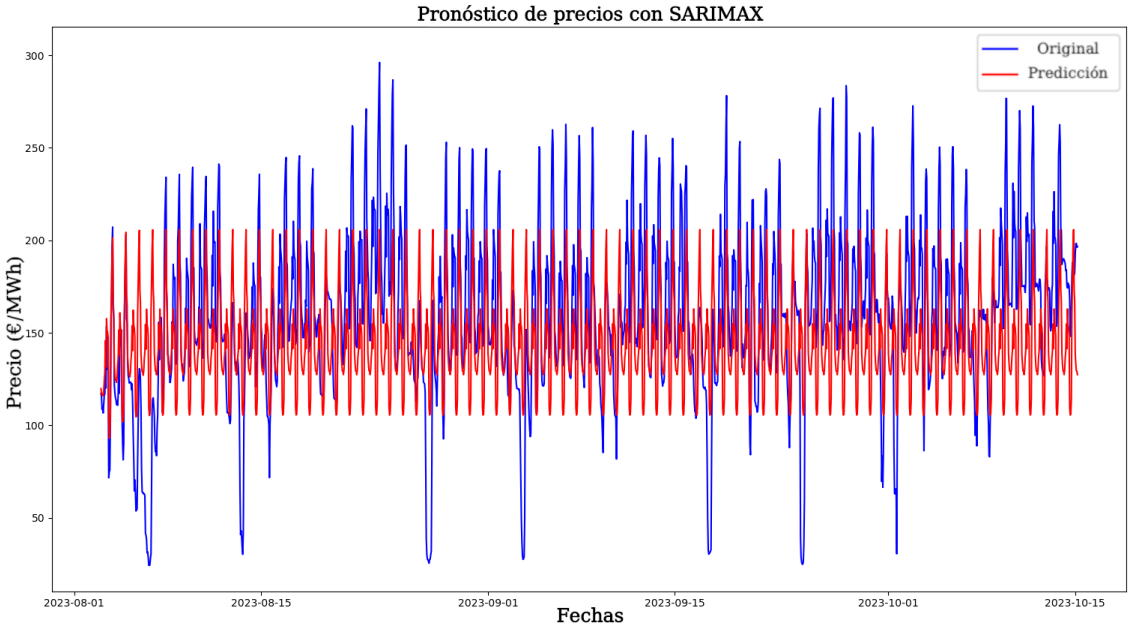
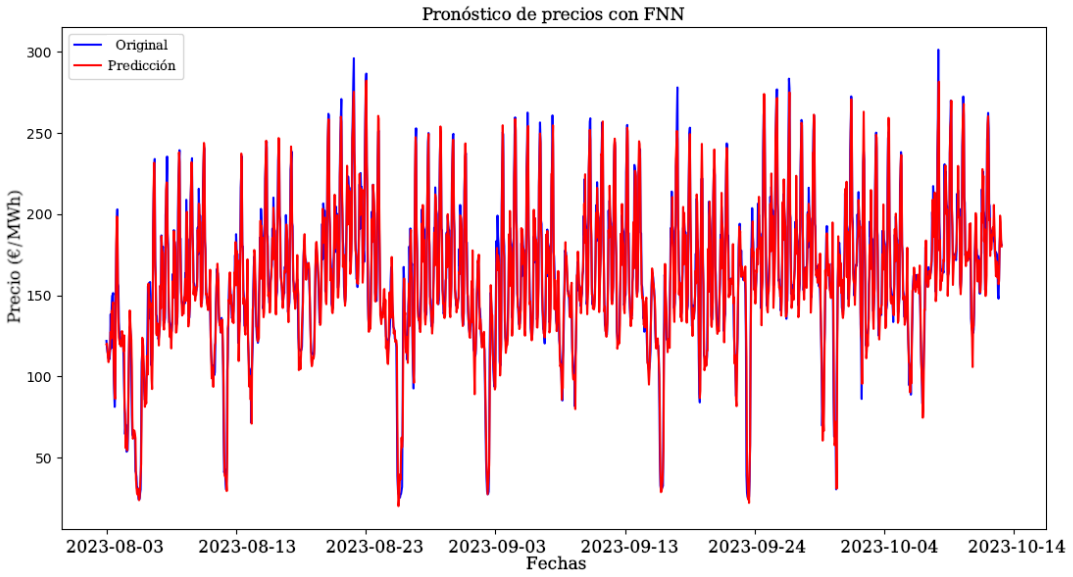
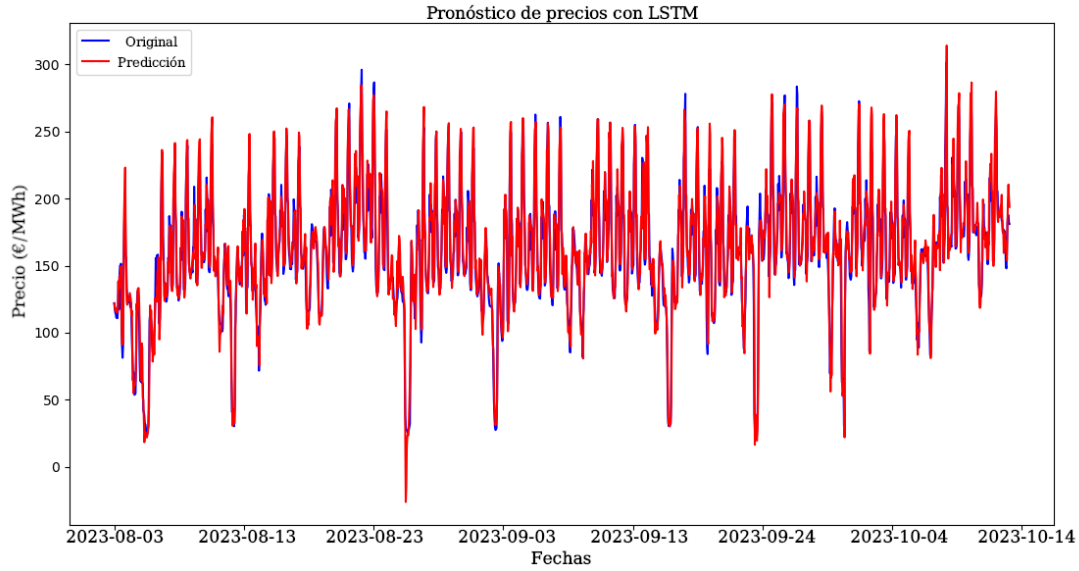


Figura 5.6: Predicción realizada por el modelo SARIMAX.

Para el modelo SARIMAX, el valor de MSE obtenido fue de 1552.73, mientras que el Hit Rate de cambio direccional fue de 83.37 %.

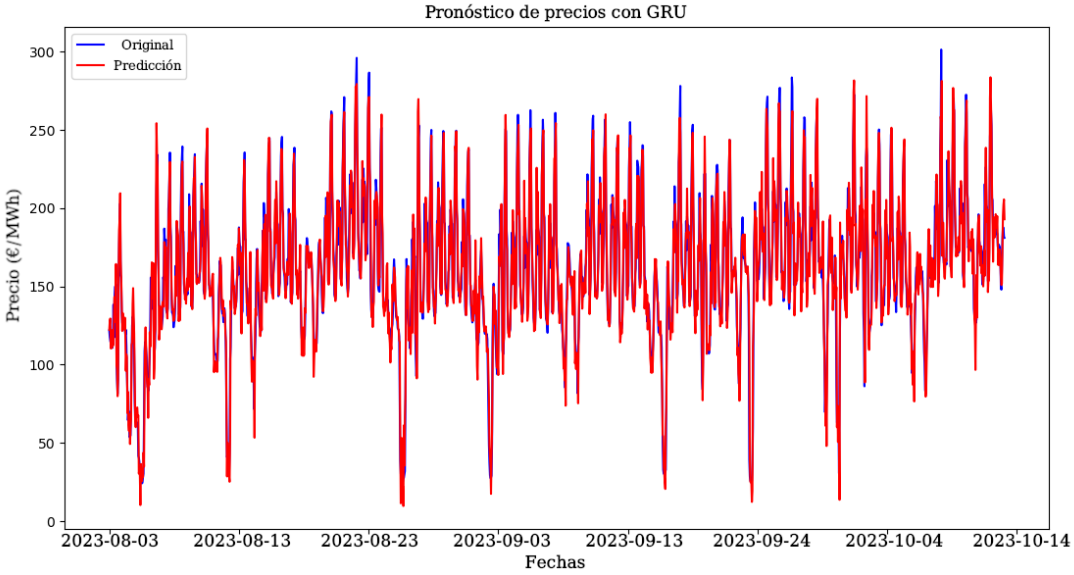


(a) FNN

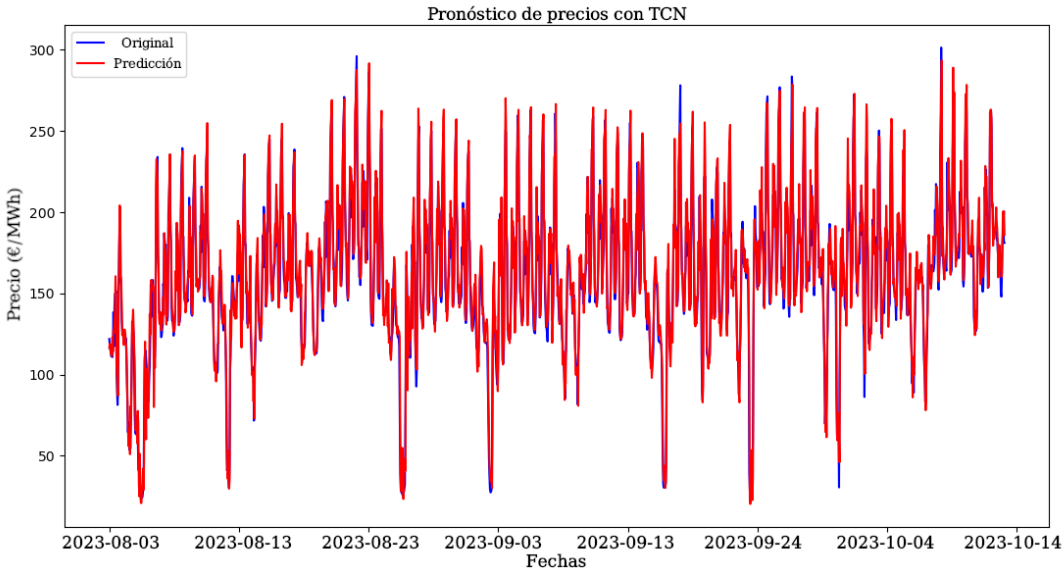


(b) LSTM

Figura 5.7: Predicciones realizadas por los modelos FNN y LSTM.



(a) GRU



(b) TCN

Figura 5.8: Predicciones realizadas por los modelos GRU y TCN.

Para los modelos de nedes neuronales, se obtuvieron los valores de MSE y Hit Rate de cambio direccional que se muestran en la tabla 5.2.

Tabla 5.2: Calidad de la predicción para diferentes modelos de redes neuronales.

Tipo	MSE	Hit Rate
FNN	111.84	75.09 %
LSTM	116.09	74.45 %
GRU	135.74	72.89 %
TCN	121.14	73.87 %

5.3.3. Análisis

Al realizar una inspección visual de la predicción generada por el modelo SARIMAX, se evidenció que los precios estimados se alejaban sustancialmente de la curva real. Frente a esto, se concluyó que el modelo no fue capaz de captar el comportamiento de la serie real de manera fiable.

El análisis fue muy distinto al observar los gráficos de las predicciones generadas por los modelos de redes neuronales, que se ajustaron de manera mucho mejor a la curva real. En todos los casos, se captó el comportamiento y volatilidad de la serie a tal grado que la serie real y la pronosticada se solapan en varios puntos.

Estudiando los valores de MSE obtenidos para cada caso, se pudo notar que la predicción con mayor error correspondió a la generada por el modelo SARIMAX. El valor de MSE obtenido en este caso equivale a un error medio de los precios de 39.4 euros.

El error de las predicciones disminuyó considerablemente para las generadas por los modelos de redes neuronales. Así, en la tabla 5.2 se observa que el menor valor de MSE, obtenido por el modelo FNN, fue de 111.84 mientras que el mayor, obtenido por el modelo GRU, correspondió a 135.74. Lo anterior se traduce en que, para la mejor predicción, el error medio de los precios pronosticados con respecto a la serie original fue de 10.58 euros, mientras que para la peor predicción fue de 11.65 euros.

En cuanto a los resultados de Hit Rate de cambio direccional obtenidos para los modelos de redes neuronales, se observó que el mayor valor, obtenido por el modelo FNN, fue de 75.09 % mientras que el menor, obtenido por el modelo GRU, correspondió a 72.89 %.

Un aspecto llamativo fue que el valor de Hit Rate de cambio direccional obtenido por el modelo SARIMAX fue mayor al obtenido por los modelos de redes neuronales. Esto significa, de forma directa, que el modelo estadístico predijo de mejor manera la dirección de cambio de los valores.

Para concluir sobre la calidad de las predicciones, se evaluaron en conjunto los valores obtenidos por ambas métricas y se consideró el análisis de la inspección visual de los resultados. Debido a esta razón, las predicciones realizadas por las redes neuronales se consideraron notablemente superiores a las generadas por el modelo SARIMAX. Por su parte, la mejor predicción, en general, fue la realizada por el modelo FNN.

Tras presentar los resultados al BSC, se corroboró que la predicción obtenida por el modelo FNN era suficientemente precisa para considerarse un método confiable de predicción

de precios de la energía.

De esta forma, se concluyó que los pronósticos generados con el modelo FNN son una manera válida de creación de escenarios aleatorios para incorporar estocasticidad. Al mismo tiempo y debido a su calidad, constituyen un método fiable para predecir los precios de la energía a lo largo del tiempo.

Capítulo 6

Validación

Durante los capítulos 4 y 5 se realizaron las implementaciones de los modelos determinista y estocástico, respectivamente. Cada una de estas secciones cerró con una validación del trabajo realizado en la iteración.

En el siguiente capítulo, se describirá la validación general de la solución desarrollada, en cuanto a la sensibilidad del modelo implementado y la calidad del código desarrollado en términos de extensibilidad y mantenibilidad.

6.1. Análisis de sensibilidad del modelo

Tras determinar un método válido para predecir los precios de la energía en el tiempo y concluir que, mediante los datos pronosticados, se podía integrar estocasticidad al modelo de costos, fue de interés para el BSC evaluar la sensibilidad del modelo frente a variaciones de este parámetro.

6.1.1. Diseño

El objetivo de este análisis fue evaluar el efecto de las variaciones del precio de la energía sobre el costo total de la cadena de suministro. De esta manera, se podría estudiar que tan robusto era el modelo frente a diferentes escenarios de éste parámetro.

Para lograr esto, se llevó a cabo un experimento en el cual se resolvió el problema de optimización de un modelo de suministro energético, frente a diferentes valores de precio de la energía.

De esta forma, considerando una serie de precios original, se aplicaron variaciones porcentuales sobre sus valores. Se resolvió el problema de optimización para un modelo con estas entradas y se anotaron los resultados obtenidos.

Ambiente de prueba

Todos los experimentos se realizaron de manera local en un computador personal. Las especificaciones técnicas del equipo son las siguientes.

- Sistema operativo: Ubuntu 22.04.3 LTS version 64 bit.
- Procesador: Intel® Core™ i7-10510U CPU @ 1.80GHz × 8
- RAM: 16,0 GB.
- Disco: 512 GB

La ejecución de los experimentos se realizó en un entorno virtual creado para este propósito, donde se instaló el paquete `cuco_opt` junto con sus requerimientos.

Datos de prueba

Para el experimento, se consideró un escenario con la cadena de suministro completa.

Los valores de los parámetros del modelo fueron datos dados por el BSC en un archivo `JSON`. El archivo contenía 18 campos, cada uno correspondiendo a un parámetro. Cada campo contiene una lista de 24 valores, a excepción del campo de horizonte temporal, con un único valor y los campos SOC de la batería y el almacenamiento de hidrógeno, que contienen 3 valores.

Experimento

Para observar la variación del valor de optimización con respecto a variaciones del precio de la energía, se resolvió el problema de optimización tomando el conjunto original de los datos de prueba y modificando únicamente el campo de precio de la energía.

Tomando como punto de partida los datos originales, en cada ejecución posterior se realizó un incremento porcentual de los valores y se anotó el valor del resultado de la optimización para cada caso. Los incrementos realizados consistieron en un 1 %, 2 %, 5 %, 10 %, 15 % y 20 % de los valores originales del precio de la energía.

6.1.2. Resultados y Análisis

Tras realizar el experimento y graficar los resultados obtenidos, se obtuvo la visualización que se muestra en la figura 6.1.

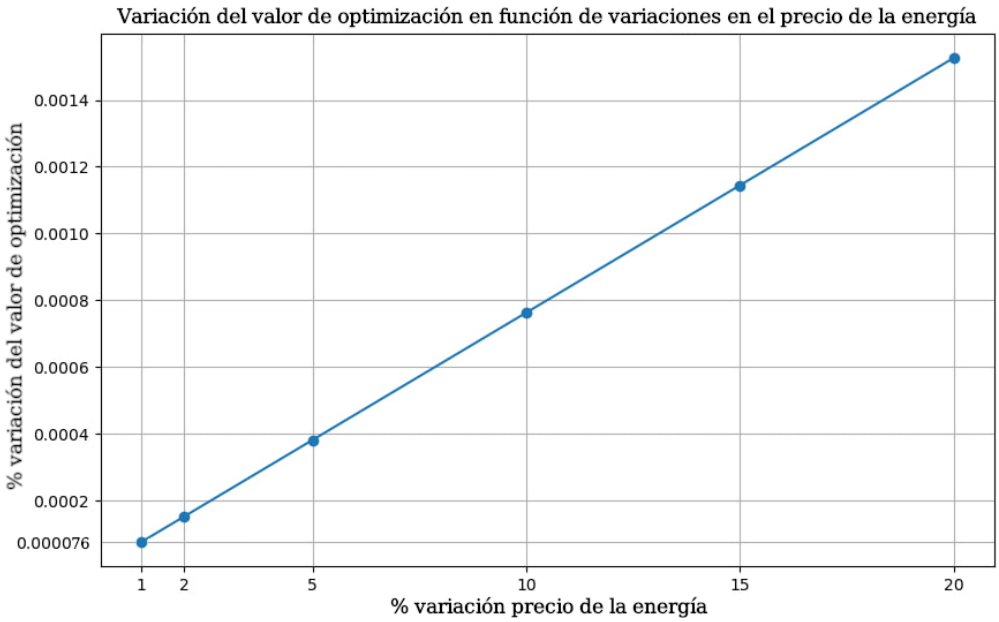


Figura 6.1: Variación del costo total optimizado de la cadena de suministro en función de variaciones en el precio de la energía.

Analizando el resultado del experimento, se pudo observar que la variación del valor de optimización de costos de la cadena suministro energético, con respecto a la variación de precios de energía de la red eléctrica, era perfectamente lineal.

De lo anterior se deduce una relación directamente proporcional, donde por cada $1\,\%$ que varíe el precio, el valor de optimización de la cadena variará en un $7,6 \times 10^{-5}\,\%$.

Con esto se concluyó que el modelo era poco sensible frente a variaciones en el precio de la energía. Además, el efecto que tuvo la variación del precio frente al valor de optimización fue consistente y predecible. Ambas observaciones dan señales de robustez del modelo desarrollado.

La baja sensibilidad del modelo de costos frente al precio de la energía se puede explicar por dos razones. La primera es que el modelo de costos considera otros 17 parámetros adicionales, lo que significa que el impacto de cualquier variación individual en el precio de la energía puede ser mitigado o equilibrado por la influencia combinada de estos otros factores.

La segunda razón se relaciona con la diferencia de magnitud entre las cantidades que se comparan. En el contexto del problema, los precios asumen valores, generalmente, en el orden de magnitud 10^2, mientras que los costos de la cadena se encuentran en el orden de 10^8. Esto implica que las variaciones porcentuales en los precios de la energía, aunque significativas en términos absolutos, representan solo una fracción muy pequeña cuando se proyectan sobre los costos totales de la cadena.

6.2. Extensibilidad

El modelo implementado en este trabajo fue una primera aproximación al modelamiento de costos de la cadena de suministro y se espera que su representación se complejice en el futuro. De esta manera y tomando en cuenta el contexto del problema, se identificó que los principales puntos de extensión correspondían a añadir o modificar componentes de la cadena de suministro y añadir o modificar restricciones del problema.

Originalmente, la implementación se basaba en un solo *script* de `Jupyter Notebook`. Este *script* creaba una instancia de un modelo de `Pyomo`, configurando sus atributos de manera manual. Cualquier cambio o adición de un componente en la cadena de suministro, o de una restricción, requería alterar múltiples secciones del código. Para comprender dónde realizar estos ajustes, era necesario revisar todo el archivo exhaustivamente.

La solución desarrollada adoptó un enfoque modular, de manera de asignar responsabilidades específicas a cada módulo. El diseño en el cual se basó la implementación, representado anteriormente en la figura 4.1, contempló desde un inicio los puntos de extensión y se planificó para facilitar la incorporación de modificaciones en estas áreas.

A continuación, se describen los pasos a seguir para agregar un nuevo componente a la cadena de suministro y para añadir una nueva restricción, con el fin de ilustrar este punto.

6.2.1. Añadir un nuevo componente

Para añadir un nuevo componente a la cadena, se debe crear un nuevo módulo de componente `name_of_component.py` en de la carpeta `components`. Dentro de este módulo, se deben crear las funciones que definan las variables y parámetros que lo caracterizan.

Luego de crear este módulo, se deben invocar las funciones definidas en su interior en `model.py` para que el nuevo componente sea añadido al modelo.

En el caso de que el nuevo componente tenga variables o parámetros que se incluyan en una restricción o en la función objetivo, se deben realizar los cambios correspondientes en `constraints.py` y `objectives.py` respectivamente.

6.2.2. Añadir una nueva restricción

Para añadir una restricción, se debe crear una nueva función en el módulo `constraints.py` que defina la expresión matemática correspondiente a la restricción.

Luego de crear esta función, se la debe invocar en el módulo `model.py` para que sea añadida como una restricción del modelo.

6.2.3. Existencia de tests

El código también cuenta con una suite de tests automatizados, que prueban todas las funcionalidades del paquete, bajo distintos escenarios. Esto proporciona la seguridad de que, de introducir modificaciones al código que rompan las funcionalidades existentes, se reportarán los errores de manera inmediata.

6.2.4. Análisis

A partir de lo expuesto anteriormente, se evidencia que la solución desarrollada permite introducir modificaciones al modelo matemático sin que resulte muy difícil, costoso o se requiera de muchos cambios en el código.

La implementación del paquete resultante es modular, se encuentra debidamente testeada, y es altamente cohesiva. Además, cuenta con bajo acoplamiento entre módulos, a excepción del módulo `model.py`, que al ser el elemento central de la implementación, depende de los demás.

De esta manera, se concluye que el código desarrollado es extensible bajo las condiciones evaluadas.

6.3. Mantenibilidad

CUCO es un proyecto que aún se encuentra en una etapa inicial y se espera que más desarrolladores formen parte de él en un futuro. En vista de esto, fue crucial implementar estrategias que facilitaran la mantención del software existente y de sus futuras ampliaciones.

Para asistir a este fin, se definieron y aplicaron diferentes estándares de programación. Estos incluyen un testeo mínimo del 80 % de las líneas de código, la adhesión al estándar PEP8 para la escritura y la elaboración de documentación pertinente.

6.3.1. Nivel de cobertura

Se introdujo una suite de testing para probar el correcto funcionamiento de todas las funcionalidades del paquete. Para diseñar y ejecutar los test se utilizó la librería `pytest`, mientras que para el reporte de la cobertura se utilizó `coverage`.

En el momento de la entrega del paquete, se ejecutaban satisfactoriamente todos los tests, alcanzando un nivel de cobertura del 99 % de las líneas de código.

Lo anterior permitió garantizar que el paquete funcionaba como se esperaba en el contexto estudiado, a la vez que contribuyó a facilitar la detección y reparación de errores futuros.

6.3.2. Flujo de Integración Continua

Para asegurar la calidad del código que se suba al repositorio del proyecto, se configuró un flujo de integración continua. De esta manera, cada vez que se integre una modificación, se gatillará la ejecución de la suite de test completa y se reportará de inmediato la existencia de cualquier error.

6.3.3. Formateo de código

Con el objetivo de facilitar la comprensión del paquete para los desarrolladores que se unan al proyecto en el futuro, se homogeneizó el código siguiendo las directrices del estándar de escritura de `Python`, PEP8.

Para resguardar que cualquier modificación al código cumpla con este estándar, se utilizó la herramienta `precommit` y se configuró para evaluar que cualquier cambio realizado se ajuste a PEP8. En caso de no ser así, la misma herramienta realiza las modificaciones necesarias, de manera automática. Todo este proceso se lleva a cabo de manera local, antes de que los cambios se suban al repositorio.

6.3.4. Documentación y ejemplos

El paquete desarrollado fue extensamente documentado, tanto en el código fuente como en el archivo `README` del proyecto. Los documentos describen de manera detallada la finalidad del paquete, sus instrucciones de instalación, requerimientos y ejemplos de uso.

Dentro del paquete mismo se incluyó la carpeta `examples` con *scripts* de ejemplo de los usos más comunes de la librería.

Como un recurso adicional y en un esfuerzo por mejorar la accesibilidad, se creó y disponibilizó un videotutorial del paquete.

6.3.5. Análisis

Considerando los puntos detallados anteriormente, se pudo concluir que el código desarrollado cuenta con estándares de programación rigurosos, un grado de cobertura exhaustivo y una documentación detallada, aspectos que facilitan la colaboración futura.

La combinación de todas estas estrategias y prácticas muestra un enfoque integral y bien planificado hacia la mantenibilidad del software, ayudando a su viabilidad a largo plazo.

Capítulo 7

Conclusiones

En el siguiente capítulo, se presentarán las conclusiones generales del trabajo realizado, considerando aspectos técnicos y personales.

Finalmente, se plantearán posibles líneas de trabajo futuro del proyecto.

7.1. Trabajo realizado

Mediante el trabajo realizado, se logró cumplir con casi todos los objetivos planteados en un inicio.

El valor que aportó la solución desarrollada para el avance del proyecto fue tremendo y muy apreciado por el BSC. Se logró construir un paquete en Python con la implementación del modelo de costos y se logró hallar una forma bastante confiable para predecir los precios de la energía de la red eléctrica, incorporando estocasticidad al modelo desarrollado.

El único objetivo específico que no se cumplió en su totalidad fue el de la modelación de variabilidad de los parámetros del modelo: esto se logró de manera parcial, ya que si bien se incorporó con éxito la estocasticidad del precio de la energía, reflejando fielmente su comportamiento real, faltó añadir la variabilidad de la demanda energética y la captación de energía solar. Lo anterior no fue posible debido a falta de tiempo, pero podría haberse realizado de la misma manera.

La labor efectuada fue bastante extensa y contempló varios desafíos. La primera parte, que consistió en la implementación del modelo determinista, fue la más sencilla, ya que lo solicitado era algo con lo que me encontraba bastante familiarizada. A lo largo de prácticamente toda la carrera, se nos ha enseñado a desarrollar buen código, en una amplia gama de lenguajes de programación.

La parte novedosa de esta sección fue la del dominio del problema en sí, debido a que nunca había participado de proyectos de optimización y menos del mercado eléctrico. Esto implicó destinar una porción del tiempo inicial de trabajo al aprendizaje de nuevos conceptos

y herramientas, como Pyomo. La información que se puede encontrar en internet de estos tópicos es sumamente amplia, se encuentra bastante actualizada y está bien explicada, por lo que no resultó en mayores complicaciones.

En contraste, la segunda parte del trabajo consistió en explorar territorio completamente nuevo para mí, con el que me costó bastante sentirme más cómoda. Esta iteración no solo fue investigativa en gran parte, sino que también contempló áreas de conocimiento que desconocía. De esta manera, una porción muy grande de tiempo se destinó a la lectura, investigación y conversaciones con expertos para adquirir el conocimiento necesario y así abordar adecuadamente las tareas pendientes.

Debido a que el proyecto *CUCO* se encuentra en una fase inicial, fui la primera persona con conocimientos formales de computación en formar parte del desarrollo. Esto significó, por el lado negativo, tomar algunas decisiones del proyecto en solitario, sin otra persona para discutir lo determinado desde una perspectiva más técnica. Sin embargo y desde el lado positivo, tuve una total libertad para orientar la realización de las tareas asignadas como estimara apropiado, empoderándome del trabajo y solidificando la confianza en mis conocimientos y habilidades.

Tomar un proyecto real una etapa temprana, entregó valiosas lecciones de desarrollo de software, como la importancia de la existencia de estándares de código. Comenzar a trabajar sobre un proyecto sin estándares es bastante frustrante e implica una muy evitable pérdida de tiempo.

Este trabajo, debido a su gran extensión, conllevó decidir cómo aportar el mayor valor posible con el escaso tiempo disponible. Si bien algunas de las partes desarrolladas podrían haberse hecho mejor, las decisiones tomadas consideraron lo mejor que se podría hacer bajo el contexto y con los recursos que se tenían.

Finalmente, desde una perspectiva de trabajo en equipo, aprender a comunicar ideas de forma clara, concisa y transparente resultó sumamente fundamental, en especial trabajando con personas de manera remota. El diálogo fluido, honesto y asertivo, entre todas las partes que conforman un grupo, ayuda a construir la confianza necesaria y un clima de trabajo óptimo para sacar adelante un proyecto.

7.1.1. Oportunidades de mejora

Tal como se adelantó anteriormente, algunas partes del trabajo desarrollado pueden ser mejoradas, contando con más recursos.

Para la implementación del modelo determinista, hubiera sido ideal realizar los experimentos en MareNostrum, el entorno de producción. En particular, el experimento de tiempo de ejecución de los solucionadores en función del horizonte temporal arrojó error para dos solucionadores cuando se consideraba un problema de 72 horas y podría deberse a limitaciones del equipo de prueba.

Probar en el superordenador permitiría aceptar o rechazar esta hipótesis, lo cual no es

trivial, puesto que el solucionador escogido para la resolución del problema, Gurobi, es de uso gratuito con licencia académica por tan solo un año.

En cuanto a la introducción de estocasticidad, de tener más tiempo, se podría haber añadido para los parámetros faltantes, como la demanda y la captación de energía solar, permitiendo por primera vez, predecir los mejores momentos de compra y venta de energía en el DAM.

Por último, se podrían haber evaluado más formas de considerar la estocasticidad de los parámetros, ya sea utilizando métodos diferentes a los explorados o probando con otras configuraciones de los modelos presentados.

7.2. Trabajo futuro

Tomando en cuenta la etapa inicial de desarrollo en la que se encuentra el proyecto *CUCO*, aún queda mucho trabajo por delante.

Con respecto al modelo matemático de costos, se hace necesario incluir nuevas consideraciones para acercarlo más a la realidad. Así, por ejemplo, existen aspectos que de momento no se están contemplado en el modelo pero que sí forman parte del problema real, como el envejecimiento de los activos, la descarga de las baterías y los componentes de generación de energía eólica.

Desde la perspectiva de la implementación y como se detalló en las oportunidades de mejora, falta considerar la estocasticidad de manera más amplia, para que el modelo realmente sea útil para entregar información que dicte la toma de decisiones en el mercado energético.

Otra línea de trabajo interesante y que se podría desarrollar de manera paralela, es analizar de forma más acabada el desempeño del modelo que se tiene, en cuanto a su robustez y sensibilidad. Alternativas a explorar en este sentido podrían ser estudiar la sensibilidad del modelo frente a variaciones de otro parámetro o un conjunto de ellos, o estudiar la robustez del modelo frente a casos extremos.

Bibliografía

[1] Z. Ahmad, R. Noor, and J. Zhang. Multiple neural networks modeling techniques in process control: a review. *Asia-Pacific Journal of Chemical Engineering*, 4:403–419, 2009.

[2] Patrick Amar. Pandæsim: An epidemic spreading stochastic simulator. *Biology*, 9, 2020.

[3] AMPL: Advanced Modeling for Optimization Solutions. Artelys knitro solver: Download, pricing & documentation. `https://ampl.com/products/solvers/solvers-we-sell/knitro/`.

[4] AMPL: Advanced Modeling for Optimization Solutions. Baron solver: Download, pricing & documentation. `https://ampl.com/products/solvers/solvers-we-sell/baron/`.

[5] AMPL: Advanced Modeling for Optimization Solutions. Lindo global solver: Download, pricing & documentation. `https://ampl.com/products/solvers/solvers-we-sell/lindoglobal/`.

[6] Barcelona Supercomputing Center. Marenostrum. `https://www.bsc.es/marenostrum/marenostrum`.

[7] Barcelona Supercomputing Center. Qué hacemos. `https://www.bsc.es/es/descubre-el-bsc/el-centro/que-hacemos`.

[8] Black documentation. Getting started. `https://black.readthedocs.io/en/stable/integrations/source_version_control.html`.

[9] G.E.P. Box, G.M. Jenkins, G.C. Reinsel, and G.M. Ljung. *Time Series Analysis: Forecasting and Control*. Wiley Series in Probability and Statistics. Wiley, 2015.

[10] Tomás Caraballo, Renato Colucci, Javier López de-la Cruz, and Alain Rapaport. A way to model stochastic perturbations in population dynamics models with bounded realizations. *Journal of Systems and Control Engineering*, 220:171–182, 2006.

[11] Young-Jin Cha, Wooram Choi, and Oral Buyukozturk. Deep learning-based crack damage detection using convolutional neural networks. *Computer-Aided Civil and Infrastructure Engineering*, 32:361–378, 2017.

[12] Jenhui Chen and Ashu Abdul. A session-based customer preference learning method by using the gated recurrent units with attention function. *Ieee Access*, 7:17750–17759, 2019.

[13] COIN-OR. Bonmin users' manual. https://www.coin-or.org/Bonmin/Intro.html.

[14] COIN-OR. Couenne, a solver for non-convex minlp problems. https://www.coin-or.org/Couenne/.

[15] G. Cybenko. Approximation by superpositions of a sigmoidal function. mathematics of control signals and systems. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

[16] Aurea Anguera de Sojo, Juan A. Lara, David Lizcano, and María A. Martínez. Sensor-generated time series events: A definition language. *Sensors*, 12:11811–11852, 2012.

[17] Alexander Dowling, Ranjeet Kumar, and Victor Zavala. A multi-scale optimization framework for electricity market participation. *Applied Energy*, 190:147–164, 2017.

[18] Carl M. Ellis. Temporal convolutional network using keras-tcn. https://www.kaggle.com/code/carlmcbrideellis/temporal-convolutional-network-using-keras-tcn.

[19] Thomas Fischer and Christopher Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270:654–669, 2018.

[20] Freie Universität Berlin. Seasonal and trend decomposition using loess (stl). https://www.geo.fu-berlin.de/en/v/soga-py/Advanced-statistics/time-series-analysis/Seasonal-decompositon/STL-decomposition/index.html.

[21] Chad Fulton. Statsmodels: Econometric and statistical modeling with python. In *21th Python in Science Conference*, pages 83–89, 2022.

[22] Jin Gao and Lihua Dai. Forecasting covid-19 cases: A comparative analysis between recurrent and convolutional neural networks. *Results in Physics*, 24, 2021.

[23] GitHub Docs. Understanding github actions. https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions.

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[25] Wilson Guasti and Isaac Santos. Solving differential equations using feedforward neural networks. In *International Conference on Computational Science and Its Applications*, page 385–399, 2021.

[26] Yi Guo, Xuejiao Han, Xinyang Zhou, and Gabriela Hug. Incorporate day-ahead robustness and real-time incentives for electricity market design. *Applied Energy*, 332, 2023.

[27] R.I.D. Harris. Testing for unit roots using the augmented dickey-fuller test: Some issues relating to the size, power and the lag structure of the test. *Economics Letters*, 38:381–386, 1992.

[28] ShiFeng Hu, ShiJian Zhu, MinJun Zhong, and QiWei He. Neural network modeling and generalized predictive control for giant magnetostrictive actuators. In *2009 Chinese Control and Decision Conference*, pages 2981–2985, 2009.

[29] Rob J. Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

[30] isort. Git hook. `https://pycqa.github.io/isort/docs/configuration/git_hook.html`.

[31] Ying Ji, Jianhui Wang, Shijie Yan, Wenzhong Gao, and Hepeng Li. Optimal microgrid energy management integrating intermittent renewable energy and stochastic load. In *IEEE Advanced Information Technology, Electronic and Automation Control Conference*, 2015.

[32] Jupyter. Project jupyter documentation. `https://docs.jupyter.org/en/latest/`.

[33] Yoshio Kajitani, Ian Mcleod, and Keith W. Hipel. Forecasting nonlinear time series with feed-forward neural networks: A case study of canadian lynx data. *Journal of Forecasting*, 24, 2005.

[34] Demetris Koutsoyiannis. The unavoidable uncertainty of renewable energy and its management. *Geophysical Research Abstracts*, 18, 2016.

[35] Holger Krekel. Coverage.py. `https://coverage.readthedocs.io/en/7.4.0/`.

[36] Holger Krekel. pytest: helps you write better programs. `https://docs.pytest.org/en/7.4.x/`.

[37] Subhash Kumar Kumawat, Alok Bansal, and Sultan Singh Saini. Design analysis and implementation of stock market forecasting system using improved soft computing technique. *International Journal on Future Revolution in Computer Science & Communication Engineering*, 8:9–16, 2022.

[38] Weixian Li, Thillainathan Logenthiran, and Wai Lok Woo. Multi-gru prediction system for electricity generation's planning and operation. *IET Generation, Transmission and Distribution*, 13:1630–1637, 2019.

[39] Zhenwei Li, Jing Han, and Yuping Song. On the forecasting of high-frequency financial time series based on arima model improved by deep learning. *Journal of Forecasting*, 39:1081–1097, 2020.

[40] Q. El Maazouzi, A. Retbi, and S. Bennani. Automatisation hyperparameters tuning process for times series forecasting: application to passenger's flow prediction on a railway network. *The International Archives of the Photogrammetry Remote Sensing and Spatial Information Sciences*, XLVIII-4/W3-2022:53–60, 2022.

[41] Abhishek Mamidi. Time series analysis - artificial neural networks. `https://www.kaggle.com/code/abhishekmamidi/time-series-analysis-artificial-neural-networks`.

[42] Minja Marinović, Dragana Makajić, and Milan Stanojević. Optimization in day-ahead planning of energy trading. *Institute for Innovation and Public Purpose, Working Paper Series*, 11:201–208, 2013.

[43] Joaquim R. R. A. Martins and Andrew Ning. *Engineering Design Optimization*. Cambridge University Press, 2021.

[44] MathWorks. Gurobi optimizer. `https://www.mathworks.com/products/connections/product_detail/gurobi-optimizer.html`.

[45] P.E. Naill M. Momani. Time series analysis model for rainfall data in jordan: Case study for using time series analysis. *American Journal of Environmental Sciences*, 5:599–604, 2009.

[46] K. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *Ieee Transactions on Neural Networks*, 1, 1990.

[47] Next Kraftwerke. What is day-ahead trading of electricity? `https://www.next-kraftwerke.com/knowledge/day-ahead-trading-electricity`.

[48] National Institute of Standards and Technology. *NIST/SEMATECH e-Handbook of Statistical Methods: Anderson-Darling Test*. 2012. `https://www.itl.nist.gov/div898/handbook/eda/section3/eda35e.htm`.

[49] National Institute of Standards and Technology. *NIST/SEMATECH e-Handbook of Statistical Methods: Kolmogorov-Smirnov Goodness-of-Fit Test*. 2012. `https://www.itl.nist.gov/div898/handbook/eda/section3/eda35g.htm`.

[50] Heiko Onnen. Temporal coils: Intro to temporal convolutional networks for time series forecasting in python. `https://towardsdatascience.com/temporal-coils-intro-to-temporal-convolutional-networks-for-time-series-forecasti`

[51] Bo Pang, Erik Nijkamp, and Ying Nian Wu. Deep learning with tensorflow: A review. *Journal of Educational and Behavioral Statistics*, 45, 2019.

[52] Josef Perktold, Skipper Seabold, and Jonathan Taylor. Sarimax: Introduction. `https://www.statsmodels.org/dev/examples/notebooks/generated/statespace_sarimax_stata.html`.

[53] Josef Perktold, Skipper Seabold, and Jonathan Taylor. Statistical models, hypothesis tests, and data exploration. `https://www.statsmodels.org/stable/index.html`.

[54] Per Pettersson, Gianluca Iaccarino, and Jan Nordström. A stochastic galerkin method for the euler equations with roe variable transformation. *Journal of Computational Physics*, 257:481–500, 2014.

[55] Sushan Poudel and Anu Radha. Speech command recognition using artificial neural networks. *JOIV International Journal on Informatics Visualization*, 4, 2020.

[56] pre-commit. Documentation. `https://pre-commit.com/`.

[57] Munish Puri, Srinivas Tipparaju, and Vijay Kumar Sutariya. *Artificial Neural Network for Drug Design, Delivery and Disposition*. Elsevier Science, 2015.

[58] Pyomo. About pyomo. http://www.pyomo.org/about.

[59] Pyomo. Mindtpy solver. https://pyomo.readthedocs.io/en/stable/contributed_packages/mindtpy.html.

[60] pyOptSparse. Ipopt. https://mdolab-pyoptsparse.readthedocs-hosted.com/en/latest/optimizers/IPOPT.html.

[61] Read the Docs. Flake8: Your tool for style guide enforcement. https://flake8.pycqa.org/en/latest/.

[62] Red Eléctrica Española. TÉrmino de facturaciÓn de energÍa activa del pvpc 2.0td penÍnsula. https://www.esios.ree.es/es/analisis/1001?vis=1&start_date=14-10-2022T05%3A00&end_date=15-10-2023T04%3A55&geoids=8741&compare_start_date=13-10-2022T05%3A00&groupby=hour&compare_end_date=14-10-2023T04%3A55.

[63] SCIP Optimization Suite. What is scip? https://www.scipopt.org/index.php#about.

[64] Skipper Seabold and Josef Perktold. Statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, pages 92–96, 2010.

[65] Omer Berat Sezer, Mehmet Ugur Gudelek, and Ahmet Murat Ozbayoglu. Financial time series forecasting with deep learning : A systematic literature review: 2005–2019. *Applied Soft Computing*, 90, 2020.

[66] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Computer Vision and Pattern Recognition*, 2014.

[67] Singularity. Introduction to singularity. https://docs.sylabs.io/guides/3.5/user-guide/introduction.html.

[68] Yujian Tang. Build a gru rnn in keras. https://pythonalgos.com/2022/01/02/build-a-gru-rnn-in-keras/.

[69] Tensorflow. Api documentation. https://www.tensorflow.org/api_docs/.

[70] Tensorflow. Keras: The high-level api for tensorflow. https://www.tensorflow.org/guide/keras.

[71] Michael Tobias. Two ways to use grid-tied batteries: Energy arbitrage and grid services. https://www.ny-engineers.com/blog/two-ways-to-use-grid-tied-batteries-energy-arbitrage-and-grid-services.

[72] Universidad de Notra Dame de Ingeniería Química y Biomolecular. 60 minutes to pyomo: An energy storage model predictive control example. https://github.com/ndcbe/CBE60499/blob/4026fb161f81b9c660293b1cc9e49b7425312f97/docs/01.01-Pyomo-Nuts-and-Bolts.ipynb.

[73] Guido van Rossum, Barry Warsaw, and Alyssa Coghlan. Pep 8 – style guide for python code. `https://peps.python.org/pep-0008/`.

[74] Zidong Wang, Fuwen Yang, and Xiaohui Liu. Robust filtering for systems with stochastic non-linearities and deterministic uncertainties. *Journal of Systems and Control Engineering*, 220:171–182, 2006.

[75] Lichao Wu and Guilherme Perin. On the importance of pooling layer tuning for profiling side-channel analysis. In *International Conference on Applied Cryptography and Network Security*, pages 114–132, 2021.

[76] Jining Yan, Lin Mu, Lizhe Wang, Rajiv Ranjan, and Albert Y. Zomaya. Temporal convolutional networks for the advance prediction of enso. *Science report*, 10, 2020.

[77] Dongmin Yu and Noradin Ghadimi. Reliability constraint stochastic unit commitment by considering correlation of random variables with copula theory. *IET Renewable Power Generation*, 13, 2019.

[78] Peter G. Zhang. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175, 2003.

[79] Xinjing Zhang, Chao Qin, Eric Loth, Yujie Xu, Xuezhi Zhou, and Haisheng Chen. Arbitrage analysis for different energy storage technologies and strategies. *Energy Reports*, 7:8198–8206, 2021.

[80] Xuehan Zhang. Financial time series forecasting based on lstm neural network optimized by wavelet denoising and whale optimization algorithm. *Academic Journal of Computing & Information Science*, 5:1–9, 2022.

[81] Noemí Zoroa, E. Lesigne, María-José Fernández Sáez, and P. Zoroa. The coupon collector urn model with unequal probabilities in ecology and evolution. *Journal of the Royal Society Interface*, 14:481–500, 2017.