2024

¿Cómo pueden aplicarse las ecuaciones diferenciales neuronales al consumo de la red eléctrica española para asegurar un voltaje constante?

APLICACIÓN DE ECUACIONES DIFERENCIALES NEURONALES PARA ESTABILIZAR EL VOLTAJE EN LA RED ELÉCTRICA ESPAÑOLA

HUGO ÁLVAREZ SEVILLA

Tabla de contenidos

1-Introducción

Las redes eléctricas operan a un voltaje constante, esto es un hecho en todas las partes del mundo. Si este voltaje cambia lo suficiente, los dispositivos enchufados a la corriente dejan de funcionar. Para mantener el voltaje constante, se realizan complejas predicciones para equilibrar la producción eléctrica con el consumo. El objetivo de esta monografía es predecir el consumo utilizando ecuaciones neuronales diferenciales para equiparar la producción eléctrica en la red española hora a hora.

Esto es importante pues tradicionalmente la predicción de consumo eléctrico se ha hecho manualmente, sin embargo, un modelo de inteligencia artificial como “long short term memory (LSTM)” o, en este caso, ecuaciones diferenciales ordinales neuronales “neural ODE’s” son especialmente útiles cuando se trata de predecir modelos continuos en el tiempo, porque modelan el sistema que rige a los datos, una función, y la utilizan para ver como van a ser los datos en el futuro. Por ende, un sistema de predicción automática puede ser más preciso y servir de ayuda en la tarea de mantener el voltaje estable en una red eléctrica.

Para medir el éxito de la red neuronal vamos a tomar un enfoque cuantitativo. Se examinarán los datos reales y se compararan contra las predicciones del modelo, de ahí obtendremos un porcentaje de acierto dentro de un margen de error estándar en el mundo real.

Esta monografía consecuentemente va a tener un alcance explorativo, ya que las ecuaciones diferenciales neuronales son un concepto rnovedoso y que hasta día de hoy tiene aplicaciones limitadas. Pero también Analítico, pues vamos a determinar si una inteligencia artificial puede alcanzar el nivel de predicción que se obtiene hoy en día con humanos programas de análisis de datos.

2-Conocimientos básicos sobre machine learning

Las ecuaciones diferenciales neuronales actúan como una especie de puente entre el aprendizaje profundo (deep learning) y la modelización matemática clásica. Las ecuaciones diferenciales neuronales fueron presentadas por primera vez en 2018 en un estudio realizado por un grupo de investigadores en la universidad de Toronto (arXiv:1806.07366 [cs.LG])

Machine learning es un aspecto del campo de estudios de la inteligencia artificial. Es la base de todos los modelos y arquitecturas que se pueden usar. El objetivo de machine learning es utilizar datos para crear un modelo capaz de realizar una tarea, y medir la eficiencia de ese modelo numéricamente. El aprendizaje es el proceso donde uno encuentra el modelo adecuado cambiando parámetros como la arquitectura, las dimensiones de los datos o el número de pasadas que el modelo le da a los datos, midiendo este aprendizaje de forma numérica con varios indicadores. El objetivo de este proceso no es de conseguir el modelo que mejor se ajuste a los datos que se tienen, sino el que sea capaz de mejor predecir los datos futuros.

La información que ya se tiene y se le da al modelo para que aprenda se conoce como “input data”. Esta va a ser un vector con un numero de dimensiones positivas y enteras. Los datos que el modelo trata de predecir se conocen como “output data”, Estos, si bien pueden ser un vector, en la mayoría de los casos van a ser un valor numerico escalar.

Cuando uno está entrenando un modelo en machine learning, los datos se dividen en tres tipos, entrenamiento, test y validación. El más importante por ahora es el set de entrenamiento, pues los otros son para asegurarse que el modelo se ha entrenado correctamente. El set de entrenamiento viene en parejas (x_1, y_1) ...(x_n,y_n). Donde X es el dato que se le da al modelo, en forma de vector y Y

es el dato esperado del modelo, en valor numérico.

Una vez el modelo esta entrenado, se mide la diferencia entre los datos reales y las predicciones del modelo con una función llamada “loss function” o función de perdidas, L. El problema entonces se reduce a optimización matemática, donde se busca minimizar esta función encontrando los valores ideales para varias variables de las que hablaremos más adelante. Tal que $\mu = \arg min_{\theta} L(\hat{y})$. Donde μ es el valor que minimiza la función de perdidas, arg min es notación para representar el argumento que minimiza, es decir, estamos buscando el valor del parámetro 0 que minimiza L(y), siendo L(y) la propia función de perdidas.. Cabe anotar que hay muchas funciones de perdidas dependiendo de la arquitectura elegida, esto es una generalización para comprender su funcionamiento.

En problemas de regresión, se busca predecir valores continuos, es decir, un valor numérico para cada valor de tiempo individual, a partir de valores de entradas, como discutido anteriormente. la función de perdidas más popular para problemas de regresión y consecuentemente la que vamos a aplicar va a ser MSE “Mean Square Error” (Error cuadrático medio). Su fórmula matemática es la siguiente:

$$MSE(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Esta fórmula representa la media entre las diferencias al cuadrado del valor real (yi) y el valor predicho ($\hat{y}_i$), donde N representa el número total de datos observados hasta ese punto. Al estar elevado al cuadrado, el valor resultante siempre va a ser positivo, y por la misma razón los errores más grandes se penalizan más duramente.

Ahora que se han expuesto los conocimientos necesarios sobre aprendizaje automático, vamos a dar una base teórica sobre ODE’s para entender el resto de esta monografía.

Lo primero que es necesario entender es el problema del valor inicial. El problema del valor inicial es aquel en el que se defina una ecuación diferencial ordinaria de forma $\frac{d\varphi}{dt} = f(t, x)$ lo que significa que $f(t, x)$ es la función que describe como cambia x con respecto al tiempo. El problema del valor inicial IVP por sus siglas en ingles, busca la función φ (t) que satisface las siguientes condiciones:

$\varphi(t0\ ) = x0$, es decir, la funcion en t0 debe comenzar en x0

$(t, \varphi(t)) \in D$, la función debe mantenerse dentro del dominio D

$\frac{d\varphi}{dt} = f(t, \varphi(t))$, la derivada de φ (t) con respecto al tiempo está determinada por la función f. Es decir, f describe cómo debe cambiar ϕ(t) a medida que el tiempo avanza, de acuerdo con la ecuación diferencial.

La solución única de un IVP se anota tal que $\varphi(t; t0\ , x0\ )$, lo que indica que la solución depende del tiempo y de las condiciones iniciales.

Por último, es necesario conocer el teorema de Picard, este teorema, que ahora explicaremos, es imperativo para verificar la validez de nuestros resultados, ya que va a comprobar si la solución que hemos encontrado al teorema del valor inicial es determinada y única, por lo que nunca se va a dar el caso de dos posibles consumos para una única demanda, pues la solución correcta va a ser la que cumpla con este teorema. El teorema de Picard dice que nuestra solución debe de cumplir con las condiciones de Lipschitz, es decir, la función con respecto a y tiene que cumplir la desigualdad:

$$\|f(t,x1)-f(t,x2)\| \leq L\|x1-x2\|$$

$$L>0$$

Esta desigualdad quiere decir que la función con respecto a su segunda variable, x, no cambia más rápido que la constante de Lipschitz, L. Esto nos asegura que no ocurre un crecimiento infinito por lo que la solución es real para cualquier par de valores de x x1, x2.

3-Los neural ODE como modelo matemático

si bien los Neural ODE pueden abordarse desde un punto de vista de código, vamos a plantearlo tal como para conseguir una definición matemática. En este caso, vamos a tomar el concepto de una ecuación diferencial ordinal y utilizar técnicas de machine learning para parametrizar el campo vectorial. Consideremos un problema de valor inicial cualquiera

$$\frac{dx}{dt}(t) = f\theta\,(t, x(t))$$

$$x(0) = x_0$$

Estas son las condiciones iniciales que hemos discutido antes.

En una ecuación diferencial ordinaria neuronal, el campo vectorial f_0 sería parametrizado por una red neuronal, de cualquier tipo de arquitectura, comúnmente LSTM.

Esto en sí es correcto, pero como hemos discutido antes, para que tenga una solución y consecuentemente sea útil, es necesario que f0 sea continuo con Lipzchig como explicado anteriormente.

En este mismo contexto, la solución de la ecuación va a venir dada por una función φ que va a estar dentro del conjunto $\Omega \subseteq R \times R \times R^d \times R^p$. En este caso, uno puede proyectar la solución donde se toma un vector de entrada x y se devuelve otro de salida en las mismas dimensiones $\varphi: Rd \to Rd$. Entonces la solución con condiciones iniciales x_0 con respecto al tiempo viene dada por $x \mapsto \varphi(T; 0, x, \theta)$, donde θ son los parametros del modelo.

Lo inconveniente con esta idea es que estamos asumiendo implícitamente que la transformación interna mantiene la misma dimensionalidad, es decir, el input y output tienen el mismo número de dimensiones, lo que muchas veces, como en nuestro caso, no es cierto, ya que la entrada tiene varios parámetros, que se representan en un espacio de altas dimensiones y la salida va a ser un escalar.

4-Ejemplo teórico de los neural ODE

Pongamos el siguiente ejemplo. Tenemos unos datos Representables con un tensor $R^{128x128x3}$

Donde 128x128 es el tamaño de la imagen y el 3 representa la intensidad de cada color, rojo, verde o azul. Y queremos clasificar la imagen en 7 distintas categorías, por lo que el output tiene que ser R^7. En este caso nuestro flujo seria tal que

$$\varphi : R^{128x128x3} \rightarrow^{\varphi} R^{128x128x3} \rightarrow^{l_\theta} R^7 \rightarrow^{softmax} Categorias$$

Aquí, la imagen pasa primero por una Neural ODE (representada por φ) que la transforma en una representación del mismo tamaño, pero con características más fáciles de interpretar por el modelo. Esta aproximación se logra considerando φ como una función que evoluciona con el tiempo, tal como se discutió en el problema del valor inicial.

Posteriormente, la salida de φ (x) se introduce en la capa lineal l_θ, que mapea dicha representación a un vector de R7. Finalmente, la función softmax convierte este vector en una distribución de probabilidad sobre las 7 categorías, indicando cuál es la más probable según el modelo.

Como hemos mencionado antes, la dimensionalidad de el vector de entrada es muy importante, ya que a más características más dimensiones van a ser necesarias, pero también es practico dejar dimensiones vacías para mejorar la precisión del modelo, esta técnica es fundamental para nuestro modelo.

Hay una técnica muy útil para estos modelos conocida como aumentación. Esta técnica inserta un mapa afín entre el input y los valores iniciales para aumentar la dimensionalidad de los datos, permitiendo así al modelo trabajar en más dimensiones y por consecuencia predecir mejor.

Esta técnica, matemáticamente hablando podemos definirla de la siguiente manera. Pensemos en un input con forma $x \in R^d$, entonces, los valores iniciales del modelo van a tener la forma $g_0(x)$ en vez de x, donde $g_0(x)$ es una función con también una cantidad de dimensiones, pero aumentada

con respecto a x porque las funciones implícitamente tienen más dimensiones que un valor numérico. El objetivo de esto entonces es aumentar las dimensiones que llegan al modelo. Esto ocurre pues el flujo de una ODE no puede cambiar la forma de sus inputs, por lo que el espacio en el que trabaja va a ser de las mismas dimensiones que sus valores iniciales.

5-Ejemplo practico de los neural ODE aplicando a los datos de la REE

Ahora que hemos visto como podríamos realizar este modelo desde un punto de vista matemático, y las técnicas que vamos a aplicar, es momento de realizarlo. Para comenzar, voy a tomar un ejemplo de mis datos obtenidos de la red eléctrica Española y explicar como funciona el modelo en detalle, sin embargo, esto tiene dos limitaciones, la primera es que una parte fundamental del modelo es el campo parametrizado por el LSTM, en este se ajustan unos pesos hasta encontrar los pesos óptimos para el sistema dado. Para encontrar los pesos más óptimos hace falta iterar con una cantidad altísima de datos, y, ante la posibilidad de hacer esto manualmente, voy a ejemplificar con unos pesos arbitrariamente elegidos, aunque no sean representativos de los pesos finales que el modelo encuentre, ya que no puedo calcularlos manualmente.
Una aclaración necesaria es que para este ejemplo vamos a ignorar el teorema de Picard que hemos mencionado antes, esto no es arbitrario. Con una cantidad de datos tan pequeños y unos pesos aleatorios es extremadamente probable que la solución que encontremos, si bien sea un numero real localmente, tienda hacia el infinito positivo o negativo en un intervalo de tiempo mayor, es por esto que, aunque en el modelo final si se va a tener en cuenta, no es razonable tenerlo en cuenta para este ejemplo porque entonces se plantearía una imposibilidad para encontrar pesos y una ecuación diferencial que pueda cumplir estas condiciones.

Lo primero que necesitamos entonces son datos limpios, si vemos el anexo 1, esta pieza de código lo que esta haciendo es combinar todos los datos brutos extraídos de la red eléctrica y procesándolos a

unos datos de más calidad y con la forma que nuestro modelo necesita. Nosotros nos vamos a basar en los datos para el 17 de septiembre de 2024. Esto de forma arbitraria, podría haberse escogido cualquier otro día. El código va a eliminar las características que no son interesantes para nuestro modelo, como los porcentajes de energía renovable, y va a convertir ciertas características temporales a senos y cosenos. Esto se hace ya que, por regla general, el consumo de electricidad suele aumentar y disminuir tanto a las mismas horas del día como en tendencias más generales como meses. La naturaleza cíclica de las funciones trigonométricas es perfectas para esta aplicación. Como el consumo es parecido año tras año, este no se trata como una función cíclica, sino como un valor escalar.

Real,Prevista,Hora_sin,Hora_cos,Mes_sin,Mes_cos,Año_num

26679,26783,0.0,1.0,-1.0,-1.8369701987210297e-16,2024

25066,25312,0.25881904510252074,0.9659258262890683,-1.0,-1.8369701987210297e-16,2024

24056,24250,0.49999999999999994,0.8660254037844387,-1.0,-1.8369701987210297e-16,2024

23688,23632,0.7071067811865476,0.7071067811865476,-1.0,-1.8369701987210297e-16,2024

23430,23383,0.8660254037844386,0.5000000000000001,-1.0,-1.8369701987210297e-16,2024

23443,23280,0.9659258262890683,0.25881904510252074,-1.0,-1.8369701987210297e-16,2024

24449,24570,1.0,6.123233995736766e-17,-1.0,-1.8369701987210297e-16,2024

27830,27812,0.9659258262890683,-0.25881904510252063,-1.0,-1.8369701987210297e-16,2024

30217,30329,0.8660254037844387,-0.4999999999999998,-1.0,-1.8369701987210297e-16,2024

30705,30985,0.7071067811865476,-0.7071067811865475,-1.0,-1.8369701987210297e-16,2024

31133,30716,0.49999999999999994,-0.8660254037844387,-1.0,-1.8369701987210297e-16,2024

30437,30432,0.258819045102521,-0.9659258262890682,-1.0,-1.8369701987210297e-16,2024

30229,30404,1.2246467991473532e-16,-1.0,-1.0,-1.8369701987210297e-16,2024

30390,30533,-0.2588190451025208,-0.9659258262890683,-1.0,-1.8369701987210297e-16,2024

29757,30337,-0.4999999999999997,-0.8660254037844388,-1.0,-1.8369701987210297e-16,2024

29367,29676,-0.7071067811865471,-0.7071067811865479,-1.0,-1.8369701987210297e-16,2024

29373,29184,-0.8660254037844385,-0.5000000000000004,-1.0,-1.8369701987210297e-16,2024

29682,29683,-0.9659258262890683,-0.25881904510252063,-1.0,-1.8369701987210297e-16,2024

30864,30637,-1.0,-1.8369701987210297e-16,-1.0,-1.8369701987210297e-16,2024

31085,31276,-0.9659258262890684,0.2588190451025203,-1.0,-1.8369701987210297e-16,2024

32406,32855,-0.8660254037844386,0.5000000000000001,-1.0,-1.8369701987210297e-16,2024

34338,34290,-0.7071067811865477,0.7071067811865474,-1.0,-1.8369701987210297e-16,2024

31733,32077,-0.5000000000000004,0.8660254037844384,-1.0,-1.8369701987210297e-16,2024

28636,28660,-0.25881904510252157,0.9659258262890681,-1.0,-1.8369701987210297e-16,2024

Para este ejemplo, vamos a centrarnos en uno de los datos de la muestra, el primero.

26679,26783,0.0,1.0,-1.0,-1.8369701987210297e-16,2024

Aquí el valor real de consumo: 26679MW

Y el valor que se predijo: 26783MW

Ahora entonces necesitamos calcular la media y desviación estándar de estos valores sobre el resto de los datos de nuestra muestra. Si realizamos estos cálculos podemos ver que la media es 27,268 MW y la desviación estándar es de 3163 MW. Estos valores son necesarios para la normalización que tenemos que realizar sobre los datos. Los datos se normalizan porque para el modelo es mucho más fácil procesar datos entre -1 y 1 que datos con valores elevados, realmente en un escenario hipotético el resultado debería de ser el mismo, pero esto reduce gratamente el tiempo de computación necesario para entrenar este modelo. Además, trae la ventaja de aumentar la facilidad al mirar a los cálculos que se van a realizar

La normalización se realiza con la siguiente formula

$$X_{std} = \frac{X - \bar{X}}{std(X)}$$

Para el valor real

$$Real_{std} = \frac{26679 - 27268}{3163} \approx -0.1862.$$

Para el valor previsto

$$Prevista_{std} = \frac{26783 - 27268}{3163} \approx -0.1533.$$

Ahora tenemos un input, siendo este el valor real de la hora anterior, y un output, el valor previsto.

El siguiente paso es entrenar el modelo, como ya he mencionado, vamos a realizar una representación simplificada pues los pesos reales y la forma final de la ecuación es el resultado de un proceso de afinación muy largo y complejo.

Un modelo simple de una ODE sobre la que podemos resolver seria

$$\frac{dx}{dt} = W.x + b$$

Donde W y b serian los pesos que definen el sistema y que se ajustarían tras cada dato.

Esta ecuación diferencial en concreto se podría resolver por métodos exactos, pero para ser fieles a lo que está ocurriendo en el modelo real vamos a integrar entre 0 y 1 con respecto a t y utilizando el método de Euler para aproximar un resultado. Los valores 0 y 1 los hemos escogido pues vamos a analizar valores en una escala de tiempo pequeña, la resolución de nuestro tiempo es de un dato por hora, por lo que integrar entre 0 y 1 va a representar un intervalo de tiempo de una hora, es decir, un solo dato, acorde al ejemplo sobre el que estamos trabajando.

$$x_{pred,std}(1) = x_{real,std}(0) + (W \cdot x_{real,std}(0) + b).$$

En base a esta ecuación integrada, y con un paso de Euler, podemos tomar un valor de W y uno de b para el ejemplo, digamos que W = 0,001, b = -0,1

Sobre estos valores calculamos la predicción

$$x_{pred,std}(1) = -0.1862 + 0.001 \cdot (-0.1862) + (-0.1).$$

$$x_{pred,std}(1) \approx -0.2863862$$

La predicción normalizada es de -0.2863862 mientras que el objetivo era -0.1533

Podemos revisar la función para calcular el MSE

$$MSE(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2$$

Aplicando esta formula obtenemos $(-0.1533-(-0.2864))^2 \approx 0.01772$

Hay que tener en cuenta que esto esta normalizado aun, realmente el valor de error real es muy grande, ahora procederemos a desnormalizar los resultados. Pero antes, vamos a ver como se actualizarían los parámetros para la teórica siguiente iteración.

Lo primero es realizar la derivada del MSE con respecto a la predicción

$$\frac{\partial MSE}{\partial x_{pred}} = 2(y_{target,std} - x_{pred,std}(1))(-1)$$

$$\approx -0{,}2662$$

Ahora derivamos la ecuación integrada con respecto a W y b

$$\frac{\partial x_{pred}}{\partial W} = -0{,}1862$$

$$\frac{\partial x_{pred}}{\partial b} = 1$$

Si aplicamos la regla de la cadena para encontrar la derivada del MSE con respecto a W

$$\frac{\partial MSE}{\partial W} = \frac{\partial MSE}{\partial x_{pred}}\frac{\partial x_{pred}}{\partial W} = (-0{,}2662)(-0{,}1862) \approx 0{,}0495$$

Con el mismo procedimiento podemos obtener $\frac{\partial MSE}{\partial b} \approx -0{,}2662$

Ahora tenemos que escoger una tasa de aprendizaje para este ejemplo, una tasa de aprendizaje menor va a tener mas resolución a la hora de encontrar los mínimos de la función MSE pero va a requerir mas iteraciones, para el modelo real tenemos η=0,001, para el ejemplo vamos a usar η=0,1

$$W_{nuevo} = W - \eta\frac{\partial MSE}{\partial W} \approx -0{,}00395$$

$$b_{nuevo} = b - \eta\frac{\partial MSE}{\partial b} \approx -0{,}07338$$

La misma ecuación de la normalización se puede utilizar para desnormalizar los valores, si realizamos esto con nuestros resultados obtenemos un output desnormalizado de 26362 MW aproximadamente.

Estos pasos son los que se repiten iteración tras iteración en una ecuación diferencial considerablemente más compleja, hasta que se encuentran los pesos óptimos para minimizar el MSE.

6-Las cuatro piezas de código

Ahora voy a brevemente explicar que hace cada una de las 4 piezas de código que se encuentran en el anexo. Para comenzar, código 1. Esta pieza de código simplemente se dedica a limpiar los datos que he descargado de la red eléctrica española, además, los reduce de un dato cada 5 minutos a un dato cada hora, y transforma los datos a la forma que hemos tratado antes, convirtiendo la fecha a senos y cosenos y limpiando las columnas innecesarias. Respecto a la pieza de código 2, esta se encarga de entrenar el modelo. Aquí es donde se utilizan los datos para entrenar el modelo e iterar los pesos hasta su forma final, además de comprobar la solución con el teorema de Picard. El código 3 se encarga de utilizar el modelo para realizar predicciones y obtener gráficos y métricas que nos permitan medir su validez y el código 4 simplemente se encarga de verificar que todas las otras piezas de código hayan funcionado efectivamente.

7-Entrenamiento y resultados

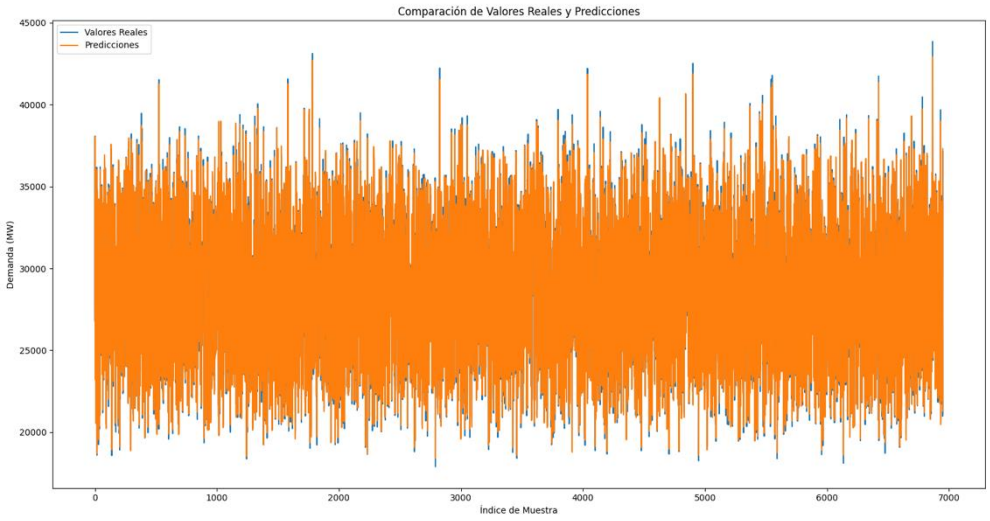
Podemos entonces ejecutar el código y entrenar el modelo. Tras alrededor de 6 horas, llegamos a un modelo con las siguientes métricas.

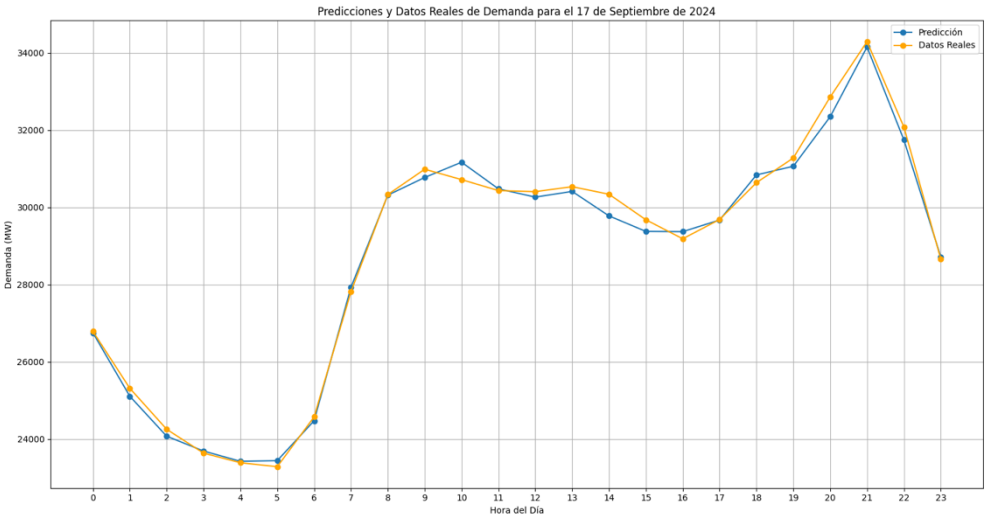
Mean Squared Error (MSE): 103,0861562MW

El modelo predijo correctamente dentro de un margen de ±2% en el 94.65% de los casos.

El modelo predijo correctamente dentro de un margen de ±10% en el 99.99994% de los casos.

Estos resultados son muy interesantes, el modelo no solo ha caído dentro del margen de 10% estándar en la industria 1, sino que ha tenido un error cuadrado promedio de 103,08 MW. Muy por debajo de los 3000MW de error que constituirían un 10% de 30000MW, una demanda aproximada a la media diaria. Pero es posible que el modelo este, por ejemplo, prediciendo muy precisamente a ciertas horas del día e incorrectamente a otras, y la media del error parezca impresionante, por esto, vamos a utilizar la pieza de código 3 para poner estos resultados en gráficas y representar un día aleatorio.





Tratemos los gráficos en orden. El primer gráfico nos enseña el valor real contra el predicho para las más de 7000 horas de las que tenemos datos. Como podemos observar, los valores predichos contra los valores reales son muy similares, no parece haber ninguna disparidad periódica lo que significa que nuestra idea de utilizar senos y cosenos para las características temporales ha sido preciso, podemos ver que el modelo puede llegar a tener problemas en los casos más extremos tanto con los datos muy elevados como los datos anormalmente bajos, viendo que el modelo infra predice sobre los datos excepcionalmente grandes y sobre predice sobre los datos anormalmente bajos. Esto es normal pues es imposible que el modelo espere estos datos anormales sin tener un contexto de que está ocurriendo ese día a esa hora. Por ejemplo, cuando hay un evento deportivo de gran escala, sea por ejemplo la apertura de las olimpiadas, la demanda eléctrica va a aumentar pues mucha gente va a querer televisarlo.

Respecto al segundo gráfico, podemos ver en detalle un día concreto, en este caso el 17 de septiembre de 2024. El modelo ha sido realmente preciso al predecir el consumo para este día estándar, probando como, aunque no tenga un contexto externo para poder adaptarse a las anormalidades, es muy preciso para predecir un día cotidiano.

8-Conclusiones

Tras haber validado el modelo contra los datos reales, podemos extraer varias conclusiones. Comenzando, podemos decir que mezclar las técnicas de modelado matemático tradicional con las redes neuronales de los LSTM para crear las neural ODE ha resultado en un modelo que ha podido predecir los datos con una tasa de éxito dentro de los márgenes estándar para la industria aproximadamente el 100% de las veces. Mezclando los beneficios de aplicar un modelaje tradicional con una red neuronal para gestionar mejor las tendencias a corto plazo mientras que las tendencias a largo plazo son gestionadas por la ecuación diferencial combina lo mejor de ambas áreas para dar un resultado satisfactorio.

La decisión de utilizar aumentación para este modelo ha sido precisa. Desde un punto de vista puramente computacional, el modelo necesita tener dimensiones vacías donde operar las mezclas entre parámetros, es por esto por lo que esta técnica es tan útil, ya que permite al modelo utilizar dimensiones extra para crear relaciones entre los diferentes inputs.

Las predicciones del modelo son sorprendentemente estables. Si bien es cierto de que hemos aplicado el teorema de Picard para descartar soluciones que tiendan al infinito, la naturaleza de las redes neuronales que son subyacentes a estas ecuaciones hace que sea común saltos aleatorios en las predicciones, cosa que aquí, al menos regularmente, no ocurre.

Podríamos entonces decir, y para responder a la pregunta, que las ecuaciones diferenciales neuronales ordinarias pueden aplicarse a los sistemas de predicción ya existente como una nueva herramienta de parcial o total control supervisado, estos sistemas son igual o más efectivos que las soluciones tradicionales a este problema, y completamente capaces de mantener el voltaje de la red eléctrica estable. Podemos observar según los gráficos obtenidos como el voltaje se mantendría estable. No solo esto, sino que seria extremadamente simple integrar un sistema de este estilo con las herramientas ya existentes, el concepto desde un punto de vista matemático es complejo, pero desde un punto de vista computacional es código bastante básico para alguien que conozca como

crear modelos de inteligencia artificial. Es cierto que una versión final requeriría ciertas mejoras, como ahora trataremos, pero conceptualmente es perfectamente viable implementar esta solución.

9-Limitaciones, mejoras y ampliaciones

Si bien esta implementación ha sido exitosa, es cierto que no ha sido sin sus compromisos y posibles mejoras. Para comenzar, solo hemos tomado un dato por cada hora cuando la resolución de los datos oficiales que ofrece la red eléctrica española es de 1 dato cada 5 minutos, lo que implicaría un nivel de detalle 12 veces mayor. Este volumen de datos hubiera sido demasiado para mis capacidades computacionales, y, por encima de esto, no es realmente necesario para un ejercicio demostrativo como este que lo que busca es probar la idea y estudiar su viabilidad. Pero para una implementación a un sistema real se tendría que utilizar datos con la mayor resolución posible.

Otra limitación es la antigüedad de nuestros datos, si bien tenemos los datos más recientes, hay registros de datos como los que hemos utilizado desde hace considerablemente más tiempo que nuestra fecha de corte, esto es por razones similares a la primera limitación planteada pero aun así es importante anotar este detalle.

Respecto a posibles mejoras. Una forma de haber mejorado este modelo hubiera sido utilizando ecuaciones diferenciales parciales. Las ecuaciones diferenciales parciales neuronales son un concepto aun más novedoso que las ecuaciones que hemos tratado en este informe, pero debido a su complejidad no solo computacional sino matemática, no eran precisas para este informe. Es un campo de investigación relacionado muy interesante pero por las restricciones planteadas y el nivel de profundidad y rigor matemático que es necesario, no es posible tratar este tema.

10-Bibliografía

Cui, W., Zhang, H., Chu, H., Hu, P., & Li, Y. (2023, junio). *On robustness of neural ODEs image classifiers*. ScienceDirect. https://www.sciencedirect.com/science/article/abs/pii/S0020025523003444

Gutermuth, D. (s.f.). *Picard's Existence and Uniqueness Theorem*. https://ptolemy.berkeley.edu/projects/embedded/eecsx44/lectures/Spring2013/Picard.pdf

Kidger, P. (2022, 4 de febrero). *On Neural Differential Equations*. arXiv.org. https://arxiv.org/abs/2202.02435

Lee, K., & J. Parish, E. (2021, 15 de septiembre). *Parameterized neural ordinary differential equations: applications to computational physics problems*. The Royal Society. https://royalsocietypublishing.org/doi/10.1098/rspa.2021.0162

Melton, R. (2019, 3 de agosto). *Initial-Value Problems | Calculus II*. Lumen Learning – Simple Book Production. https://courses.lumenlearning.com/calculus2/chapter/initial-value-problems/

Paoletti, M., M Haut, J., Plaza, J., & Plaza, A. (2019, 12 de noviembre). *Neural Ordinary Differential Equations for Hyperspectral Image Classification*. ResearchGate. https://www.researchgate.net/publication/337071873_Neural_Ordinary_Differential_Equations_for_Hyperspectral_Image_Classification

Stewart, K. (2023, 22 de marzo). *Mean squared error (MSE) | Definition, Formula, Interpretation, & Facts | Britannica*. Encyclopedia Britannica. https://www.britannica.com/science/mean-squared-error

T. Q. Chen, R., Rubanova, Y., Bettencourt, J., & Duvenaud, D. (2018, 19 de junio). *Neural Ordinary Differential Equations*. arXiv.org. https://arxiv.org/abs/1806.07366

11-Anexo

Codigo 1

```
import os
import pandas as pd

# Función para limpiar las comas al final de las líneas
def limpiar_archivo(ruta_archivo):
```

```
    with open(ruta_archivo, 'r', encoding="ISO-8859-1") as file:
        lineas = file.readlines()

    # Eliminar la coma al final de cada línea si existe
    lineas_limpias = [linea.rstrip().rstrip(',') + "\n" for linea in lineas]

    # Guardar el archivo limpio temporalmente
    with open(ruta_archivo, 'w', encoding="ISO-8859-1") as file:
        file.writelines(lineas_limpias)

# Función para filtrar los datos del día correcto y eliminar registros fuera del rango de 00:00 a 23:55
def filtrar_por_dia_y_hora(df, dia):
    # Verificar que no haya encabezados adicionales
    df = df[df["Hora"] != "Hora"]

    # Filtrar las filas que coinciden con el día especificado en el nombre del archivo
    df = df[df["Hora"].str.startswith(dia)]

    # Filtrar las filas que tienen horas entre 00:00 y 23:55
    filtro = df["Hora"].str.slice(11, 16) >= "00:00"  # Seleccionar desde 00:00 en adelante
    filtro &= df["Hora"].str.slice(11, 16) <= "23:55"  # Seleccionar hasta 23:55

    # Aplicar el filtro y devolver el DataFrame filtrado
    return df[filtro]

# Directorio relativo donde están los archivos CSV
directorio_csv = os.path.join(os.getcwd(), "data")

# Lista para almacenar los dataframes
lista_datos = []

# Leer todos los archivos en el directorio
for archivo in sorted(os.listdir(directorio_csv)):
    if archivo.endswith(".csv") and "Custom-Report" in archivo:
        # Extraer el día desde el nombre del archivo, asumiendo que el formato es 'Custom-Report-YYYY-MM-DD...'
        dia = archivo.split("-")[2] + "-" + archivo.split("-")[3] + "-" + archivo.split("-")[4]

        # Limpiar el archivo antes de procesarlo
        ruta_archivo = os.path.join(directorio_csv, archivo)
        limpiar_archivo(ruta_archivo)
```

```
        # Cargar el CSV actual, saltando las dos primeras líneas y sin incluir
el encabezado
        try:
            datos_csv = pd.read_csv(ruta_archivo, skiprows=2, header=None,
encoding="ISO-8859-1")
        except Exception as e:
            print(f"Error al leer el archivo {archivo}: {e}")
            continue

        # Asignar nombres de columnas temporalmente para poder manipular los
datos
        datos_csv.columns = ["Hora", "Real", "Prevista", "Programada"]

        # Filtrar los datos para que solo incluyan el día correcto y las horas
entre 00:00 y 23:55
        datos_filtrados = filtrar_por_dia_y_hora(datos_csv, dia)

        # Añadir los datos filtrados a la lista
        lista_datos.append(datos_filtrados)

# Concatenar todos los dataframes en uno solo
if lista_datos:
    datos_combinados = pd.concat(lista_datos, ignore_index=True)

    # Añadir encabezado a los datos combinados solo una vez
    datos_combinados.columns = ["Hora", "Real", "Prevista", "Programada"]

    # Guardar el dataframe combinado en un nuevo archivo CSV con codificación
UTF-8
    datos_combinados.to_csv("seguimiento_demanda_combinado.csv", index=False,
encoding="utf-8")

    print("Archivos combinados y guardados correctamente en formato UTF-8.")
else:
    print("No se encontraron archivos CSV válidos.")
```

Codigo 2

```
import torch
import torch.nn as nn
import pandas as pd
import numpy as np
from torchdiffeq import odeint
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

```python
import joblib
from datetime import datetime

# Configuración del dispositivo (CPU o GPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# 1. Cargar los datos
data = pd.read_csv('data_transformed_filtered.csv')

# 2. Preprocesamiento de datos
# Seleccionar las columnas relevantes
# Vamos a ajustar 'Año_num' para que sea relativo al año mínimo en los datos
anio_minimo = data['Año_num'].min()
data['Año_relativo'] = data['Año_num'] - anio_minimo

features = ['Real', 'Hora_sin', 'Hora_cos', 'Mes_sin', 'Mes_cos',
'Año_relativo']
target = 'Prevista'

X = data[features].values
y = data[target].values.reshape(-1, 1)  # Aseguramos que y sea una matriz
columna

# Dividir en conjuntos de entrenamiento y prueba antes de la normalización
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Normalización de los datos
scaler_X = StandardScaler()
scaler_y = StandardScaler()

# Ajustar el escalador en el conjunto de entrenamiento y transformar
X_train = scaler_X.fit_transform(X_train)
y_train = scaler_y.fit_transform(y_train)

# Usar el mismo escalador para transformar el conjunto de prueba
X_test = scaler_X.transform(X_test)
y_test = scaler_y.transform(y_test)

# Convertir a tensores de PyTorch
X_train_tensor = torch.tensor(X_train, dtype=torch.float32).to(device)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32).to(device)

X_test_tensor = torch.tensor(X_test, dtype=torch.float32).to(device)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32).to(device)
```

```python
# 3. Definir el modelo de Neural ODE aumentada

# Definir la función de aumento
class Augmenter(nn.Module):
    def __init__(self, augment_dim):
        super(Augmenter, self).__init__()
        self.augment_dim = augment_dim

    def forward(self, x):
        # Aumentar la dimensionalidad agregando ceros
        batch_size = x.size(0)
        aug = torch.zeros(batch_size, self.augment_dim).to(x.device)
        x_aug = torch.cat([x, aug], dim=1)
        return x_aug

# Definir la función ODE
class ODEFunc(nn.Module):
    def __init__(self, input_dim, hidden_dim):
        super(ODEFunc, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.Tanh(),
            nn.Linear(hidden_dim, input_dim),
        )

        for m in self.net.modules():
            if isinstance(m, nn.Linear):
                nn.init.normal_(m.weight, mean=0, std=0.1)
                nn.init.constant_(m.bias, val=0)

    def forward(self, t, x):
        return self.net(x)

# Definir el modelo completo
class NeuralODE(nn.Module):
    def __init__(self, input_dim, augment_dim, hidden_dim, output_dim):
        super(NeuralODE, self).__init__()
        self.augmenter = Augmenter(augment_dim)
        self.odefunc = ODEFunc(input_dim + augment_dim, hidden_dim)
        self.linear = nn.Linear(input_dim + augment_dim, output_dim)

    def forward(self, x):
        x_aug = self.augmenter(x)
        # Resolver la ODE desde t=0 hasta t=1
        t = torch.tensor([0., 1.]).to(x.device)
        out = odeint(self.odefunc, x_aug, t, method='rk4')
        # Tomar el último paso de tiempo
```

```
        out = out[1]
        out = self.linear(out)
        return out.squeeze()

# Parámetros del modelo
input_dim = X_train.shape[1]
augment_dim = 2  # Puedes ajustar este valor
hidden_dim = 50
output_dim = 1  # Ya que estamos prediciendo una variable continua

model = NeuralODE(input_dim, augment_dim, hidden_dim, output_dim).to(device)

# 4. Definir la función de pérdida y el optimizador
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# 5. Entrenar el modelo
num_epochs = 1000  # Puedes ajustar el número de épocas

for epoch in range(num_epochs):
    model.train()
    optimizer.zero_grad()
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor.squeeze())
    loss.backward()
    optimizer.step()

    if (epoch+1) % 50 == 0:
        # Evaluar en el conjunto de prueba
        model.eval()
        with torch.no_grad():
            test_outputs = model(X_test_tensor)
            test_loss = criterion(test_outputs, y_test_tensor.squeeze())
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}, Test
Loss: {test_loss.item():.4f}')

# 6. Predicción final
model.eval()
with torch.no_grad():
    predictions_scaled = model(X_test_tensor)

# Invertir la normalización para obtener los valores originales
predictions = predictions_scaled.cpu().numpy()
predictions = scaler_y.inverse_transform(predictions.reshape(-1, 1)).flatten()

y_test_actual = y_test.flatten()
```

```
y_test_actual =
scaler_y.inverse_transform(y_test_tensor.cpu().numpy()).flatten()

# 7. Visualización de resultados
# Gráfico 1: Comparación de valores reales y predicciones en el conjunto de
prueba
plt.figure(figsize=(10,6))
plt.plot(y_test_actual, label='Valores Reales')
plt.plot(predictions, label='Predicciones')
plt.legend()
plt.title('Comparación de Valores Reales y Predicciones en el Conjunto de
Prueba')
plt.xlabel('Índice de Muestra')
plt.ylabel('Demanda (MW)')
plt.show()

# 8. Evaluación del modelo

# Calcular el error absoluto y porcentual
error_absoluto = np.abs(predictions - y_test_actual)
error_porcentual = (error_absoluto / y_test_actual) * 100

# Manejar posibles divisiones por cero en y_test_actual
error_porcentual = np.where(y_test_actual != 0, error_porcentual, 0)

# Calcular el MAPE y la precisión
mape = np.mean(error_porcentual)
print(f"\nError Porcentual Medio Absoluto (MAPE): {mape:.2f}%")

precision = 100 - mape
print(f"Precisión del modelo (1 - MAPE): {precision:.2f}%")

# Métricas adicionales
mse = mean_squared_error(y_test_actual, predictions)
mae = mean_absolute_error(y_test_actual, predictions)
r2 = r2_score(y_test_actual, predictions)

print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"Coeficiente de Determinación (R²): {r2:.4f}")

# **Guardando los escaladores**
# Guardar los escaladores en archivos
joblib.dump(scaler_X, 'scaler_X.save')
joblib.dump(scaler_y, 'scaler_y.save')
```

Codigo 3

```
import pandas as pd
import numpy as np
from datetime import datetime

# Datos proporcionados para el 17 de septiembre de 2024
data_17sep2024 = """
Real,Prevista,Hora_sin,Hora_cos,Mes_sin,Mes_cos,Año_num
26679,26783,0.0,1.0,-1.0,-1.8369701987210297e-16,2024
25066,25312,0.25881904510252074,0.9659258262890683,-1.0,-1.8369701987210297e-
16,2024
24056,24250,0.49999999999999994,0.8660254037844387,-1.0,-1.8369701987210297e-
16,2024
23688,23632,0.7071067811865476,0.7071067811865476,-1.0,-1.8369701987210297e-
16,2024
23430,23383,0.8660254037844386,0.5000000000000001,-1.0,-1.8369701987210297e-
16,2024
23443,23280,0.9659258262890683,0.25881904510252074,-1.0,-1.8369701987210297e-
16,2024
24449,24570,1.0,6.123233995736766e-17,-1.0,-1.8369701987210297e-16,2024
27830,27812,0.9659258262890683,-0.25881904510252063,-1.0,-1.8369701987210297e-
16,2024
30217,30329,0.8660254037844387,-0.4999999999999998,-1.0,-1.8369701987210297e-
16,2024
30705,30985,0.7071067811865476,-0.7071067811865475,-1.0,-1.8369701987210297e-
16,2024
31133,30716,0.49999999999999994,-0.8660254037844387,-1.0,-1.8369701987210297e-
16,2024
30437,30432,0.258819045102521,-0.9659258262890682,-1.0,-1.8369701987210297e-
16,2024
30229,30404,1.2246467991473532e-16,-1.0,-1.0,-1.8369701987210297e-16,2024
30390,30533,-0.2588190451025208,-0.9659258262890683,-1.0,-1.8369701987210297e-
16,2024
29757,30337,-0.4999999999999997,-0.8660254037844388,-1.0,-1.8369701987210297e-
16,2024
29367,29676,-0.7071067811865471,-0.7071067811865479,-1.0,-1.8369701987210297e-
16,2024
29373,29184,-0.8660254037844385,-0.5000000000000004,-1.0,-1.8369701987210297e-
16,2024
29682,29683,-0.9659258262890683,-0.25881904510252063,-1.0,-
1.8369701987210297e-16,2024
30864,30637,-1.0,-1.8369701987210297e-16,-1.0,-1.8369701987210297e-16,2024
31085,31276,-0.9659258262890684,0.2588190451025203,-1.0,-1.8369701987210297e-
16,2024
32406,32855,-0.8660254037844386,0.5000000000000001,-1.0,-1.8369701987210297e-
16,2024
```

```
34338,34290,-0.7071067811865477,0.7071067811865474,-1.0,-1.8369701987210297e-16,2024
31733,32077,-0.5000000000000004,0.8660254037844384,-1.0,-1.8369701987210297e-16,2024
28636,28660,-0.25881904510252157,0.9659258262890681,-1.0,-1.8369701987210297e-16,2024
"""

# Leer los datos en un DataFrame
from io import StringIO

df_17sep2024 = pd.read_csv(StringIO(data_17sep2024))

# Añadir la columna 'Año_relativo'
df_17sep2024['Año_relativo'] = df_17sep2024['Año_num'] - anio_minimo

# Crear la columna 'Fecha'
df_17sep2024['Hora'] = range(24)  # Asignar las horas de 0 a 23
df_17sep2024['Fecha'] = pd.to_datetime({
    'year': df_17sep2024['Año_num'],
    'month': 9,  # Septiembre
    'day': 17,
    'hour': df_17sep2024['Hora']
})

# Seleccionar las características necesarias
features_new = ['Real', 'Hora_sin', 'Hora_cos', 'Mes_sin', 'Mes_cos',
'Año_relativo']
X_new = df_17sep2024[features_new].values

# Normalizar las características usando el escalador entrenado
X_new_scaled = scaler_X.transform(X_new)

# Convertir a tensor
X_new_tensor = torch.tensor(X_new_scaled, dtype=torch.float32).to(device)

# Realizar las predicciones
model.eval()
with torch.no_grad():
    predictions_scaled = model(X_new_tensor)

# Invertir la normalización
predictions = predictions_scaled.cpu().numpy()
predictions = scaler_y.inverse_transform(predictions.reshape(-1, 1)).flatten()

# Datos reales
```

```
real_values = df_17sep2024['Prevista'].values  # Asumiendo que 'Prevista' son los valores reales

# Horas para el gráfico
horas = df_17sep2024['Hora'].values

# Gráfico: Predicciones y datos reales para el 17 de septiembre de 2024
plt.figure(figsize=(10,6))
plt.plot(horas, predictions, marker='o', label='Predicción')
plt.plot(horas, real_values, marker='o', label='Datos Reales', color='orange')
plt.title('Predicciones y Datos Reales de Demanda para el 17 de Septiembre de 2024')
plt.xlabel('Hora del Día')
plt.ylabel('Demanda (MW)')
plt.xticks(horas)
plt.grid(True)
plt.legend()
plt.show()

# Imprimir las predicciones y los valores reales para cada hora
for hour, pred, real in zip(horas, predictions, real_values):
    print(f"Hora {hour:02d}:00 - Predicción: {pred:.2f} MW - Real: {real:.2f} MW")
```

Codigo 4

```
import os
import pandas as pd

# Función para limpiar las comas al final de las líneas
def limpiar_archivo(ruta_archivo):
    with open(ruta_archivo, 'r', encoding="ISO-8859-1") as file:
        lineas = file.readlines()

    # Eliminar la coma al final de cada línea si existe
    lineas_limpias = [linea.rstrip().rstrip(',') + "\n" for linea in lineas]

    # Guardar el archivo limpio temporalmente
    with open(ruta_archivo, 'w', encoding="ISO-8859-1") as file:
        file.writelines(lineas_limpias)

# Función para filtrar los datos del día correcto y eliminar registros fuera del rango de 00:00 a 23:55
def filtrar_por_dia_y_hora(df, dia):
    # Verificar que no haya encabezados adicionales
    df = df[df["Hora"] != "Hora"]
```

```
    # Filtrar las filas que coinciden con el día especificado en el nombre del
archivo
    df = df[df["Hora"].str.startswith(dia)]

    # Filtrar las filas que tienen horas entre 00:00 y 23:55
    filtro = df["Hora"].str.slice(11, 16) >= "00:00"  # Seleccionar desde
00:00 en adelante
    filtro &= df["Hora"].str.slice(11, 16) <= "23:55"  # Seleccionar hasta
23:55

    # Aplicar el filtro y devolver el DataFrame filtrado
    return df[filtro]

# Directorio relativo donde están los archivos CSV
directorio_csv = os.path.join(os.getcwd(), "data")

# Lista para almacenar los dataframes
lista_datos = []

# Leer todos los archivos en el directorio
for archivo in sorted(os.listdir(directorio_csv)):
    if archivo.endswith(".csv") and "Custom-Report" in archivo:
        # Extraer el día desde el nombre del archivo, asumiendo que el formato
es 'Custom-Report-YYYY-MM-DD...'
        dia = archivo.split("-")[2] + "-" + archivo.split("-")[3] + "-" +
archivo.split("-")[4]

        # Limpiar el archivo antes de procesarlo
        ruta_archivo = os.path.join(directorio_csv, archivo)
        limpiar_archivo(ruta_archivo)

        # Cargar el CSV actual, saltando las dos primeras líneas y sin incluir
el encabezado
        try:
            datos_csv = pd.read_csv(ruta_archivo, skiprows=2, header=None,
encoding="ISO-8859-1")
        except Exception as e:
            print(f"Error al leer el archivo {archivo}: {e}")
            continue

        # Asignar nombres de columnas temporalmente para poder manipular los
datos
        datos_csv.columns = ["Hora", "Real", "Prevista", "Programada"]

        # Filtrar los datos para que solo incluyan el día correcto y las horas
entre 00:00 y 23:55
```

```
        datos_filtrados = filtrar_por_dia_y_hora(datos_csv, dia)

        # Añadir los datos filtrados a la lista
        lista_datos.append(datos_filtrados)

# Concatenar todos los dataframes en uno solo
if lista_datos:
    datos_combinados = pd.concat(lista_datos, ignore_index=True)

    # Añadir encabezado a los datos combinados solo una vez
    datos_combinados.columns = ["Hora", "Real", "Prevista", "Programada"]

    # Guardar el dataframe combinado en un nuevo archivo CSV con codificación
UTF-8
    datos_combinados.to_csv("seguimiento_demanda_combinado.csv", index=False,
encoding="utf-8")

    print("Archivos combinados y guardados correctamente en formato UTF-8.")
else:
    print("No se encontraron archivos CSV válidos.")
```