Trabajo de Fin de Grado

del

Grado en Física

de la

Universidad de Zaragoza

Modelización de sistemas térmicos en experimentos criogénicos usando redes neuronales

Trabajo realizado por

Javier Colominas Villalba

a lo largo del segundo semestre del curso académico 2021-2022.

Trabajo dirigido por:

Carlos Pobes Aranda

Sergio Gutiérrez Rodrigo

Memoria entregada el 12 de septiembre de 2022.

Índice

1. Introducción

En la actualidad, las tecnologías de bajas temperaturas son indispensables en una gran variedad de campos. Fenómenos como la superconductividad, la superfluidez o el efecto Josephson encuentran múltiples aplicaciones en la obtención de campos magnéticos elevados por medio de imanes superconductores, para su uso en imagen por resonancia magnética o en aceleradores de partículas; el estudio de los fenómenos cuánticos a escalas macroscópicas, como la condensación de Bose-Einstein; y la fabricación de magnetómetros de alta precisión, los conocidos como SQUIDs, que son ampliamente utilizados en exploración geofísica y en biomagnetismo [1].

Una de las tecnologías que permite alcanzar temperaturas inferiores a 10 mK son los refrigeradores de dilución, que basan su funcionamiento en las transiciones de fase que experimenta una mezcla de ^{3}He y ^{4}He. Una vez alcanzada su temperatura base, puede emplearse un calefactor (*heater*) para, variando su potencia, ajustar la temperatura del baño a cualquier valor en una región situada por encima de dicha temperatura. Para ajustar esta potencia de manera que el baño se lleve a la temperatura deseada y se mantenga en ella, pueden diseñarse distintas estrategias de realimentación, siendo una de las más sencillas y efectivas el control PID. Este control, instalado por el fabricante del dispositivo experimental, utiliza la información sobre la temperatura del baño a lo largo de un intervalo de tiempo para decidir qué valor de potencia es conveniente suministrar en un momento dado.

Además de las aplicaciones ya mencionadas, las tecnologías de bajas temperaturas permiten fabricar bolómetros y calorímetros de muy alta precisión, basados en los sensores de transición superconductora (en inglés, transition-edge sensors, comúnmente abreviado como TES). Estos sensores presentan una sensibilidad muy elevada en un rango muy estrecho de temperatura, lo que obliga a situar su punto de operación con gran precisión colocándolos en un baño térmico, ajustando la temperatura de este a la temperatura de transición de fase superconductora del TES (en torno a 0.1 K) y manteniéndola frente a posibles cambios en las condiciones ambientales.

En los experimentos basados en esta tecnología, surgen una serie de inconveniencias a tener en cuenta y, dentro de lo posible, tratar de solucionar. La primera de ellas es que el proceso de estabilización de la temperatura del baño a la temperatura requerida por el sensor suele ser relativamente largo, superior en ocasiones a la media hora; esto supone un problema cuando se quieren realizar medidas I(V) y de impedancia compleja para caracterizar el dispositivo a distintas temperaturas, pues entre medida y medida deberá esperarse un largo tiempo hasta que el baño alcance la nueva temperatura y se mantenga suficientemente estable en torno a ella [2]. Por ello, es de interés optimizar la estrategia de control utilizada con el objetivo de minimizar este tiempo. Para el control PID, al quedar este determinado por tres parámetros, puede plantearse el problema como la obtención, dadas la temperatura inicial del baño y la temperatura deseada, de los valores para dichos parámetros que proporcionan una respuesta lo más parecida posible a la deseada idealmente. Un posible enfoque para intentar solucionar este problema sería realizar una serie de experimentos de cambio de una temperatura a otra variando los parámetros del sistema y, a continuación, utilizar alguna herramienta de inteligencia artificial (una red neuronal) que, con estos datos, aprenda a cuantificar la bondad de la respuesta a partir de los parámetros del sistema; así, dadas las temperaturas inicial y final, sería relativamente sencillo hallar los valores de los parámetros del PID que proporcionen una respuesta óptima, sin más que minimizar numéricamente la función que define el parámetro de optimalidad respecto a los parámetros del PID, manteniendo las temperaturas inicial y final constantes. Sin embargo, esta

estrategia presenta un problema importante: la cantidad de experimentos necesarios para que la red pueda aprender la función del parámetro de optimalidad es enorme, y el tiempo asociado para la realización de los experimentos, inadmisible. Por tanto, una estrategia mucho más práctica, aunque previsiblemente de peor resultado, consiste en modelizar primero el sistema y simular computacionalmente los experimentos, tomando abundantes datos de las simulaciones para entrenar la red o, directamente, optimizando numéricamente el modelo para unas temperaturas inicial y final dadas. No obstante, obtener un modelo tan fiable para el sistema como para poder optar por esta vía parece complicado, debido a la complejidad del sistema con el que estamos tratando, en especial en las proximidades de la temperatura base. Por ello, la opción más razonable es combinar adecuadamente los dos tipos de datos (experimentales y simulados) para entrenar la red, y a continuación proseguir como se ha explicado anteriormente. Este será, en consecuencia, el objeto del trabajo.

2. Objetivos

Así pues, como objetivos perseguidos en este Trabajo de Fin de Grado pueden diferenciarse los siguientes:

- Modelizar el refrigerador simulando el control PID de la temperatura, y desarrollar un programa que permita hallar su solución numérica para unos datos iniciales dados y unos parámetros del PID prefijados.
- Entrenar una red neuronal combinando datos generados por el modelo (datos aproximados, disponibles en gran cantidad) y datos medidos experimentalmente (datos "exactos" salvo errores de medida, disponibles en mucho menor número) para predecir la bondad de la respuesta en función de los parámetros del controlador.
- Desarrollar un programa que calcule, para cada par de temperaturas (T_{in}, T_f), los parámetros del control PID que optimizan la respuesta térmica del sensor, según cada uno de los criterios considerados.

3. Fundamentos

3.1. Sensores de transición superconductora (TES)

Los sensores de transición superconductora (TES) están formados por una lámina superconductora situada en la estrecha región de temperaturas que separa el estado normal del superconductor. En esta región, la resistencia eléctrica aumenta desde 0 hasta su valor normal de manera muy brusca, resultando en una sensibilidad enorme frente a pequeños cambios de temperatura. Concretamente, la sensibilidad logarítmica, $\frac{d\log R}{d\log T}$, es hasta dos órdenes de magnitud superior a la de los termistores semiconductores utilizados en calorímetros criogénicos. Entre las ventajas derivadas de esto, destaca la posibilidad de fabricar detectores térmicos con una mayor rapidez de respuesta, mayor capacidad calorífica y mejor resolución que los basados en dichos termistores. Sin embargo, la implementación práctica de estos sensores es tecnológicamente más complicada, debido a la inestabilidad y baja energía de saturación asociadas a la brusquedad de la transición y al pequeño valor de la resistencia normal, que requiere el acoplamiento a un

amplificador de corriente SQUID en lugar de un FET convencional (o bien el empleo de circuitos complejos). Además, para evitar el desplazamiento del punto de operación por efecto Joule, debe situarse en él por medio de la aplicación de un voltaje (que resulta en una realimentación negativa), no de una corriente (que provocaría una realimentación positiva).

Entre las aplicaciones de los sensores TES en la actualidad, destaca su uso en bolómetros, para medir potencia, y en calorímetros, para medir pulsos de energía. Estos instrumentos permiten realizar medidas en un amplio rango del espectro electromagnético, desde longitudes de onda del orden del mm hasta rayos γ. También permiten estudiar partículas de interacción débil, materia oscura, supersimetría, biomoléculas y la composición química de materiales, así como diversos aspectos relacionados con la información cuántica. Los calorímetros basados en TES permiten detectar fotones individuales hasta el infrarrojo cercano, con perspectivas de llegar también al lejano.

Para más información acerca del funcionamiento de estos sensores, consúltese [3].

3.2. Refrigeradores de dilución

Los refrigeradores de dilución son ampliamente utilizados en la actualidad, ya que permiten alcanzar temperaturas inferiores a 4 mK y, en el caso de los más modernos, sin necesidad de emplear helio líquido de manera externa. Algunas de sus aplicaciones más prometedoras son la refrigeración de detectores de ondas gravitacionales, la fabricación de ordenadores cuánticos, la búsqueda de la partícula de Majorana y el estudio de muchas otras propiedades cuánticas.

Su funcionamiento se basa en el empleo de una mezcla de ^{3}He y ^{4}He. El núcleo de ^{4}He es un bosón, de modo que este isótopo presenta una transición de fase superfluida a 2.17 K. Por su parte, el núcleo de ^{3}He es un fermión, por lo que esta transición de fase ocurre a temperaturas mucho menores. Así, para una mezcla de ^{3}He y ^{4}He, la temperatura de la transición dependerá de la fracción de ^{3}He. Enfriando aún más una mezcla de ambos isótopos en estado superfluido, la mezcla se separa en dos fases: una superior de ^{3}He puro y una inferior enriquecida en ^{4}He, pero con un 6.4 % de ^{3}He (a 0 K). Esta solubilidad finita del ^{3}He en el ^{4}He se debe a que la energía de enlace de un átomo de ^{3}He en una masa de ^{4}He es mayor que en una de ^{3}He, pero al tratarse de fermiones se van ocupando sucesivamente estados de mayor energía, hasta llegar a la concentración (6.4 %) a la cual el siguiente estado no ocupado tiene una energía igual a la de enlace en el ^{3}He (es decir, su calor latente de vaporización), de modo que se alcanza el equilibrio termodinámico.

Para aprovechar las características de este sistema, el refrigerador dispone de dos cámaras: la de mezcla (*mixing chamber*) y el destilador (*still*), entre las cuales circula el helio en un circuito cerrado. La primera contiene la mezcla de ^{3}He y ^{4}He, con las dos fases descritas anteriormente. El *still*, por su parte, utiliza la diferencia en la presión de vapor del ^{3}He y del ^{4}He para destilar el primero de la fase rica en ^{4}He. Esto provoca el paso de átomos de ^{3}He desde la fase rica en él hasta la de ^{4}He, con la resultante potencia de refrigeración debida a la diferencia de entalpía del ^{3}He en ambas fases, cuya expresión es:

$$\dot{Q}_{\text{refr}} = -(95T_{\text{mc}}^2 - 11T_{\text{ex}}^2)\dot{n}_3 \tag{1}$$

donde T_{mc} es la temperatura de la *mixing chamber* en K, T_{ex} es la del último intercambiador de calor por el que pasa el ^{3}He en su regreso a ella, también en K, y $\dot{n}_3$ es el flujo de ^{3}He en mol/s. Esta dependencia con T^2 es la principal ventaja frente a los refrigeradores por evaporación, cuya

dependencia de la potencia de refrigeración con $e^{-\frac{1}{T}}$ supone una disminución mucho más rápida de esta con la temperatura, y, en consecuencia, no permite alcanzar temperaturas tan bajas. Para ver el diseño de estos refrigeradores en detalle, consúltese [4].

3.3. Control PID

Para mantener una situación de equilibrio en un sistema frente a cambios del entorno, es necesario emplear técnicas de realimentación. Estas modifican la propiedad de interés del sistema en cuestión proporcionándole una determinada señal en función del estado actual de esta propiedad, con el objeto de oponerse a cambios en dicho estado respecto al que se desea.

El control PID es el algoritmo de control más utilizado en ingeniería. Destaca su sencillez, pues depende solo de tres parámetros, así como su enorme versatilidad, por su amplio rango de aplicación. Denotando por $e(t)$ la diferencia entre el valor deseado v_s de la propiedad y su valor $v(t)$ en el tiempo t (es decir, $e = v_s - v(t)$) y por $u(t)$ el de la señal proporcionada al sistema por el controlador en el tiempo t, la expresión de esta última viene dada por:

$$u(t) = K\left(e(t) + \frac{1}{T_I}\int_0^t e(s)ds + T_D\frac{de(t)}{dt}\right) \tag{2}$$

Se entiende que una señal u positiva (respectivamente, negativa) induce a un aumento (respectivamente, decrecimiento) en el valor de v. Notar que, para que el control según la expresión anterior sea efectivo, los tres parámetros K, T_I y T_D han de tomar valores no negativos (y $T_I \neq 0$, aunque puede valer ∞). Además, para asegurar la estabilidad del sistema en torno a la posición de equilibrio y para lograr que se llegue a ella lo antes posible partiendo de un estado diferente, es preciso hacer una buena elección de estos valores. La manera de conseguir esto en un sistema particular será, como ya se ha dicho, uno de los objetivos fundamentales de este trabajo. Para más información sobre este tipo de control, consultar [5].

3.4. Redes neuronales e inteligencia artificial

La inteligencia artificial (IA o AI, por sus siglas en inglés) es uno de los campos más recientes de la ciencia y la tecnología. Podría definirse como el estudio y desarrollo de sistemas capaces de realizar trabajos que requieren inteligencia en alguna de sus múltiples formas; es decir, que (a priori) no son tareas fácilmente mecanizables, sino que requieren, para su efectiva conclusión, un proceso análogo al, o más general que el, pensamiento humano. Estas tareas incluyen el reconocimiento de imágenes, el juego al ajedrez, la demostración de teoremas matemáticos y la comprensión del lenguaje natural, entre otras muchas cosas. El objetivo final de esta ciencia es, por tanto, diseñar algoritmos efectivos para la realización de estas tareas [6].

Uno de los sistemas más utilizados en IA por su simplicidad y su enorme versatilidad son las redes neuronales. Estas permiten, con un diseño adecuado, reconstruir funciones desconocidas a partir de una serie de datos de entrada-salida, lo cual puede utilizarse, por ejemplo, para clasificar imágenes en distintas categorías. Así pues, constituyen una forma de aprendizaje supervisado. En su forma más básica, estas redes consisten en varias capas de unidades, denominadas neuronas, que, en analogía a las neuronas de los seres vivos, tienen diversas conexiones entre sí. Concretamente, cada neurona (salvo las de la primera capa) está conectada a todas las de la capa anterior, y recibe una señal de entrada (*input*) que es un número real, obtenido como combinación lineal de las señales de salida (*output*) de las neuronas de la capa anterior, con unos

ciertos pesos w_{ij}. Entonces, la neurona genera una señal de salida sumando un cierto valor fijo (*bias*) a su señal de entrada recibida y aplicando una función no lineal prefijada (por ejemplo, una sigmoide) al resultado. La neurona estará conectada a todas las de la siguiente capa a través de unos ciertos pesos, de modo que su señal de salida contribuirá a las señales de entrada de todas ellas. De esta forma, la red neuronal transforma una determinada señal de entrada (los valores asignados a las neuronas de la primera capa) en otra de salida (el conjunto de valores de las neuronas de la última capa después de llevar a cabo el proceso descrito anteriormente, desde la segunda capa hasta la última). La idea es que, con el número apropiado de capas y de neuronas en cada capa y con los valores adecuados para los parámetros (pesos y *biases*), es posible aproximar cualquier función tan bien como se quiera; y, con una estructura de red fija (nº de capas y de neuronas en cada capa), dando valores adecuados a los parámetros, suele ser posible obtener una aproximación razonablemente buena para una función dada.

Simplificadamente, la manera de proceder para reconstruir una función es la siguiente: en primer lugar, se obtiene un conjunto de datos; es decir, se genera una serie de valores para la(s) variable(s) de la función y, para cada uno de ellos, se calcula el valor asociado de la función. A continuación, se define una función de coste, que ha de ser diferenciable, que dependerá de los parámetros de la red y de los datos generados y que será menor cuanto más se aproxime el *output* de la red para cada valor de la variable del conjunto de datos al valor de la función en él; normalmente, se toma la suma de errores cuadráticos como función de coste, en virtud de su simplicidad. Entonces, se *entrena* la red, dando unos valores iniciales a sus parámetros y encontrando, por medio de un descenso de gradiente, el conjunto de valores que minimiza la función de coste; esto es, los valores de los parámetros que hacen más acertadas las predicciones de la red sobre los datos del conjunto proporcionado. El descenso de gradiente estándar consiste en calcular el gradiente de la función de coste con los valores actuales de los parámetros (los datos se mantienen constantes) y modificarlos dando, en el espacio de parámetros, un paso proporcional al gradiente con una cierta constante de proporcionalidad negativa. Así, al avanzar siempre en la dirección de decrecimiento más rápido, es de esperar que el valor de la función de coste converja rápidamente a su mínimo. Es importante que esta constante tenga, en magnitud, un valor adecuado para asegurar la convergencia en el menor tiempo posible, aunque es difícil estimar este valor de antemano. El proceso se repite hasta que se satisfaga un criterio de convergencia prefijado, y los valores obtenidos para los parámetros se toman como los valores que mejor representan la función. En este punto, se dice que la red ha sido entrenada. A continuación, se genera otro conjunto de datos de test, y se comprueba si las predicciones de la red sobre los valores de las variables coinciden aproximadamente con los valores de la función en ellos. Si es así, se considera que la red ha sido entrenada correctamente, mientras que en caso contrario se debe identificar dónde está el problema y tratar de corregirlo; puede ser que la constante de proporcionalidad no tenga un valor adecuado, que la cantidad de datos de entrenamiento sea insuficiente o que la estructura de la red no sea apropiada para reconstruir la función en cuestión.

4. Desarrollo

4.1. Modelo integro-diferencial para el control PID

4.1.1. Formulación del modelo sin controlador

En primer lugar, como ya se ha explicado en la introducción, es necesario modelar el sistema físico del laboratorio. La potencia absorbida por el refrigerador será:

$$\dot{Q} = P_{\text{ext}} + P_{\text{h}} + \dot{Q}_{\text{refr}} = C(T)\dot{T} \tag{3}$$

donde P_{ext} es la potencia que absorbe del entorno, P_{h} es la que absorbe del *heater* y $\dot{Q}_{\text{refr}}$ es la potencia (negativa) de refrigeración, que viene dada por la ecuación (1). Por otro lado, al estar la zona de interés del refrigerador constituida por un metal, su capacidad calorífica será, despreciando la contribución de los fonones, proporcional a la temperatura: $C(T) = k_C T$. De esta manera, para una P_{h} constante en el tiempo, se obtiene la siguiente ecuación diferencial para la evolución de la temperatura del sistema:

$$\dot{T} = \frac{1}{k_C T}\left[P_{\text{ext}} + P_{\text{h}} - (95 - 11f_T^2)\dot{n}_3 T^2\right] \tag{4}$$

donde la temperatura T es la de la *mixing chamber*, y el factor f_T relaciona dicha temperatura con la del último intercambiador de calor. Esta ecuación tiene solución analítica, que puede expresarse como:

$$\left(P_{\text{ext}} + P_{\text{h}} - (95 - 11f_T^2)\dot{n}_3 T^2(t)\right) = \left(P_{\text{ext}} + P_{\text{h}} - (95 - 11f_T^2)\dot{n}_3 T_0^2\right) e^{-\frac{2(95-11f_T^2)\dot{n}_3}{k_C}(t-t_0)} \tag{5}$$

O, de manera más compacta:

$$T^2(t) = T_\infty^2 + (T_0^2 - T_\infty^2)e^{-\frac{t-t_0}{\tau}} \tag{6}$$

con

$$T_\infty = \sqrt{\frac{P_{\text{ext}} + P_{\text{h}}}{(95 - 11f_T^2)\dot{n}_3}} \qquad\qquad \tau = \frac{k_C}{2(95 - 11f_T^2)\dot{n}_3} \tag{7}$$

4.1.2. Determinación de los valores de los parámetros del modelo

De todos los parámetros que intervienen en la ecuación (6), conocemos el valor de $\dot{n}_3$, y supondremos un valor razonable para f_T. Estos valores son los que recoge la Tabla (1):

$\dot{n}_3$ (mol/s)	$6 \cdot 10^{-4}$
f_T^2	1.1

Tabla 1: Valores conocidos o supuestos para el flujo de ^{3}He y la relación entre la temperatura del sensor y la del último intercambiador de calor

También conocemos, en cada situación, los valores de t_0 y T_0 (tiempo y temperatura iniciales, respectivamente), pues el programa que controla el refrigerador genera automáticamente ficheros (*logs*) que registran, entre otras cosas, los valores de la temperatura de la *mixing chamber* en función del tiempo; así como el valor de P_{h}, pues lo proporcionamos nosotros manualmente o bien el controlador PID. Por otro lado, según la expresión anterior, el valor de P_{ext} puede obtenerse a

partir de la ecuación (6) como $P_{\text{ext}} = (95-11f_T^2)\dot{n}_3 T_{\text{mín}}^2$, donde $T_{\text{mín}}$ es la temperatura asintótica que se alcanza cuando el *heater* está apagado. Esta potencia será, en general, variable con el tiempo, por su dependencia con las condiciones ambientales; este hecho queda reflejado en los *logs* del refrigerador, que muestran una temperatura mínima algo diferente a lo largo de los días. Sin embargo, por simplicidad y por la dificultad de modelar este fenómeno en detalle, consideraremos la potencia externa como constante, y la calcularemos como el promedio de las potencias externas halladas para cada día según los diferentes *logs* disponibles. En el cálculo, solo utilizaremos los *logs* cuya temperatura mínima sea inferior a 14 mK, pues para aquellos en los que esto no se verifica se observa que en ningún momento se ha dejado el *heater* apagado durante el tiempo suficiente como para que la temperatura se aproxime lo suficiente a su valor mínimo. Con estas consideraciones, el valor hallado para la potencia externa es:

$$P_{\text{ext}} = 5{,}12\ \mu\text{W}$$

Así, solo falta por estimar el valor de la constante k_C para tener bien definido el modelo. Para hacerlo, se recopilan, dentro de los *logs*, series de datos en las que el *heater* se mantenga apagado (es decir, $P_h = 0$) y la temperatura disminuya durante el mayor tiempo posible. A continuación, se representa gráficamente $\ln\left|P_{\text{ext}} - (95-11f_T^2)\dot{n}_3 T^2(t)\right|$ frente a $(t-t_0)$; según la ecuación (6), la relación debería ser lineal, en cuyo caso la constante k_C vendría dada, en términos de la pendiente m, por la siguiente expresión:

$$k_C = -\frac{2(95-11f_T^2)\dot{n}_3}{m} \tag{8}$$

No obstante, en los datos experimentales solo se observa una recta (aproximadamente) en el primer tramo, la cual se curva cuando la temperatura es próxima a la temperatura mínima (más concretamente, en cuanto baja de 30 mK). A modo de ejemplo de este comportamiento, la figura (1) muestra un caso particular correspondiente a uno de los fragmentos de *logs* utilizados:

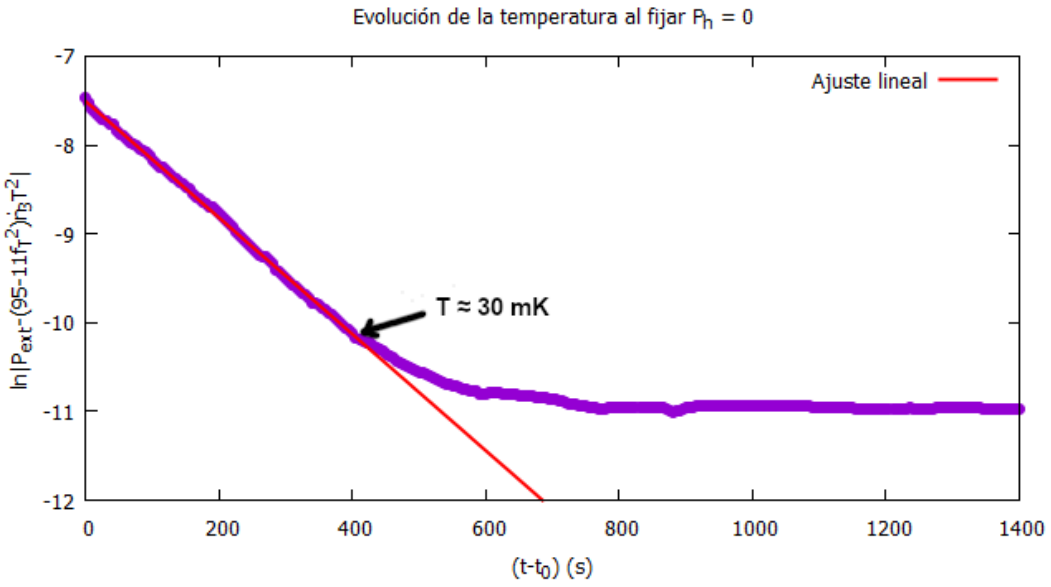


Figura 1: Gráfica empleada para determinar experimentalmente el valor del parámetro k_C del modelo

Con toda seguridad, las discrepancias del comportamiento térmico observado experimentalmente con las predicciones del modelo se deben a todos los efectos que este no tiene en cuenta por simplicidad, y que incluyen una cierta variabilidad de la potencia externa que llega al sistema. La magnitud de esta es despreciable a altas temperaturas; sin embargo, conforme estas disminuyen, también lo hace la capacidad calorífica, y en consecuencia un valor bajo de potencia puede afectar significativamente a la temperatura.

En consecuencia, para hallar el valor de k_C, se tomó una serie de 29 fragmentos de *logs* similares al de la figura (1), aunque con temperatura inicial diferente en cada caso, y se ajustó el primer tramo de cada uno de ellos a una recta siguiendo el criterio de mínimos cuadrados. A continuación, se calculó el valor de k_C para cada uno a partir de la pendiente de la recta de ajuste, según la expresión (8). La mejor estimación para el valor de k_C será, por tanto, el promedio de todos ellos:

$$k_C = 16{,}64 \pm 0{,}29 \text{ J/K}$$

4.1.3. Formulación del modelo con un controlador PID continuo

Hasta ahora, hemos modelado el sistema con un valor de P_h constante. Sin embargo, con la excepción del descenso a la temperatura base, esto en la realidad no será así, pues el control PID irá ajustando el valor de P_h con el objetivo de converger a la temperatura deseada. Veamos primero cómo se plantearía la implementación de un control PID continuo en nuestro modelo:

Para incluir el controlador en el modelo, basta con sustituir P_h por la expresión correspondiente al control en la ecuación (3). Así, queda:

$$P_{\text{ext}} + K\left(T_{\text{ref}} - T(t) + \frac{1}{T_I}\int_0^t (T_{\text{ref}} - T(s))ds + T_D\frac{d(T_{\text{ref}} - T(t))}{dt}\right) + \dot{Q}_{\text{refr}}(t) = k_C T(t)\dot{T}(t) \quad (9)$$

Sustituyendo $\dot{Q}_{\text{refr}}(t)$ por su expresión $-(95 - 11f_T^2)\dot{n}_3 T^2(t)$, derivando ambos miembros de la ecuación respecto al tiempo y considerando las variables $x(t) = T(t)$ e $y(t) = \dot{T}(t)$, se tiene el siguiente sistema de dos ecuaciones diferenciales ordinarias acopladas de primer orden:

$$\begin{cases} \dot{x} &= y \\ \dot{y} &= \frac{-Ky + \frac{K}{T_I}(T_{\text{ref}} - x) - 2(95 - 11f_T^2)\dot{n}_3 xy - k_C y^2}{k_C x + KT_D} \end{cases} \quad (10)$$

Por su parte, las condiciones iniciales apropiadas pueden hallarse haciendo $t = 0$ en la ecuación (9) y sabiendo que, por definición, $T(t_0) = T_0$. De esta forma, se obtiene:

$$\begin{cases} x_0 &= T_0 \\ y_0 &= \frac{P_{\text{ext}} + K(T_{\text{ref}} - T_0) - (95 - 11f_T^2)\dot{n}_3 T_0^2}{k_C T_0 + KT_D} \end{cases} \quad (11)$$

El sistema de EDOs junto con las condiciones iniciales constituyen un problema de valor inicial de primer orden, que puede resolverse con una gran variedad de métodos numéricos sencillos. En este trabajo, se ha elegido el método de Runge-Kutta explícito de 4º orden (RK4), por su alto orden de convergencia, su bajo coste computacional y su simplicidad en cuanto a la programación. El código puede encontrarse en el Anexo D.

Notar que este control es en esencia continuo, pues la única discretización que interviene (además del almacenamiento de números reales en una memoria finita) es la inherente a la aplicación del algoritmo de Runge-Kutta, y esta puede hacerse tan pequeña como convenga (aunque, eso sí, con el coste computacional correspondiente).

4.1.4. Formulación del modelo con un controlador PID discreto

Pese a lo que se acaba de explicar, el dispositivo experimental utilizado presenta dos limitaciones que invalidan el desarrollo anterior, correspondiente a un controlador PID continuo. La primera de ellas es la más relevante, y consiste en que el termómetro que monitoriza la temperatura del sistema para poder llevar a cabo la realimentación permite tomar únicamente una medida en cada intervalo de 5 segundos. En consecuencia, el controlador está configurado para suministrar escalones de potencia de valor constante durante este tiempo; esto podría cambiarse, pero, al no disponer de *feedback* en lo que dura uno de estos intervalos, no tendría demasiado interés hacerlo.

Esta limitación cambia completamente la naturaleza del problema, pues ahora P_h sí es constante a trozos y, por tanto, la evolución del sistema en cada intervalo de 5 segundos viene dada por la ecuación (4), que es de primer orden (en el caso del control continuo, era de segundo). Así pues, para hallar la evolución de la temperatura en un intervalo de tiempo arbitrario, basta con dividir este en intervalos de 5 segundos; y, a continuación, resolver la ecuación (4) en cada uno de ellos, en orden, usando el valor de P_h que corresponda en cada caso y tomando como condición inicial la última temperatura calculada en el intervalo de 5 s anterior.

Notar que no puede despreciarse este discretizado temporal de la manera en que despreciamos el del algoritmo RK4. La razón es que, con el primero, la respuesta del sistema empeora muy notablemente al aumentarlo, provocando incluso que no se alcance la temperatura deseada con unos parámetros del PID que sí permitirían alcanzarla con un control continuo. Esto se debe a que, con una sola medida de la temperatura cada 5 segundos, el tiempo de respuesta del controlador es demasiado elevado en comparación con los tiempos característicos que tarda la temperatura del sistema en variar apreciablemente. Para ilustrar este punto, se considera el siguiente escenario: supongamos que el sistema se encuentra a 40 mK, y desea llevarse a 60 mK. Para ello, se utilizan los siguientes parámetros del PID: (K,TI,TD) = (3, 750 s, 0). La siguiente gráfica permite comparar las predicciones del modelo con control discreto, aumentando y disminuyendo el tiempo entre medidas en un orden de magnitud, y las del modelo con control continuo. En ambos casos, se ha incluido la segunda corrección al controlador, la cual se explicará a continuación. Los resultados muestran claramente lo que se acaba de comentar sobre la importancia del tiempo entre medidas en la respuesta cualitativa del sistema: al aumentarlo, aparecen oscilaciones de amplitud cada vez mayor en torno a la temperatura deseada.

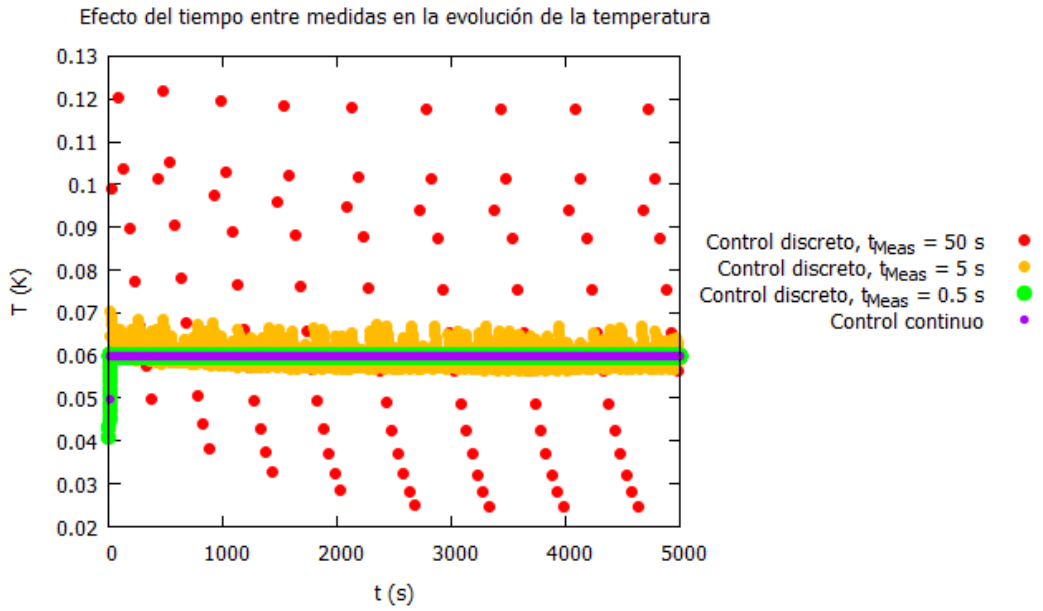


Figura 2: Estudio del comportamiento del sistema al variar el tiempo entre medidas

Por otro lado, se comprobó que la evolución de la temperatura apenas variaba al aumentar o reducir el discretizado del algoritmo RK4 en un factor 10, con lo cual se confirma que el discretizado empleado es apropiado para realizar las simulaciones.

En cuanto a la segunda limitación, esta consiste en la imposibilidad de suministrar una potencia negativa por medio del *heater*, pese a que el controlador PID pudiera requerirlo; cuando esto sucede, el *heater* simplemente se apaga ($P_h = 0$) hasta que deja de darse esta condición. Paralelamente, para evitar posibles daños por sobrecalentamiento en el sistema, se fija el valor máximo de potencia ($P_{\mathrm{h\ max}}$) en 3 mW; en caso de que el controlador exija más, simplemente se mantiene $P_h = 3\ mW$. Para añadir este hecho al modelo de un controlador PID continuo, se debe emplear una ecuación de evolución definida a trozos; esta será igual al sistema (10) en aquellos intervalos de tiempo en los que el PID requiera una potencia no negativa y menor que $P_{\mathrm{h\ max}}$, mientras que en aquellos en los que exija una potencia negativa o mayor que $P_{\mathrm{h\ max}}$ será igual a la ecuación (4), cuya solución ya hemos obtenido de forma analítica, aunque también se puede calcular numéricamente con el RK4. Sin embargo, obsérvese que estos intervalos no son conocidos *a priori*, de modo que tendrá que modificarse el código para poder hacerlo en el propio tiempo de ejecución del programa; esto es justamente lo que se hizo para obtener la curva del control continuo en la gráfica (2) anterior.

Sin embargo, para el controlador PID discreto, que es el que nos interesa modelizar, la modificación a realizar para tener en cuenta esta segunda limitación es mucho más sencilla: basta con evaluar el valor de potencia que se va a suministrar en cada intervalo antes de hacerlo, y, si es negativo o mayor que la potencia máxima permitida, sustituirlo por 0 o por el valor de dicha potencia, respectivamente. De nuevo, esto es lo que se hizo en la gráfica (2) para los tres casos con control discreto. El código para implementar este control en el modelo puede encontrarse en el Anexo (C).

4.1.5. Comparación del modelo con el sistema del laboratorio

Antes de seguir adelante, conviene comprobar la correspondencia del modelo con el sistema físico que modeliza, para hacernos una idea de cuánto pueden ayudar los datos generados por simulaciones a complementar los datos reales en el entrenamiento de la red neuronal (recordemos que esta era la motivación inicial para elaborar el modelo). Para ello, se han seleccionado dos escalones de temperatura crecientes y otros dos decrecientes, y dos conjuntos de parámetros para el PID distintos: unos que funcionan razonablemente bien en todos los escalones, y otros que no. Comencemos por los que sí resultan apropiados; la tabla (2) recoge sus valores, y la figura (3) ilustra la comparación entre la evolución predicha por el modelo para la temperatura y la observada experimentalmente:

K	0.05
T_I (s)	150
T_D (s)	0

Tabla 2: Valores para los parámetros del PID que funcionan correctamente

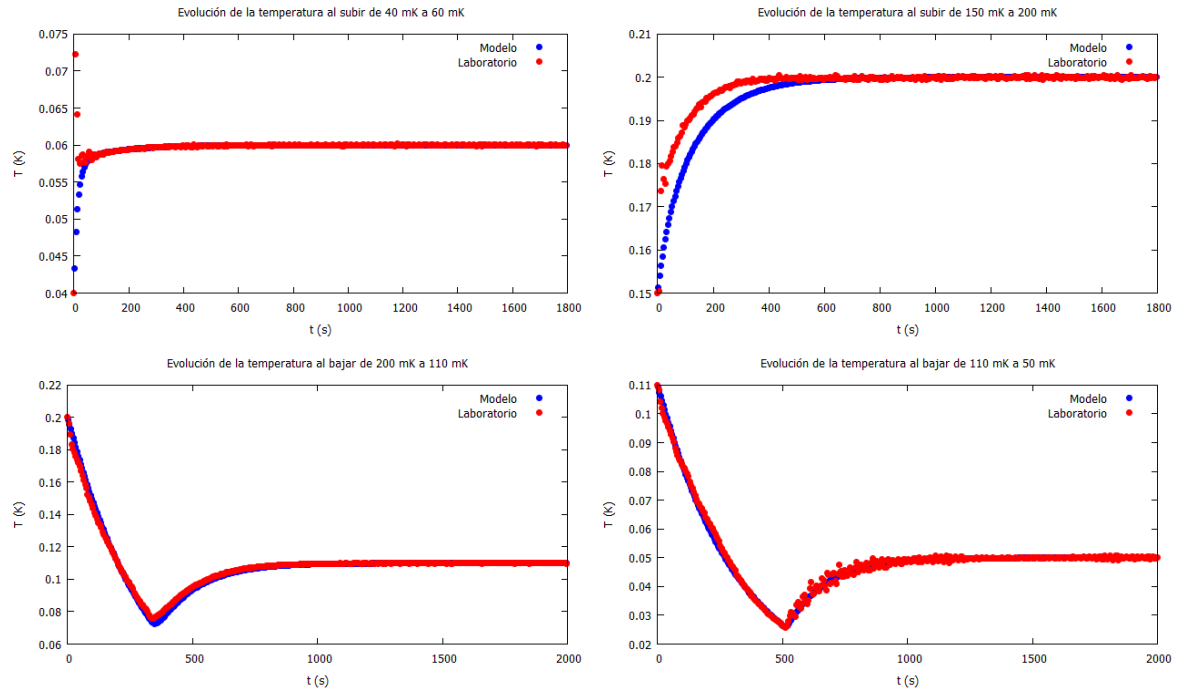


Figura 3: Comparativa entre la evolución predicha para la temperatura por el modelo y la real, en cuatro situaciones diferentes y con valores razonables para los parámetros

Se observa que la correspondencia entre el modelo y la realidad es mucho mayor, tanto cualitativa como cuantitativamente, en las bajadas de temperatura que en las subidas. Esto podría explicarse observando que, en primer lugar, los valores para los parámetros del modelo se han obtenido a partir de datos de bajadas libres de temperatura, con lo cual es de esperar que reproduzcan razonablemente bien el primer tramo de escalones decrecientes, pues en esta región $P_h = 0$. Además, se ve que donde más falla el modelo es en el paso de 150 mK a 200 mK; es decir, a altas temperaturas (en comparación con las que corresponden a la mayoría de

los escalones). Esto era de esperar, ya que previamente se había comprobado que la ecuación (1), que modela el refrigerador de dilución, es bastante imprecisa por encima de 150 mK. Esta comprobación se explica en detalle en el Anexo A.

Vayamos ahora con el otro conjunto de valores para los parámetros, que, como veremos, funcionan mucho peor. Estos valores están recogidos en la tabla (3), y la figura (4) permite comparar la evolución predicha por el modelo para la temperatura con la observada en el laboratorio:

K	3
T_I (s)	750
T_D (s)	0

Tabla 3: Valores para los parámetros del PID que no funcionan de manera razonable

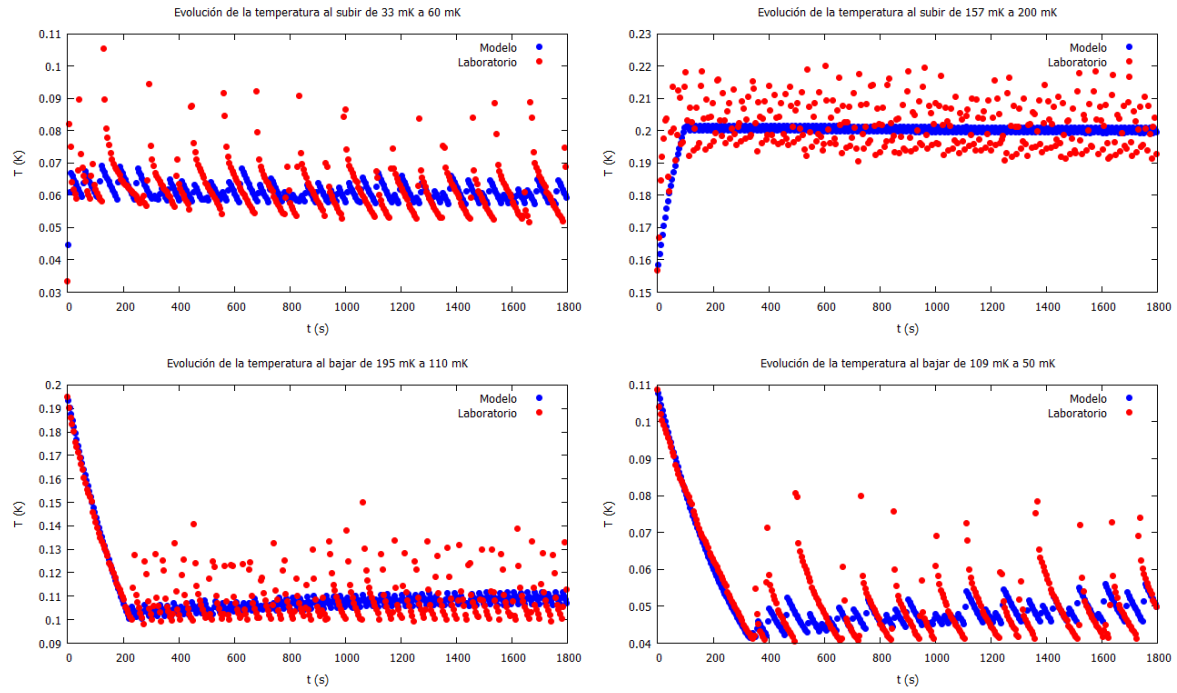


Figura 4: Comparativa entre la evolución predicha para la temperatura por el modelo y la real, en cuatro situaciones diferentes y con valores inapropiados para los parámetros

Con estos valores para los parámetros y en las situaciones consideradas, la correspondencia cualitativa entre modelo y realidad es mayor que para el caso anterior, pues en ambos se observa, para las cuatro configuraciones, una sucesión de aumentos repentinos de la temperatura seguidos por descensos libres (correspondientes al *heater* apagado). No obstante, cuantitativamente estos aumentos son mucho mayores, en un factor superior a 3, en la realidad que en las simulaciones. En vista a nuestro objetivo final, que es encontrar los valores óptimos para los parámetros del controlador en función de las temperaturas inicial y final, esto no es preocupante. La razón para ello es que lo único importante es que haya una relación de monotonía creciente entre el modelo y la realidad; es decir, que cuanto mejor resulte un conjunto de parámetros en la realidad, mejores sean las predicciones asociadas en el modelo, pero sin ningún tipo de requerimiento de proporcionalidad entre ambos. Así, optimizando el modelo habremos optimizado también el sistema real. Las figuras (3) y (4) avalan esta monotonía, aunque lógicamente no puede asegurarse

en el caso general; de hecho, como se verá más adelante, la relación no es de monotonía perfecta. Por ello, será necesario disponer también de datos experimentales.

4.2. Formulación matemática de la condición de optimalidad de la respuesta térmica deseada

Como ya se ha comentado en la introducción, el objetivo fundamental del trabajo, además de la elaboración de un modelo para el dispositivo experimental por medio de inteligencia artificial, es poder determinar, para cada pareja de temperaturas inicial y final, los parámetros del PID que dan lugar a una respuesta del sistema lo más próxima a la deseada idealmente. Concretamente, se busca, por un lado, que la temperatura final se alcance en el menor tiempo posible; y, por otro, que una vez en la temperatura deseada, el sistema se mantenga lo más próximo a ella durante un cierto tiempo, evitando oscilaciones en torno a ella o bien minimizando su amplitud. Así pues, intuitivamente, el objetivo está claro. Sin embargo, para poder comparar las respuestas asociadas a los distintos conjuntos de parámetros y determinar cuál de ellas es la mejor, debe definirse un parámetro de optimalidad. Este parámetro tomará valores reales y será función tanto de las temperaturas inicial y final como de los tres parámetros del controlador PID, y el conjunto de valores óptimos para los parámetros del PID se obtendrá, una vez fijadas las dos temperaturas, minimizando la restricción de la función al subconjunto con estas temperaturas fijas.

Es importante notar que no hay una única definición válida para el parámetro de optimalidad, y que diferentes definiciones pueden conllevar soluciones óptimas distintas (de hecho, es de esperar que lo hagan). Esto no supondrá ningún problema siempre y cuando, con el criterio (definición) que hayamos fijado, la solución óptima y las soluciones próximas a ella sean altamente deseables a nivel práctico. A continuación, plantearemos dos posibles definiciones razonables para el parámetro de optimalidad, y más tarde compararemos las soluciones óptimas halladas con cada una de ellas para distintos valores de las temperaturas.

4.2.1. Minimización de la norma en el espacio $\mathbf{L^2(0, t_{máx})}$ de la función diferencia

Nuestra primera definición estará motivada por la suma de residuos cuadráticos que se emplea habitualmente como criterio de calidad en regresión, y que matemáticamente corresponde a la norma en el espacio $L^2(X)$ para un cierto espacio dado X (habitualmente, $X = \mathbb{R}^n$ para algún $n \in \mathbb{N}$). Para ello, consideraremos el conjunto de evoluciones de la temperatura, en un intervalo de longitud prefijada, que pueden obtenerse variando los valores de los parámetros del PID. Como longitud del intervalo, se ha decidido tomar 5000 s. La razón para ello es que, experimentalmente, se observa que la temperatura deseada en los rangos de interés siempre puede alcanzarse y estabilizarse en un tiempo bastante menor a este; así, nos aseguramos de poder encontrar las soluciones "buenas" sin más que ver su comportamiento en dicho intervalo, pues si verifican lo anterior, teniendo en cuenta el funcionamiento del control PID, es de esperar que el resto del tiempo tengan un valor muy próximo a la temperatura objetivo. Además, el hecho de que el intervalo sea bastante mayor que el tiempo óptimo de estabilización nos sirve para penalizar las soluciones que, si bien alcanzan la temperatura objetivo en un tiempo pequeño, sufren marcadas oscilaciones en torno a ella durante un largo intervalo posterior.

Con la consideración anterior, las evoluciones de la temperatura en dicho intervalo para los diferentes conjuntos posibles de valores para los parámetros serán funciones pertenecientes al

espacio $L^2([0,5000])$, aunque lógicamente no podrán abarcar todas las funciones de este espacio (en particular, no podrán incluir ninguna de sus funciones discontinuas). Además, la razón de que sean de cuadrado integrable en dicho intervalo es que este es acotado y la temperatura también (es imposible obtener temperaturas divergentes en el laboratorio, por lo menos con el sistema con el que estamos tratando). Notar que la respuesta ideal que nos gustaría poder obtener es una función escalón $h_{T_0,T_{\text{ref}}}$ con valor T_0 en $t = 0$ y T_{ref} en el resto del intervalo $(0, 5000]$; obsérvese que esta función también está en el espacio mencionado anteriormente.

Es ahora momento de introducir un resultado de Análisis Funcional, que determina que, si I es un intervalo (o, en realidad, cualquier espacio de medida), entonces el espacio $L^2(I)$ es un espacio vectorial normado (más aún, espacio de Hilbert) en el que la norma de una función f viene dada por:

$$\|f\| = \sqrt{\int_I |f|^2 dx} \tag{12}$$

y en el que dicha norma proviene de un producto escalar:

$$\langle f, g\rangle = \int_I fg dx \qquad\qquad \|f\| = \sqrt{\langle f, f\rangle} \tag{13}$$

Notar que, en este caso, estamos considerando el caso particular $I = [0, 5000]$.

Por otra parte, denotaremos por $S_{T_0,T_{\text{ref}}}$ el subconjunto de funciones de dicho espacio que pueden obtenerse en el laboratorio como evolución de la temperatura habiendo fijado la temperatura inicial a T_0 y la final a T_{ref}, pero pudiendo establecer los valores de los tres parámetros del PID en cualquier número real positivo (y con T_I pudiendo valer ∞, pero no 0). Además, el espacio $L^2(I)$, al igual que cualquier otro espacio vectorial normado, es un espacio métrico. Entonces, parece razonable plantear el problema como encontrar la función de S cuya distancia a la función $h_{T_0,T_{\text{ref}}}$ es mínima. Como no conocemos bien el conjunto S, no podemos garantizar la existencia de este mínimo; sí que podríamos hacerlo, por ejemplo, si supiéramos con certeza que S es cerrado y convexo, pero esto no podemos saberlo. Sin embargo, en caso de no existir este mínimo, sí que existirá un ínfimo al que nos podremos acercar arbitrariamente, y a efectos prácticos cualquier solución que esté cerca de él nos servirá como solución "cuasi-óptima".

En resumen, según este planteamiento, el parámetro de optimalidad es:

$$p(T_0, T_{\text{ref}}, K, T_I, T_D) = \int_I (T_{T_0,T_{\text{ref}},K,T_I,T_D}(t) - h_{T_0,T_{\text{ref}}}(t))^2 dt \tag{14}$$

donde $T_{T_0,T_{\text{ref}},K,T_I,T_D}(t)$ da la evolución de la temperatura fijados los valores que aparecen en los subíndices como parámetros del sistema. Por otro lado, la ventaja de esta definición es que el parámetro de optimalidad valdría 0 para la función ideal buscada ($h_{T_0,T_{\text{ref}}}$) y un valor estrictamente positivo para cualquier otra función, que, en general, será mayor cuanto menos deseable sea la evolución de la temperatura obtenida. Notar también que, como ya se ha comentado, este parámetro de optimalidad penaliza simultáneamente los tiempos largos de alcance de la temperatura deseada y las oscilaciones en torno a ella, y es sencillo de implementar (la integral se va calculando sobre la marcha). Este parámetro de optimalidad se denotará por $I_2(T_0, T_{\text{ref}}, K, T_I, T_D)$.

4.2.2. Minimización del tiempo de convergencia con una tolerancia dada

Además del parámetro I_2, también puede tomarse como parámetro de optimalidad el tiempo de convergencia, denotado por t_1 y definido como el primer tiempo a partir del cual, dentro de

una simulación de 5000 segundos, el valor absoluto de la diferencia entre la temperatura y su valor objetivo no excede una cierta tolerancia, establecida en 0.2 mK. La motivación de esta definición y sus consecuencias se explican en detalle en el Anexo B.

4.3. Diseño de la red neuronal: estructura, tamaño y topología

Para programar la red neuronal, se empleó la herramienta Keras de Tensorflow, una librería de Python muy sencilla de utilizar a la par que versátil.

La topología utilizada consistió en una secuencia de capas, de las cuales la primera contenía una neurona por cada parámetro de entrada, y la última, una sola neurona para predecir el valor del parámetro de optimalidad. Entre ellas, se dispusieron tres capas ocultas, cada una de ellas con 40 neuronas (para la primera fase del entrenamiento), una capa intermedia con una sola neurona (*output* de la primera red) y 2 capas con 10 neuronas (para la segunda fase); todo esto se explicará en detalle más tarde. Como función de activación para todas las neuronas se estableció una sigmoide. El *momentum* de Keras se fijó en 0.001.

4.4. Generación de datos de entrenamiento

Por extraño que parezca, la parte más complicada en el trabajo con técnicas de inteligencia artificial (más concretamente, en la rama conocida como aprendizaje supervisado) es obtener una cantidad de datos, adecuadamente etiquetados, que permita entrenar a la red para reconstruir una determinada función, de manera que, al presentarle un nuevo conjunto de valores para los parámetros de entrada no presente en los datos de entrenamiento, la red sea capaz de dar una buena estimación del valor de la función para esos valores de entrada. Cuanto mayor es el número de estos parámetros, mayor será el número de datos requerido para poder entrenar a la red exitosamente. La razón de esto es sencilla de entender. Por ejemplo, supongamos que para reconstruir una función de $\mathbb{R}$ en $\mathbb{R}$ con un buen grado de fidelidad son necesarios 10 puntos. Entonces, para reconstruir una función similar pero de $\mathbb{R}^2$ en $\mathbb{R}$ harán falta del orden de $10^2 = 100$ puntos, y para una función de $\mathbb{R}^3$ en $\mathbb{R}$, unos $10^3 = 1000$ puntos. En nuestro caso, la función que define el parámetro de optimalidad (cualquiera de los dos definidos anteriormente) toma 5 parámetros como entrada y devuelve un número real, por lo que podrían hacer falta hasta $10^5 = 100000$ datos de entrenamiento. No obstante, esto supondría una cantidad descomunal de horas de simulación y/o de trabajo en el laboratorio, por lo que nos conformaremos con menos de la décima parte.

4.4.1. Datos generados por el modelo

Con las consideraciones comentadas al principio de esta sección, se generarán $6^5 = 7776$ datos simulados, lo que sería análogo a reconstruir una función de $\mathbb{R}$ en $\mathbb{R}$ a partir de 6 puntos. Como este es un número más bien reducido, deberemos tomar los datos proporcionando a los parámetros de entrada valores distribuidos en unos intervalos razonables, que permitan a la red intuir cuál es la tendencia de la función incógnita fuera de ellos, y, a ser posible, que contengan las soluciones óptimas para un amplio número de combinaciones de temperaturas inicial y final. A continuación, la tabla (4) recoge los distintos valores que se dio a cada parámetro, tomados de esa manera en base a pruebas realizadas con el descenso de gradiente sobre el propio modelo integro-diferencial (esto se explicará en detalle más adelante). Los datos se obtuvieron generando todas las combinaciones posibles con estos valores:

T_0 (K)	T_{ref} (K)	K (W/K)	T_I (s)	T_D (s)
0.010	0.020	0.01	30	0
0.040	0.058	0.13	180	0.01
0.070 (K)	0.096	0.25	330	0.02
0.100 (K)	0.134	0.37	480	0.03
0.130 (K)	0.172	0.49	630	0.04
0.160 (K)	0.210	0.61	780	0.05

Tabla 4: Valores proporcionados a cada parámetro, con los cuales se generaron todas las combinaciones posibles

4.4.2. Datos medidos experimentalmente

Puesto que, en definitiva, el experimento es lo que queremos reproducir con nuestro modelo, hará falta el mayor número posible de datos experimentales para entrenar a la red neuronal. Si pudiéramos obtener suficientes, sería innecesario (de hecho, también sería desaconsejable) utilizar al mismo tiempo datos provenientes de simulaciones para entrenar la red. Por desgracia, este no es el caso, lo cual nos obliga a combinar ambos tipos de datos para asegurarnos de que disponemos del número suficiente de puntos para reconstruir la función y, al mismo tiempo, de que estos reflejan la realidad lo más fielmente posible. Por ello, se recopiló, a partir de los *logs* del refrigerador, el mayor número de escalones posible (pasos de una temperatura a otra, con unos valores prefijados para los parámetros del PID). El número total de escalones obtenidos de esta manera fue 160, con la desventaja de que la mayoría usaban los mismos valores para los parámetros del PID; el número total de combinaciones empleadas para estos parámetros fue 5. Por su parte, los escalones eran en su mayoría de subida, con casi todos los saltos de temperatura con altura comprendida entre 1 y 60 mK. Las combinaciones de parámetros del PID probadas son las que se recogen en la tabla (5):

K (W/K)	T_I (s)	T_D (s)
0.05	100	0
0.01	250	0
0.1	80	0
3	750	0
0.3	40	0

Tabla 5: Combinaciones de los parámetros del PID ensayadas experimentalmente

El escaso número de estas combinaciones (tan solo 5) hace patente la necesidad de complementar estos datos con los provenientes de simulaciones. Además, todos los datos experimentales obtenidos se tomaron con $T_D = 0$, lo cual habría sido un grave error si este parámetro no hubiera demostrado tener, en todas las pruebas previas que se hicieron, una influencia más bien reducida en las predicciones del modelo.

Por otro lado, los datos experimentales presentaban una pequeña inconveniencia: la mayoría de los escalones tenía una duración muy inferior a 5000 s (en muchos casos, unos 250 s), con lo cual era necesario hacerles un tratamiento previo para que el valor del parámetro de optimalidad fuese comparable con el obtenido por medio del modelo. Se consideró que la solución más efectiva

a este problema sería, en el caso del parámetro I_2, repetir los últimos 50 trapecios que forman parte de la discretización de la integral (ya que esta se calculó por trapecios, en lugar de por rectángulos, para mejorar la estimación proporcionada) una y otra vez hasta alcanzar los 1000 (uno por cada intervalo de 5 s que compone el intervalo total de 5000 s). Para aquellos escalones con menos de 120 datos, se repitieron únicamente los 15 últimos trapecios. Esta aproximación es razonable cuando la temperatura ya se halla relativamente estabilizada al acabar el tiempo que se mantiene el escalón, y permite establecer comparaciones con el modelo especialmente en las configuraciones que resulten en oscilaciones nada despreciables en torno a la temperatura objetivo.

4.5. Entrenamiento de la red combinando ambos tipos de datos

Como ya se ha explicado anteriormente, un objetivo fundamental de este trabajo es elaborar un modelo del refrigerador de dilución con los sensores TES combinando datos experimentales (precisos, pero poco abundantes) con datos generados por simulaciones basadas en un modelo integro-diferencial más sencillo (menos precisos debido a la simplicidad del modelo, pero mucho más fáciles de obtener en gran cantidad). El enfoque por el que se optó para combinar ambos tipos de datos es lo que se conoce como *transfer learning*. En este caso particular, requiere, en primer lugar, entrenar una red neuronal empleando únicamente datos provenientes de las simulaciones. A continuación, se construye una segunda red, cuyas primeras capas son idénticas a las de la red inicial (tanto en estructura como en los valores de los pesos y *biases*) y se mantienen "congeladas"; es decir, no modificarán los valores de sus parámetros durante el entrenamiento. No obstante, después de estas capas, se añaden otras, aunque con menos parámetros (pues se configuran con un menor número de capas y de neuronas), que sí modificarán los valores de estos en el entrenamiento, el cual se realizará en esta segunda etapa con datos reales. Es de esperar que las predicciones de esta segunda red para el parámetro de optimalidad sean mejores que las de la primera, pues se habrá incorporado información de alta fidelidad en la red.

El entrenamiento en cada fase se realizó en un total de 200 épocas, con un tamaño de *batch* igual a 32. Para reducir el riesgo de *overfitting* (es decir, de que la red aprenda muy bien a reproducir los datos que se le han proporcionado pero, sin embargo, sea incapaz de predecir el valor del parámetro de optimalidad para otras combinaciones de parámetros), se añadió, después de cada una de las capas ocultas, una capa de *dropout* con una tasa de 0.2; es decir, que en cada paso del entrenamiento, cada una de estas capas pondrá a 0 en torno al 20 % de los valores de entrada correspondientes. Por otra parte, el *learning rate* (el factor que multiplica al vector opuesto al gradiente en cada paso del descenso) se fijó en 0.01. Finalmente, el conjunto de datos en cada fase se dividió en dos grupos: el de entrenamiento, que comprendía el 80 % de los datos disponibles, y el de validación, formado por el 20 % restante.

La evolución del aprendizaje tanto en el *set* de entrenamiento como en el de validación con respecto al avance de las épocas, así como la comparación entre las predicciones de la red y los valores reales, se muestran en la sección *Resultados*.

4.6. Optimización de los parámetros del PID para cada par de temperaturas

Tal y como se comentó en la introducción, el segundo objetivo fundamental del trabajo, y el más relevante en cuanto a sus aplicaciones prácticas, es servirse del modelo para poder hallar, dadas las temperaturas inicial y final, los valores de los parámetros del PID que permiten

llegar a la temperatura de referencia en el menor tiempo posible, manteniéndose en ella el resto del tiempo sin realizar oscilaciones pronunciadas. Esto, como ya se ha explicado anteriormente, equivale a encontrar los parámetros del PID que minimizan el valor del parámetro de optimalidad para unas temperaturas inicial y final fijas. Para ello, uno de los métodos estándares más sencillos es el descenso de gradiente. Para poder aplicar este método, se debe disponer de la función $p(T_0, T_{\text{ref}}, K, T_I, T_D)$, que proporciona el valor de un parámetro de optimalidad genérico p para unos parámetros del modelo arbitrarios (temperaturas y coeficientes del PID). En nuestro caso, disponemos de una aproximación a dicha función, de modo que es de esperar que el resultado final que obtengamos no coincida con el óptimo, pero que sí se encuentre cerca de él (lo cual, a efectos prácticos, nos sirve perfectamente). A continuación, se debe fijar el *learning rate*, que ya se ha definido antes y que se denotará por lr. Para finalizar los preparativos, se escoge un punto inicial razonable (en $\mathbb{R}^5$). Entonces, se realiza una serie de iteraciones hasta que se verifica una determinada condición; por ejemplo, que el valor hallado en una iteración y el hallado en la anterior difieran en una cantidad menor que una determinada tolerancia. En cada iteración, se estima numéricamente el gradiente de la función en el punto actual. Para ello, se calculan las derivadas parciales por medio de la expresión (5), sustituyendo t_1 por p en caso de estar considerando un parámetro de optimalidad diferente a t_1. En este cálculo, es fundamental asegurarse de que h es lo suficientemente pequeño como para que esta sea una buena aproximación, pero también lo suficientemente grande (en el caso de t_1) para sobreponerse al problema de constancia local que presenta este parámetro, y que se explica en el Anexo B. Entonces, se da un paso (en $\mathbb{R}^5$) en la dirección opuesta al gradiente (dirección de máximo decrecimiento local) y de magnitud $lr|\nabla p|$. Es de esperar que, en este nuevo punto, el parámetro de optimalidad tome un valor menor (si lr es lo suficientemente pequeño, esto siempre ocurrirá). Este proceso se repite en cada iteración, y el punto en que nos encontremos al finalizar la última de ellas será nuestra estimación para el conjunto óptimo de parámetros del PID con las temperaturas inicial y final especificadas.

Como se ve, este algoritmo de minimización es, conceptualmente, muy sencillo. Sin embargo, presenta una serie de problemas:

En primer lugar, el algoritmo nos llevará a un mínimo local, que no tiene por qué ser el mínimo global. Este problema es imposible de solucionar en el caso general, aunque sí puede ser conveniente aplicar el proceso varias veces desde puntos iniciales diferentes y comprobar si se llega o no siempre al mismo punto final.

En segundo lugar, el valor de lr debe fijarse *a priori*, y en muchas ocasiones no se tiene intuición sobre cuál podría ser un valor razonable para él. Además, este valor razonable irá variando conforme nos acerquemos al mínimo, ya que el gradiente disminuirá en sus proximidades y, por tanto, los pasos dados podrán llegar a ser excesivamente pequeños. Por otro lado, puede ocurrir que lr sea demasiado grande para una región dada y que, en consecuencia, el valor del parámetro de optimalidad aumente de una iteración a la siguiente. Para evitar lo último, siempre se comprueba que esto no va a ocurrir antes de dar el paso; y, en caso de no ser así, se reduce el valor del lr a la mitad hasta que lo verifique. Con esta modificación, su valor inicial puede ser elevado sin ningún problema, lo cual favorece por lo general una velocidad de convergencia alta.

El tercer problema es el enorme desajuste de escalas entre los tres parámetros del PID; es decir, que un cambio de una unidad en el parámetro K resulta en una variación de p mucho mayor (varios órdenes de magnitud) que si se hubiera modificado el parámetro T_I en la misma cantidad. En el caso del parámetro T_D, la sensibilidad es todavía mayor. Esto supone un grave

problema a la hora de ejecutar el descenso, ya que en cuanto el parámetro más sensible tome un valor cercano a su mínimo local, el valor de lr deberá reducirse mucho para que el valor del parámetro de optimalidad no vuelva a incrementarse; sin embargo, al hacer esto, los pasos que se darán en las direcciones de los otros parámetros serán demasiado pequeños, forzando al algoritmo a realizar un número inabarcable de iteraciones para poder llegar a las proximidades del óptimo.

Para solucionar esto último, en lugar de dar cada paso en la dirección opuesta al gradiente, lo que se hace es multiplicar primero cada una de sus componentes por un determinado factor positivo (distinto para cada componente) que garantice que, en la mayoría de los puntos, las tres derivadas parciales tomen valores que no difieran en demasiados órdenes de magnitud (a ser posible, que se encuentren dentro del mismo). Los pesos relativos adecuados para cada componente se determinaron haciendo pruebas en varios casos particulares, y son los que muestran en la tabla (6):

f_K	f_{T_I}	f_{T_D}
1	$5{\cdot}10^6$	0.01

Tabla 6: Factores por los que se multiplican las respectivas derivadas parciales antes de dar cada paso en el descenso de gradiente

4.6.1. Descenso de "gradiente" aplicado al modelo integro-diferencial

Inicialmente, se aplicó un descenso de gradiente, con las modificaciones explicadas en el apartado anterior, al modelo integro-diferencial para tener una idea de los intervalos en los que se encuentran los valores óptimos para los parámetros del PID, y poder así generar datos con valores en un rango apropiado. Por tanto, esto se hizo antes de la generación de datos descrita anteriormente (es lo que proporcionó la intuición comentada en dicha sección para elegir los valores de los parámetros). Se comprobó además que, por lo general, bastaba con aplicar unas 5 iteraciones del descenso para encontrar una solución muy próxima a la óptima, por lo que, para dar un poco de margen pero manteniendo un tiempo de ejecución bajo (menos de 3 minutos), se fijó el número de iteraciones a 10 (pudiendo aumentarse si se observa una gran variación entre los últimos puntos). Las soluciones óptimas halladas para distintas configuraciones de temperaturas inicial y final se encuentran en el apartado *Resultados*.

4.6.2. Descenso de "gradiente" aplicado a la red neuronal

Con el objetivo de encontrar unos valores para los parámetros del PID que optimicen la evolución de la temperatura dados sus valores inicial y de referencia, sería conveniente, en principio, aplicar el descenso de gradiente a la red neuronal, pues de los dos modelos que tenemos es el único que contiene información de alta fidelidad sobre el experimento. Sin embargo, como se verá en el apartado *Resultados*, la red neuronal se muestra incapaz de predecir el parámetro de optimalidad con exactitud, por lo que se prescindirá de esto.

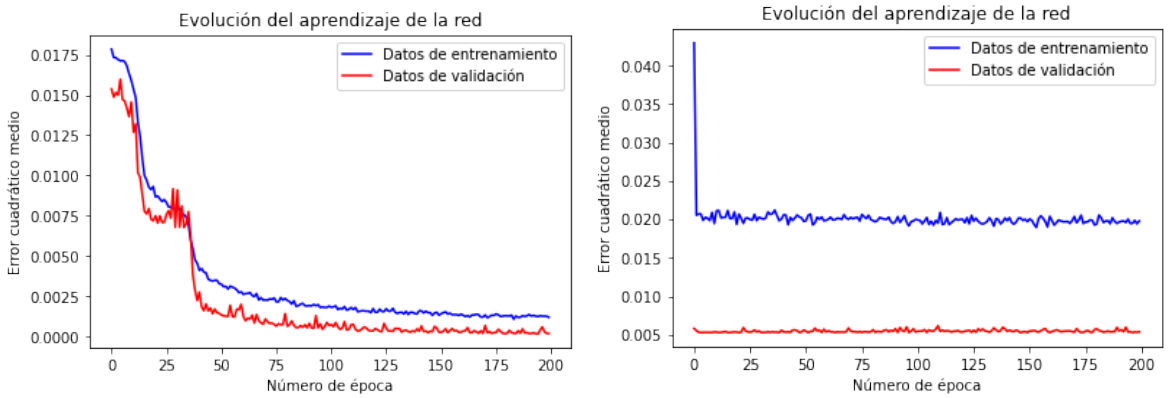
5. Resultados

En esta sección se recogen todos los resultados relativos al entrenamiento de la red neuronal y a la optimización de los parámetros del PID dadas las temperaturas inicial y final. Los resultados intermedios (inferencia de los valores de los parámetros del modelo integro-diferencial, desempeño de este, etc.) ya se han mostrado en los apartados correspondientes, con el objeto de facilitar la comprensión del trabajo. Por otra parte, por simplicidad, en los dos primeros apartados 5.1 y 5.2 se trabajará únicamente con el parámetro de optimalidad I_2.

5.1. Evolución del aprendizaje de la red con el paso de las épocas

El aprendizaje de la red se cuantifica por medio del error cuadrático medio (*mse*), tanto respecto a los datos de entrenamiento como respecto a los de validación. En un proceso de aprendizaje correcto, este error debe disminuir con el paso de las épocas para ambos grupos de datos. Además, en las últimas épocas debe ser próximo a 0 en los datos de entrenamiento (de lo contrario, se dice que tenemos *underfitting*) y no mucho mayor en los datos de validación que en los de entrenamiento (en caso contrario, tendremos *overfitting*).

Veamos en primer lugar las curvas de aprendizaje de la red inicial, tanto si la entrenamos exclusivamente con datos simulados como si lo hacemos solo con datos reales:



(a) Entrenamiento con datos simulados (7776) (b) Entrenamiento con datos reales (160)

Figura 5: Evolución del aprendizaje de la red inicial

En la figura (5) se observa como, en el caso del entrenamiento con datos reales, el error cuadrático medio final es un orden de magnitud superior al obtenido para los datos simulados, lo que apunta a una situación de *underfitting* al emplear únicamente datos experimentales.

Veamos ahora lo que ocurre en el entrenamiento de la segunda red, que utiliza la información aprendida por la primera (entrenada exclusivamente con datos simulados) y la complementa con datos reales:

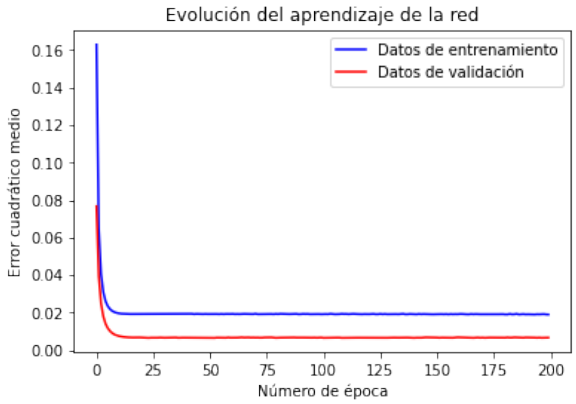
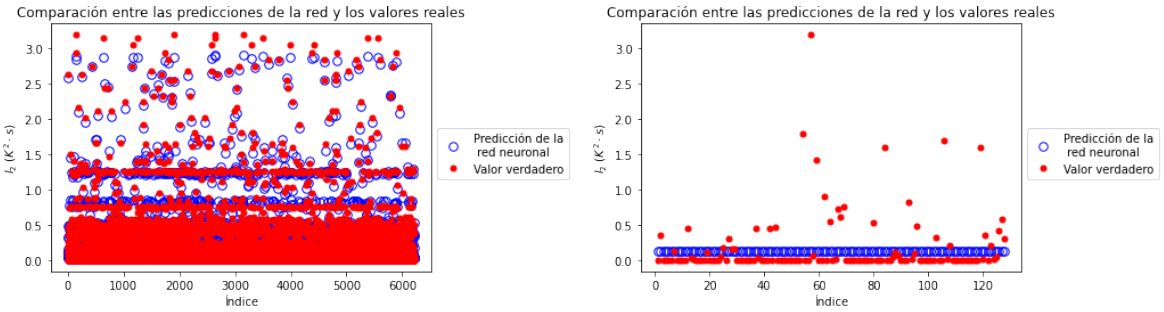


Figura 6: Evolución del aprendizaje de la red final

Según se observa en la figura (6), en la red final se alcanza una situación similar a la de la figura (5b), en cuanto a que se vuelve a tener *underfitting*. No obstante, las curvas de aprendizaje son marcadamente más suaves.

5.2. Exactitud de las predicciones de la red sobre los datos

En este apartado, se compararán las predicciones de ambas redes para sus respectivos datos de entrada utilizados en el entrenamiento con los datos de salida que se han empleado para entrenarlas (y que, en consecuencia, se corresponden con lo que se debería obtener, bien en la simulación o bien en el laboratorio). Comencemos por la primera red:

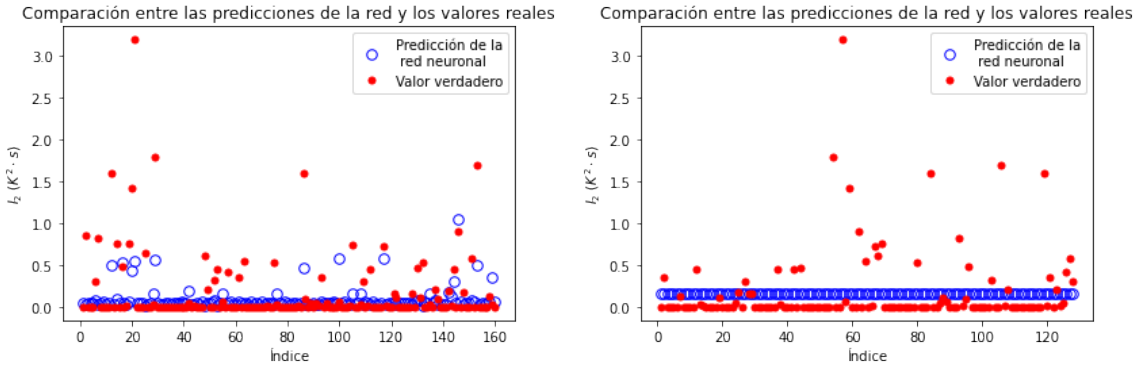


(a) Comparación con datos simulados - red inicial (b) Comparación con datos reales - red inicial

Figura 7: Evolución del aprendizaje de la red inicial

Se observa como, tal y como cabía esperar a la vista de los resultados de la sección anterior, la red entrenada con los datos simulados proporciona predicciones mucho más acertadas que la entrenada con datos reales (que predice, esencialmente, una función constante, lo cual no tiene ningún tipo de sentido). La razón principal de esta discrepancia es, sin lugar a dudas, el número de datos empleados para entrenar la red: 7776 en el caso de los datos simulados y tan solo 160 en el de los experimentales. Por tanto, en este último caso estaríamos intentando reconstruir una función de $\mathbb{R}^5$ en $\mathbb{R}$ a partir de 160 puntos, lo que, según la discusión que se hizo en el apartado 4.4, equivaldría en dificultad a tratar de reconstruir una función de $\mathbb{R}$ en $\mathbb{R}$ con menos de 3 puntos ($\sqrt[5]{160} \approx 2.76$), lo cual es evidentemente imposible de realizar sin cometer un error enorme.

Veamos ahora lo que ocurre con las predicciones de la red final:



(a) Comparación con datos reales - sin reentrenar (b) Comparación con datos reales - red final

Figura 8: Comparación con datos reales - red final

Se observa que, en esencia, se vuelve a tener el comportamiento de la figura (7b), al igual que ocurría con las gráficas de entrenamiento. Esto puede parecer contrario a la intuición, pues, tal y como se explicó en la sección 4.5, lo lógico sería que, al reentrenar la red con unos pocos parámetros libres, mejorase el acuerdo entre sus predicciones y la realidad. El hecho es que, cuantitativamente y según el criterio escogido (error cuadrático medio), sí que mejora, pero es evidente que la respuesta cualitativa empeora bastante, pues la red pasa de predecir picos en las mismas posiciones en que están (antes de reentrenarla) a predecir una función constante tras la segunda fase del entrenamiento. Inicialmente, se pensó que podía ser problema de la cantidad de parámetros libres añadidos en dicha fase; sin embargo, después de hacer varias pruebas aumentando y disminuyendo el número de capas y neuronas nuevas y comprobar que este resultado se mantiene prácticamente inalterado, se concluye que el problema está en la escasez de datos experimentales y en la muy reducida variedad de valores que toman los parámetros del PID en ellos. Además, notar que, si dos combinaciones de parámetros dan lugar al mismo valor de I_2 según las predicciones de la red inicial pero no en la realidad, esto seguirá ocurriendo independientemente de la estructura que adopte la nueva parte de la red. Para solucionar esto, habría que "descongelar" algunos de los parámetros de la primera red, pero no es evidente cómo hacer esto sin que el resultado equivalga a entrenar la nueva red desde cero a partir de los datos reales, prescindiendo de la información proporcionada por el modelo.

5.3. Soluciones óptimas

Como ya se ha explicado a lo largo del trabajo, lo ideal para hallar los valores óptimos para los parámetros del PID en función de las temperaturas inicial y final sería disponer de un modelo que predijese el valor del parámetro de optimalidad con gran exactitud y aplicarle un descenso de gradiente (modificado) respecto a los tres parámetros del PID. Sin embargo, como se acaba de comprobar, las predicciones del modelo que combina datos reales con datos simulados distan mucho de la realidad. Más aún, predicen una función constante, de modo que según dicho modelo cualquier combinación de valores para los parámetros del PID daría pie a una solución igual de buena. Sin embargo, en la sección 4.1.5 pudimos comprobar que este no es en absoluto el caso, pues hay valores del PID que funcionan muy bien y otros con los que ni siquiera se puede estabilizar el sistema en la temperatura deseada. Por tanto, dadas las circunstancias, este enfoque no parece apropiado.

Por otro lado, en la figura (8a) se observa que, si bien una red entrenada con datos simulados falla cuantitativamente al predecir la bondad de una combinación determinada de parámetros, sí que tiende a asignar mayores valores del parámetro de optimalidad a aquellas combinaciones que, en el laboratorio, resultan en valores elevados para dicho parámetro. Puesto que se ha entrenado empleando exclusivamente datos simulados, es de esperar que el modelo integro-diferencial inicial posea también esta propiedad de monotonía creciente, al menos de manera aproximada. Para comprobarlo, se calculó, mediante este modelo, el valor de I_2 correspondiente a todas las combinaciones de parámetros de las que se tienen datos experimentales, obteniéndose la siguiente gráfica:

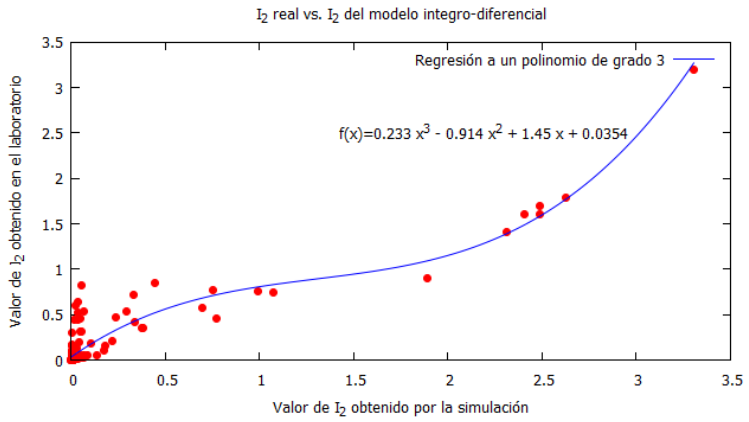


Figura 9: Relación entre los valores de I_2 predichos por el modelo integro-diferencial y los reales

Aunque es cierto que la figura (9) no muestra una monotonía creciente perfecta, sí que permite apreciar a primera vista una tendencia general creciente. Por tanto, esto apunta a que, para hallar los parámetros óptimos para el PID, basta con aplicar el descenso de gradiente sobre el modelo integro-diferencial, pues aunque el valor que este prediga para el parámetro de optimalidad sea diferente al real, la solución óptima en un caso también lo será (aproximadamente, al menos) en el otro.

Por otro lado, si a partir del valor predicho para este parámetro se quisiera saber su valor real, como primera aproximación podría emplearse la regresión cúbica por mínimos cuadrados que se muestra también en la figura (9), cuya ecuación puede verse en la gráfica. No obstante, en principio este valor no será relevante.

Para finalizar este trabajo, se muestran, en la figura (10), las soluciones óptimas halladas, en unidades del SI, para cada criterio de optimalidad (I_2 y t_1), por descenso de gradiente modificado sobre el modelo integro-diferencial, para las combinaciones de temperaturas presentadas anteriormente en la sección 4.1.5. Estas soluciones se superponen a las halladas en dicha sección con los valores "razonables" para los parámetros del PID, así como a la evolución de la temperatura medida experimentalmente en cada caso. Interpretemos los resultados:

En la figura 10a, las dos soluciones óptimas (según sus respectivos criterios) son prácticamente coincidentes, y es evidente que mejoran de manera notable la solución anterior. Esto se corresponde con lo que cabría esperar, pues ambos criterios favorecen los mismos comportamientos.

En la figura 10b, se observa un hecho curioso: la solución óptima para el parámetro I_2 tiene un valor de t_1 mucho menor que la solución óptima para este último parámetro (105 s frente a

366 s), lo cual, *a priori*, carece de toda lógica. No obstante, lo que ha ocurrido en realidad es que el descenso de gradiente para el parámetro t_1 ha fallado debido a la constancia local de la función que lo define, quedándose la optimización a medias al llegar a un punto de gradiente nulo (incluso con las medidas que se han tomado para evitar esta situación). Se comprueba así que este parámetro es considerablemente más problemático que I_2 a la hora de llevar a cabo la optimización.

Por último, las figuras 10c y 10d muestran soluciones óptimas cualitativamente muy diferentes para los dos parámetros. La razón de esto es que, para el parámetro I_2, la solución óptima emplea valores de K y T_I muy elevados para oponerse a la disminución inicial de la temperatura por debajo de su valor deseado; sin embargo, esto provoca oscilaciones en torno a la temperatura de equilibrio que no verifican el criterio de calidad de t_1, de modo que para este parámetro la solución óptima será una que, pese a decrecer más de lo que se desearía inicialmente, no experimenta oscilaciones en torno a la temperatura de referencia, sino que se aproxima a ella de manera asintótica.

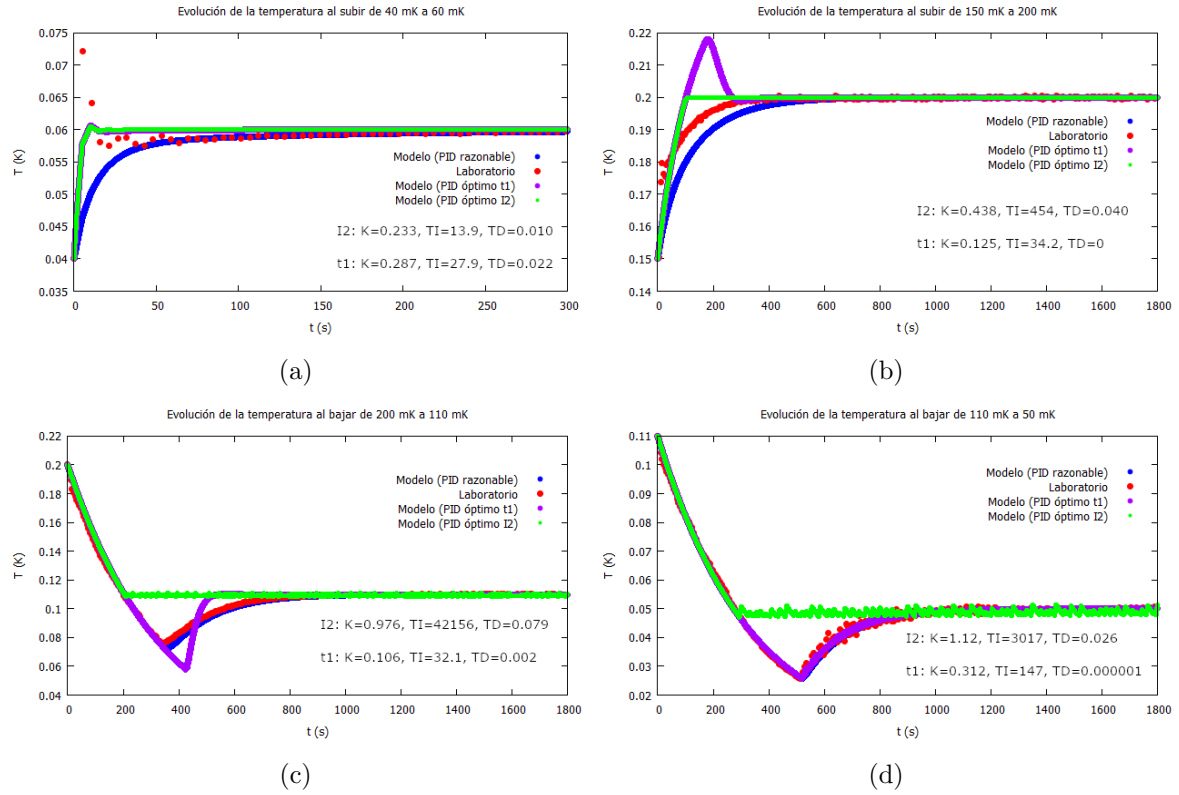


Figura 10: Comparativa entre la respuesta del sistema con los parámetros "razonables" de la sección 4.1.5 y su respuesta con los parámetros óptimos, hallados por descenso de gradiente modificado sobre el modelo integro-diferencial, según los dos parámetros de optimalidad definidos

Notar que los resultados obtenidos en el proceso de optimización abren varias vías de investigación para el futuro. Por ejemplo, el hecho de que, en escalones de bajada, parezca imposible conseguir simultáneamente una convergencia rápida y la ausencia de oscilaciones (ver figuras 10c y 10d) con un único conjunto de valores para el PID sugiere el empleo consecutivo de dos conjuntos: uno para converger rápido y otro para mantenerse estable en la temperatura deseada. Por otro lado, en escalones de subida, la solución óptima aplica una elevada potencia en los

primeros intervalos; esto sugiere mantener el *heater* encendido inicialmente a potencia máxima durante un tiempo variable para acercar la temperatura a la de referencia todo lo posible, y utilizar después un conjunto de valores para el PID optimizados para estabilidad.

6. Conclusiones

En este trabajo se ha modelado un refrigerador de dilución, simulando el control PID de tres maneras diferentes: un modelo integro-diferencial, una red neuronal entrenada con datos generados por dicho modelo y, finalmente, un modelo de *transfer learning* alimentado con datos tanto teóricos como experimentales. Los dos primeros modelos proporcionan resultados muy similares entre sí, mientras que el modelo de *transfer learning* predice para el conjunto entero de datos unas propiedades muy similares, próximas a las de la mayoría de los puntos pero bastante diferentes para una minoría; la razón principal de este hecho es la escasez de datos experimentales y su poca variedad. Así pues, queda abierto para investigaciones futuras mejorar este modelo para lograr un alto grado de acuerdo con la totalidad de los datos experimentales, no únicamente con la mayoría de ellos.

Por otro lado, el descenso de gradientes se ha implementado satisfactoriamente en el modelo integro-diferencial, permitiendo encontrar valores más apropiados para los parámetros del PID que los empleados hasta el momento.

7. Bibliografía

[1] DIETL, T. y otros. Low Temperature Physics. *Physics Now: Reviews by leading physicists in the International Union of Pure and Applied Physics* [en línea]. 2004, vol. 39 [consulta: 11 septiembre 2022]. https://www.dpg-physik.de/vereinigungen/fachlich/skm/fvtt/low-temperature-physics-an-introduction#:~:text=The%20most%20significant%20real%20application,field%20with%20no%20Joule%20heating.

[2] POBES, Carlos y otros. Development of cryogenic X-ray detectors based on Mo/Au Transition Edge Sensors. *IEEE Transactions on Applied Superconductivity* [en línea]. 2017, vol. 27, nº 4 [consulta: 17 agosto 2022]. ISSN 1051-8223. Disponible en: http://hdl.handle.net/10261/151089

[3] IRWIN, K.D. y HILTON, G.C. Transition-Edge Sensors. En: C. ENSS, editor. *Cryogenic Particle Detection.* Springer-Verlag Berlin Heidelberg, 2005, p. 63-149. ISBN 978-3-540-31478-3.

[4] BATEY, Graham y TELEBERG, Gustav. Principles of dilution refrigeration. Abingdon, Reino Unido: Oxford Instruments Nanoscience, 2015.

[5] ÅSTRÖM, K.J. y MURRAY, R.M. *Feedback Systems: An Introduction for Scientists and Engineers* [en línea]. Versión v2.11b. Princeton, NJ: Princeton University Press, 2012. [consulta: 18 agosto 2022]. ISBN 978-0-691-13576-2. Disponible en: http://www.cds.caltech.edu/~murray/books/AM08/pdf/am08-complete_28Sep12.pdf

[6] RUSSELL, S. y NORVIG, P. *Artificial Intelligence: A Modern Approach.* 3ª ed. Upper Saddle River, NJ: Prentice Hall, 2010. ISBN 978-0-13-604259-4.

A. Limitaciones del modelo

De acuerdo con la ecuación (4), si el *heater* aplica una potencia constante P_h, entonces en el equilibrio se verificará (haciendo $\dot{T} = 0$):

$$P_h = (95 - 11 f_T^2)\dot{n}_3 T_{\text{eq}}^2 - P_{\text{ext}} \tag{15}$$

Es decir, que si tenemos un conjunto de datos de la forma (T_{eq},P_{ext}), donde P_{ext} es la potencia constante del *heater* que da lugar a una temperatura de equilibrio T_{eq}, entonces la relación entre P_{ext} y T_{eq}^2 deberá ser lineal. Sin embargo, recopilando datos de la forma anterior de los *logs* del sistema, se obtiene la siguiente gráfica:

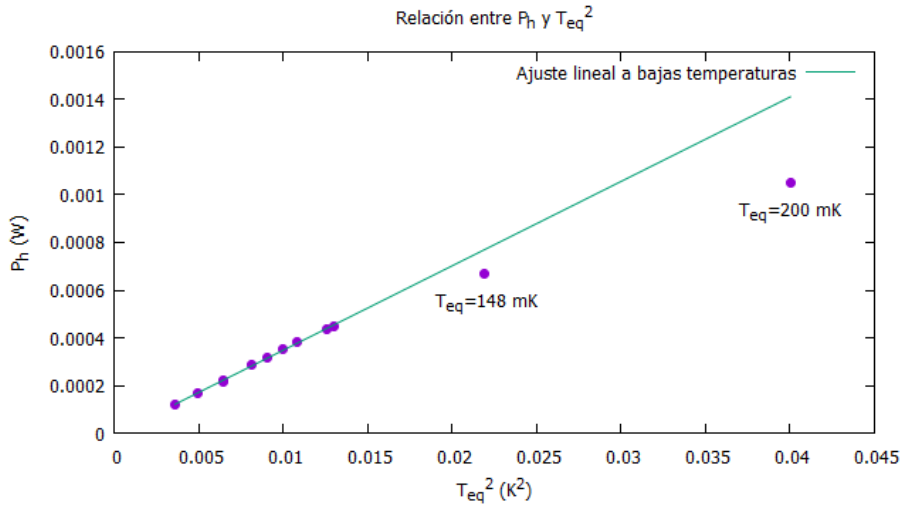


Figura 11: Relación entre la temperatura de equilibrio y la potencia constante del *heater* que permite alcanzarla

Se observa que todos los puntos correspondientes a temperaturas entre 60 mK y 114 mK (es decir, todos los puntos de la figura (A) salvo los dos de la derecha) responden muy bien a las predicciones del modelo, pues para ellos la relación entre P_{ext} y T_{eq}^2 presenta un ajuste muy bueno a una recta. Sin embargo, el punto con $T_{\text{eq}} = 148$ mK se desvía considerablemente de la recta de ajuste, y el de $T_{\text{eq}} = 200$ mK experimenta una desviación aún mayor. Esto apunta a que la expresión (1) de la potencia de refrigeración es válida por debajo de 114 mK (por lo menos, hasta 60 mK), pero no lo es a temperaturas iguales o superiores a 150 mK, de modo que es de esperar que haya discrepancias importantes entre las predicciones del modelo y los resultados de los experimentos por encima de esta temperatura.

B. Motivación y consecuencias de la definición del parámetro de optimalidad t_1

Otra manera razonable, además del parámetro I_2, de definir el parámetro de optimalidad sería considerar el tiempo que tarda la temperatura en estar definitivamente confinada en el intervalo $(T_{\text{ref}} - \epsilon, T_{\text{ref}} + \epsilon)$, donde ϵ es una tolerancia de error para la temperatura final que debemos fijar previamente. En vista de las medidas experimentales, un valor razonable para esta tolerancia es ϵ =0.2 mK. Por otro lado, no es posible simular la evolución de la temperatura en un tiempo infinito; sin embargo, basta con que la temperatura se mantenga estable durante el tiempo que se va a tardar en tomar las medidas, que en el caso de las más largas es aproximadamente 1h30' (5400 s). No obstante, para facilitar la generación de datos, se mantendrá el tiempo de simulación en 5000 s, y se tomará como parámetro de optimalidad, denotado por t_1, el primer tiempo a partir del cual, dentro de la duración de la simulación, la diferencia entre la temperatura y su valor objetivo no excede (en valor absoluto) la tolerancia prefijada. Esta definición del parámetro de optimalidad, pese a corresponderse mejor con lo que se desea minimizar, presenta dos inconvenientes. En primer lugar, las soluciones malas tendrán el mismo valor para él, concretamente 5000.05 s (cuando ni siquiera en t=5000 s se satisfaga el criterio de calidad). En segundo lugar, debido al, relativamente grande, discretizado del tiempo en el RK4 (0.05 s), resulta que la función $t_1(T_0, T_{\text{ref}}, K, T_I, T_D)$ es localmente constante. En efecto, para unos valores $\vec{P} = (T_0, T_{\text{ref}}, K, T_I, T_D)$ de los parámetros que resulten en un cierto valor de t_1, habrá un entorno de $\mathbb{R}^5$ conteniendo al punto $\vec{P}$ en cuestión cuyos puntos resultarán en el mismo valor de t_1, ya que el valor absoluto de la diferencia entre la temperatura a tiempo t_1 (y posteriores) y la temperatura objetivo variará lo suficientemente poco como para seguir cumpliendo el criterio de calidad, y la misma diferencia a tiempo $t_1 - dt$ también se verá modificada lo suficientemente poco como para seguir incumpliéndolo. Todo esto puede ser un problema importante a la hora de aplicar un descenso de gradiente para encontrar el mínimo valor de t_1, ya que al ser el gradiente nulo en todos los puntos de dicho entorno, no daremos ningún paso, sino que nos mantendremos en el mismo punto eternamente. Por tanto, para solucionar esto, a la hora de calcular numéricamente las derivadas parciales que componen el gradiente según la expresión:

$$\frac{\partial t_1}{\partial P_i} = \frac{t_1(\vec{P} + h\hat{e}_i) - t_1(\vec{P})}{h} \tag{16}$$

habrá que cerciorarse de tomar un valor de h suficientemente grande como para que nos saque del entorno de constancia en al menos una de las 5 direcciones cartesianas de $\mathbb{R}^5$, de modo que el gradiente no sea nulo. Por otro lado, también será necesario asegurarse de que el punto inicial desde el que comienza el descenso de gradiente no corresponde a una de las soluciones malas descritas anteriormente (las que resultan en t_1=5000.05 s), pues entonces esta región de constancia local será demasiado grande como para poder salir de ella con un valor razonable de h. Sin embargo, además de corresponderse mucho mejor con lo que se desea, esta definición del parámetro de optimalidad excluye soluciones inaceptables con oscilaciones excesivas que, con el parámetro I_2, podrían pasar por razonables siempre y cuando se llegara a las proximidades de la temperatura objetivo lo suficientemente rápido.

Obsérvese que, con el parámetro de optimalidad I_2 considerado en el apartado anterior, no teníamos ninguno de estos problemas, pues el único discretizado existente es el del almacenamiento de variables en la memoria del ordenador, y es suficientemente fino como para ser

irrelevante (pues habría que evaluar la función I_2 en un punto de $\mathbb{R}^5$ extremadamente cercano al actual para obtener un gradiente nulo, y esto en la práctica no ocurrirá nunca).

C. Código para resolver el modelo integro-diferencial del control PID y generar los datos de entrenamiento

```
import numpy as np
import matplotlib.pyplot as plt

kC = 16.64
n3dot = 0.6e-3
Pext = 5.12e-6
fT = 1.1
Pheater = 0
Phmax = 0.003

Dt = 5000
errT = 2e-4

def Qdot(T):
    return (95-11*fT**2)*n3dot*T**2

def f(T,t):
    return 1/(kC*T) * (Pext+Pheater-Qdot(T))

def RK4(f, t0, tf, x0, N):
    dt = (tf-t0)/(N-1)
    tList = np.linspace(t0, tf, N)
    xList = np.zeros(N)
    xList[0] = x0
    for i in range(N-1):
        k1 = f(xList[i], tList[i])
        k2 = f(xList[i] + k1*dt/2, tList[i] + dt/2)
        k3 = f(xList[i] + k2*dt/2, tList[i] + dt/2)
        k4 = f(xList[i] + k3*dt, tList[i] + dt)
        xList[i+1] = xList[i] + 1/6*dt*(k1+2*k2+2*k3+k4)

    return (tList, xList)

N = 101
tMeas = 5.0

dt = tMeas/(N-1)

t0 = 0
tf = 5000

n = (int)((tf-t0)/tMeas)

tlist = np.zeros(n*(N-1)+1)
Tlist = np.zeros(n*(N-1)+1)

tlistAux = np.zeros(N)
```

```
TlistAux = np.zeros(N)

tsamples = np.zeros(n)
Tsamples = np.zeros(n)

def paramValue(T0,Tref,K,TI,TD,tT,errT):

    global Pheater

    x0 = T0
    y0 = 1/(kC*x0)*(Pext+K*(Tref-x0)-(95-11*fT**2)*n3dot*x0**2) / (1+K*TD/(kC*x0))
    Pheater = K*((Tref-x0)-TD*y0)
    if (Pheater < 0):
        Pheater = 0
    elif (Pheater > Phmax):
        Pheater = Phmax

    integral = 0
    integralE2 = 0

    tlist[0] = t0
    Tlist[0] = x0

    t1 = t0

    for i in range(n):
        (tlistAux, TlistAux) = RK4(f, 0, tMeas, x0, N)
        for j in range(N-1):
            tlist[i*(N-1)+j+1] = i*tMeas + tlistAux[j+1]
            Tlist[i*(N-1)+j+1] = TlistAux[j]
            integralE2 += (Tref-TlistAux[j])*(Tref-TlistAux[j])*dt
            if (np.fabs(Tref-TlistAux[j]) > errT and tlist[i*(N-1)+j+1]-t1 < Dt):
                t1 = tlist[i*(N-1)+j+1] + dt
        integral += (Tref-TlistAux[N-1])*tMeas
        x0 = TlistAux[N-1]
        y0 = (TlistAux[N-1]-TlistAux[N-2])/dt

        Pheater = K*((Tref-x0)+1.0/TI*integral-TD*y0)
        if (Pheater < 0):
            Pheater = 0
        elif (Pheater > Phmax):
            Pheater = Phmax

        tsamples[i] = i*tMeas + tlistAux[N//2]
        Tsamples[i] = TlistAux[N//2]

    return (integralE2, t1)

filep = open("DatosModelo.txt","w")

filep.write("t0 = %f s\n" % t0)
```

```
filep.write("tf = %f s\n" % tf)
filep.write("tMeas = %f s\n" % tMeas)
filep.write("dt = %f s\n" % dt)
filep.write("n = %i\n" % n)
filep.write("N = %i\n" % N)
filep.write("T0(K)\tTref(K)\tK(W/K)\tTI(s)\tTD(s)\tI(K^2)\tt1(s)\n")

n1par = 6

for ctr1 in range(n1par):
    T0 = 0.01+0.03*ctr1
    for ctr2 in range(n1par):
        Tref = 0.02+0.038*ctr2
        for ctr3 in range(n1par):
            K = 0.01+0.12*ctr3
            for ctr4 in range(n1par):
                TI = 30.0+150.0*ctr4
                for ctr5 in range(n1par):
                    TD = 0.0+0.01*ctr5

                    (integralE2,t1) = paramValue(T0,Tref,K,TI,TD,Dt,errT)

                    filep.write("%f\t%f\t%f\t%f\t%f\t%f\t%f\n" %
                        (T0,Tref,K,TI,TD,integralE2,t1))
filep.close()
```

D. Código para el caso de control continuo

```
import numpy as np
import matplotlib.pyplot as plt

kC = 16.64
n3dot = 0.6e-3
Pext = 5.12e-6
fT = 1.1
Pheater = 0
Phmax = 0.003

N = 1000001

t0 = 0
tf = 5000
T0 = 0.040
Tref = 0.060

K = 3
TI = 750
TD = 0.0

def Qdot(T):
    return (95-11*fT**2)*n3dot*T**2

integral = 0

def f2(x, y, t):
    if (K*(Tref-x + 1/TI*integral - TD*y) < 0):
        return (1/(kC*x)*(Pext-(95-11*fT**2)*n3dot*x*x),
            -Pext/(kC*x*x)*y-(95-11*fT**2)*n3dot/kC*y)
    elif (K*(Tref-x + 1/TI*integral - TD*y) > Phmax):
        return (1/(kC*x)*(Pext+Phmax-(95-11*fT**2)*n3dot*x*x),
            -(Pext+Phmax)/(kC*x*x)*y-(95-11*fT**2)*n3dot/kC*y)
    else:
        return (y, (-K*y + K/TI*(Tref-x)- 2*(95-11*fT**2)*n3dot*x*y - kC*y*y) /
            (kC*x+K*TD))


def RK42(f, t0, tf, x0, y0, N):
    global integral
    dt = (tf-t0)/(N-1)
    tList = np.linspace(t0, tf, N)
    xList = np.zeros(N)
    xList[0] = x0
    yList = np.zeros(N)
    yList[0] = y0
    for i in range(N-1):
        Plist[i] = K*(Tref-xList[i] + 1/TI*integral - TD*yList[i])
```

```
        k1 = f2(xList[i], yList[i], tList[i])
        k2 = f2(xList[i] + k1[0]*dt/2, yList[i] + k1[1]*dt/2, tList[i] + dt/2)
        k3 = f2(xList[i] + k2[0]*dt/2, yList[i] + k2[1]*dt/2, tList[i] + dt/2)
        k4 = f2(xList[i] + k3[0]*dt, yList[i] + k3[1]*dt, tList[i] + dt)
        xList[i+1] = xList[i] + 1/6*dt*(k1[0]+2*k2[0]+2*k3[0]+k4[0])
        yList[i+1] = yList[i] + 1/6*dt*(k1[1]+2*k2[1]+2*k3[1]+k4[1])
        integral += (Tref-xList[i])*dt

    return (tList, xList)

tlist = np.zeros(N)
Tlist = np.zeros(N)
Plist = np.zeros(N)

n = 1000
nr = (N-1)//n

x0 = T0
y0 = (Pext+K*(Tref-x0)-(95-11*fT**2)*n3dot*x0**2) / (kC*x0+K*TD)
if (K*(Tref-x0 - TD*y0) < 0):
    y0 = 1/(kC*x0)*(Pext-(95-11*fT**2)*n3dot*x0*x0)
elif (K*(Tref-x0 - TD*y0) > Phmax):
    y0 = 1/(kC*x0)*(Pext+Phmax-(95-11*fT**2)*n3dot*x0*x0)


(tlist, Tlist) = RK42(f2, t0, tf, x0, y0, N)
plt.plot(tlist, Tlist)

filep = open("DatosModelo.txt","w")
filep.write("t (s)\tP (W)\T (K)\n")
for i in range(n):
    if (Plist[nr//2+nr*i]<0):
        filep.write("%f\t%f\t%f\n" % (tlist[nr//2+nr*i],0,Tlist[nr//2+nr*i]))
    elif (Plist[nr//2+nr*i]>Phmax):
        filep.write("%f\t%f\t%f\n" % (tlist[nr//2+nr*i],Phmax,Tlist[nr//2+nr*i]))
    else:
        filep.write("%f\t%f\t%f\n" %
            (tlist[nr//2+nr*i],Plist[nr//2+nr*i],Tlist[nr//2+nr*i]))
filep.close()
```

E. Código para optimizar los parámetros del PID por descenso de gradiente modificado (pesando las distintas componentes)

```
import numpy as np
import matplotlib.pyplot as plt

kC = 16.64
n3dot = 0.6e-3
Pext = 5.12e-6
fT = 1.1
Pheater = 0
Phmax = 0.003

Dt = 5000
errT = 2e-4

def Qdot(T):
    return (95-11*fT**2)*n3dot*T**2

def f(T,t):
    return 1/(kC*T) * (Pext+Pheater-Qdot(T))

def RK4(f, t0, tf, x0, N):
    dt = (tf-t0)/(N-1)
    tList = np.linspace(t0, tf, N)
    xList = np.zeros(N)
    xList[0] = x0
    for i in range(N-1):
        k1 = f(xList[i], tList[i])
        k2 = f(xList[i] + k1*dt/2, tList[i] + dt/2)
        k3 = f(xList[i] + k2*dt/2, tList[i] + dt/2)
        k4 = f(xList[i] + k3*dt, tList[i] + dt)
        xList[i+1] = xList[i] + 1/6*dt*(k1+2*k2+2*k3+k4)

    return (tList, xList)

N = 101
tMeas = 5.0  # 5

dt = tMeas/(N-1)

t0 = 0
tf = 5000

n = (int)((tf-t0)/tMeas)

tlist = np.zeros(n*(N-1)+1)
Tlist = np.zeros(n*(N-1)+1)

tlistAux = np.zeros(N)
```

```
TlistAux = np.zeros(N)

tsamples = np.zeros(n)
u = np.zeros(n)
Tsamples = np.zeros(n)

def intValue(T0,Tref,K,TI,TD):

    global Pheater

    x0 = T0
    y0 = 1/(kC*x0)*(Pext+K*(Tref-x0)-(95-11*fT**2)*n3dot*x0**2) / (1+K*TD/(kC*x0))
    Pheater = K*((Tref-x0)-TD*y0)
    if (Pheater < 0):
        Pheater = 0
    elif (Pheater > Phmax):
        Pheater = Phmax

    integral = 0
    integralE2 = 0

    tlist[0] = t0
    Tlist[0] = x0

    for i in range(n):
        (tlistAux, TlistAux) = RK4(f, 0, tMeas, x0, N)
        for j in range(N-1):
            tlist[i*(N-1)+j+1] = i*tMeas + tlistAux[j+1]
            Tlist[i*(N-1)+j+1] = TlistAux[j]
            integralE2 += (Tref-TlistAux[j])*(Tref-TlistAux[j])*dt
        integral += (Tref-TlistAux[N-1])*tMeas
        x0 = TlistAux[N-1]
        y0 = (TlistAux[N-1]-TlistAux[N-2])/dt

        Pheater = K*((Tref-x0)+1.0/TI*integral-TD*y0)
        if (Pheater < 0):
            Pheater = 0
        elif (Pheater > Phmax):
            Pheater = Phmax

        tsamples[i] = i*tMeas + tlistAux[N//2]
        u[i] = Pheater
        Tsamples[i] = TlistAux[N//2]

    return integralE2

def t1Value(T0,Tref,K,TI,TD,tT,errT):

    global Pheater

    x0 = T0
```

```
    y0 = 1/(kC*x0)*(Pext+K*(Tref-x0)-(95-11*fT**2)*n3dot*x0**2) / (1+K*TD/(kC*x0))
    Pheater = K*((Tref-x0)-TD*y0)
    if (Pheater < 0):
        Pheater = 0
    elif (Pheater > Phmax):
        Pheater = Phmax

    integral = 0
    #integralE2 = 0

    tlist[0] = t0
    Tlist[0] = x0

    t1 = t0

    for i in range(n):
        (tlistAux, TlistAux) = RK4(f, 0, tMeas, x0, N)
        for j in range(N-1):
            tlist[i*(N-1)+j+1] = i*tMeas + tlistAux[j+1]
            Tlist[i*(N-1)+j+1] = TlistAux[j]
            if (np.fabs(Tref-TlistAux[j]) > errT and tlist[i*(N-1)+j+1]-t1 < Dt):
                t1 = tlist[i*(N-1)+j+1] + dt
            #integralE2 += (Tref-TlistAux[j])*(Tref-TlistAux[j])*dt
        integral += (Tref-TlistAux[N-1])*tMeas
        x0 = TlistAux[N-1]
        y0 = (TlistAux[N-1]-TlistAux[N-2])/dt

        Pheater = K*((Tref-x0)+1.0/TI*integral-TD*y0)
        if (Pheater < 0):
            Pheater = 0
        elif (Pheater > Phmax):
            Pheater = Phmax

        tsamples[i] = i*tMeas + tlistAux[N//2]
        u[i] = Pheater
        Tsamples[i] = TlistAux[N//2]

    return t1

def OptimizePID(T0,Tref,K,TI,TD):

    h = 0.000001
    lr = 100.0


    for i in range(10):
        intVal = intValue(T0,Tref,K,TI,TD)
        print("int=%f\t" % intVal)
        print("K=%f\tTI=%f\tTD=%f\t" % (K, TI, TD))
        print("lr=%f\n" % lr)
```

```
        dK = (intValue(T0, Tref, K+h, TI, TD)-intVal)/h
        dTI = (intValue(T0, Tref, K, TI+h, TD)-intVal)/h
        dTD = (intValue(T0, Tref, K, TI, TD+h)-intVal)/h

        fK = 10
        fTI = 5000
        fTD = 0.01

        while (intValue(T0, Tref, K - fK*lr*dK, TI - fTI*lr*dTI, TD - fTD*lr*dTD) >
            intVal):
            lr = lr/2

        K = K - fK*lr*dK
        TI = TI - fTI*lr*dTI
        TD = TD - fTD*lr*dTD

        if (K<0):
            K = K + fK*lr*dK
        if (TI<0):
            TI = TI + fTI*lr*dTI
        if (TD<0):
            TD = TD + fTD*lr*dTD

    return (K,TI,TD)

def Optimizet1(T0,Tref,K,TI,TD,Dt,errT):

    h = 0.001
    lr = 100

    for i in range(10):
        t1Val = t1Value(T0,Tref,K,TI,TD,Dt,errT)
        print("t1=%f s\t" % t1Val)
        print("K=%f\tTI=%f\tTD=%f\t" % (K, TI, TD))
        print("lr=%f\n" % lr)

        fK = 10
        fTI = 5000
        fTD = 0.01

        dK = (t1Value(T0, Tref, K+h, TI, TD, Dt, errT)-t1Val)/h
        dTI = (t1Value(T0, Tref, K, TI+fTI/100*h, TD, Dt, errT)-t1Val)/(fTI/100*h)
        dTD = (t1Value(T0, Tref, K, TI, TD+h, Dt, errT)-t1Val)/h

        while (t1Value(T0, Tref, K - fK*lr*dK, TI - fTI*lr*dTI, TD - fTD*lr*dTD, Dt,
            errT) > t1Val):
            lr = lr/2

        K = K - fK*lr*dK
        TI = TI - fTI*lr*dTI
        TD = TD - fTD*lr*dTD
```

```
        if (K<0):
            K = K + fK*lr*dK
        if (TI<0):
            TI = TI + fTI*lr*dTI
        if (TD<0):
            TD = TD + fTD*lr*dTD

    return (K,TI,TD)

T0 = 0.110
Tref = 0.050

K = 0.606770
TI = 2913.318982
TD = 0.026243

(K,TI,TD) = OptimizePID(T0,Tref,K,TI,TD)
print("K=%f\tTI=%f\tTD=%f\n" % (K, TI, TD))
print("%f\n" % intValue(T0,Tref,K,TI,TD))

(K,TI,TD) = Optimizet1(T0, Tref, K, TI, TD, Dt, errT)
print("K=%f\tTI=%f\tTD=%f\n" % (K, TI, TD))
print("t1=%f\n" % t1Value(T0,Tref,K,TI,TD,Dt,errT))
```