



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE QUÍMICA

**USO DE ARDUINO EN EL APRENDIZAJE DE DINÁMICA Y
CONTROL DE PROCESOS**

TESINA

**QUE PARA OBTENER EL TÍTULO DE:
INGENIERO QUÍMICO**

PRESENTA:

ADÁN ISRAEL LARA AGUIRRE

ASESOR:

DR. MODESTO JAVIER CRUZ GÓMEZ

Ciudad Universitaria, CDMX AÑO 2023





Universidad Nacional
Autónoma de México



UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

JURADO ASIGNADO:

PRESIDENTE: **Profesor: Montiel Maldonado Celestino**

VOCAL: **Profesor: Jiménez Bedolla Juan Carlos**

SECRETARIO: **Profesor: Cruz Gómez Modesto Javier**

1er. SUPLENTE: **Profesor: Pérez Salinas Patricia**

2° SUPLENTE: **Profesor: Talavera Rosales Marco Antonio**

SITIO DONDE SE DESARROLLÓ EL TEMA

Laboratorio E-212

Facultad de Química UNAM

ASESOR DEL TEMA:



Modesto Javier Cruz Gómez

SUSTENTANTE (S):

Lara Aguirre Adán Israel

AGRADECIMIENTOS

ÍNDICE

1	RESUMEN.....	1
2	INTRODUCCIÓN	1
3	OBJETIVO.....	2
4	MARCO TEÓRICO	2
4.1	Laboratorio de control de temperatura TCL	2
4.2	Dinámica y control de procesos en TCL	5
4.2.1	Balances de energía del sistema TCL	5
4.2.2	Señales de entrada	9
4.2.3	Modelos de primer orden del TCL.....	9
4.2.4	Modelos de segundo orden del TCL.....	11
4.2.5	Optimización	14
4.2.6	Control	14
4.2.7	Sintonización y Correlaciones de Ajuste.....	17
4.2.8	Control solo Proporcional (P)	18
4.2.9	Control Proporcional Integral (PI).....	19
4.2.10	Control proporcional integral derivativo (PID).....	20
4.2.11	Control feedforward (ff).....	21
4.2.12	Control en cascada.....	22
4.2.13	Diagrama de bloques	23
4.3	Python	25
4.3.1	SciPy 	25
4.3.2	Gekko 	25
5	Programas.....	26
5.1	Simulación de respuesta de paso ODE	26
5.2	Transferencia de calor por convección	29
5.3	Transferencia de calor radiactivo	30
5.4	Balance de energía linealizado	32
5.5	FOPDT ajuste gráfico	33
5.6	FOPDT regresión.....	35
5.7	Control Proporcional P.....	38
5.8	Control proporcional integral PI.....	41
5.9	Control proporcional integral derivativo PID.....	42
5.10	Sintonización de control PI	43

5.11	Sintonización de control PID	46
5.12	PID con retroalimentación	48
5.13	Modelo FOPDT para el calentador 2 (Q2)	50
5.14	Prueba de sensores de temperatura.....	52
5.15	Diagrama de bloques.....	53
5.16	Espacio de estado	56
5.17	Control On-Off.....	59
5.18	Regresión en tiempo real de segundo orden	62
5.19	Regresión de tercer orden	67
5.20	Estabilidad y oscilaciones.....	72
5.21	Control en cascada.....	75
6	Conclusión.....	77
7	Apéndice.....	78
A.	Lista de carpeta de programas	78
8	Bibliografía.....	78

1 RESUMEN

Los cursos tradicionales de dinámica y control están limitados a la teoría y moderadamente a la utilización de software para la simulación, el modelado dinámico y control de procesos, dejando a docentes y estudiantes sin la posibilidad de poner en práctica los conceptos de la asignatura, por lo tanto, la utilización de un dispositivo portátil que nos permita experimentar y obtener datos en una computadora, se vuelve una herramienta fundamental en el desarrollo de estudiantes de ingeniería. En el presente trabajo se utilizó un curso online de APMonitor de dinámica y control de procesos (Hedengren, apmonitor, 2022), en el que se utiliza un sistema térmico denominado TCL (*temperature control lab*). Este sistema está montado en una placa Arduino que cuenta con dos calentadores y dos sensores de temperatura, dispositivos suficientes para modelar, controlar y diseñar sistemas de control de temperatura básicos.

Los programas analizados fueron desarrollados en el lenguaje de programación Python en los que se procedió a la revisión exhaustiva del código y se proporcionó una explicación detallada de cada línea programada. Los programas fueron puestos a prueba y se analizaron los resultados obtenidos. Toda la información se introdujo en una carpeta de *Dropbox* (*Apéndice: A*), el cual contiene los programas resueltos con el TCL utilizado en este trabajo. Con la guía adecuada el TCL es el complemento perfecto para el estudio de dinámica y control de procesos, es transportable y fácil de utilizar, la obtención de datos y gráficos es esclarecedor incluso en conceptos complejos. Además, al estar programado en Python es accesible debido a que es un lenguaje de programación de código abierto.

2 INTRODUCCIÓN

La experimentación es una actividad fundamental para el aprendizaje de dinámica y control de procesos. Haciendo uso de TCL es posible que los estudiantes tengan a la mano un laboratorio portátil capaz de funcionar como un sistema de control de temperatura. Además, el lenguaje de programación utilizado es Python, que actualmente es uno de los lenguajes de programación más utilizados y con una de las comunidades más grandes a nivel mundial. Al implementar el uso de TCL en problemas de dinámica y control de temperatura, se logra que el usuario tenga una experiencia práctica permitiendo un aprendizaje completo de este tema. Se ha utilizado una serie de videos de APMonitor.com siguiendo el orden de una lista de reproducción de YouTube llamada "TCLab Dynamics and Control". El capítulo 4 contiene el marco teórico. Comienza con una descripción del TCL, sus accesorios y funcionamiento. La información de dinámica y control de procesos necesaria para todos los programas y el lenguaje de programación utilizado Python. El capítulo 5 contiene 21 aplicaciones que se corresponden con los programas analizados (ver Anexo A). En esta sección se encuentra la información necesaria y los resultados obtenidos con el TCL, que darán al lector datos y una descripción detallada de cada programa.

3 OBJETIVO

Implementar el uso de TCL en problemas de dinámica y control de temperatura, para el aprendizaje integral en dinámica y control de procesos.

4 MARCO TEÓRICO

4.1 Laboratorio de control de temperatura TCL

El TCL es un sistema térmico que contiene dos sensores que miden la temperatura y dos calentadores. La temperatura es la variable de salida o variable que se desea controlar, mientras que el porcentaje de calentador es la variable de entrada o manipulada. Es por eso que para la generación de modelos se parte de un sistema con entradas Q_i y salidas T_i (Fig.1). La variable Q puede ser generada por el calentador 1 (Q_1) o el calentador 2 (Q_2). T puede ser medida por el sensor 1 (T_1) o por el sensor 2 (T_2) (Fig.2).

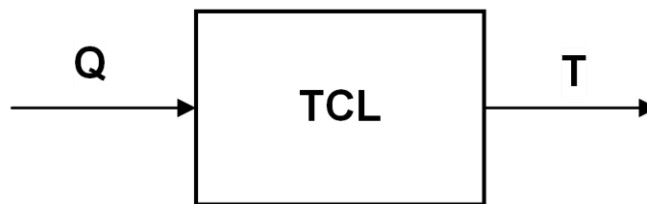


Figura 1. Sistema TCL.

- Q : porcentaje del calentador (variable entrada o de control).
- T : valor de temperatura (variable de salida o de proceso).
- TCL: sistema de control de temperatura



Figura 2. Dispositivo TCL conexiones con ordenador y a la corriente eléctrica. Extraída de: https://apmonitor.com/pds/notebooks/TCLab_Help.html

Para poder hacer uso del TCL es necesario instalar el paquete dentro del ambiente de trabajo. Esto puede hacerse abriendo anaconda, después Jupyter Notebook y pegando en una celda de código lo siguiente:


```

1 #Instalar tclab
2 try:
3     import tclab
4 except:
5     !pip install --user pyserial
6     !pip install --user tclab
7     import tclab

```

Calentadores

Los calentadores en TCL son transistores de unión bipolar NPN (negativo, positivo negativo, ver Fig. 3) lo que significa que la base está cargada positivamente para controlar el flujo de corriente del colector al emisor, además cuentan con un encapsulado TO-220 lo que le permite manejar potencias altas y gracias al reverso metálico con un agujero en la parte superior, permite montar un disipador. Debido a que el TIP31C disipa energía, actúa como calentador e interruptor. El calentador TIP31C se controla a través de modulación de ancho de pulso (PWM: *pulse width modulation*), lo cual ayuda que la señal transmitida no pierda potencia o sufra distorsiones por interferencias.

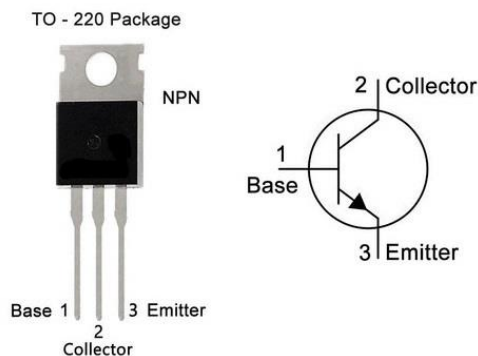


Figura 3. Calentador y diagrama de flujo de corriente.

Sensores de temperatura

Los sensores de temperatura utilizados en TCL son los TMP36GZ, en los cuales el voltaje de salida es linealmente proporcional a la temperatura como se muestra en la función (1). La precisión del sensor es $\pm 1^{\circ}C$ a $25^{\circ}C$ y $\pm 2^{\circ}C$ en el rango $-40^{\circ}C$ a $150^{\circ}C$. El TMP36 utiliza la propiedad de los diodos que cambian repetidamente el voltaje con la temperatura. El sensor mide el pequeño cambio y emite un voltaje DC (*corriente continua*) analógico entre 0 y 1,75 V. El mV se lee en el Arduino con un ADC (*Analog to digital converter*).

El ADC de Arduino trabaja con un voltaje de 3.3V (3300mV) con resolución de 10 bits. Cada bit contiene dos valores (0 y 1) por lo tanto habrá 2^{10} DL (*discrete levels*) o 1024 DL.

Empezando en 0 y siendo el ultimo valor 1023, el ADC de Arduino devolverá un valor en 0y 1023 que es transformado a milivoltios con $mV = DL * \frac{3300}{1024}$. El 0 DL corresponde a 0 mV y 1023 DL corresponden a 3300 mV. Después dentro de la programación en Arduino se pasa de mV a temperatura en grados centígrados con la ecuación (1) que será la temperatura leída en ese instante o variable de salida.

$$T^{\circ}C = 0.1mV - 50 \quad (1)$$

La siguiente celda contiene una sección de la programación interna del TCL realizada en C para la toma de temperatura con los sensores. Lo que se hace en el ciclo *for* (línea 6) es indicarle que realice la lectura de la temperatura n veces y sumarlas, notar que en las líneas 7 y 8 se está convirtiendo de DL a mV y de mV a °C en cada una de las iteraciones, el valor de n es 10 dentro de la programación en Arduino por lo tanto la temperatura se tomara 10 veces. Al salir del ciclo (línea 10) vemos que se saca el promedio de las 10 lecturas, con la relación $\frac{\text{suma de lecturas en } ^{\circ}C}{\text{numero flotante } n (n=10)}$, para finalmente imprimir la temperatura promedio del TCL.

```

1 // Arduino code tclab.ino
2 const int pinT1 = 0;
3 const int pinT2 = 2;
4 float mV = 0.0;
5 float degC = 0.0;
6 for (int i = 0; i < n; i++) {
7     mV = (float) analogRead(pinT1) * (3300.0/1024.0);
8     degC = degC + (mV - 500.0)/10.0;
9 }
10 degC = degC / float(n);
11 Serial.println(degC);

```

Uno de los sistemas de control de temperatura generados con el TCL puede verse en la Fig.4 donde se muestra la disposición de cada uno de los elementos que forman parte del dispositivo en un circuito cerrado de control.

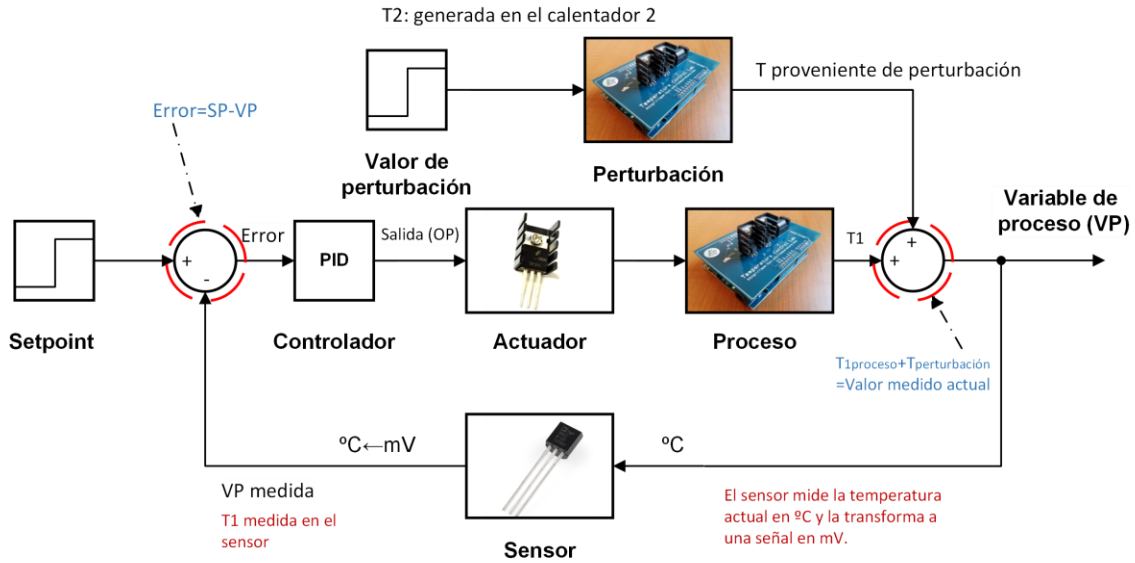


Figura 4. Sistema de control de temperatura TCL

4.2 Dinámica y control de procesos en TCL

4.2.1 Balances de energía del sistema TCL

Primer orden

En la figura 5 se muestra la transferencia de calor al encender uno de los calentadores (Q1). T_1 es el sensor de temperatura acoplado a Q1, las flechas rojas representan la transferencia de calor por convección y las flechas verdes representan la transferencia de calor por radiación.

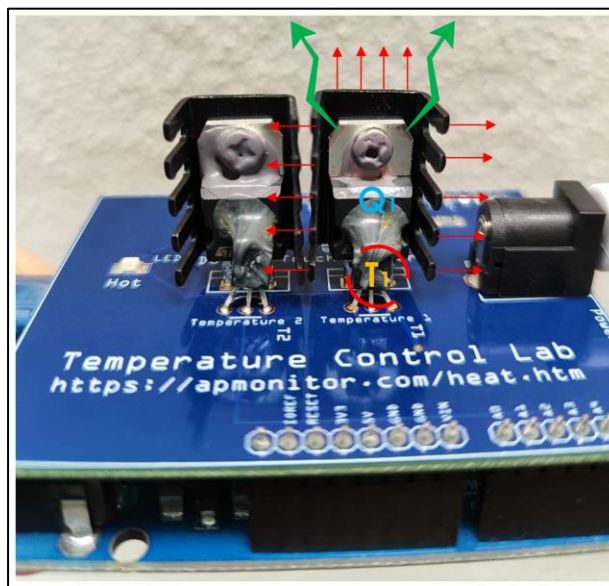


Figura 5. Relación de primer orden utilizando el calentador Q1

El balance de energía para el TCL es sobre un metal calentado por una fuente eléctrica, por lo que se considerarán solamente las pérdidas de calor por convección y radiación hacia los alrededores y la producción en la fuente de calentamiento que está dada por: αQ donde α es la relación entre la potencia del calentador y el porcentaje de salida del calentador y Q es el porcentaje de salida del calentador (0-100%).

Para esto se utiliza un balance de energía simplificado del TCL:

$$\text{Acumulación} = \text{producción} - \text{perdida} \quad (2)$$

Si se consideran solamente las pérdidas por convección natural la ecuación (2) queda:

$$mc_p \frac{dT}{dt} = \alpha Q - UA(\Delta T) \quad (3)$$

El término convectivo es:

$$UA(\Delta T) \quad (4)$$

- U : coeficiente global de transferencia de calor.
- A : área.

Si el $\Delta T = T - T_a$, donde T_a es la temperatura ambiente y T es la temperatura del calentador, la ecuación quedaría:

$$mc_p \frac{dT}{dt} = \alpha Q - UA(T - T_a) \quad (5)$$

Reacomodando el signo negativo del término convectivo en (5) se obtiene la siguiente ecuación:

$$mc_p \frac{dT}{dt} = UA(T_a - T) + \alpha Q \quad (6)$$

Para considerar la transferencia de calor radiación se añade al balance el siguiente término, proveniente de la ley de Stefan-Boltzman:

$$\epsilon \sigma A (T_{\infty}^4 - T^4) \quad (7)$$

- ϵ : Es un número adimensional que relaciona la radiación emitida por una superficie contra la radiación que emitiría un cuerpo negro.
- σ : Constante de Stefan-Boltzmann. ($5.676 \times 10^{-8} W / m^2 \cdot K^4$).
- T_{∞}^4 : Temperatura objetos circundantes.

Finalmente se obtiene una ecuación diferencial de primer orden, no lineal, que representa el balance de energía del TCL al encender un calentador:

$$mc_p \frac{dT}{dt} = UA(T_a - T) + \epsilon\sigma A(T_\infty^4 - T^4) + \alpha Q \quad (8)$$

Balance orden superior

Un modelo basado en la física de cuarto orden consta de cuatro ecuaciones diferenciales, que son balances de energía con transferencias por convección, conducción y radiación térmica. (Fig.6)

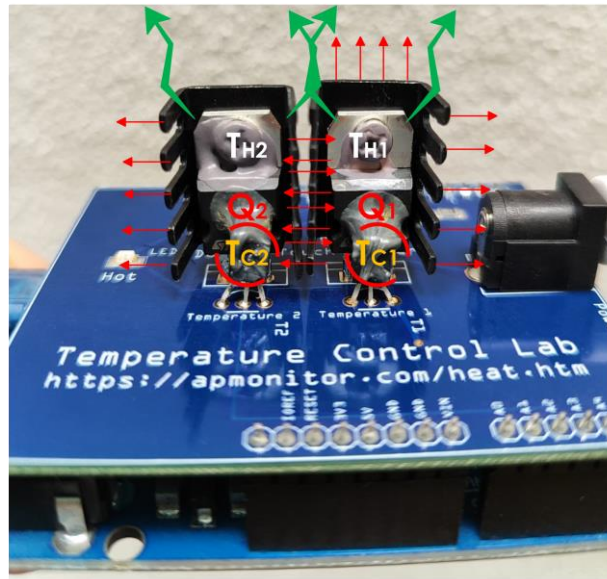


Figura 6. Relación de 4to orden utilizando dos calentadores

Un modelo basado en la física de cuarto orden consta de cuatro ecuaciones diferenciales, que son balances de energía con transferencias por convección, conducción y radiación térmica.

Tenemos que los balances de energía para los dos calentadores son:

$$mc_p \frac{dT_{H_1}}{dt} = UA(T_\infty - T_{H_1}) + \epsilon\sigma A(T_\infty^4 - T_{H_1}^4) + Q_{C_{12}} + Q_{R_{12}} + \alpha_1 Q_1 \quad (9)$$

$$mc_p \frac{dT_{H_2}}{dt} = UA(T_\infty - T_{H_2}) + \epsilon\sigma A(T_\infty^4 - T_{H_2}^4) - Q_{C_{12}} - Q_{R_{12}} + \alpha_2 Q_2 \quad (10)$$

Y si se realiza un balance de calor por conducción en los sensores de temperatura 1 y 2 tendremos:

$$\tau \frac{dT_{C_1}}{dt} = T_{H_1} - T_{C_1} \quad (11)$$

$$\tau \frac{dT_{C_2}}{dt} = T_{H_2} - T_{C_2} \quad (12)$$

Para simplificar las expresiones Q_{R12} : calor radiactivo del calentador 1 al 2, y Q_{C12} : calor convectivo del calentador 1 al 2. Se proporcionan los siguientes términos.

$$\begin{aligned} Q_{C12} &= U_s A_s (T_{H2} - T_{H1}) \\ Q_{R12} &= \epsilon \sigma A_s (T_{H2}^4 - T_{H1}^4) \\ \tau &= \frac{m_s c_{p,s} \Delta x}{k A_c} \end{aligned} \quad (13)$$

Tabla 1. Variables y constantes del balance de energía de orden superior del TCL.

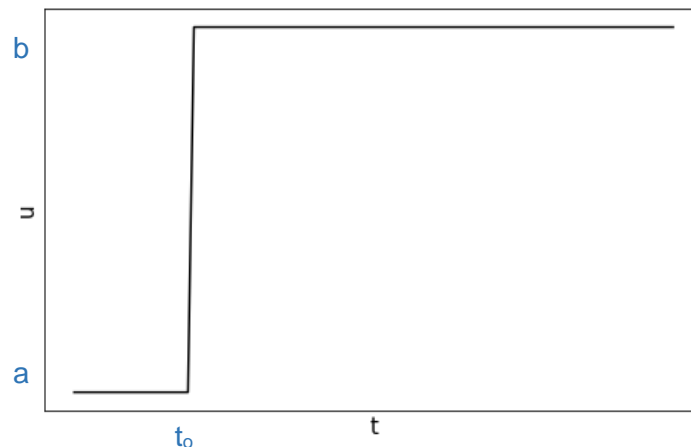
Variable	Nombre	Unidades
T_o	Temperatura inicial	K
T_∞	Temperatura ambiente	K
Q_1	Salida del calentador 1	0 – 100 %
Q_2	Salida del calentador 2	0 – 100 %
α_1	Factor del calentador 1	$W/\%calentador$
α_2	Factor del calentador 2	$W/\%calentador$
C_p	Capacidad calorífica	$J/(kg - K)$
A	Área de superficie no entre calentadores	m^2
A_s	Área de superficie entre calentadores	m^2
m	Masa	kg
τ	Constante tiempo de conducción	s
U	Coficiente general de transferencia de calor	$W/(m^2s)$
U_s	Coficiente general de transferencia de calor entre calentadores	$W/(m^2s)$
ϵ	Emisividad	
σ	Constante de Stefan Boltzman	$W/(m^2s)$
m_s	Masa del sensor	kg
$C_{p,s}$	Capacidad calorífica del sensor	$J/(kg - K)$
Δx	Distancia del calentador al sensor	m
A_c	Área de contacto calentador-sensor	m^2

4.2.2 Señales de entrada

Entrada escalón

La señal de entrada del tipo escalón se genera con un cambio abrupto de la variable de entrada. El sistema sufrirá un cambio que se verá reflejado en la variable de salida. Esto nos permite conocer la respuesta del sistema debido a un cambio del tipo escalón.

$$u(t) = \begin{cases} a & \text{si } t < 0 \\ b & \text{si } t \geq t_0 \end{cases}$$



Donde $u(t)$ es un escalón unitario y b es la amplitud del escalón. En TCL el escalón puede generarse por un cambio repentino del porcentaje del calentador y mantenerlo constante durante un periodo de tiempo.

4.2.3 Modelos de primer orden del TCL

Linealización de balance de energía no lineal

Es el proceso por el cual aproximamos los sistemas no lineales con los lineales. Se usa ampliamente en el estudio de la dinámica de procesos y el diseño de sistemas de control por las siguientes razones (Stephanopoulos, Linearization of systems with one variable, 1984):

1. Podemos tener soluciones analíticas de forma cerrada para sistemas lineales. Así podemos tener una imagen completa y general del comportamiento de un proceso independientemente de los valores particulares de los parámetros y variables de entrada. Esto no es posible para los sistemas no lineales, y la simulación por computadora nos proporciona solo el comportamiento del sistema en valores específicos de entradas y parámetros.
2. Todos los desarrollos significativos hacia el diseño de sistemas de control efectivos se han limitado a procesos lineales.

Vemos que el balance expresado en (8) es no lineal por el termino de T^4 , por lo tanto, despejando $\frac{dT}{dt}$ nos queda la ecuación siguiente:

$$\frac{dT}{dt} = f(T, Q) = \frac{UA(T_a - T) + \epsilon\sigma A(T_\infty^4 - T^4) + \alpha Q}{mc_p} \quad (14)$$

Si aplicamos series de Taylor:

$$\frac{dT}{dt} = f(\bar{T}, \bar{Q}) + \frac{\partial f}{\partial T} \Big|_{\bar{T}, \bar{Q}} (T - \bar{T}) + \frac{\partial f}{\partial Q} \Big|_{\bar{T}, \bar{Q}} (Q - \bar{Q}) \quad (15)$$

El primer término se hace cero pues es la función evaluada en el estado estacionario. T en estado estacionario será la temperatura al inicio que es la temperatura ambiente (T_a). Y Q al inicio esta apagado, por lo tanto, será igual a cero ($Q=0$).

$$\frac{dT}{dt} = \frac{\partial f}{\partial T} \Big|_{\bar{T}, \bar{Q}} (T - \bar{T}) + \frac{\partial f}{\partial Q} \Big|_{\bar{T}, \bar{Q}} (Q - \bar{Q}) \quad (16)$$

Si derivamos la ecuación (14) con respecto a la temperatura:

$$\gamma = \frac{\partial f}{\partial T} \Big|_{\bar{T}, \bar{Q}} = -\frac{UA}{mc_p} - \frac{4\epsilon\sigma AT^3}{mc_p} \quad (17)$$

Ahora (14) con respecto al calor:

$$\beta = \frac{\partial f}{\partial Q} \Big|_{\bar{T}, \bar{Q}} = \frac{\alpha}{mc_p} \quad (18)$$

A la diferencia entre una variable menos la variable en estado estacionario se le denomina variable de desviación, por lo que se definirán variables de desviación de la siguiente manera:

$$\begin{aligned} T' &= T - \bar{T} \\ Q' &= Q - \bar{Q} \end{aligned} \quad (19)$$

Donde \bar{T} y \bar{Q} son variables de estado estacionario.

Sustituyendo (17), (18) y las variables de desviación (19) en la ecuación (16) se obtiene la ecuación lineal simplificada:

$$\frac{dT'}{dt} = \gamma T' + \beta Q' \quad (20)$$

First-Order Plus Deadtime (FOPDT)

Para un modelo dinámico se puede hacer uso de un modelo de primer orden más tiempo muerto, que puede observarse en la siguiente ecuación diferencial.

$$\tau_p \frac{dy(t)}{dt} = -y(t) + K_p u(t - \theta_p) \quad (21)$$

τ_p : constante tiempo del proceso

k_p : ganancia del proceso

θ_p : tiempo muerto del proceso

Donde τ_p es la medida de cambio del proceso, k_p es la sensibilidad de la salida con respecto a la entrada, y θ_p es el tiempo en el que el sensor mide un cambio en la variable de salida por un cambio en la variable manipulada. Las variables de salida y entrada son respectivamente: $y(t)$ y $u(t)$. Utilizando la temperatura como variable de salida y a Q como la variable de entrada (ver fig. 2), el modelo FOPDT sería:

$$\tau_p \frac{dT'}{dt} = -T' + k_p Q'(t - \theta_p) \quad (22)$$

Método gráfico FOPDT

Para poder obtener los parámetros del modelo FOPDT con el método gráfico es necesario seguir los siguientes puntos:

1. Encuentra Δy de la respuesta de paso.
2. Encuentra Δu de la respuesta de paso.
3. Calcula $k_p = \frac{\Delta y}{\Delta u}$.
4. Encontrar θ_p , tiempo muerto, en la respuesta de paso.
5. Encuentra $0.632\Delta y$ de la respuesta de paso.
6. Encuentra $t_{0.632}$ para $y(t_{0.632}) = 0.632\Delta y$ de la respuesta de paso.
7. Calcula $\tau_p = t_{0.632} - \theta_p$.

4.2.4 Modelos de segundo orden del TCL

Sistema lineal de segundo orden

Son sistemas cuyo comportamiento dinámico es representado con la siguiente ecuación diferencial de segundo orden en su forma estándar. (Babatunde, 1994)

$$\tau_s^2 \frac{d^2 y}{dt^2} + 2\zeta \tau_s \frac{dy}{dt} + y = K_p u(t) \quad (23)$$

Donde y y u son respectivamente las variables de salida y de entrada del sistema. Los parámetros son: la ganancia del proceso K_p , el factor de amortiguamiento ζ , y la constante de tiempo de segundo orden τ_s o el periodo natural de oscilación.

Para el TCL el sistema de segundo orden quedaría de la siguiente manera:

$$\tau^2 \frac{d^2 T}{dt^2} + 2\zeta \tau \frac{dT}{dt} + T = K_p Q(t) \quad (24)$$

Método gráfico para sistemas de segundo orden

1. Encontrar ΔT_1 de la respuesta escalón.
2. Encuentra ΔT_{SP} como entrada a la respuesta escalón.
3. Calcula $K_p = \frac{\Delta T_1}{\Delta T_{SP}}$
4. Calcular el factor de amortiguamiento ζ de sobrepaso (OS: *overshoot*) OS o relación de decaimiento (DR: *decay ratio*) DR. (Las ecuaciones se encuentran en "Ajuste grafico a respuesta de segundo orden.ipynb" en P17).
5. Calcula τ_s de las ecuaciones para el tiempo de subida (*rise time*) t_r , tiempo pico (*peak time*) t_p , o periodo P.

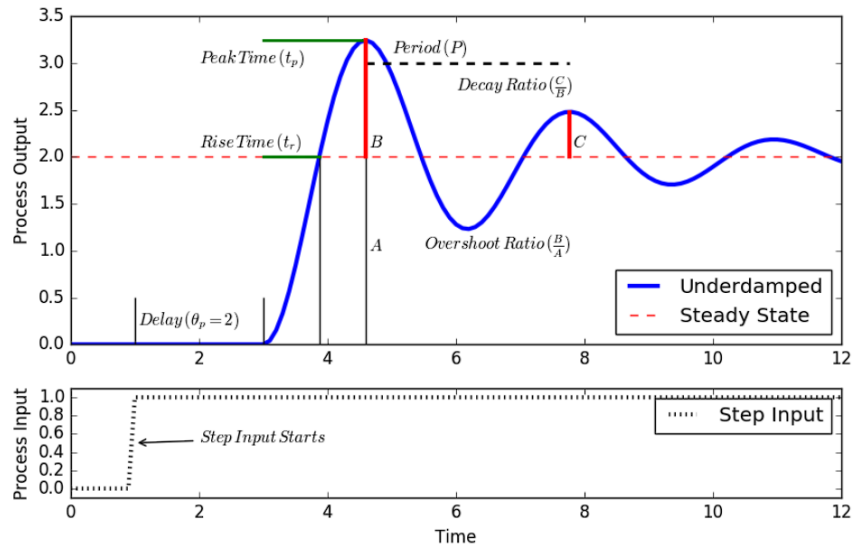


Figura 7. Ajuste gráfico para sistemas de segundo orden. Extraída de: <https://apmonitor.com/pdc/index.php/Main/TCLabOnOffControl>

Espacio de estado

Un modelo de espacio de estado es una ecuación diferencial matricial que representa a un sistema físico. Un sistema físico puede ser descrito por variables de estado de la siguiente manera:

$$\begin{aligned}\dot{x}_1 &= f_1(x_1, x_2, \dots, x_n, u_1, u_2, \dots, u_m) \\ \dot{x}_2 &= f_2(x_1, x_2, \dots, x_n, u_1, u_2, \dots, u_m) \\ &\vdots \\ \dot{x}_n &= f_n(x_1, x_2, \dots, x_n, u_1, u_2, \dots, u_m)\end{aligned}\tag{25}$$

Donde x_1, x_2, \dots, x_n son n variables de estado y u_1, u_2, \dots, u_m son m entradas. Con derivadas de estado $\dot{x} = \frac{dx}{dt} \in \mathbb{R}^n$. Visto de forma matricial sería:

$$\dot{x} = f(x, u)\tag{26}$$

Para un sistema invariante en el tiempo, podemos escribir el término general \dot{x}_i de (25) como sigue:

$$\dot{x}_i = a_{i1}x_1 + a_{i2}x_2 + \dots + a_{in}x_n + b_{i1}u_1 + \dots + b_{im}u_m\tag{27}$$

La matriz equivalente de la ecuación anterior es:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix}\tag{28}$$

Escribiendo esto en la forma matricial más compacta, tenemos:

$$\dot{x} = Ax + Bu\tag{29}$$

En esta expresión hay m entradas diferentes donde $m \leq n$. Además, tenga en cuenta que, u en esta expresión es una cantidad vectorial, no una escalar. La naturaleza del sistema físico lineal expresado por la ecuación anterior está completamente expresada por las matrices A y B . Para sistemas invariantes en el tiempo los elementos de A y B son constantes.

Las salidas de interés para el ingeniero de control pueden diferir de las variables de estado x_i . La declaración más general para relacionar la salida con la variable de estado es:

$$y = Cx + Du\tag{30}$$

Donde 'y' es el vector de salidas (y_1, y_2, \dots, y_n) elegido por el ingeniero de control por alguna razón práctica. La matriz C es una matriz (p x n) que contiene elementos constantes. La matriz D es cero porque las entradas no suelen afectar directamente a las salidas. Las dimensiones de las matrices se muestran debajo, con **m** entradas, **n** estados, y **p** salidas. (Coughanowr, State Space Representation of Physical Systems, 2009)

$$\begin{aligned} A &\in \mathbb{R}^{n \times n} \\ B &\in \mathbb{R}^{n \times m} \\ C &\in \mathbb{R}^{p \times n} \\ D &\in \mathbb{R}^{p \times m} \end{aligned} \tag{31}$$

El sistema que se analiza en este programa es un sistema no lineal, por lo que es necesario linealizar, en este caso se utilizan las series de Taylor (ver linealización en "Modelos de espacio de estado.ipynb" en [P18](#)).

4.2.5 Optimización

La optimización se ocupa de seleccionar la mejor opción entre una serie de opciones posibles que son factibles o que no violan las restricciones. Python se puede utilizar para optimizar parámetros en un modelo para ajustar mejor los datos, aumentar la rentabilidad de un diseño de ingeniería potencial o cumplir con algún otro tipo de objetivo que se pueda describir matemáticamente con variables y ecuaciones.

Al optimizar un modelo este se alinea con los datos medidos usando un solucionador. El solucionador utiliza una suma de errores cuadráticos o una suma de errores absolutos para llegar a un valor mínimo. En solucionador en Python es la función **scipy.optimize.minimize**. El cual es utilizado en esta tesina.

scipy.optimize.minimize: Es un módulo de SciPy Python que tiene un método de minimización, el cual toma una función escalar (función objetivo) de una o más variables.

4.2.6 Control

Estructuras de control

La forma en que se obtiene información del proceso para la toma de decisiones y la implementación de control se puede realizar con varias estructuras de control. A continuación, se muestra una de las más comunes

Control hacia atrás (*feedback*):

Basa su funcionamiento en la salida del sistema (y), por lo tanto, tiene que existir un cambio en la salida para que empiece a trabajar. Una de las ventajas es que no es necesario conocer a fondo el proceso por lo que el modelado es simple.

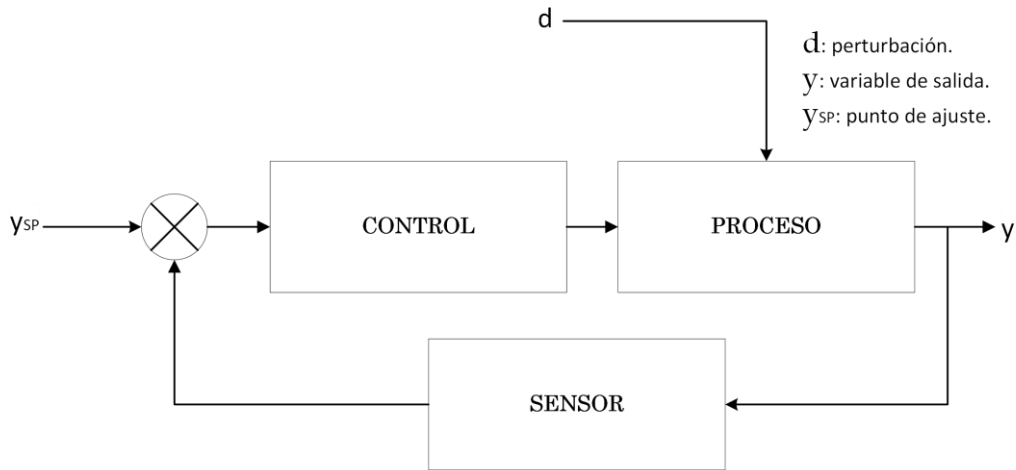


Figura 8. Diagrama de bloques control hacia atrás.

Control hacia adelante (feedforward):

Esta estructura no requiere conocer el valor a la salida para iniciar su funcionamiento por lo que la variable de entrada (Q) es la que producirá cambios en el proceso. La ventaja de esta estructura es que se adelanta a la perturbación, aunque esto complique el modelado ya que se necesita conocer profundamente el proceso y las perturbaciones que influyen en él.

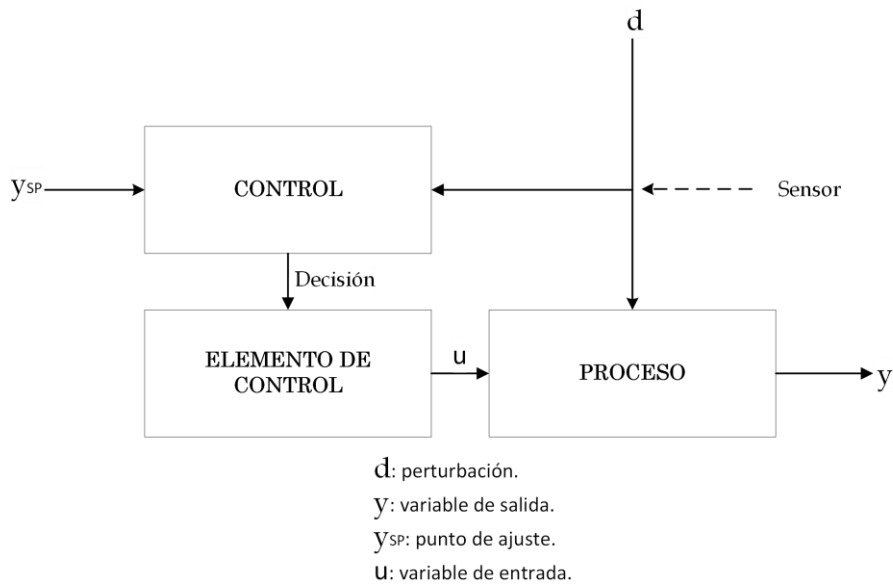


Figura 9. Diagrama de bloques control hacia adelante.

Control en cascada

En una configuración de control de cascada hay una variable manipulada y más de una medida. Está claro que con una sola manipulación podemos controlar solo una salida. La ventaja de usar un control en cascada es que las perturbaciones que surgen dentro del lazo secundario son corregidas por el control secundario antes de que puedan afectar el valor de salida primaria controlada (ver Fig. 10). (Stephanopoulos, Control Systems with Multiple Loops, 1984)

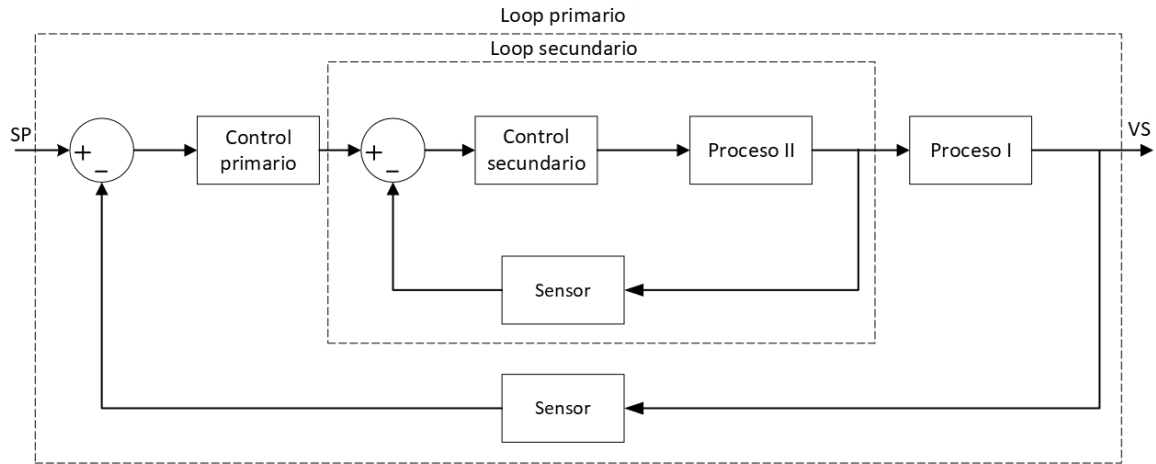


Figura 10. Representación esquemática de un control en cascada.

Diseño de control en cascada con TCL

El loop primario se compone por la variable de salida T_2 , el sensor 2, el SP de la temperatura 2 T_{2SP} y un control PID2 que recibe el error entre $T_{2SP} - T_{sensor2}$. El loop secundario tiene como entrada la T_{1SP} que es la salida del controlador PID2, un controlador PID1 que mide el error entre $T_{1SP} - T_{sensor1}$ y tiene como salida la variable manipulada Q1, esta cambiará el valor de T_1 y será medida como $T_{sensor1}$. (ver Fig. 11)

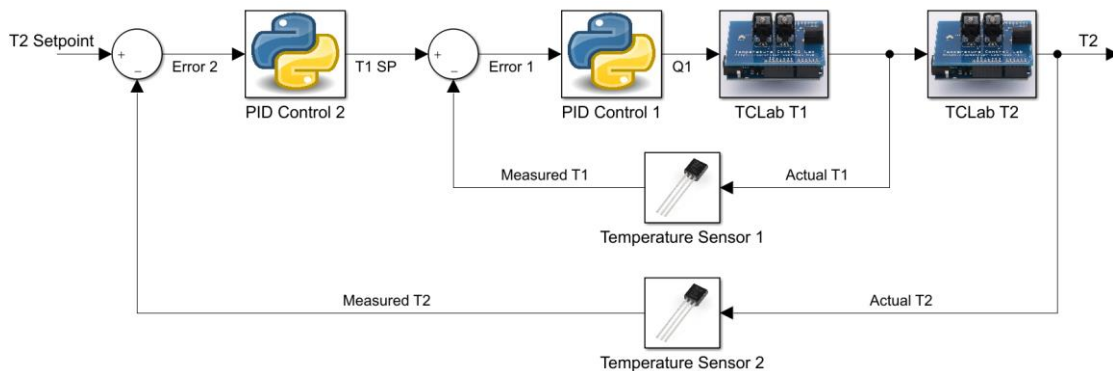


Figura 11. Diagrama de bloques en cascada del TCL. Extraída de: <https://apmonitor.com/pdc/index.php/Main/TCLabCascadeControl>

4.2.7 Sintonización y Correlaciones de Ajuste

Sintonización P, PI, PID

La sintonización consiste en buscar los mejores parámetros del controlador (K_c , τ_I y τ_D) para obtener un comportamiento ajustado a las necesidades del proceso. Por lo tanto, la selección de parámetros es un proceso de optimización. Es posible sintonizar los parámetros siguiendo un proceso de prueba y error, o bien puede usarse un proceso de optimización partiendo de un modelo matemático. Algunos criterios para elegir un buen desempeño del controlador al realizar una prueba de paso son: evitar el sobre impulso y evitar comportamiento oscilatorio. (Coughanowr, Controller Tuning and Process Identification, 2009)

La correlaciones que se muestran debajo son las que se utilizaron para resolver los problemas realizados en la tesina y han sido extraídas del curso APMonitor (Hedengren, aponitor, 2022)

Correlaciones para controladores P

ITAE

En los controladores P es comúnmente utilizada la correlación de ajuste integral de error absoluto ponderado en el tiempo (ITAE). Estas correlaciones utilizan los parámetros de FOPDT (k_p, τ_p, θ_p). El sistema que se analiza dictará que correlación será la mejor. A continuación, se muestran dos posibles arreglos ITAE.

Para el rechazo de perturbaciones o control regulatorio:

$$K_c = \frac{0.50}{K_p} \left(\frac{\tau_p}{\theta_p} \right)^{1.08} \quad (32)$$

Seguimiento del punto de ajuste (SP) o servocontrol.

$$K_c = \frac{0.20}{K_p} \left(\frac{\tau_p}{\theta_p} \right)^{1.22} \quad (33)$$

IMC

$$\text{Sintonización agresiva: } \tau_c = \max(0.1\tau_p, 0.8\theta_p) \quad (34)$$

$$\text{Sintonización moderada: } \tau_c = \max(1.0\tau_p, 8.0\theta_p) \quad (35)$$

$$\text{Sintonización conservativa: } \tau_c = \max(10\tau_p, 80.0\theta_p) \quad (36)$$

$$K_c = \frac{1}{K_p} \frac{\tau_p}{(\theta_p + \tau_c)} \quad \tau_I = \tau_p \quad (37)$$

Si se utiliza un ajuste moderado y un tiempo muerto insignificante ($\theta \rightarrow 0$ y $1.0\tau_p$) IMC se reduce a:

$$K_c = \frac{1}{K_p} \quad \tau_I = \tau_p \quad \text{correlación de sintonización simple} \quad (38)$$

Correlaciones de sintonización para PI

ITAE

$$K_c = \frac{0.586}{K_p} \left(\frac{\theta_p}{\tau_p} \right)^{-0.916} \quad \tau_I = \frac{\tau_p}{1.03 - 0.165(\theta_p/\tau_p)} \quad \text{Seguimiento del punto de ajuste} \quad (39)$$

$$K_c = \frac{0.859}{K_p} \left(\frac{\theta_p}{\tau_p} \right)^{-0.977} \quad \tau_I = \frac{\tau_p}{0.674} \left(\frac{\theta_p}{\tau_p} \right)^{0.680} \quad \text{Rechazo de perturbaciones} \quad (40)$$

IMC sintonización agresiva para PID

$$K_c = \frac{1}{K_p} \frac{\tau_p + 0.5\theta_p}{\tau_c + 0.5\theta_p} \quad \tau_I = \tau_p + 0.5\theta_p \quad \tau_D = \frac{\tau_p\theta_p}{2\tau_p + \theta_p} \quad (41)$$

4.2.8 Control solo Proporcional (P)

El controlador *feedback* más sencillo es el controlador proporcional o controlador p. Su función es recibir la señal de salida medida $y(t)$ para compararla con el punto de ajuste y_{SP} y producir la salida del controlador $u(t)$ de tal manera que regrese la salida al valor deseado y_{SP} . Por lo tanto, la entrada al controlador es el error $e(t) = y_{SP} - y(t)$. El parámetro ajustable es la ganancia del controlador K_c . Una ganancia alta produce un controlador que reacciona agresivamente a una diferencia entre la variable de proceso (PV) y la variable de ajuste o *setpoint* (SP). (Stephanopoulos, Types of feedback controllers, 1984)

Para el TCL la ecuación de control P queda de la siguiente manera:

$$Q(t) = Q_{bias} + K_c(T_{SP} - T_{PV}) = Q_{bias} + K_c e(t) \quad (42)$$

El término Q_{bias} es una constante que toma el valor del calentador $Q(t)$ al comienzo del proceso. Para el TCL el $Q_{bias} = 0$ pues comienza con el calentador apagado. El Q_{bias} proporciona una transferencia sin saltos si el error es cero cuando se enciende el controlador. T_{SP} es la temperatura SP, T_{PV} es la temperatura de la PV.

Un controlador P tiene una compensación (*offset*) persistente entre SP y PV en estado estable.

4.2.9 Control Proporcional Integral (PI)

El término integrador asegura que el error a largo plazo se acerque a cero. Este término garantiza que no habrá compensación o desviación a largo plazo del setpoint. (Roffel, 2006)

Si queremos eliminar el offset se debe introducir un modo de control adicional. Si agregamos un término integral al controlador proporcional, tenemos lo que se denomina PI, o controlador proporcional integral. De esta forma se lleva el error a cero, o visto de otro modo la eliminación del offset. Este controlador tiene dos parámetros ajustables, la ganancia y el tiempo integral (K_c, τ_c). (Coughanowr, Proportional Integral (PI) Control, 2009)

El controlador proporcional quedaría:

$$Q(t) = Q_{bias} + K_c(T_{SP} - T_{PV}) + \frac{K_c}{\tau_I} \int_0^t (T_{SP} - T_{PV}) dt \quad (43)$$

Si el error es $e(t) = T_{SP} - T_{PV}$ entonces la ecuación (43) se transforma en:

$$Q(t) = Q_{bias} + K_c e(t) + \frac{K_c}{\tau_I} \int_0^t e(t) dt \quad (44)$$

El término integral se aproxima en forma discreta como la suma del error multiplicado por tiempo de la muestra.

$$\frac{K_c}{\tau_I} \int_0^t e(t) dt \approx \frac{K_c}{\tau_I} \sum_{i=1}^{n_t} e(t) \Delta t \quad (45)$$

Las correlaciones comunes para el control PI son el método ITAE y el método IMC (*Internal Model Control*). Nuevamente las correlaciones utilizan los parámetros $K_p, \tau_p,$ y θ_p del modelo FOPDT.

Anti-Reset Windup

Un controlador con término integral puede tener una salida del controlador $u(t)$ que satura en el límite superior o inferior durante un periodo prolongado de tiempo. Esto hace que el término integral se acumule en una gran suma que hace que el controlador permanezca en el límite de saturación hasta que se reduzca la suma integral. *Anti-reset windup* (ARW: *Anti-reset windup*) hace que el término integral no se acumule si la salida del controlador está saturada en un límite superior o inferior. En la figura 12 se muestran dos ejecuciones del programa [P9](#). La curva con ARW no sobrepasa el SP, mientras que la curva sin ARW fue más agresiva, sobrepasa el SP y tarda en bajar a la temperatura SP, esto debido a la saturación de la integral. En la gráfica del calentador notamos que con ARW se mantiene menos tiempo al 100%.

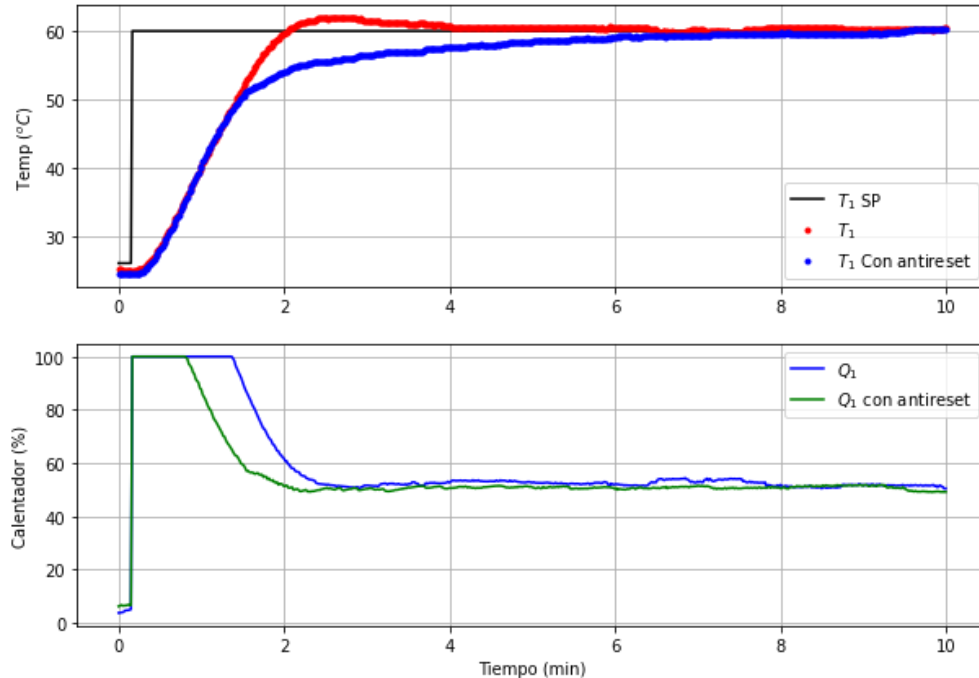


Figura 12. Funcionamiento del *antireset* en control PI

4.2.10 Control proporcional integral derivativo (PID)

La acción derivativa se basa en la rapidez con la que cambia el error o cuanto tiempo ha persistido el error. Se basa en la pendiente de la curva del error vs tiempo en cualquier instante de tiempo. Por lo tanto, una señal de error que cambia rápidamente inducirá una gran respuesta derivativa. Las señales de error ruidosas causan problemas significantes para la acción derivativa debido al rápido cambio de la pendiente del error causada por el ruido. El control derivativo debe evitarse en estas situaciones a menos que la señal de error pueda ser filtrada para eliminar el ruido. (Coughanowr, Proportional Integral Derivative (PID) control, 2009)

La ecuación de control PID es:

$$Q(t) = Q_{bias} + K_c e(t) + \frac{K_c}{\tau_I} \int_0^t e(t) dt + K_c \tau_D \frac{d(e)}{dt} \quad (46)$$

El nuevo parámetro es la constante de tiempo derivativo τ_D , el cual proporciona un peso separado a la derivada, como cambia el error con el tiempo. El termino derivativo considera que tan rápido está cambiando el error, usando una tasa de cambio o pendiente en la curva de error. Si se está cerca del SP la acción derivativa evitará que exista sobre impulso frenando al proceso antes del SP, eliminando además oscilaciones en la variable de salida.

Patada derivativa

Aunque el término derivativo implica $\frac{de(t)}{dt}$, la derivada de la variable de proceso $\frac{d(PV)}{dt}$ se utiliza en la práctica para evitar un fenómeno denominado patada derivativa. La patada derivativa ocurre porque el valor del error cambia repentinamente cada vez que se ajusta el SP. La derivada de un salto repentino en el error hace que la derivada del error sea instantáneamente grande y hace que la salida del controlador se sature durante un ciclo en un límite superior o inferior. Si bien este salto momentáneo no suele ser un problema para la mayoría de los sistemas, una saturación repentina de la salida del controlador puede ejercer una presión indebida sobre el elemento de control final o potencialmente perturbar el proceso. (Hedengren, Proportional Integral Derivative (PID), 2022)

Para evitar esto se supone que el punto de referencia es constante $\frac{d(SP)}{dt} = 0$

$$\frac{de(t)}{dt} = \frac{d(SP - PV)}{dt} = \frac{d(SP)}{dt} - \frac{d(PV)}{dt} = -\frac{d(PV)}{dt} \quad (47)$$

Además, el término derivado es una aproximación en diferencias finitas. Un método común utiliza las diferencias entre los valores actuales y anteriores de PV para obtener la pendiente actual.

$$\frac{d(T_{PV})}{dt} \approx \frac{T_{PV,n_t} - T_{PV,n_t-1}}{\Delta t} \quad (48)$$

Sustituyendo (47) y (48) en (46) la ecuación de control PID que se programa en Python queda:

$$Q(t) = Q_{bias} + K_c e(t) + \frac{K_c}{\tau_I} \sum_{i=1}^{n_t} e(t) \Delta t - K_c \tau_D \frac{T_{PV,n_t} - T_{PV,n_t-1}}{\Delta t} \quad (49)$$

4.2.11 Control feedforward (ff)

Los lazos de control feedback no pueden mantener la variable de proceso en el punto de ajuste debido a perturbaciones o cambios del SP. Esto sucede porque un control feedback reacciona solo si la salida presenta una desviación del punto de ajuste. A diferencia de los sistemas feedback los controladores feedforward miden la perturbación directamente y toma medidas de control para disminuir su impacto en la salida del proceso (ver Fig. 9). Por lo que estos controladores son sumamente estables. (Stephanopoulos, Feedforward and Ratio Control, 1984)

El control ff es un término adicional en la ecuación de control PID que agrega o elimina la acción de control $Q_1(t)$ basado en una perturbación medida T_2 . La ecuación de control resultante es:

$$Q_1(t) = Q_{1,bias} + K_c e(t) + \frac{K_c}{\tau_I} \int_0^t e(t) dt - K_c \tau_D \frac{d(T_1)}{dt} + K_{ff} T_2' \quad (50)$$

Donde el parámetro de la perturbación es el marcado en azul. K_{ff} es la ganancia de control ff y se calcula de la siguiente manera:

$$K_{ff} = \frac{K_d}{K_p} \quad (51)$$

La ganancia de la perturbación (K_d) puede obtenerse con la siguiente relación:

$$K_d = \frac{\Delta T_1}{\Delta T_2} \quad (52)$$

4.2.12 Control en cascada

En una configuración de control de cascada hay una variable manipulada y más de una medida. Está claro que con una sola manipulación podemos controlar solo una salida. La ventaja de usar un control en cascada es que las perturbaciones que surgen dentro del lazo secundario son corregidas por el control secundario antes de que puedan afectar el valor de salida primaria controlada (Fig. 13). (Stephanopoulos, Control Systems with Multiple Loops, 1984)

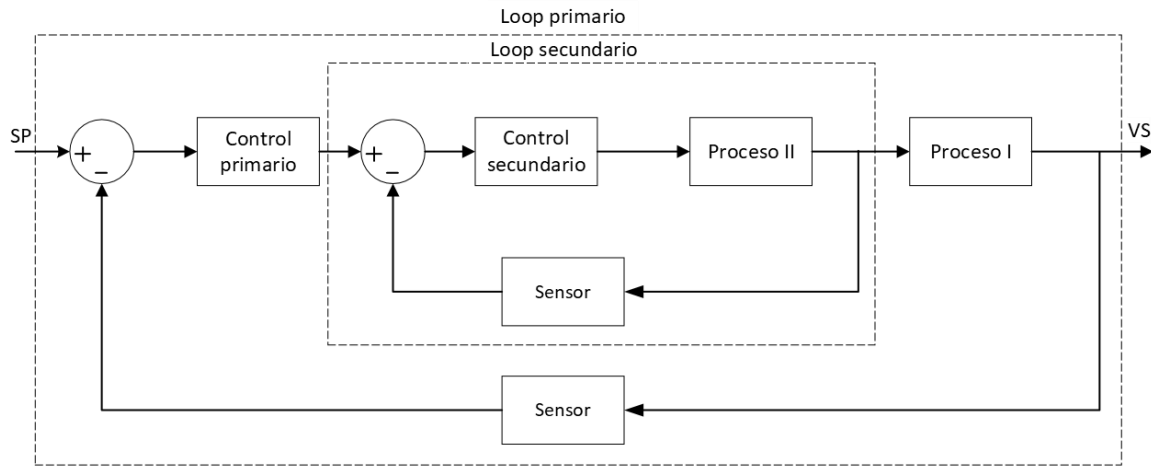


Figura 13. Representación esquemática de un control en cascada.

Control en cascada con TCL

El loop primario se compone por la variable de salida T_2 , el sensor 2, el SP de la temperatura 2 T_{2SP} y un control PID2 que recibe el error entre $T_{2SP} - T_{sensor2}$. El loop secundario tiene como entrada la T_{1SP} que es la salida del controlador PID2, un controlador PID1 que mide el error entre $T_{1SP} - T_{sensor1}$ y tiene como salida la variable manipulada Q_1 , esta cambiará el valor de T_1 y será medida como $T_{sensor1}$. (Fig.14)

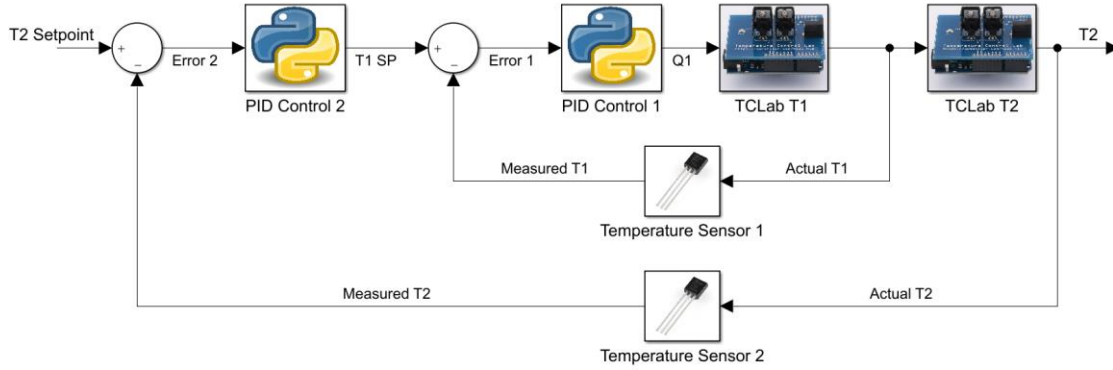


Figura 14. Diagrama de bloques en cascada del TCL. Extraída de: <https://apmonitor.com/pdc/index.php/Main/TCLabCascadeControl>

4.2.13 Diagrama de bloques

El uso de las transformadas de Laplace nos permite formar representaciones simples e interpretables de la dinámica de los procesos químicos. Es simple porque se utilizan solo ecuaciones algebraicas en lugar de ecuaciones diferenciales. Permite un rápido análisis de la dinámica del proceso y nos da la relación entre las entradas (perturbaciones, variable manipulada) y las salidas (variables controladas) de un proceso. Lo primero es obtener una función de transferencia $G(s)$, utilizando las transformadas de Laplace, esto nos dará la relación que existe entre la salida, $Y(s)$ y la entrada $U(s)$, de un proceso. (Stephanopoulos, Transfer Functions and Input-Output Models, 1984)

Función de transferencia

Para obtener la función de transferencia de primer orden del dispositivo TCL se utilizará la ecuación FOPDT (21):

$$\tau_p \frac{dy(t)}{dt} = -y(t) + K_p u(t - \theta_p) \quad (53)$$

Utilizando las transformadas de Laplace obtenemos (para solución completa ver Tclab Diagrama de bloques [P17](#)):

$$\tau_p s Y(s) + Y(s) = K_p U(s) e^{-\theta_p s} \quad (54)$$

Y reacomodando términos nos queda:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{K_p e^{-\theta_p s}}{\tau_p s + 1} \quad (55)$$

La ecuación (55) es la función de transferencia del proceso.

Ecuación PI en el dominio de Laplace

La ecuación PID puede representarse como un bloque en un diagrama de bloques del proceso. Partiendo de la ecuación de control PID (46) tendríamos:

$$u(t) = u_{bias} + K_c e(t) + \frac{K_c}{\tau_I} \int_0^t e(t) dt - K_c \tau_D \frac{d(PV)}{dt} \quad (56)$$

Si se aplica Laplace a cada uno de los términos de (56) nos queda:

$$U(s) = K_c E(s) + \frac{K_c}{\tau_I s} E(s) - K_c \tau_D PV(s) \quad (57)$$

Para un controlador PI el término derivativo se hace cero, y reacomodando términos para relacionar la entrada al controlador con la salida obtenemos la función de transferencia del controlador PI $G_c(s)$:

$$G_c(s) = \frac{U(s)}{E(s)} = K_c + \frac{K_c}{\tau_I s} = K_c \frac{\tau_I s + 1}{\tau_I s} \quad (58)$$

Función de transferencia para sistema de segundo Orden

Partiendo de la ecuación (24) podemos determinar la transformada de Laplace de cada uno de los términos:

$$\mathcal{L} \left(\tau^2 \frac{d^2 T(t)}{dt^2} \right) = \tau^2 [s^2 T(s) - sT(0) - T'(0)] \quad (59)$$

$$\mathcal{L} \left(2\zeta \tau \frac{dT(t)}{dt} \right) = 2\zeta \tau [sT(s) - sT(0)] \quad (60)$$

$$\begin{aligned} \mathcal{L}(T) &= T(s) \\ \mathcal{L}(QK_p) &= K_p Q(s) \end{aligned} \quad (61)$$

Sustituyendo en (24) las transformas anteriores con $(T(0)=0)$ queda:

$$s^2 \tau^2 T(s) + 2\zeta \tau s T(s) + T(s) = K_p Q(s) \quad (62)$$

Sacando factor común:

$$\mathbf{T}(s)[s^2 \tau^2 + 2\zeta \tau s + 1] = K_p \mathbf{Q}(s) \quad (63)$$

Despejando para obtener la forma de función de transferencia queda:

$$\frac{\mathbf{T}(s)}{\mathbf{Q}(s)} = \frac{K_p}{s^2 \tau^2 + 2\zeta \tau s + 1} \quad (64)$$

4.3 Python

Es un lenguaje de programación de alto nivel, orientado a objetos y con semántica dinámica. La facilidad de la sintaxis en Python hace de este un lenguaje muy atractivo para estudiantes que aún no han tenido acercamiento a otro lenguaje de programación.

Antes de empezar a trabajar con el TCL se recomienda instalar Anaconda, que contiene Anaconda Navigator, la cual es una interfaz gráfica de usuario que nos permitirá correr los programas más utilizados para programar en Python como son JN (Jupyter Notebook) y Spyder. En esta tesina se utilizó principalmente JN y se recomienda al lector utilizarlo para abrir los archivos con extensión .ipynb que se encuentran dentro de la carpeta de programas (Apéndice A). El capítulo 5 junto con los archivos .ipynb ampliarán la información correspondiente a Python.

4.3.1 SciPy

Es una biblioteca libre y de código abierto para Python construida sobre la extensión Numpy. Proporciona algoritmos de optimización, integración, interpolación, problemas de valores propios, ecuaciones algebraicas diferenciales, estadísticas y otros problemas. (scipy.org)

4.3.2 Gekko

Es una librería de Python orientada a objetos utilizada para el aprendizaje automático y la optimización de ecuaciones algebraicas diferenciales y enteras mixtas. Los modos de operación incluyen regresión de parámetros, reconciliación de datos, optimización en tiempo real, simulación dinámica y control predictivo no lineal. Es gratuito para uso académico y comercial bajo la licencia MIT. <https://gekko.readthedocs.io/en/latest/>

Para poder utilizar la librería gekko es necesario instalarla en el entorno. Para esto es necesario abrir Jupyter notebook y escribir las siguientes líneas en una celda de código.

```

1 # Instalar Gekko
2 try:
3     from pip import main as pipmain
4 except:
5     from pip._internal import main as pipmain
6 pipmain(['install', 'gekko'])

```

5 Programas

5.1 Simulación de respuesta de paso ODE

En este programa se utiliza el TCL para generar un cambio de temperatura debido a la modificación del valor de un calentador, imprimir una gráfica de temperatura medida vs tiempo y luego simular el comportamiento con un modelo dinámico del TCL utilizando la función **odeint** y la librería **gekko**.

El código completo puede encontrarse en P1, revisar apéndice **A**.

En la primera celda se especifica nuevamente la instalación de TCL (este paso puede omitirse si se hizo previamente). Si se inicia por esta celda simplemente se corre el programa para que el paquete TCL sea instalado.

A continuación, es necesario importar las librerías necesarias para realizar la conexión con el dispositivo TCL.

- **import numpy as np:** para crear vectores y matrices, además funciones matemáticas.
- **import matplotlib.pyplot as plt:** generar gráficos en Python.
- **import tclab:** Conexión con el TCL.
- **import time:** Funciones relacionadas con el tiempo.

Se crea un arreglo lineal con 301 elementos con la función **np.linspace()**. Será de 0 a 301 al dividirlo entre 301 nos dará un resultado de números enteros de 0 a 300, que representan 5 minutos en segundos. Se llama al TCL y se crea la variable T1 que contendrá un arreglo y una lectura del sensor 1. Se programa también el calentador con un valor del 50%.

```
1 n = 300
2 tm = np.linspace(0,n,n+1)
3
4 # datos
5 lab = tclab.TCLab()
6 T1 = [lab.T1]
7 lab.Q1(50)
```

Los datos son guardados con un ciclo **for** y finalmente se pide que se imprima un gráfico con los datos medidos como se muestra abajo:

```
1 # Temperaturas a intervalos regulares
2 for i in range(n):
3     time.sleep(1)
4     print(lab.T1)
5     T1.append(lab.T1)
6 lab.close()
7
8 # Imprimir resultados
9 plt.figure(1)
10 plt.plot(tm,T1,'r.',label='Medidos')
11 plt.ylabel('Temperatura (°C)')
12 plt.xlabel('Tiempo (seg)')
13 plt.legend()
```


Para visualizar la gráfica con un mejor formato se pueden hacer modificaciones a la impresión.

```
1 plt.figure(figsize=(12,5))
2 plt.plot(tm,Tl,'r.',label='Valor medido')
3
4 plt.legend(fontsize = 20)
5
6 plt.xlabel('Tiempo, seg', fontsize = 20)
7 plt.ylabel('Temperatura, °C', fontsize = 20)
8 plt.title('La temperatura en el artefacto Arduino', fontsize = 20)
```

Con lo que se obtiene el gráfico de los datos medidos:

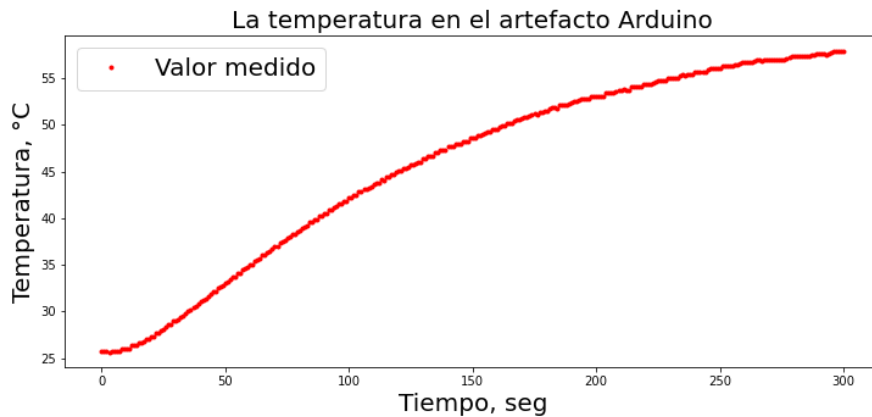


Figura 15. Curva de temperatura debida a un cambio de Q1 en TCL

Se observa una curva con los valores de temperatura durante 5 minutos medidos por el sensor 1.

En la siguiente celda se procede a hacer la simulación de la respuesta de paso utilizando las librerías Scipy y GEKKO. Utilizando el modelo dinámico visto en (22) simplificado:

$$\tau_p \frac{dT}{dt} = (T_a - T) + K_p Q \quad (65)$$

- τ_p : constante tiempo del proceso.
- K_p : ganancia del proceso.
- T_a : temperatura ambiente.
- T : temperatura.
- Q : porcentaje de calentador

Simulación con odeint

Para esta parte es necesario importar la función **odeint**.

- **from scipy.integrate import odeint:** función de librería scipy para resolver ecuaciones diferenciales.

Con la siguiente línea de código se define la ecuación diferencial. TC es la variable dependiente y t es la variable independiente. Respetando la indentación adecuada se escribe el modelo con valores indicados ($T_a=25$, $Q=50$, $\tau_p=170$, $K_p=0.8$). Se puede modificar τ_p y K_p para mejorar el resultado de la simulación. En programas posteriores se indicará la forma de calcular estos parámetros.

```
1 def labsim(TC,t):
2     dTCdt = ((25-TC) + 0.8*50)/170.0
3     return dTCdt
```

Para llamar la solución de la ecuación diferencial se utiliza la siguiente línea, cuyos argumentos son la función, la temperatura inicial, y un arreglo de tiempo donde se obtendrá la solución de la variable dependiente. Por último, se programa la impresión de la gráfica.

```
1 Tsim = odeint(labsim,25,tm)
```

Simulación con GEKKO

Dentro del programa se tiene una celda con la instalación de gekko, en caso de que no se tenga instalado es necesario correr para tener el paquete dentro del equipo.

Una vez instalada la paquetería gekko es necesario importarlo al inicio de la celda en la que se está trabajando.

- **from gekko import GEKKO**

Para crear un modelo en gekko se utiliza **GEKKO()**. Para generar el arreglo de tiempo se crea la variable m.time utilizando la función **np.linspace()**. La variable dependiente TC se crea con **m.Var(temperatura inicial)**. La ecuación diferencial se coloca en **m.Equation()** sin despejar. **m.options.IMODE** define el tipo de problema que se resuelve. **m.solve(dis=False)** resuelve el problema, el False solo le dice al programa que no imprima una hoja de resultados.

```
1 # simulación
2 m = GEKKO()
3 m.time = np.linspace(0,n,n+1)
4
5 TC = m.Var(25)
6 m.Equation(170*TC.dt()==(25-TC)+0.8*50)
7 m.options.IMODE = 4
8 m.solve(dis=False)
```

Al graficar la curva simulada con la curva de valores medidos se obtiene:

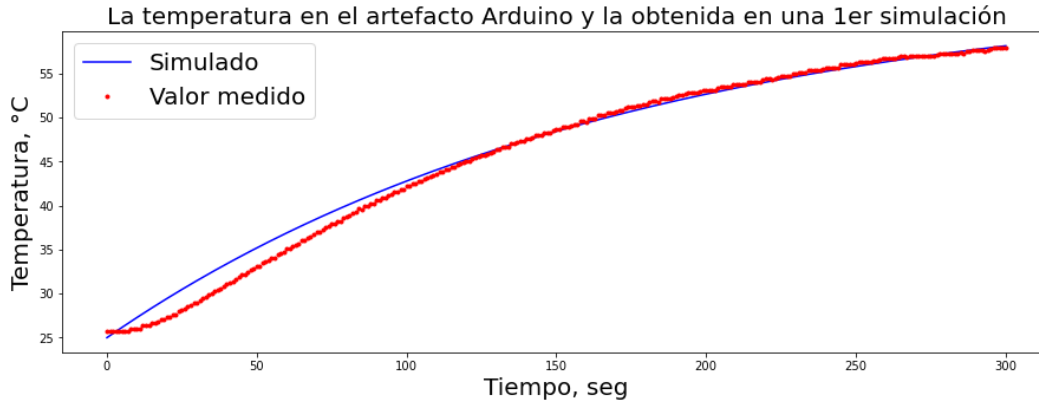


Figura 16. Curvas de temperatura simulada y medida en el TCL

Puede observarse (Fig.16) que odeint y gekko dan el mismo resultado. La respuesta de la curva simulada ajustarse modificando el valor de la temperatura inicial, que será la temperatura ambiente, con esto nos aseguramos de que el valor simulado iniciará en la misma temperatura que los valores medidos. También observamos que en los primeros dos minutos ambas curvas son distintas, esto es debido a que en el modelo simulado no se está tomando en cuenta el retraso que hay en la medición (tiempo muerto). El tiempo muerto es el lapso que el sensor de temperatura tarda en medir un cambio en la temperatura debido a un cambio de la variable de entrada. Más adelante se incluirán más términos en el modelo y se observarán las diferencias que esto provoca.

5.2 Transferencia de calor por convección

En el programa P2 se realiza la simulación de un balance de energía con transferencia de calor por convección. Ver ecuación (6).

Para tomar los datos de temperatura desde el TCL se programa una celda como la utilizada en el programa anterior. En la parte de simulación se utiliza nuevamente odeint con los datos necesarios para resolver la ecuación (6). Los datos son $m=0.004$ kg, $A=0.0012$ m², $C_p = 500 \frac{J}{kg-K}$ y $T_a= 21$ °C. La T_a debe ser la que se tenga en el lugar en el que se hace la prueba. La primera lectura que hace el TCL puede tomarse como la temperatura ambiente o si desea puede hacer una medición de la temperatura con la siguiente línea de código:

```

1 import tclab
2
3 lab= tclab.TCLab()
4 print(lab.T1)
5 lab.close()

```

Elegir un correcto valor de la temperatura inicial mejorará mucho la simulación.

Para llamar al TCL y obtener valores de temperatura se utiliza la primera parte del código expuesto en el capítulo anterior. La parte que cambia es donde se escribe la simulación, para resolver con odeint y posteriormente resolver con la librería Gekko.

Resolviendo con odeint:

```

1 def labsim(TC,t):
2     U = 10.0
3     A = 0.0012
4     Cp = 500
5     m = 0.004
6     alpha = 0.009
7     Ta = 21
8     dTCdt = (U*A*(Ta-TC) + alpha*50)/(m*Cp)
9     return dTCdt
10 Tsim = odeint(labsim,21,tm)

```

Al final se obtiene el siguiente gráfico:

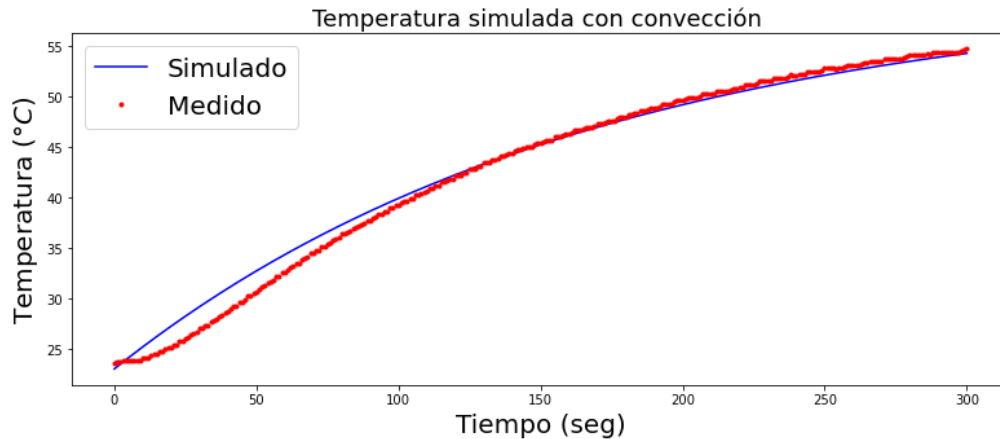


Figura 17. Simulación de modelo convectivo del TCL

Observando este gráfico notamos que del tiempo 0 al 125 las curvas son distintas, esto es debido a que el balance no contiene información que simule correctamente al tiempo muerto, mientras que del tiempo 125-300 la simulación predice bastante bien el comportamiento de la temperatura medida.

En este programa la curva de simulación puede ser distinta a la obtenida con TCL. Si se quisiera mejorar el resultado se recomienda mover el valor de α pues puede variar de un equipo TCL a otro. Para ajustar la simulación en el dispositivo utilizado en el presente trabajo el valor elegido de α es de 0.009 W/%.

5.3 Transferencia de calor radiactivo

El programa P3 toma en cuenta la transferencia de calor por radiación. Partiendo del balance de calor (8).

En la primera celda de código dentro del programa, lo primero es llamar a TCL para que obtenga mediciones de temperatura, con el calentador al 50%, seguido por las constantes necesarias para la simulación con odeint de la ecuación (8). A continuación, para poder observar cuales son las pérdidas debidas a la transferencia por convección y radiación, se programa por separado el término convectivo (4) y el radiactivo (7), como se observa en líneas 11 y 12 de la siguiente celda.

```

1 def labsim(TC,t):
2     TK = TC + 273.15
3     dTCdt = (U*A*(Ta-TC) + sigma*eps*A*(TaK**4-TK**4) + alpha*50)/(m*Cp)
4     return dTCdt
5
6 #resolver
7 tm = np.linspace(0,n,n+1)
8 Tsim = odeint(labsim,30,tm)
9
10 # Calculo de las perdidas de calor por convección y radiación
11 conv = U*A*(Ta-Tsim)
12 rad = sigma*eps*A*(TaK**4-(Tsim+273.15)**4)
13 loss = conv+rad
14 gain = alpha*50

```

Finalmente se programa el gráfico utilizando la paquetería **matplotlib**, veremos que nos permite realizar una gráfica con subgráficas, en este caso se le indicó que las acomodara de forma vertical para mejorar la visualización.

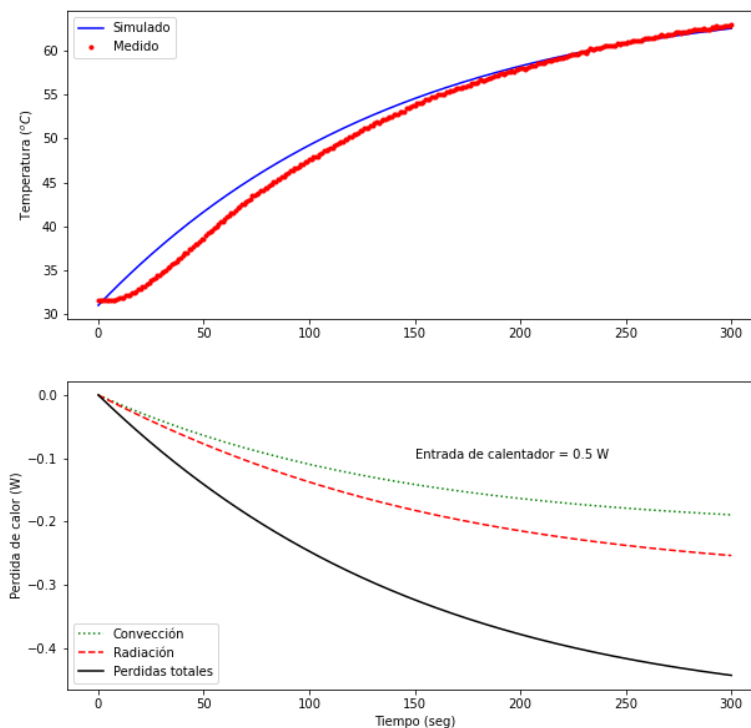


Figura 18. Simulación de modelo con transferencia por radiación y pérdidas de calor.

En la gráfica superior de la figura 18 podemos observar que la simulación no coincide con los datos medidos por lo mencionado en programas anteriores sobre el tiempo muerto o retraso de la medición por el sensor de temperatura.

Del gráfico inferior (Fig.18) se puede sacar mucha información acerca de las pérdidas de calor por ambos mecanismos. Se observa que la pérdida por radiación es mayor conforme aumenta la temperatura. Si observamos el término radiactivo (7) podemos deducir que al tener la variable T^4 a la cuarta potencia, la pérdida será mayor a mayor

temperatura. Por último, se observa la pérdida total de calor que se encuentra alrededor de 0.4 W, por lo tanto, al tiempo 300 la temperatura puede seguir aumentando, se volverá constante cuando la pérdida de calor (0.4W) se iguale a la entrada del calentador (0.5W). Pues se iguala en términos de calor que entra y sale.

5.4 Balance de energía linealizado

En este programa P4 se realizó la linealización del balance de energía (20), con el método de serie de Taylor.

Se incluye en el programa la solución de la linealización por el método de Taylor y una celda para la solución de ecuaciones diferenciales de este tipo utilizando **sympy**.

- **Import sympy as sp**: librería de Python para obtener soluciones simbólicas.

Después se tiene una celda con la toma de lecturas del TCL. Las siguientes 2 celdas muestran las soluciones gráficas en su forma no lineal (8) con odeint y gekko.

Al final contiene una celda con la solución de la ecuación diferencial linealizada. Como puede observarse se crean las variables gamma y beta (línea 1 y 2) que son las constantes obtenidas (17) y (18). También se obtiene la solución de dos ecuaciones diferenciales de forma simultánea utilizando odeint, la primera es la simulación no lineal (línea 8) mientras que la segunda es la simulación de la ecuación linealizada (línea 10), como puede verse en el código mostrado abajo.

```

1 gamma = -U*A/(m*Cp) - 4*eps*sigma*A*TaK**3/(m*Cp)
2 beta = alpha/(m*Cp)
3 def labsim(x,t):
4     TC,TC2 = x
5     # convertir a Kelvin
6     TK = TC + 273.15
7     # No lineal
8     dTCdt = (U*A*(Ta-TC) + sigma*eps*A*(TaK**4-TK**4) + alpha*Q)/(m*Cp)
9     # Lineal (condiciones de estado estacionario)
10    dTC2dt = gamma * (TC2-30) + beta * (Q-0)
11    return [dTCdt,dTC2dt]
12
13 tm = np.linspace(0,n,n+1)
14 Tsim = odeint(labsim,[27,27],tm)
15
16 T_nonlinear = Tsim[:,0]
17 T_linear = Tsim[:,1]

```

Resolviendo la simulación anterior y utilizando los datos obtenidos con TCL se le pide al programa que imprima ambas simulaciones y la curva de temperaturas medidas (Fig.19). En esta gráfica podemos observar como la curva de ecuación linealizada (línea punteada) continúa aumentando muy por encima de los valores medidos mientras que la no lineal termina con valores muy similares a los valores medidos. Esto debido principalmente al hecho de haberle quitado el componente no lineal a la ecuación de balance. Se vuelve más evidente a mayor temperatura pues el termino radiactivo (término no lineal) tiene mayor influencia.

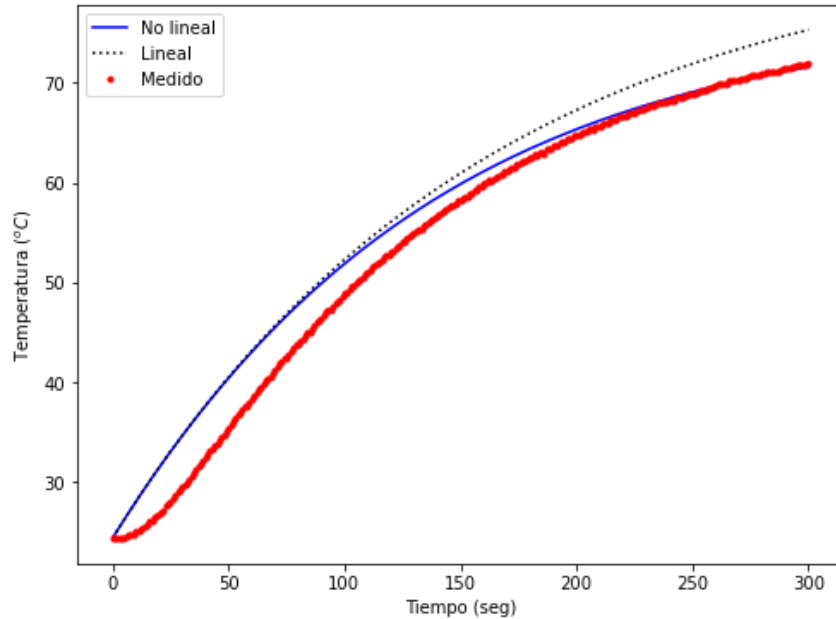


Figura 19. Simulación con modelo linealizado

5.5 FOPDT ajuste gráfico

Como se verá en el archivo TClab FOPDT Ajuste Grafico.ipynb dentro de P5 lo primero es obtener los datos utilizando TCL. Esta vez se indicará que encienda el calentador a partir del segundo 30 a un porcentaje del 70% para generar una respuesta escalón, durante 8 minutos.

Además, es el primer programa en el que se indica que se debe generar un archivo .csv, el cual es un archivo con los valores separados por comas o un arreglo similar al de tablas y filas, esto posibilita poder usar los datos obtenidos del TCL en cualquier momento en el que se utilice el programa o en otros programas sin necesidad de volver a encender el TCL. El código utilizado para guardar los datos en el archivo .csv, es el siguiente:

```

1 data = np.vstack((tm,Q1,T1)).T
2 np.savetxt('Ajuste_Grafico.csv',data,delimiter=',',\
3           header='Time,Q1,T1',comments='')

```

El gráfico que obtenemos se puede ver en la figura 20, tan solo contiene los valores medidos del TCL y los valores del calentador durante 8 minutos, pero nos será muy útil para resolver por el método gráfico. La figura 18 se puede imprimir o pasar a un programa de edición para ir marcando los valores necesarios que nos darán los valores de los parámetros k_p , τ_p y θ_p .

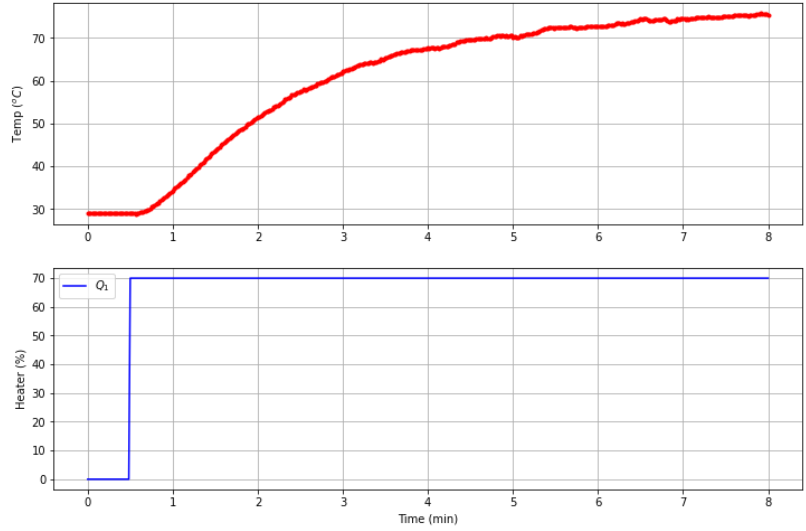


Figura 20. Prueba de paso con $Q1=70$.

Con la figura 20 es posible resolver el método gráfico siguiendo los pasos vistos en el capítulo 4.2.3:

Al final obtendremos la gráfica siguiente que contiene todos los puntos resueltos (Fig. 21).

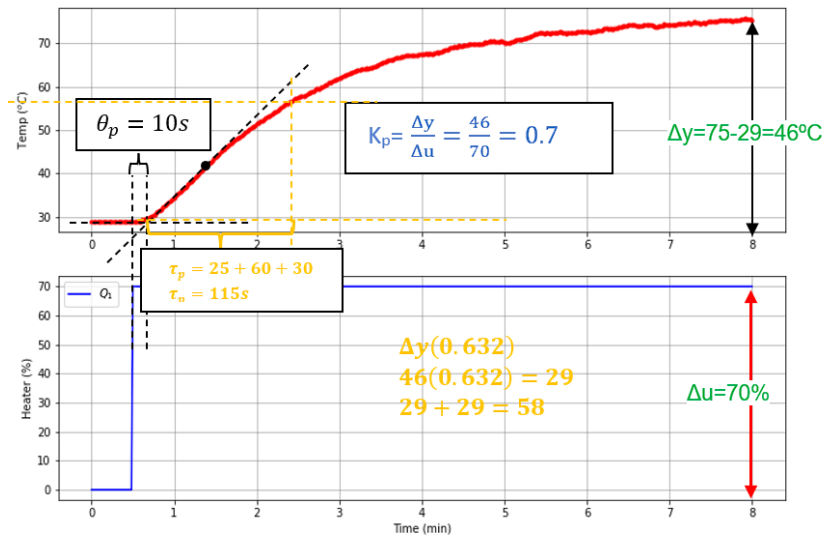


Figura 21. Método gráfico para determinar parámetros FOPDT.

Los resultados de los parámetros con el método gráfico son:

- $k_p = 0.7$
- $\tau_p = 115$
- $\theta_p = 10$

Para comprobar la efectividad de los parámetros obtenidos se comparan la simulación del modelo FOPDT con parámetros obtenidos con la curva de valores medidos en el TCL (Fig.22).

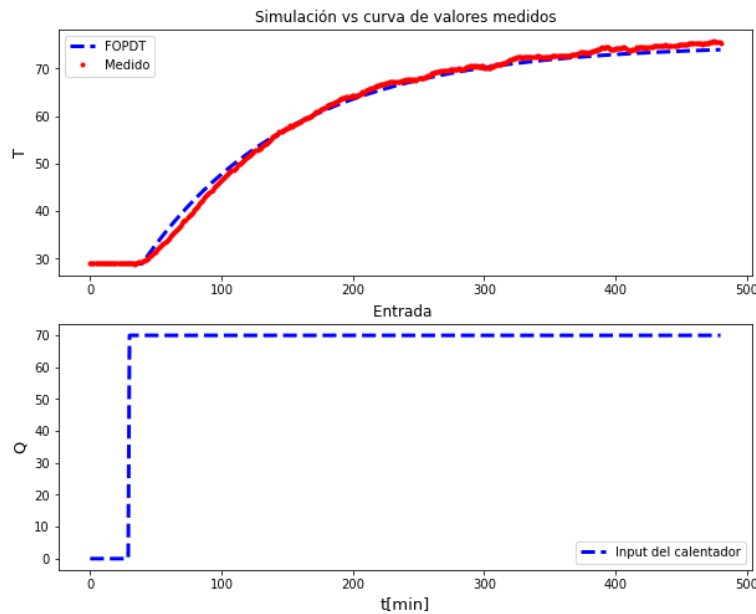


Figura 22. Simulación FOPDT para verificar parámetros.

Puede observarse que los parámetros que obtuvimos de forma gráfica fueron una buena aproximación. Se tiene una buena simulación del tiempo muerto. Si quisiéramos mejorar la simulación podríamos realizar una regresión. Dicho paso se realizará en el siguiente programa.

5.6 FOPDT regresión

En este problema se obtendrán los parámetros k_p , τ_p y θ_p , del modelo FOPDT, con un método de optimización el cual nos dará los mejores valores para simular el comportamiento de nuestro dispositivo TCL. Tener en cuenta que al usar otro dispositivo estos valores pueden cambiar, pero este programa ayudará a obtener los parámetros de cualquier dispositivo TCL utilizado.

El siguiente diagrama de flujo del programa muestra de forma general lo que se hace en cada iteración para llegar a los parámetros optimizados (Fig.23).

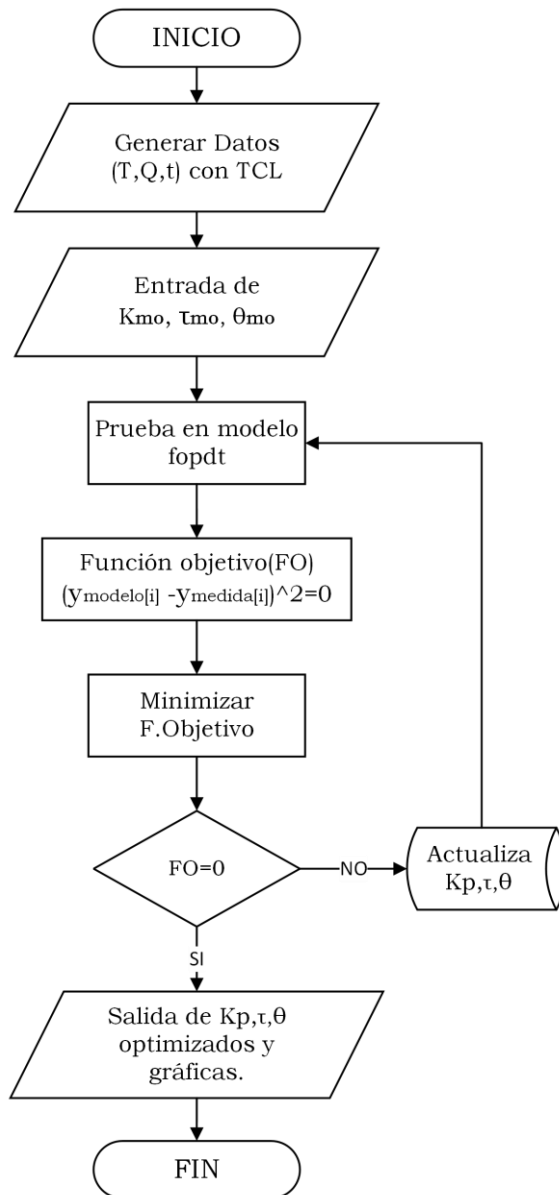


Figura 23. Diagrama de flujo para la obtención de parámetros fopdt optimizados

Como se verá en la celda para la obtención de datos se ha incluido la librería **pandas** que nos ayudará a leer el archivo Step_Response.csv en el caso de que estuviera guardado dentro de la carpeta donde se ejecuta el programa.

- **Import pandas as pd:** manipulación y análisis de datos.

Optimización en python

El uso de múltiples pasos programados moviendo los valores del calentador, nos ayudará a tener suficientes datos que nuestro programa podrá resolver por medio de una regresión la minimización del error y por lo tanto una mejor aproximación de los valores de los parámetros del modelo FOPDT.

En la celda para la optimización se añadirán algunos subpaquetes más pertenecientes a la librería SciPy.

- **`from scipy.optimize import minimize`**: para minimizar parámetros.
- **`from scipy.interpolate import interp1d`**: para interpolar.

El subpaquete **`minimize`** nos dará el mínimo valor de la función objetivo creada que es la suma de los errores cuadrados (SSE: *sum squared error*) en la que se utilizan las variables de temperatura modelada y temperatura medida. Este proceso se realiza en cada intervalo de tiempo y obtiene el resultado final en algunos segundos, lo cual comprueba la eficiencia de python al resolver problemas complejos con cientos de datos.

Dentro de la carpeta del programa P6 se tiene el archivo “Step_Response.csv” que contiene los datos del dispositivo utilizado en este trabajo. Si se cuenta con el dispositivo propio es necesario borrar dicho archivo para obtener los valores correspondientes.

Los resultados de la optimización pueden observarse en la tabla 2. Vemos como se pasa de un error 22773 a 162, esto quiere decir que nuestros parámetros se aproximan bastante bien al resultado deseado. Y seguido de esto se encuentran los valores de los parámetros optimizados.

Tabla 2. Parámetros k_p , τ_p y θ_p , optimizados

1	Initial SSE Objective: 22773.591378037556
2	Optimizing Values...
3	Final SSE Objective: 162.3775389470644
4	Kp: 0.757545012422904
5	taup: 134.0385083931629
6	thetap: 13.301719663789475

Para tener más claro lo que se hizo en la optimización puede observarse la figura 24, donde vemos como la curva de estimación inicial en la que se utilizan los parámetros obtenidos con el método gráfico se alejan del comportamiento real visto en la curva de temperatura medida. Mientras que al aplicar la optimización se obtiene la FOPDT Optimizada (curva en líneas punteadas). Lo cual nos indica que la simulación de nuestro dispositivo es correcta.

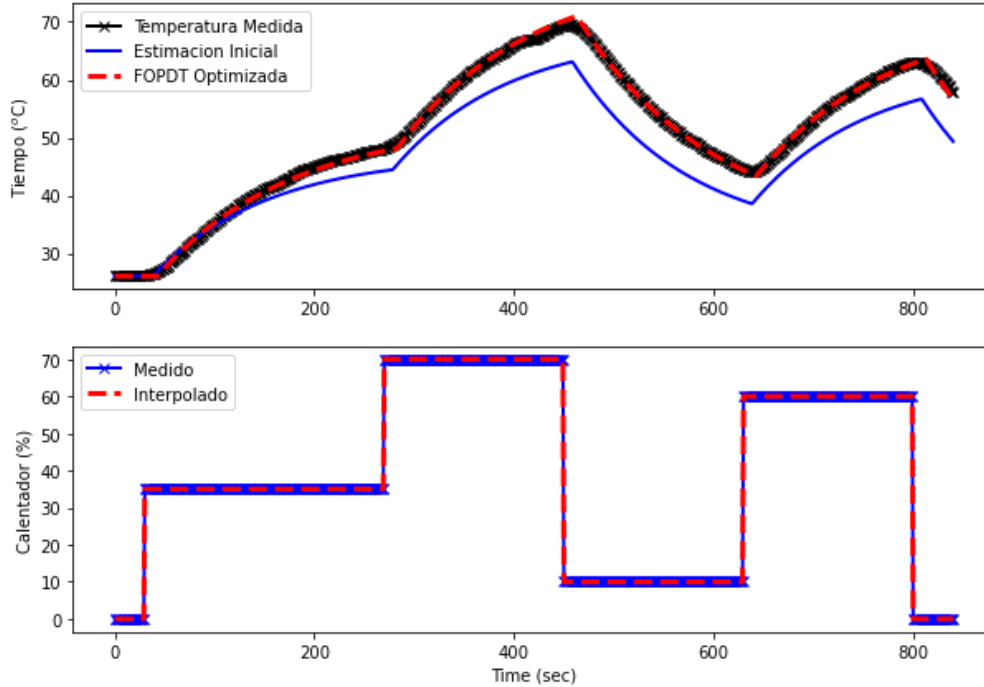


Figura 24. Gráfica de optimización de parámetros FOPDT

5.7 Control Proporcional P

Cálculo de K_c

Para este programa se utilizará la ecuación (33) para el cálculo de K_c . Si utilizamos los parámetros obtenidos en la regresión (ver tabla 2) tendremos:

$$K_c = \frac{0.20}{0.8} \left(\frac{134}{13} \right)^{1.22} = 4.305 \quad (66)$$

Utilización del simulador

En la primera celda del programa “Tclab Control solo proporcional.ipynb” en P7 se tiene un simulador de la respuesta a K_c . Al colocar los valores de los parámetros FOPDT y establecer la temperatura inicial igual que la temperatura ambiente, se obtiene una gráfica como la mostrada en la figura 25, que además si se está corriendo dentro del entorno de programación tiene una barra deslizante que modifica en tiempo real el efecto que tendría en el proceso el cambio de K_c . Es muy útil pues si se deseara un comportamiento diferente de nuestro controlador podríamos ubicar a K_c en el valor deseado. En la primera sección (Fig. 25, izquierda superior) se muestra la curva de temperatura, el valor establecido del SP, el valor del offset calculado y el valor del K_c . La siguiente (superior derecha) muestra el valor que toma el termino proporcional del controlador, al inicio es alto pues la diferencia entre SP y PV es alta. Luego (inferior izquierda) vemos la curva del error similar a la gráfica anterior pues es proporcional y depende de las mismas variables.

Y por último (inferior derecha) tenemos el comportamiento de la salida del calentador vemos que nos sobrepasa el 100, pues el máximo valor que puede tomar el calentador es 100.

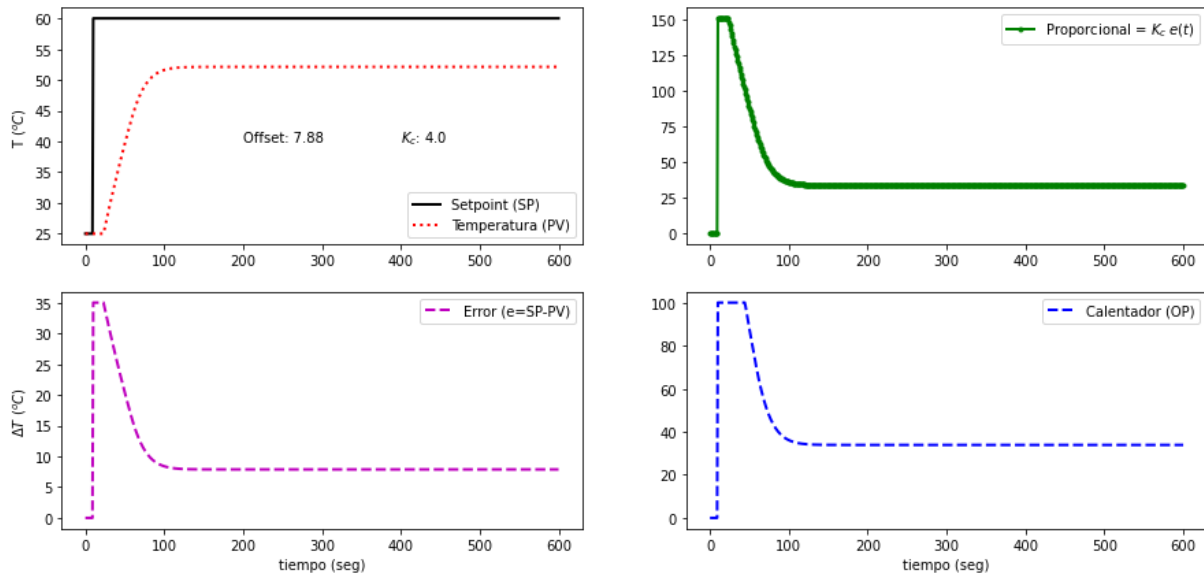


Figura 25. Simulador de control P.

Cálculo del offset

La ecuación para el cálculo de offset que se obtiene partiendo de (22) y (42) , cuya demostración puede verse en P7, es la siguiente:

$$offset = T_{SP} - T_{ee} \quad (67)$$

Donde T_{ee} es la temperatura de estado estacionario.

$$T_{ee} = \frac{T_a + K_p K_c T_{SP}}{1 + K_p K_c} \quad (68)$$

Para calcular el offset quedaría:

$$offset = 60 - \frac{25 + (0.8)(4.3)(60)}{1 + (0.8)(4.3)} = 7.89 \quad (69)$$

Ciclo de control P en Python

Podemos ver la simplicidad del control P en el siguiente recuadro. Se recibe la lectura de temperatura del sensor 1, se activa la ecuación de control y devuelve un valor para el calentador (Q1) que se limita entre valores de 0-100. Este nuevo valor de Q1 es enviado al TCL para que el calentador trabaje con ese valor. Las líneas finales son para imprimir las variables deseadas cada segundo, encabezados y cierre del TCL.

Las variables que se imprimen cada segundo son:

- Valor del calentador (*Heater*: calentador).
- Temperatura del sensor 1 (Temp).
- Punto de ajuste (Setpoint).

```

1 for i in range(n):
2     # Registro de medición
3     T1[i] = lab.T1
4     # -----
5     # Complete la ecuación del controlador P-only para cambiar Q1[i]
6     # -----
7     Q1[i] = Q1_bias + Kc * (SP1[i]-T1[i]) # Ecuacion del controlador P-only
8     # implementar nuevo valor de calentador
9     Q1[i] = max(0,min(100,Q1[i]))
10    lab.Q1(Q1[i]) #actuador Q1 toma el valor indicado por el controlador
11    if i%20==0: #módulo
12        print(' Heater, Temp, Setpoint') #Si residuo es cero, imprimir.
13    print(f' {Q1[i]:7.2f}, {T1[i]:7.2f}, {SP1[i]:7.2f}') #imprimirá cada segundo los datos
14    # wait for 1 sec
15    time.sleep(1)
16 lab.close()

```

En la figura 26 podemos observar la existencia de offset, la temperatura que se alcanza es de 51°C y nunca se alcanza el SP deseado que era de 60°C. En cuanto al offset medido respecto al calculado, vemos una diferencia pequeña (aproximadamente de 8%), por lo que puede decirse que solución teórica es bastante cercana al comportamiento del sistema TCL. La diferencia se puede deber a perturbaciones, parámetros FOPDT redondeados o deficiencias en el modelado.

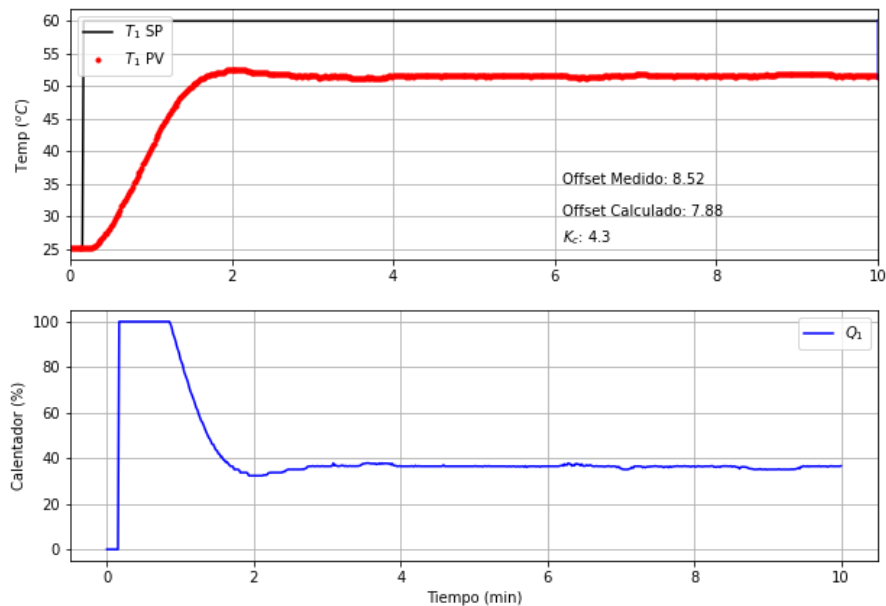


Figura 26. Aplicación de un controlador P en TCL.

5.8 Control proporcional integral PI

Aplicación del controlador PI

En siguiente celda se muestra el ciclo de control PI. A diferencia del control P vemos que se han incluido nuevas variables, estas son el error (err), integral absoluta del error (iae), y el error integral (ierr). Seguido de esto vemos la ecuación de control PI. Además, se agrega el ARW como puede observarse en las líneas 9-10.

```
1 for i in range(n):
2     # registro de medición
3     T1[i] = lab.T1
4     err =SP1[i]-T1[i]
5     iae += np.abs(err)
6     ierr += err
7     Q1[i] = Q1_bias + Kc*err + Kc/tauI * ierr #Ec. control PI
8     # implementar el nuevo valor del calentador con integral anti-reset windup
9     if Q1[i]>100 or Q1[i]<0:
10        ierr -= err
11    Q1[i] = max(0,min(100,Q1[i]))
12    lab.Q1(Q1[i])
13    if i%20==0:
14        print(' Heater, Temp, Setpoint')
15        print(f'{Q1[i]:7.2f},{T1[i]:7.2f},{SP1[i]:7.2f}')
16        # wait for 1 sec
17        time.sleep(1)
18    lab.close()
```

En la figura 27 se observa que al aplicar un controlador PI se ha eliminado el offset. En un tiempo aproximado de 8 minutos se ha llegado al SP=60. Si se deseara llegar más rápido al SP se podría aumentar el valor de K_c o disminuir τ_I lo cual puede verse observando el termino integral (45).

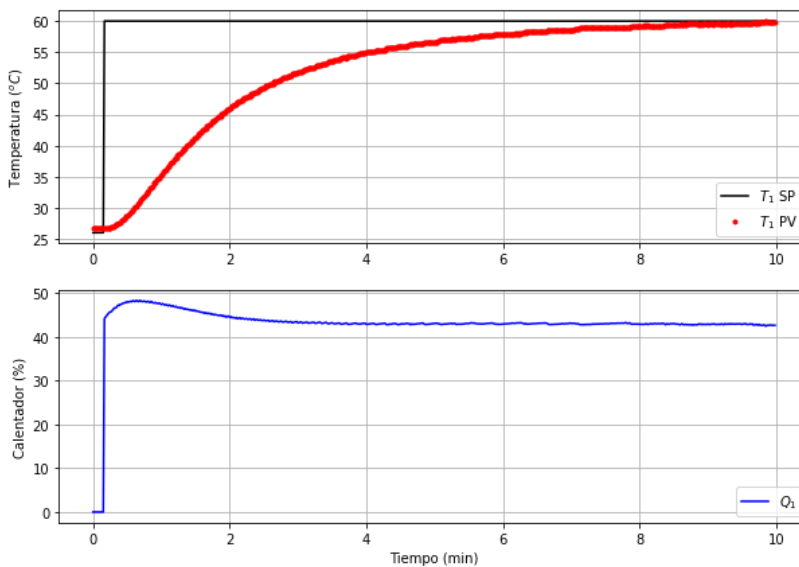


Figura 27. Control PI en TCL.

5.9 Control proporcional integral derivativo PID

Función PID en Python

En la primera celda, mostrada abajo podemos observar la función PID, las variables de entrada son: sp (punto de ajuste), pv (temperatura actual), pv_last (temperatura anterior), ierr (error integral), y dt (incremento de tiempo entre mediciones). Mientras que las de salida son: op (salida del controlador), P (contribución proporcional), I (contribución integral), y D (contribución derivativa). Esta función puede ser utilizada como controlador PID en cualquier ejercicio.

```
1  ###-----Función PID-----###
2  def pid(sp,pv,pv_last,ierr,dt):
3      KP = Kc
4      KI = Kc/tauI
5      KD = Kc*tauD
6      op0 = 0
7      # límites superior e inferior del calentador
8      ophi = 100 #output hi
9      oplo = 0
10     # calcular el error
11     error = sp-pv
12     # Calcular el error integral
13     ierr = ierr + KI * error * dt
14     # Calcular la derivada medida
15     dpv = (pv - pv_last) / dt
16     # calcular la salida PID
17     P = KP * error
18     I = ierr
19     D = -KD * dpv
20     op = op0 + P + I + D
21     # implement anti-reset windup; si OP<0 or op>100
22     if op < oplo or op > ophi:
23         I = I - KI * error * dt
24         # acortar salida
25         op = max(oplo,min(ophi,op))
26     # devolver la salida del controlador y los términos PID
27     return [op,P,I,D]
```

Aplicación del control PID

A diferencia de los controladores anteriores la ecuación que se programa es la llamada a la función PID descrita anteriormente, lo cual puede verse en la siguiente línea de código.

```
[Q1[i],P,ierr,D] = pid(SP1[i],T1[i],T1[max(0,i-1)],ierr,1.0)
```

Donde en cada segundo la función PID devuelve la salida del controlador o variable manipulada (Q1), la contribución proporcional, la contribución integral (ierr) y la contribución derivativa (D). Mientras se alimenta de los valores de SP programado sp1[i], la temperatura medida (T1[i]), la temperatura medida anterior empezando en 0 (T1[max...]), el error integral (ierr) y el incremento de tiempo entre mediciones o dt que se establece en un valor de 1.

IMC sintonización agresiva PID

Para calcular τ_c se utiliza la sintonización agresiva vista en (34) y la correlación IMC agresiva (41) vistas en el capítulo 4.2.6.

Al correr la celda del TCL con control PID obtenemos la gráfica siguiente (Fig. 28). Vemos que se llegó al SP de forma rápida ya que se usó una sintonización agresiva, pero no hay sobre impulso gracias a la acción derivativa la cual disminuye la salida del controlador antes de llegar al SP suavizando la curva de PV. En el gráfico del calentador vemos que aparece ruido durante todo el proceso, una de las desventajas de la acción derivativa. El ruido no es deseable en procesos reales, si esta fuera una válvula la oscilación implicaría mucho desgaste al estar cerrando y abriendo. Una manera de controlar el ruido podría hacerse añadiendo un filtro, el cual es un parámetro extra que se añade en la ecuación de control.

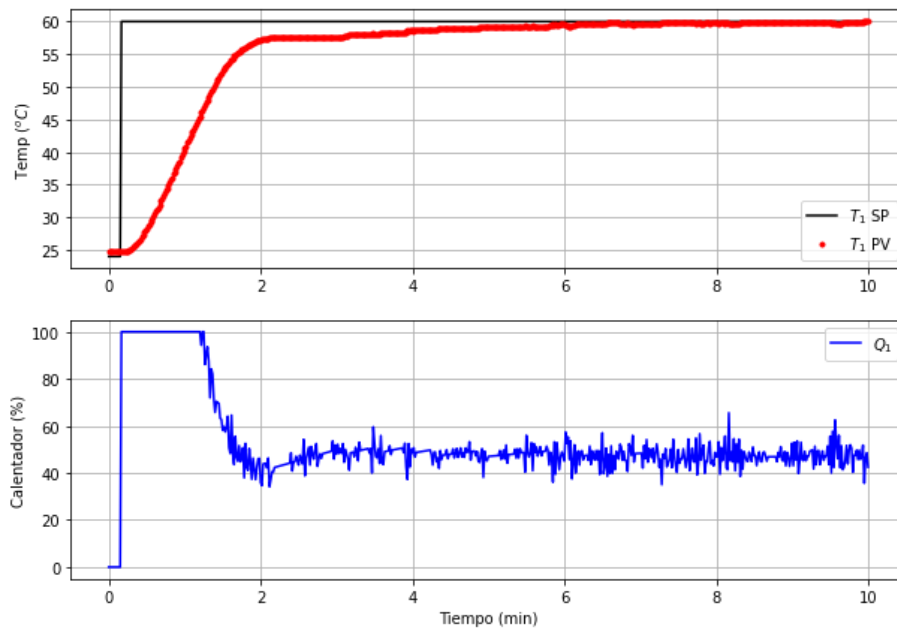


Figura 28. Respuesta a un controlador PID

5.10 Sintonización de control PI

Simulador de sintonización PI

En este programa se utiliza un simulador de control con un modelo FOPDT el cual al correrlo devuelve una figura con dos barras deslizables, una para K_c y otra para τ_I con lo cual podemos modificar ambos valores y ver como se comporta la curva de variable de proceso (Fig.29). Además, se imprime el valor del error absoluto integral (IAE: *Integral absolute error*), lo cual nos ayuda obtener los mejores parámetros si minimizamos dicho

error. Lo cual nos lleva a obtener el mejor comportamiento de la curva PV y el mínimo error en $K_c = 6$ y $\tau_I = 55$.

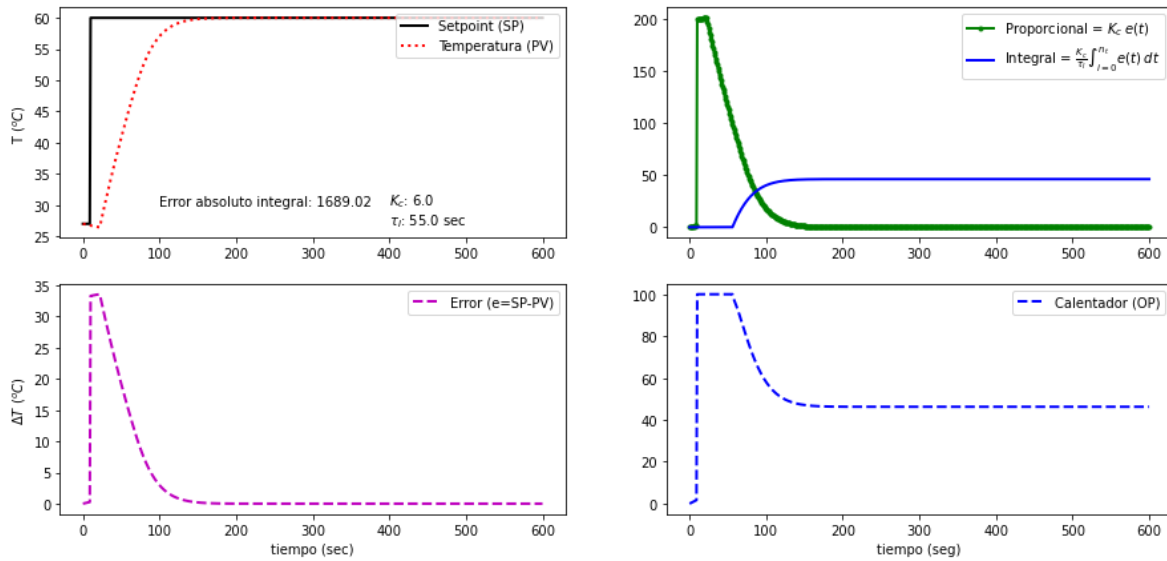


Figura 29. Simulador de sintonización PI.

Los valores anteriores podrían ser utilizados para evaluar el controlador en TCL, pero en el programa se evalúa la sintonización con IMC agresivo (34) y con ITAE seguimiento del punto de ajuste (39). El objetivo es ver cuál de las dos correlaciones obtiene un menor IAE.

Como podemos ver en las figuras 30 y 31 se puso a prueba el controlador PI con las correlaciones indicadas en el párrafo anterior. Observando las curvas veríamos un comportamiento muy similar y sería difícil determinar cuál sintonización es mejor, pero con el IAE podemos distinguir fácilmente entre uno y otro, siendo el de menor valor el que más interese al momento de la selección, pues se pretende disminuir el error.

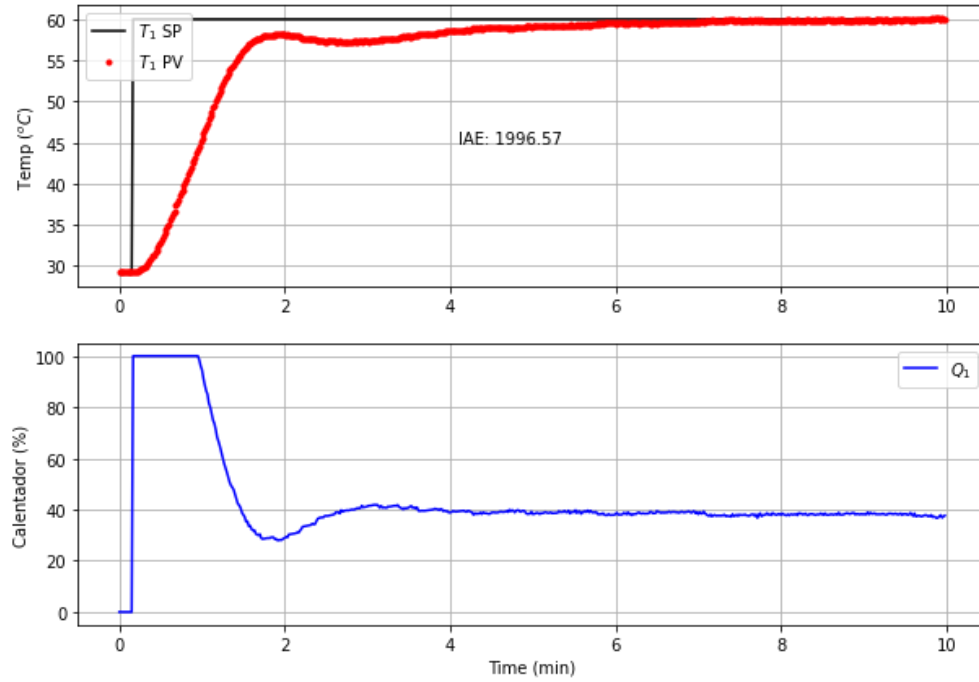


Figura 30. Sintonización con IMC agresivo

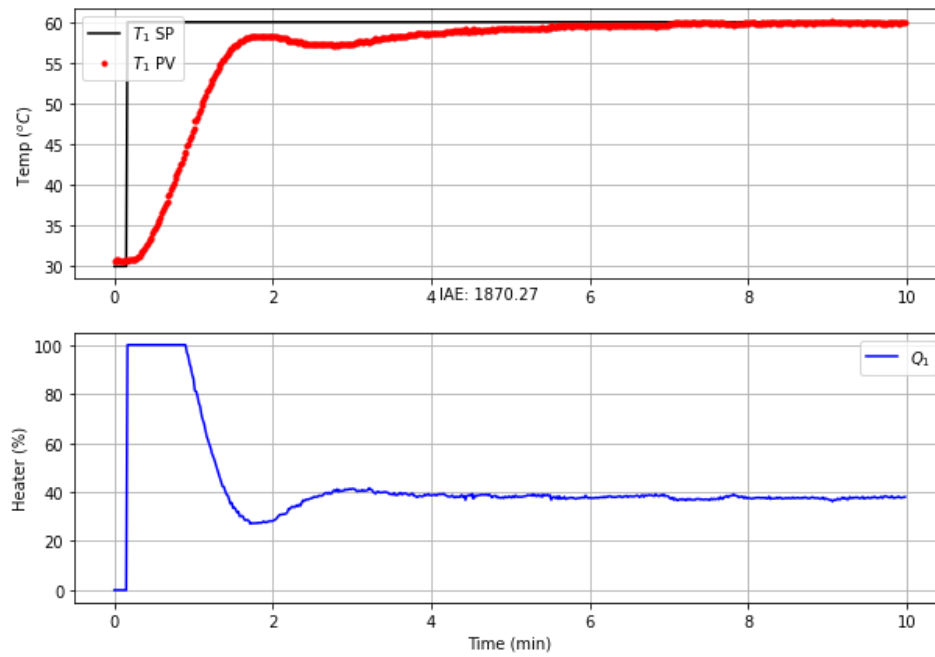


Figura 31. Sintonización con ITAE

En la última celda se utiliza simple-pid que nos permite generar controladores pid de una forma simple, solo es necesario instalar e importar dicha función.

Para instalar usar:

```
6 !pip install simple-pid
```

Y para importar:

- ***from simple_pid import PID***

Se programa de forma que se realicen dos pruebas, una con IMC y la otra con ITAE con una sola celda, el proceso en total dura 40 minutos pues se detiene durante 10 minutos entre cada operación para que el sistema se enfríe. Se obtienen dos gráficas y aunque el IAE tiene el mismo comportamiento, a diferencia de las celdas programadas con el PID utilizado anteriormente muestran sobre impulso, lo cual puede ser debido a un ajuste diferente en la función simple-pid que debe ser analizada en la documentación del programa.

Se observa que la sintonización más eficiente es la ITAE por tener un menor error integral durante el experimento con TCL.

- Error IMC = 1996
- Error ITAE = 1870

Utilizando simple-pid

- Error IMC = 1942
- Error ITAE = 2385

5.11 Sintonización de control PID

En este programa se realiza la sintonización de un control PID, probando su rendimiento con algunos cambios del valor del SP, los parámetros ajustables son K_c , τ_I y τ_D . En la sintonización de un PID debe evitarse además del sobre impulso y el comportamiento oscilatorio de la señal de salida. Debe evitarse también el incremento de oscilaciones en la salida del controlador provocado por el aumento del parámetro derivativo.

Simulador de sintonización PID

Este simulador contiene al modelo FOPDT así como el cambio de SP de 50°C a los 10 segundos y 40° a los 300 segundos. Contiene tres barras deslizantes con los tres parámetros a sintonizar K_c , τ_I y τ_D . Los cuales fueron ajustados para que el IAE disminuya. Al aumentar τ_D se puede seguir disminuyendo el valor de IAE, pero se deja en un valor bajo pues al aumentar es posible incrementar el ruido en la señal de salida del controlador.

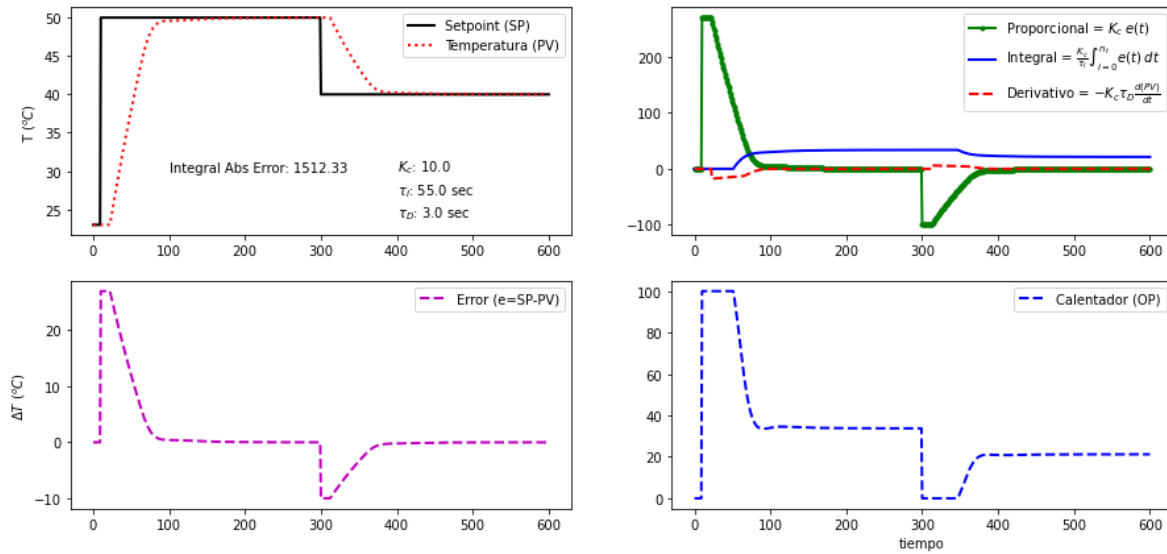


Figura 32. Sintonización con simulador PID

La celda para ejecutar en TCL se recomienda correr en IDLE. Los archivos utilizados para usar IDLE se encuentran en la carpeta “Simulación en IDLE” dentro del programa P11. Para abrir IDLE seguir los siguientes pasos:

1. Abrir anaconda prompt.
2. Teclar IDLE seguido de enter.
3. Dentro de IDLE click en file luego open.
4. Abrir archivo test.py. (este archivo puede crearse en Notepad++)
5. Click en run.
6. Run module.

Corriendo en IDLE se puede visualizar el cambio en la gráfica en tiempo real sin que se guarde la figura anterior. Además, solo corriendo en IDLE el archivo PID_control.png puede visualizarse.

Al final se obtiene el gráfico PID_control.png (fig. 33). En ella se observa que la variable de proceso presenta sobre impulso a diferencia de la simulación en la que nunca se rebaza el SP. El sistema TCL se debe principalmente al ruido del sistema real, lo cual incrementa el IAE y produce el comportamiento oscilatorio de la variable de proceso.

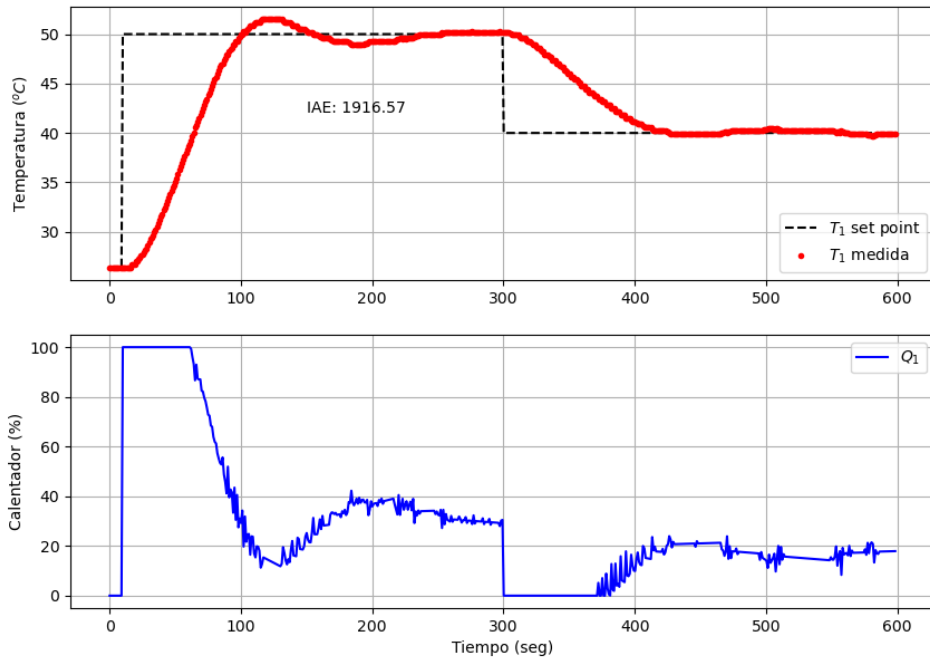


Figura 33. Sintonización PID del TCL.

5.12 PID con retroalimentación

En el programa P12 Tclab “PID con retroalimentación.ipynb” se realiza primero una prueba escalonada con el calentador 2 al 100% para poder determinar K_d (Fig.34).

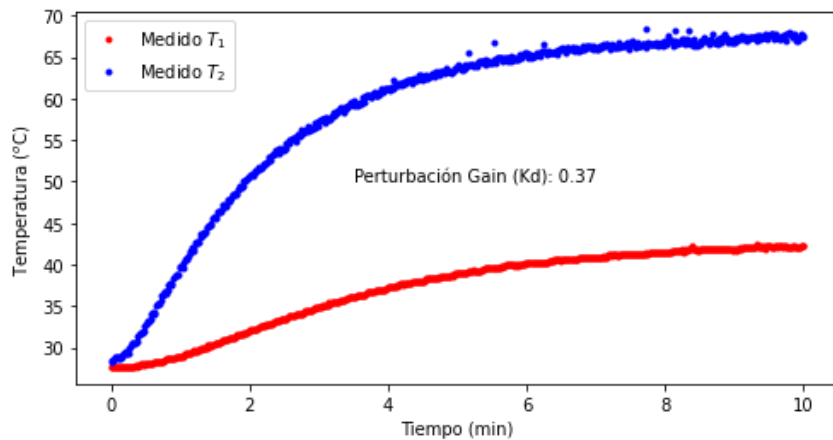


Figura 34. Prueba escalonada para determinar K_d

Con K_d podemos calcular K_{ff} usando la siguiente ecuación que proviene de la relación vista en (51):

$$K_{ff} = -\frac{K_d}{K_p} = -\frac{0.37}{0.8} = -0.4625 \quad (70)$$

Con lo anterior es posible utilizar el simulador para ver el desempeño de los valores obtenidos, contiene cuatro barras deslizables $K_c, \tau_I, \tau_D, K_{ff}$ (Fig.35). Vemos que la variable controlada se mantiene en el SP aunque la perturbación sea grande y persistente en el proceso, además no se muestra sobre impulso notorio y oscilaciones en la temperatura controlada (Temperatura 1(PV)). También notamos que el calentador Q1 no muestra oscilaciones. Con esta información es posible aplicarlo en TCL para probar el control ff.

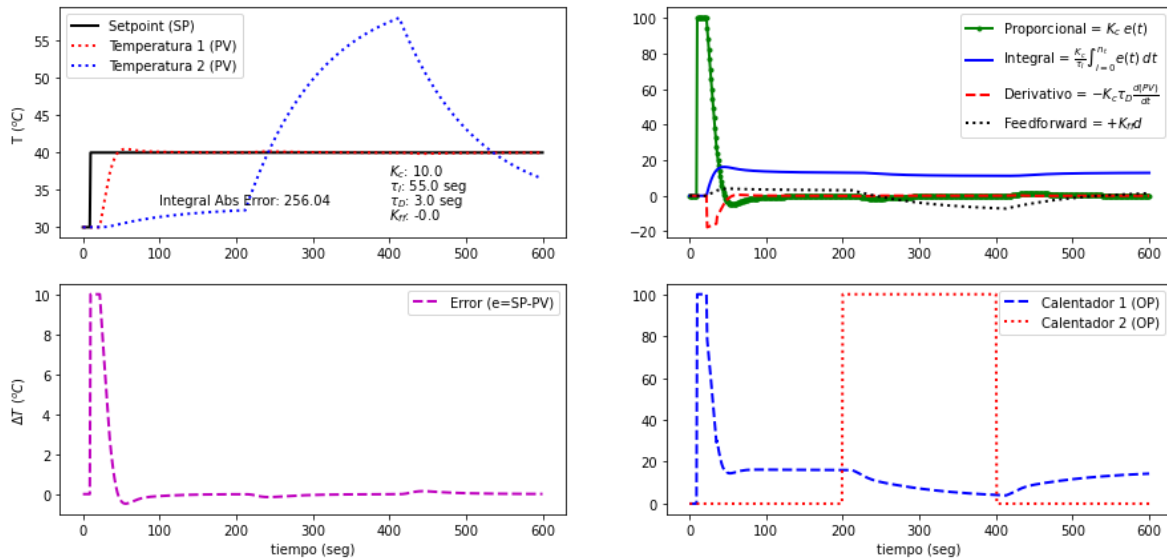


Figura 35. Simulador PID feedforward

Para aplicar el control ff en TCL se recomienda usar IDLE. En la siguiente gráfica (fig.36) podemos notar un sobre impulso que no era significativo en la simulación, pero regresa rápidamente al SP. Al encender el calentador 2 (perturbación) a los 200 segundos el parámetro ff reduce el valor del calentador 1, para poder mantener la temperatura 1 en el SP. Al apagarse el calentador 2 el calentador 1 nuevamente empieza a aumentar para compensar la ausencia de la perturbación.

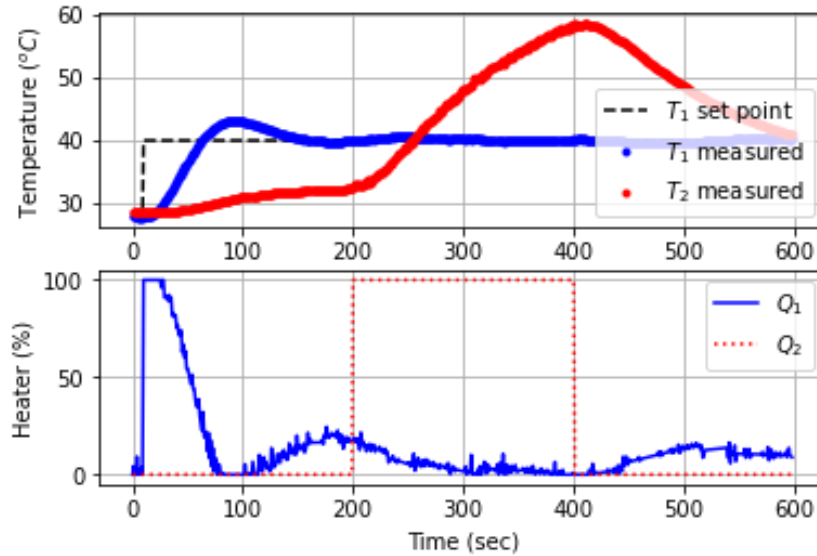


Figura 36. Control PID feedforward del TCL.

5.13 Modelo FOPDT para el calentador 2 (Q2)

En este programa P13 se desarrolló un modelo FOPDT para el segundo calentador Q2. Partiendo de la ecuación FOPDT (22) podemos resolver con transformada de Laplace para obtener la ecuación (71) cuya obtención puede verse dentro de P5 en el archivo “Retardo de tiempo en sistemas dinámicos.ipynb”:

$$y(t) = K_p \left(1 - \exp\left(\frac{(t - \theta_p)}{\tau_p}\right) \right) \Delta u S(t - \theta_p) \quad (71)$$

Donde $y(t)$ será la temperatura, Δu es el cambio de paso Q2 y $S(t - \theta_p)$ es una función escalonada que cambia de 0 a 1 en $t = \theta_p$.

Modelo FOPDT del calentador 2

El único parámetro que cambia es el de ganancia ($K_p=0.8/2$) pues solo está recibiendo la mitad de la potencia. Los otros parámetros θ_p y τ_p serán los mismos que en Q1 porque provienen del mismo hardware. Al registrar las temperaturas con una prueba de paso en Q2 y añadiendo también el modelo FOPDT (71) con los parámetros antes mencionados, se obtuvo el gráfico de la figura 37 donde podemos observar que el modelo FOPDT se acerca bastante a los datos medidos, lo que permite saber que el modelado es correcto.

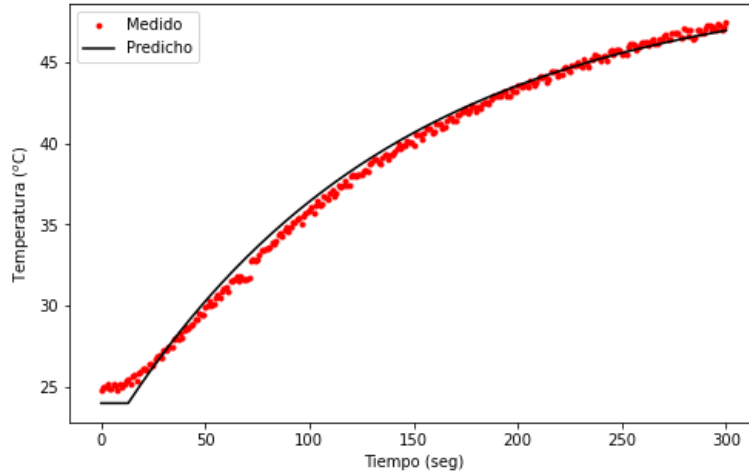


Figura 37. Simulación y comprobación del modelo FOPDT de Q2

Prueba de escalón animada del calentador 2 (Q2)

En esta prueba escalón se programan múltiples valores para Q2 y se añade un escalón en el calentador 1. La ecuación utilizada para la simulación es la solución analítica del modelo FOPDT expresada de la siguiente manera:

$$T(t) = (T_1 - T_{ss}) \exp\left(\frac{-dt}{\tau_p}\right) + (Q_1 - Q_{ss}) \left(1 - \exp\left(\frac{-dt}{\tau_p}\right)\right) K_p + T_{ss} \quad (72)$$

La Fig.38 muestra la gráfica final de esta prueba escalón animada. Se ve que el comportamiento simulado es similar al de los datos medidos. Vemos que la T1 medida se encuentra por encima de la simulación, esto es debido a que no tiene información de la perturbación provocada por Q2. Así mismo la simulación de la T2 no contiene información de lo que ocurre en Q1, notándose más la desviación cuando Q1 se activó.

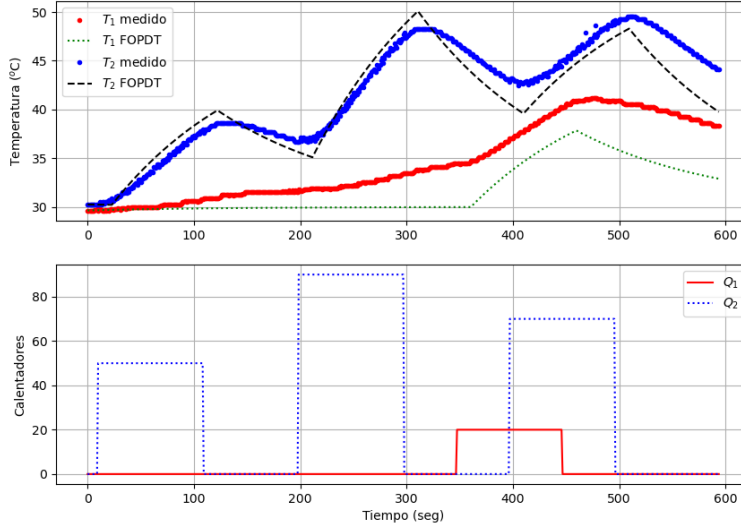


Figura 38. Prueba de paso múltiple para Q1 y Q2.

5.14 Prueba de sensores de temperatura

Los problemas propuestos en P14 se resuelven utilizando la relación vista en el capítulo 4.1 Ec. (1). Se definen dos funciones, una que convierte de mV a °C y otra que pasa de °C a mV. Si despejamos mV de la ecuación (1) obtenemos:

$$mV = (T + 50) * 10 \quad (73)$$

Y programado en Python la ecuación (1) quedaría como se observa debajo en las líneas 2 y 3 y la ecuación (73) sería la función de las líneas 5 y 6. Mientras que la llamada a la función se hace poniendo el nombre definido y el argumento (que es la variable que entra) dentro de paréntesis como se muestra en las líneas 8 y 9. El código completo con los problemas del programa resueltos se encuentran en “Tclab Sensores para automatización.ipynb” dentro de P14.

```

1 #Función que pasa de mV a °C
2 def T(mV):
3     return round(0.1*mV-50.0,1)
4 #Función que pasa de °C a mV
5 def mV(T):
6     return round((T+50.0)*10.0,2)
7
8 mV(25) #Pasa de T°C a mV
9 T(0.5*1000) #Pasa de mV a T°C; el 1000 es para pasar de V a mV

```

5.15 Diagrama de bloques

Diagrama de bloques con control PI

El controlador PI ($K_c=2$ y $\tau_i=180$) es una función de transferencia (ver ecuación (58)) que transforma la señal de error (entrada) en un valor del calentador (Q_1 , salida del controlador). La función de transferencia del proceso se obtiene utilizando la ecuación (55) y sustituyendo los valores de los parámetros PI obtenemos $G_p = \frac{0.9}{180s+1} e^{-15s}$ que recibe el valor del calentador (Q_1) como entrada y produce un cambio en la salida de la variable de proceso de temperatura 1 (TA_1). La función de transferencia de la perturbación es $G_d = \frac{0.3}{180s+1} e^{-15s}$ tiene una entrada de la temperatura 2 (T_2) y produce un cambio en la salida de la variable de proceso de temperatura T_1 (T_{B1}). Las dos variables de temperatura se suman $T_1 = T_{A1} + T_{B1}$. El lazo se cierra con el sensor proporcionando información al controlador como un error ($E=T_{SP}-T_1$) entre el SP y el valor medido.

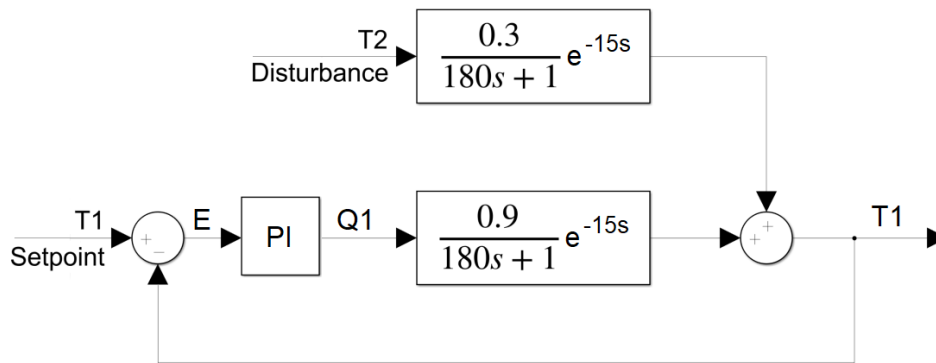


Figura 39. Diagrama de bloques con funciones de transferencia. Extraída de: <https://apmonitor.com/pdc/index.php/Main/TCLabBlockDiagram>

Solución analítica a la respuesta del controlador PI a un aumento de 20°C en T2

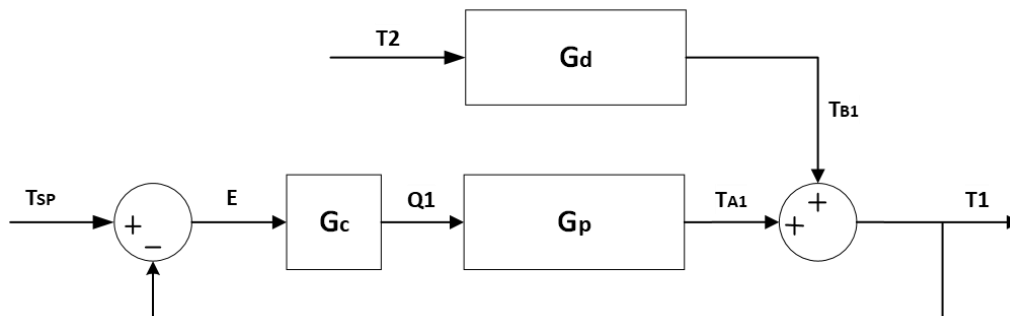


Figura 40. Diagrama de bloques con una perturbación.

Utilizando algebra de bloques en la fig.40 se obtiene la función de transferencia general, comenzamos definiendo la señal de error $E(s)$ que como vemos procede de un punto de suma:

$$E(s) = T_{SP}(s) - T_1(s)$$

Podemos observar que tenemos dos bloques en serie G_c y G_p por lo tanto

$$T_{A1}(s) = E(s)G_c(s)G_p(s)$$

Sustituyendo

$$T_{A1} = G_c(s)G_p(s)(T_{SP}(s) - T_1(s))$$

Ahora en el punto de suma de la salida vemos que entra en juego la perturbación por lo tanto primero definimos la señal T_{B1} :

$$T_{B1}(s) = G_d(s)T_2(s)$$

Por lo que el punto de suma quedaría:

$$T_1(s) = T_{A1} + T_{B1} = G_c(s)G_p(s)(T_{SP}(s) - T_1(s)) + G_d(s)T_2(s)$$

Para este caso $T_{SP}=0$, pues no hay cambio del SP

$$T_1(s) = -G_c(s)G_p(s)T_1(s) + G_d(s)T_2(s)$$

Reacomodando términos y sustituyendo las funciones de transferencia, la función de transferencia global queda:

$$\frac{T_1(s)}{T_2(s)} = \frac{G_d(s)}{1 + G_c(s)G_p(s)} = \frac{\frac{0.3}{180s+1} e^{-15s}}{1 + K_c \frac{\tau_I s + 1}{\tau_I s} \frac{0.9}{180s+1} e^{-15s}} \quad (74)$$

La función global nos da información relevante, como puede verse en (74) se produce un cambio en la salida $T_1(s)$ debido a un cambio en la entrada que es la perturbación $T_2(s)$.

Utilizando una función de Python simbólico podemos obtener la transformada de Laplace inversa.

```
from sympy.integrals import inverse_laplace_transform
```

Para observar más claro el resultado del primer problema planteado en este programa se obtiene un gráfico (fig.41). Donde vemos la predicción del cambio de la T_1 debida a una perturbación T_2 .

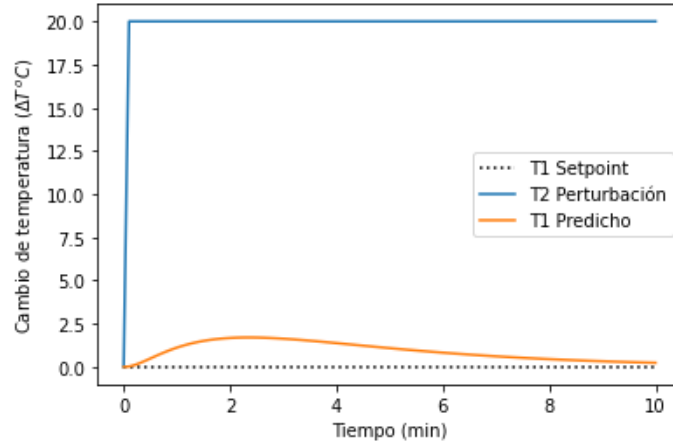


Figura 41. Predicción de T₁ debido a un cambio de 20°C en T₂

Solución analítica T₁ a un cambio en de SP de 20°C

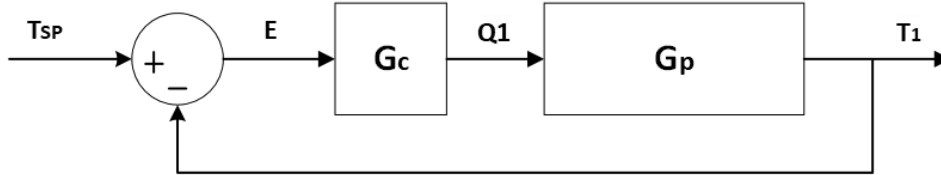


Figura 42. Diagrama de bloques con un cambio del SP

Habiendo omitido la perturbación la algebra de bloques se simplifica, la función global de transferencia queda:

$$\frac{T_1(s)}{T_{SP}(s)} = \frac{G_c G_p(s)}{1 + G_c G_p(s)}$$

La solución simbólica simplificada es:

$$T(t) = (6.14e^{-0.054} - 26.13e^{-0.012} + 20)S(t - 1)$$

Prueba de control PI con solución analítica

La curva de valores predichos no se ajusta completamente a la de valores medidos (PV), principalmente debido a que se están usando valores de parámetros (K_p , τ_p , y θ_p) que no son los del dispositivo TCL usado en este proyecto. Pero es suficiente para ilustrar que la solución analítica predice el comportamiento de la temperatura controlada con un PI (ver Fig.43).

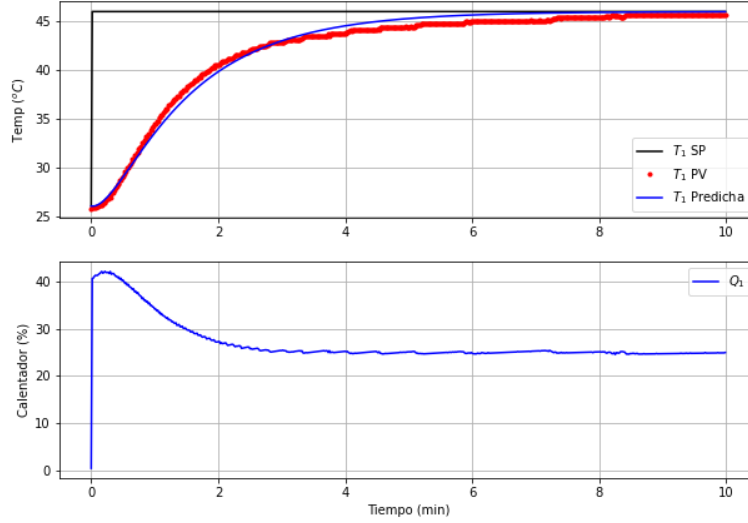


Figura 43. Prueba de control PI con solución analítica de un cambio en SP.

5.16 Espacio de estado

El sistema que se analiza en este programa es un sistema no lineal, por lo que es necesario linealizar, en este caso se utilizan las series de Taylor (ver linealización en “Modelos de espacio de estado.ipynb” P16).

El balance del TCL es el mostrado en el capítulo 4.1.2 para un sistema de cuarto orden (9), (10), (11), (12) y (13).

El modelo de espacio de estado partiendo de la información de los balances de energía sería el siguiente:

$$\begin{bmatrix} \dot{T}'_{H1} \\ \dot{T}'_{H2} \\ \dot{T}'_{C1} \\ \dot{T}'_{C2} \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} \begin{bmatrix} T'_{H1} \\ T'_{H2} \\ T'_{C1} \\ T'_{C2} \end{bmatrix} + \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \\ b_{3,1} & b_{3,2} \\ b_{4,1} & b_{4,2} \end{bmatrix} \begin{bmatrix} Q'_1 \\ Q'_2 \end{bmatrix} \quad (75)$$

$$\begin{bmatrix} T'_{C1} \\ T'_{C2} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} T'_{H1} \\ T'_{H2} \\ T'_{C1} \\ T'_{C2} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} Q'_1 \\ Q'_2 \end{bmatrix} \quad (76)$$

Las dimensiones de las matrices se obtienen partiendo de definir m, n y p. En m=2 pues solo hay dos entradas Q1 y Q2. En n=4 se cuentan los estados, provenientes de cada una

de las ecuaciones diferenciales que se tienen. Y finalmente para $p=2$ se cuentan las salidas que serán T1 y T2. Utilizando las definiciones en (31) se obtiene: $A \in \mathbb{R}^{4 \times 4}$, $B \in \mathbb{R}^{4 \times 2}$, $C \in \mathbb{R}^{2 \times 4}$ y $D \in \mathbb{R}^{2 \times 2}$, para el sistema analizado en este programa.

Los elementos de las matrices de (75) son las derivadas parciales de la función con respecto de la variable que se indique. ($a_{función(n),variable(n)}$) en condiciones de estado estacionario. En (77) se realizaron algunas derivadas parciales para ilustrar el proceso que se sigue antes de programar las matrices.

$$\begin{aligned}
 a_{1,1} &= \frac{\partial f_1}{\partial T_{H1}} \Big|_{\bar{T}} = - \frac{UA + 4\epsilon\sigma AT_0^3 + U_s A_s + 4\epsilon\sigma A_s T_0^3}{mC_p} \\
 a_{1,2} &= \frac{\partial f_1}{\partial T_{H2}} \Big|_{\bar{T}} = \frac{U_s A_s + 4\epsilon\sigma A_s T_0^3}{mC_p} \\
 b_{1,1} &= \frac{\partial f_1}{\partial Q_1} \Big|_{\bar{T}} = \frac{\alpha_1}{mC_p}
 \end{aligned}
 \tag{77}$$

Matrices en Numpy

La primera celda de código solo contiene la manera en que se deben programar las matrices. Para generar una matriz de ceros utilizando numpy se escribe de la siguiente manera:

```
Am = np.zeros((4,4))
Bm = np.zeros((4,2))
Cm = np.zeros((2,4))
Dm = np.zeros((2,2))
```

El primer argumento son filas y el segundo son columnas.

Para introducir valores a las matrices solo hay que indicarle al programa en que posición se colocará dentro de la matriz:

```
Am[0,0] = -(c1+c2+c3+c4)/c5
Am[0,1] = (c3+c4)/c5
Cm[0,2] = 1
Cm[1,3] = 1
```

En la primera línea del código anterior se estaría indicando que en la primera columna y fila debe introducirse la operación señalada.

Espacio de estado con Gekko

Para generar un modelo de espacio de estado con Gekko tan solo hay que añadir el código mostrado debajo, donde la función **ss.state_space** recibe como argumentos las matrices previamente creadas y que contienen información del modelo basado en la física del sistema, esta devolverá las variables: estados (x), salidas (y) y entradas (u). Se le

añaden los valores programados en los 2 calentadores y el tiempo en que se ejecuta previamente guardados en una data. Se define el tipo de problema con **IMODE=7** de tipo secuencial (SQS: *Sequential*). Y al final se le pide que resuelva el modelo de espacio de estado con **ss.solve()**.

```
ss = GEKKO ()
x,y,u = ss.state_space (Am,Bm,Cm,D=None)
u[0].value = data['Q1'].values
u[1].value = data['Q2'].values
ss.time = data['Time'].values
ss.options.IMODE = 7
ss.solve (disp=False)
```

Espacio de estado con Scipy

Un modelo de espacio de estado con Scipy es muy similar al realizado con Gekko, aunque utiliza menos líneas de código y la sintaxis comúnmente utilizada en Python. El modelo (sys: modelo espacio de estado) se crea con la función **signal.StateSpace()** que recibe como argumentos las matrices previamente creadas. Requerirá que añadamos las variables de tiempo y los dos calentadores de la data. Para leer los datos correctamente hay que acomodarlos primero en una matriz vertical con **np.vstack** y luego transponerlos para que queden una matriz con 2 columnas y n filas (n: en este problema es 601) con Qsys.T. Para resolver se utiliza **signal.lsim()** que recibe argumentos sys, Qsys, tsys y devuelve los vectores de espacio de estados (ver ecuación (75) y (76)): ysys=y (Tc1 y Tc2: temperaturas simuladas de los sensores 1 y 2) y xsys=ẋ.

```
sys = signal.StateSpace (Am,Bm,Cm,Dm)
tsys = data['Time'].values
Qsys = np.vstack ((data['Q1'].values,data['Q2'].values))
Qsys = Qsys.T
tsys,ysys,xsys = signal.lsim(sys,Qsys,tsys)
```

Solución de simulación linealizada

En la última celda de código del programa “Tclab Espacio de estado.ipynb” dentro de P16 se realizó la comprobación de la simulación de los modelos basado en la física y por espacio de estado y se comparó con la ejecución del proceso en TCL, en la que se le indicaron 4 pasos distintos en Q1 (80,20,70 y 50) y tres pasos en Q2 (100,10, 80). El tiempo para la ejecución se ajusta a 10 minutos. Después de programar lo que ocurrirá en TCL se crea un modelo con Gekko para simular el modelo basado en la física, recibirá algunas constantes y otras variables que se ajustarán con un método de optimización dentro de la misma celda. Después se crean las matrices A, B, C y D del modelo de espacio de estado y se añade la simulación con gekko. Al final se programa el gráfico (Fig. 44).

Modelo basado en la física (modelo no lineal). vemos como los modelos de los balances de energía predicen con exactitud el comportamiento de las curvas de valores medidos, esto se logra con la optimización de los parámetros. En contraste el modelo de espacio de

estado (modelo lineal) aunque provenga de los parámetros ajustados del modelo basado en la física se aleja de la curva de valores medidos pues este modelo es una linealización de los balances de energía que son no lineales.

Tabla 3. Parámetros del modelo basado en la física.

Parámetros estimados	
U	: 8.4697418288
Us	: 40.0
alpha1	: 0.01474934556
alpha2	: 0.0077526420055
tau	: 17.108090708
Constantes	
Ta	: 298.15
m	: 0.004
Cp	: 500.0
A	: 0.001
As	: 0.0002
eps	: 0.9
sigma	: 5.67e-08
SAE Energy Balance	: 507.50644904000035

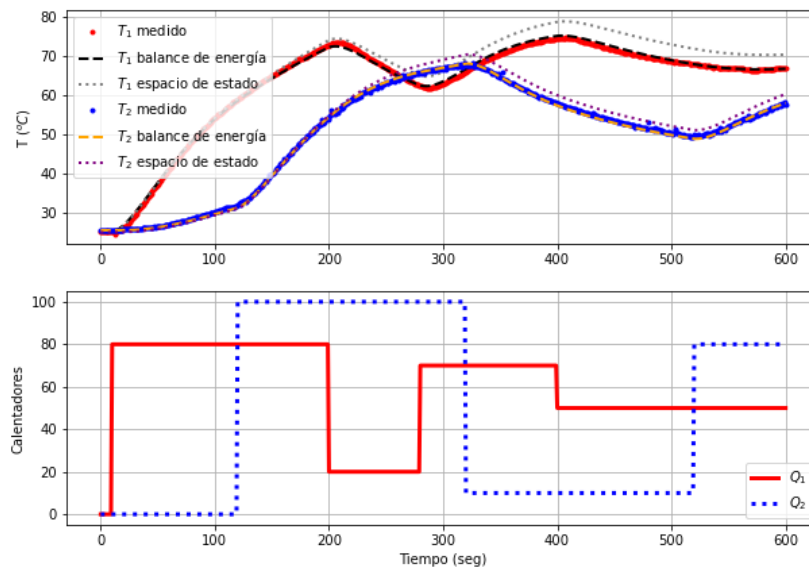


Figura 44. Simulación de modelos SS y basado en la física

5.17 Control On-Off

En este programa se genera un sistema de segundo orden utilizando un control On-Off y un método gráfico para obtener los parámetros del modelo (K_p , ζ y τ_s). El tiempo muerto no es necesario para este modelo. (ver ecuación (24))

Control On-Off en Python

Un controlador On-Off como su nombre lo indica enciende y apaga al sistema para llegar a la temperatura que se le está indicando. Es comúnmente utilizado en sistemas de ventilación y calefacción. En este programa se le indicó al TCL que alcance una temperatura de 40°C durante un periodo de tiempo de 4 min. Lo que al final arroja un comportamiento de un sistema de segundo orden.

El control On-Off se logra escribiendo el siguiente ciclo for:

```
9 for i in range(240):
10     time.sleep(1)
11     T1 = a.T1
12     Q1 = 100 if T1<=40.0 else 0
13     a.Q1(Q1)
14     print('Time: '+str(i)+' Q1: '+str(Q1)+' T1 (SP=40): '+str(T1))
15     fid = open('data.csv','a')
16     fid.write(str(i)+' '+str(Q1)+' '+str(T1)+'\n')
17     fid.close()
18 a.close()
```

Del código anterior vemos que se le está indicando al TCL que se encienda por 240 segundos tome lecturas de temperatura del sensor 1 y establezca los nuevos valores para el calentador. Las demás líneas están creando un archivo de datos. El control On-Off se encuentra en la línea 12, con el condicional if...else. Q1 Será 100% si la temperatura leída es menor a 40°C y 0% cuando tome valores mayores a 40°C.

El resultado del control On-Off puede observarse en la Fig. 45. Puede verse que cuando la temperatura llego a 40°C el calentador se va a cero, dicho de otro modo, se apaga por completo.

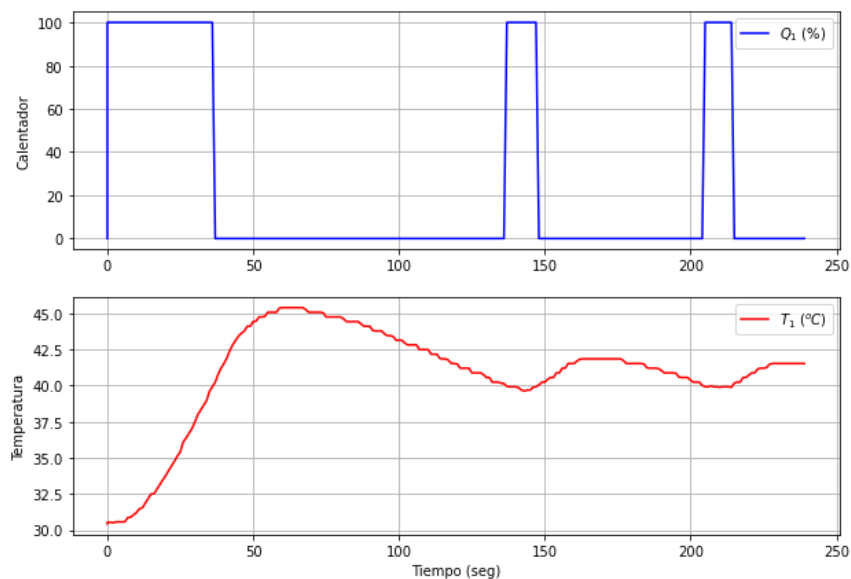


Figura 45. Control On-Off en TCL

Desarrollando los pasos descritos en 4.2.4 y con la gráfica de la Fig. 45 tenemos:

1. ΔT_1 empieza en 30 y tiene un valor final promedio de 41. El cambio de temperatura es $41 - 30 = 11^\circ C$.
2. ΔT_{SP} : EL SP es el cambio de 30 a 40, entonces $\Delta T_{SP} = 40 - 30 = 10^\circ C$.
3. $K_p = \frac{\Delta T_1}{\Delta T_{SP}} = \frac{11}{10} = 1.1$
4. La relación OS es B/A (ver fig.7). $\frac{B}{A} = \frac{45-41}{41-30} = 0.363$

Utilizando la ecuación de sobre impulso OS:

$$\zeta = \sqrt{\frac{(\ln(OS))^2}{\pi^2 + (\ln(OS))^2}} = 0.306$$

ζ se encuentra entre $0 \leq \zeta < 1$ por que se considera un sistema subamortiguado.

5. Calcular τ_s de la ecuación de tiempo pico $t_p = 60$ (ver fig.7).

$$\tau_s = \frac{1 - \zeta^2}{\pi} t_p = \frac{1 - 0.215^2}{\pi} (60) = 18.18$$

Para comprobar los parámetros los parámetros obtenidos, se utiliza la data generada en el control On-Off que se ha guardado en un archivo .csv (ver archivo "data.csv" dentro de P17). Al tener zeta con un valor de 0.306, la solución analítica para un sistema subamortiguado ($0 \leq \zeta < 1$) es la siguiente (para ver otras relaciones y ecuaciones ver archivo "Sistemas de Segundo Orden.ipynb" en P17:

$$y(t) = K_p M \left(1 - e^{-\zeta t / \tau_s} \left[\cos \left(\frac{t}{\tau_s} \sqrt{1 - \zeta^2} \right) + \frac{\zeta}{\sqrt{1 - \zeta^2}} \sin \left(\frac{t}{\tau_s} \sqrt{1 - \zeta^2} \right) \right] \right) \quad (78)$$

Lo primero que se hizo en esa celda es calcular nuevamente los parámetros por medio del ajuste gráfico, después se programa la solución analítica (78) como se muestra abajo. Vemos como se están creando las variables de t (tiempo) y T0 (temperatura inicial) utilizando la data previamente cargada con `pd.read_csv('data.csv')`. Después se crean las variables a, b, c, d con el fin de simplificar la expresión (78). En la línea 27 mostrada abajo podemos ver que se escribe la ecuación (78) ya simplificada.

```

21 t = data['Time'].values
22 T0 = data['T1'].values[0]
23 a = np.sqrt(1-zeta**2)
24 b = t/taus
25 c = np.cos(a*b)
26 d = (zeta/a)*np.sin(a*b)
27 T1 = Kp*Delta_SP*(1-np.exp(-zeta*b)*(c+d))+T0

```

En la última parte de esta celda se pide que imprima las gráficas (ver fig.46). Una para el comportamiento del calentador y otra para las curvas de temperatura medida y predicha. Puede verse que las curvas T1 Medido y T1 Predicho, se encuentran cercanas en

comportamiento, pero no predicen todas las oscilaciones con exactitud. Como una primera aproximación el método gráfico es el mejor camino para entender el comportamiento de un sistema de segundo orden. Para mejorar el resultado se tendría que utilizar un método de optimización para obtener los mejores parámetros que describirán a nuestro sistema.

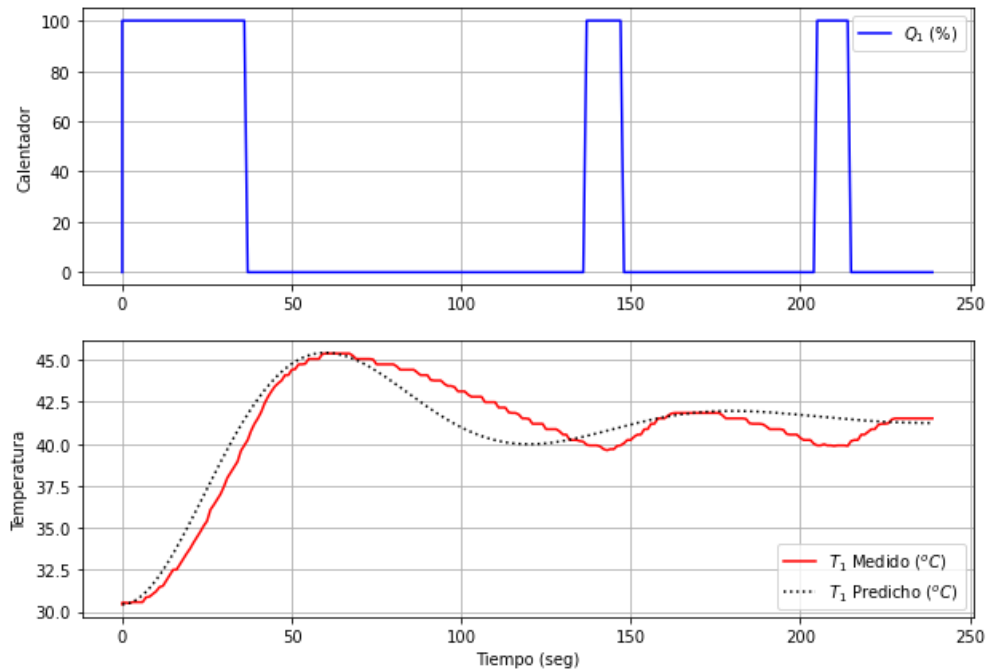


Figura 46. Ajuste gráfico del sistema de segundo orden del TCL.

5.18 Regresión en tiempo real de segundo orden

Por lo visto en el programa anterior el modelo necesita una optimización de los parámetros de segundo orden (K_p , ζ y τ_s) para mejorar la predicción. En este programa se presentan dos maneras en las que se puede hacer una regresión de los datos medidos. La primera utilizando scipy y la segunda con Gekko.

Se utilizará un modelo para un sistema sobre amortiguado (OD: *overdamped*) es decir que el factor de amortiguamiento es mayor a 1 ($\zeta > 1$). La solución analítica de (24) para un sistema sobre amortiguado es:

$$y(t) = K_p M \left(1 - e^{-\zeta t / \tau_s} \left[\cosh \left(\frac{t}{\tau_s} \sqrt{\zeta^2 - 1} \right) + \frac{\zeta}{\sqrt{\zeta^2 - 1}} \sinh \left(\frac{t}{\tau_s} \sqrt{\zeta^2 - 1} \right) \right] \right) \quad (79)$$

El tamaño de estos programas es grande por lo tanto se pide al lector que revise el archivo 'Tclab Regresion en tiempo real de 2do orden.ipynb' en P18, en donde encontrará el código con la información necesaria para entender lo que se está haciendo.

Regresión adaptativa de 2do orden con Scipy Minimize

La función de Scipy que se utiliza es:

```
from scipy.optimize import minimize
```

Para activarla recibe una función objetivo que tenemos que crear antes de llamarla, los parámetros del modelo de segundo orden y los argumentos `tm` y `T1` son el segmento de tiempo y temperatura en el que se encuentra el ciclo.

```
minimize(func.objet,parametros,args=(tm[0:i+1],T1[0:i+1]))
```

Esta primera optimización toma valores de TCL cada 1 segundo y optimiza después de 15 segundos solo en los puntos que son múltiplos de 3. Visto de otra manera con el condicional `if` sería:

```
if i>=15 and (np.mod(i,3)==0):
```

Np.mod permite obtener el residuo de la división de `i`. Si es 0 quiere decir que es múltiplo de 3. Ejemplo **21 % 3 = 0**.

Debajo se proporciona un diagrama de flujo del programa esto con la finalidad de que el lector tenga una idea general de cómo se está llevando a cabo la regresión (Fig.47).

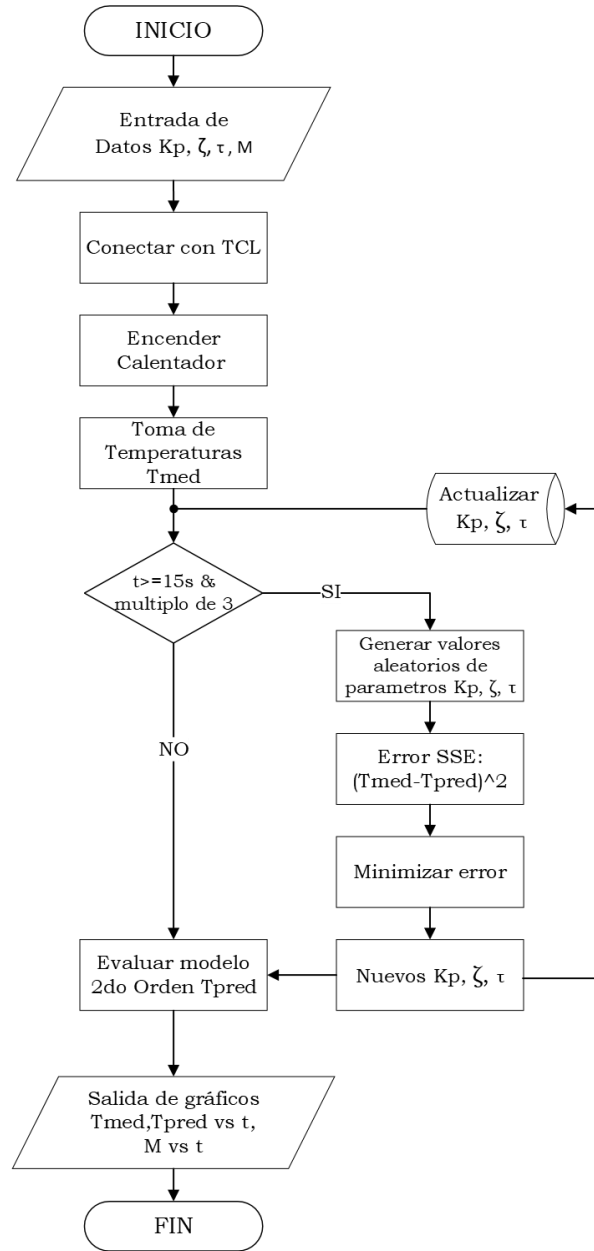


Figura 47. Diagrama de Flujo Retrogresión de 2do Orden Con Scipy

Los parámetros optimizados son:

Kp: 0.66
 taus: 61.348
 zeta: 1.265

Vemos que los parámetros optimizados logran simular completamente el sistema de segundo orden pues en la figura 48 puede verse que el comportamiento de la curva de temperaturas medidas (T1 Medi.) es igual que el de la curva de predicción (T1 Pred).

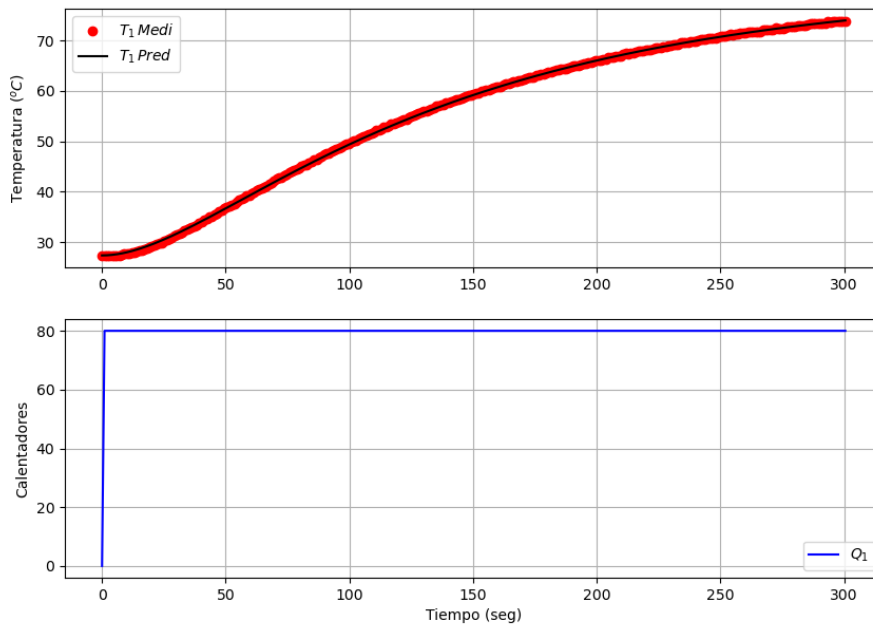


Figura 48. Optimización de 2do Orden con Scipy

Regresión adaptativa de 2do orden con Gekko

Para utilizar este método de optimización tan solo hay que importar Gekko como se hizo En el programa 1. El diagrama de flujo es prácticamente el mismo, con excepción de que se toman lecturas cada 2 segundos y se optimiza cada 5.

Los parámetros optimizados obtenidos son:

```
Kp: 0.68
taus: 62.43
zeta: 1.31
```

El optimizador logró predecir la curva de valores medidos (Fig.49). Los parámetros obtenidos con Gekko y Scipy son muy parecidos, las variaciones pueden ser debido al método de regresión que usa cada biblioteca.

Nota: Es recomendable correr ambos programas por medio de IDLE de Python.

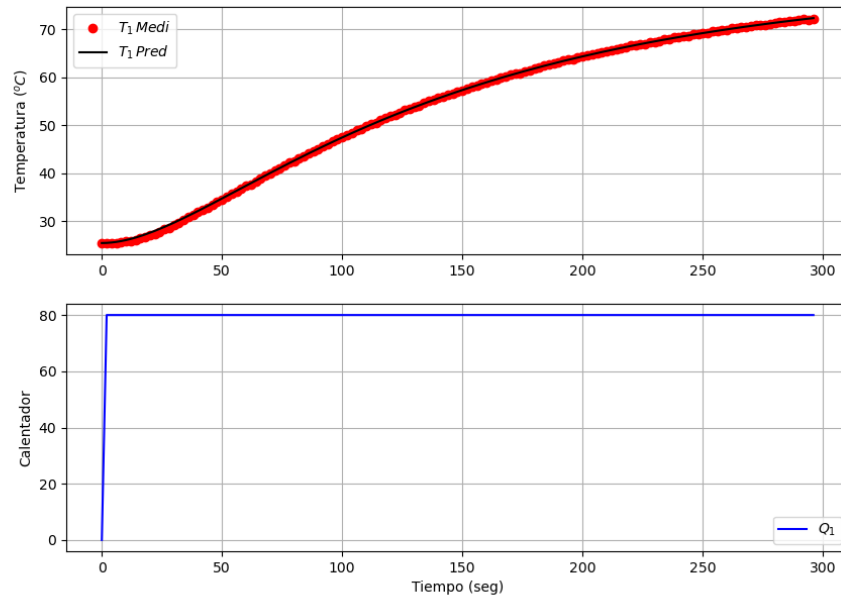


Figura 49. Optimización de 2do Orden con Gekko

5.19 Regresión de tercer orden

En este programa se realiza el modelado de TCL con una dinámica de tercer orden. Esto puede lograrse controlando la temperatura del calentador 2 utilizando el calentador 1. El desarrollo del modelo de espacio de estado se vio en P19. Por lo que los balances de energía de los dos calentadores serán (9) y (10). Por el hecho de usar solo un sensor de temperatura se toma el balance por conductividad en el sensor 2, ecuación (12). Para simplificar las ecuaciones de balance se utilizan las relaciones vistas en (13).

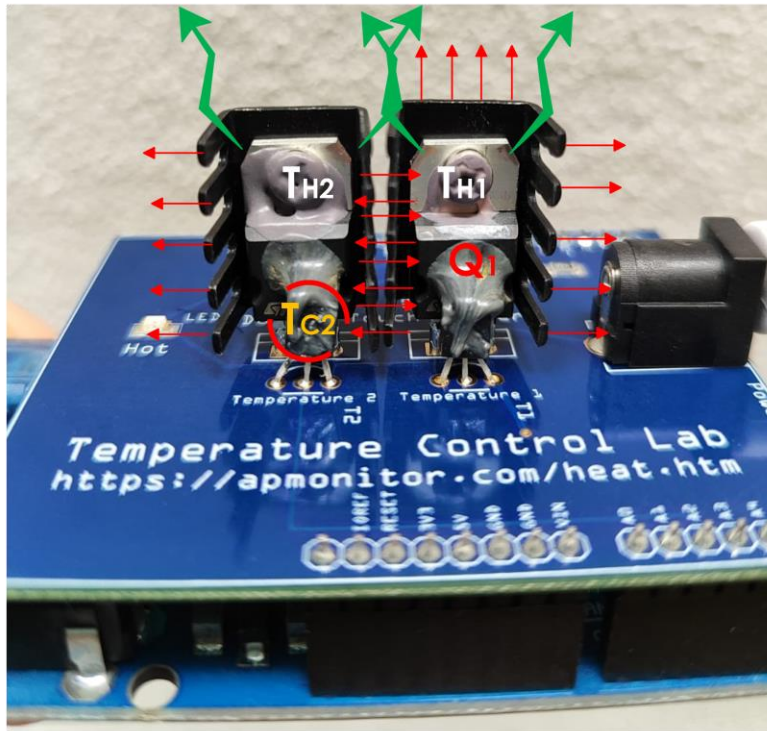


Figura 50. Modelo de 3er Orden con TCL.

Un vistazo rápido a las relaciones en (31) nos ayudan a desarrollar este modelo de espacio de estado de tercer orden. Se tienen tres estados T_{H1} , T_{H2} y T_{C2} ($n=3$), una entrada Q_1 ($m=1$) y una salida T_{C2} ($p=1$). Por lo tanto, las dimensiones de las matrices quedan $A \in \mathbb{R}^{3 \times 3}$, $B \in \mathbb{R}^{3 \times 1}$, $C \in \mathbb{R}^{1 \times 3}$ y $D \in \mathbb{R}^{1 \times 1}$.

El modelo de espacio de estado para este sistema queda como sigue:

$$\begin{bmatrix} \dot{T}'_{H1} \\ \dot{T}'_{H2} \\ \dot{T}'_{C2} \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \begin{bmatrix} T'_{H1} \\ T'_{H2} \\ T'_{C2} \end{bmatrix} + \begin{bmatrix} b_{1,1} \\ b_{2,1} \\ b_{3,1} \end{bmatrix} [Q'_1] \quad (80)$$

El sistema elegido tendrá como salida la temperatura del sensor 2 (T_{C2}).

$$[T'_{C2}] = [0 \quad 0 \quad 1] \begin{bmatrix} T'_{H1} \\ T'_{H2} \\ T'_{C2} \end{bmatrix} + [0] [Q'_1] \quad (81)$$

De forma simplificada de espacio de estado:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \quad (82)$$

La $a_{n,n}$ son derivadas parciales y podemos resolverlas manualmente:

$$a_{1,1} = \frac{\partial f_1}{\partial T_{H1}} \Big|_{\bar{Q}, \bar{T}} = -\frac{UA + 4\epsilon\sigma AT_0^3 + U_s A_s + 4\epsilon\sigma A_s T_0^3}{mC_p} = -0.0115952$$

$$a_{1,2} = \frac{\partial f_1}{\partial T_{H2}} \Big|_{\bar{Q}, \bar{T}} = \frac{U_s A_s + 4\epsilon\sigma A_s T_0^3}{mC_p} = 0.00456306$$

Nota: Las demás $a_{n,n}$ se presentan en "Tclab Regresión de 3er orden.ipynb" dentro de P19.

En la primera celda se generó una serie de pasos del calentador 1, midiendo las temperaturas tanto del sensor 1 como el sensor 2 durante 10 min. Luego se programan las ecuaciones y se utiliza una función de optimización para obtener los parámetros del balance de energía con GEKKO ($U, U_s, \alpha_1, \alpha_2, \tau$). La línea punteada de la Fig. 51 es el modelo optimizado de balance de energía mientras que la punteada en azul son los valores medidos del sensor 2. La curva roja son los valores programados de Q_1 . Además, podemos observar que la curva de balance de energía se ajusta correctamente a los valores medidos

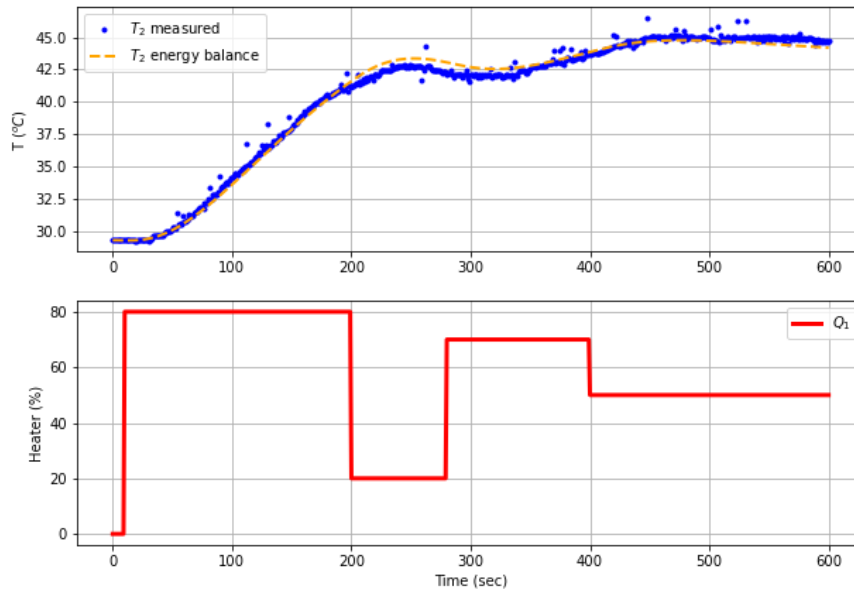


Figura 51. Optimización de modelo basado en la física.

Los parámetros obtenidos del modelo basado en la física para este sistema de tercer orden son:

Tabla 4. Variables del modelo basado en la física

	Cantidad	Valor
	Temperatura inicial (T_0)	302.15 K (20°C)
	Temperatura ambiente T_∞	302.15 K (20°C)
	Salida del calentador 1 (Q_1)	0-100 %
	Factor del calentador 1 (α_1)	0.0151 W/(% calentador)
	Salida del calentador 2 (Q_2)	0-100 %
	Factor del calentador 2 (α_2)	0.005 W/(% calentador)
	Capacidad calorífica (C_p)	500 J/kg-K
	Área de superficie no entre calentadores (A)	$1.0 \times 10^{-3} \text{ m}^2$ (10cm ²)
	Área de superficie entre calentadores (A_s)	$2.0 \times 10^{-4} \text{ m}^2$ (2cm ²)
	Masa (m)	0.004 kg (4g)
	Constante de tiempo de conducción (τ)	16.6 s
	Coefficiente general de transferencia de calor (U)	$8.4 \text{ W/m}^2 - \text{K}$
	Coefficiente general de transferencia de calor entre calentadores (U_s)	$40.0 \text{ W/m}^2 - \text{K}$
	Emisividad (ϵ)	0.9
	Constante de Stefan Boltzman (σ)	$5.67 \times 10^{-8} \text{ W/m}^2 - \text{K}^4$

Modelos restantes:

- Espacio de estado:
 - Requiere los parámetros de balance de energía optimizados.
 - Programar las matrices y las $a_{n,n}$
 - Resolver con:
 - `x,y,u = ss.state_space(Am,Bm,Cm,D=None)`-----Gekko
 - `sys = signal.StateSpace(Am,Bm,Cm,Dm)`-----SciPy
- Modelo de función de transferencia:
 - Solo se utiliza función que transforma de espacio de estado a función de transferencia.
 - `tf=sys.to_tf()` -----SciPy
- Modelo de series de tiempo utiliza los datos medidos del TCL. Temperatura del sensor 2 (T2), valores del calentador 1 (Q1) y el tiempo (Time). Para que el modelo de series de tiempo sea de 5to orden na y nb tienen que ser igual a 5 ($na=5, nb=5$).
 - `yp,p,K = ts.sysid(tt,tu,ty,na,nb...)`----- SciPy

Al final se obtiene la Fig. 52 y algunos resultados de los modelos planteados, por lo que se puede concluir lo siguiente:

- El modelo de series de tiempo de quinto orden es el que mejor se adapta a los valores medidos. (curva cortada verde)
- La curva de espacio de estado es la que menos se adapta a los valores medidos, recordar que es una linealización, por lo tanto, se alejaría del comportamiento no lineal. (curva punteada morada)
- No se graficó la curva de función de transferencia porque proviene del modelo de espacio de estado, por lo tanto, una curva idéntica.

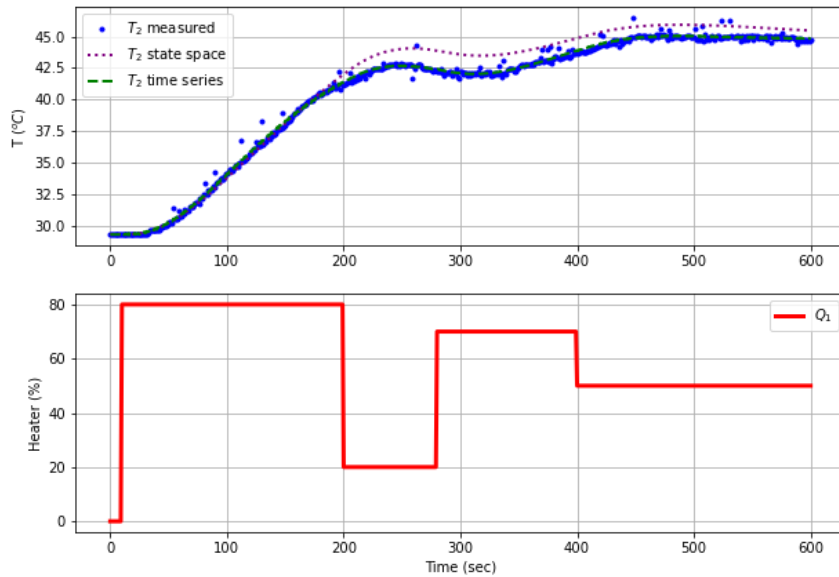


Figura 52. Simulación de modelos, espacio de estado y series de tiempo.

Modelo de espacio de estado:

$$A \in \mathbb{R}^{3 \times 3} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} = \begin{bmatrix} -0.0115952 & 0.00456306 & 0 \\ 0.00456306 & -0.0115952 & 0 \\ 0 & 0.06032572 & -0.06032572 \end{bmatrix}$$

$$B \in \mathbb{R}^{3 \times 1} = \begin{bmatrix} 0.00756636 \\ 0 \\ 0 \end{bmatrix}$$

$$C \in \mathbb{R}^{1 \times 3} = [0 \quad 0 \quad 1]$$

$$D \in \mathbb{R}^{1 \times 1} = [0]$$

Función de transferencia:

$$G(s) = \frac{T_{C2}(s)}{Q_1(s)} = \frac{0.304}{145886.7s^3 + 12183.9s^2 + 220.7s + 1}$$

5.20 Estabilidad y oscilaciones

En este programa se determinó la estabilidad y las oscilaciones de un sistema de segundo orden con un controlador P, analizando su matriz de ruta.

Se utilizó solo el sensor 1 (T_{C1}) y el calentador 1 (Q_1), por lo que la ecuación de segundo orden que representa al sistema sería la siguiente:

$$\tau^2 \frac{d^2 T}{dt^2} + 2\zeta\tau \frac{dT}{dt} + T = K_p Q(t) \quad (83)$$

Si la ecuación (83) está en términos de desviación (ver relación(19)), las condiciones iniciales son cero y su transformada de Laplace queda (ver desarrollo en “Tclab Estabilidad y oscilaciones.ipynb” en P20) (Stephanopoulos, Dynamic Behavior of second-Order Systems, 1984):

$$s^2 \tau^2 \mathbf{T}(s) + 2\zeta\tau s \mathbf{T}(s) + \mathbf{T}(s) = K_p \mathbf{Q}(s) \quad (84)$$

Despejando y reacomodando términos obtenemos la forma de función de transferencia

$$G_p(s) = \frac{T_{C1}(s)}{Q_1(s)} = \frac{k_p}{\tau_s^2 s^2 + 2\zeta\tau_s s + 1} \quad (85)$$

Lo primero que se hace es obtener los parámetros (K_p , ζ y τ_s) con una optimización en gekko. Se hace una prueba con cuatro pasos (80, 20, 70 y 50 %) para generar datos de T_{C1} debido a los cambios de Q_1 . La ecuación que se utiliza para optimizar los parámetros es la (83) que se divide en dos ecuaciones diferenciales en lugar de usar la ecuación diferencial de segundo orden. Las sustituciones que se hacen son las siguientes:

$$x = \frac{T_{C1}}{dt} \quad dx = \frac{T_{C1}^2}{dt^2} \quad (86)$$
$$\tau^2 \frac{dx}{dt} + 2\zeta\tau x + (T_{C1} - T_o) = K_p Q_1$$

Los valores obtenidos son los siguientes: $K_p = 0.745$, $\zeta = 1.546$ y $\tau_s = 38.89$. En la fig.53 podemos ver que la optimización tiene un buen desempeño, pues la curva de simulación se encuentra sobre la curva de valores medidos.

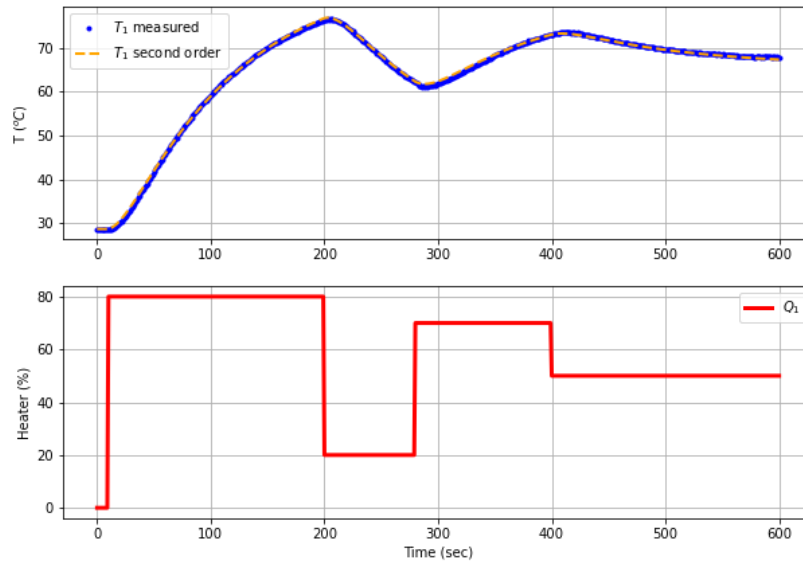


Figura 53. Obtención de parámetros de sistema de 2do orden con Gekko.

Con la función de transferencia (85) y los parámetros (K_p , ζ y τ_s) podemos determinar la estabilidad del controlador.

Tabla 5. Parámetros del modelo de 2do Orden con optimización.

Parámetros del modelo	
K_p	0.74507
ζ	1.5462
τ_s	38.890

Matriz de ruta

La función de transferencia del lazo cerrado con una ganancia de controlador P K_c es:

$$G_{cl}(s) = \frac{TC1(s)}{TSP(s)} = \frac{K_c G_p(s)}{1 + K_c G_p(s)} = \frac{\frac{K_c K_p}{\tau_s^2 s^2 + 2\zeta \tau_s s + 1}}{1 + \frac{K_c K_p}{\tau_s^2 s^2 + 2\zeta \tau_s s + 1}} = \frac{K_c K_p}{\tau_s^2 s^2 + 2\zeta \tau_s s + 1 + K_c K_p} \quad (87)$$

Los coeficientes del denominador se analizan para determinar el rango de estabilidad (ver archivo “Análisis de estabilidad del controlador.ipynb” en P20):

$a_n = \tau_s^2$	$a_{n-2} = 1 + K_c K_p$
$a_{n-1} = 2\zeta \tau_s$	$a_{n-3} = 0$
$b_1 = \frac{(2\zeta \tau_s)(1 + K_c K_p)}{2\zeta \tau_s} = 1 + K_c K_p$	$b_2 = 0$
$c_1 = 0$	0

La columna de la izquierda no debe cambiar de signo para que el sistema sea estable. Por lo tanto, se deben cumplir las siguientes condiciones:

$$\begin{aligned} a_n &= \tau_s^2 = 1512 > 0 \\ a_{n-1} &= 2\zeta\tau_s = 120.3 > 0 \\ b_1 &= 1 + K_c K_p > 0 \\ c_1 &= 0 > 0 \end{aligned}$$

La restricción positiva de b_1 lleva a $K_c > -\frac{1}{K_p}$. Por lo tanto, el siguiente rango es aceptable para la estabilidad del controlador.

$$K_c = -1.3422 \quad (88)$$

Gráfica del lugar geométrico de las raíces

La última celda dentro de “Tclab Estabilidad y oscilaciones.ipynb” dentro de P20 contiene un programa que genera un gráfico con una barra deslizante de K_c para observar la estabilidad del modelo. Solo es necesario sustituir las variables K_p, ζ y τ_s obtenidas en la primera celda, correr el programa y mover la barra deslizante. El eje horizontal mostrará las raíces reales mientras que el vertical contendrá las raíces imaginarias.

Nota: Correr esta celda en IDLE para poder interactuar con la gráfica.

Hay una raíz positiva en $K_c < -1.35$ por lo tanto el sistema será inestable en valores menores de -1.35 (ver figura 54).

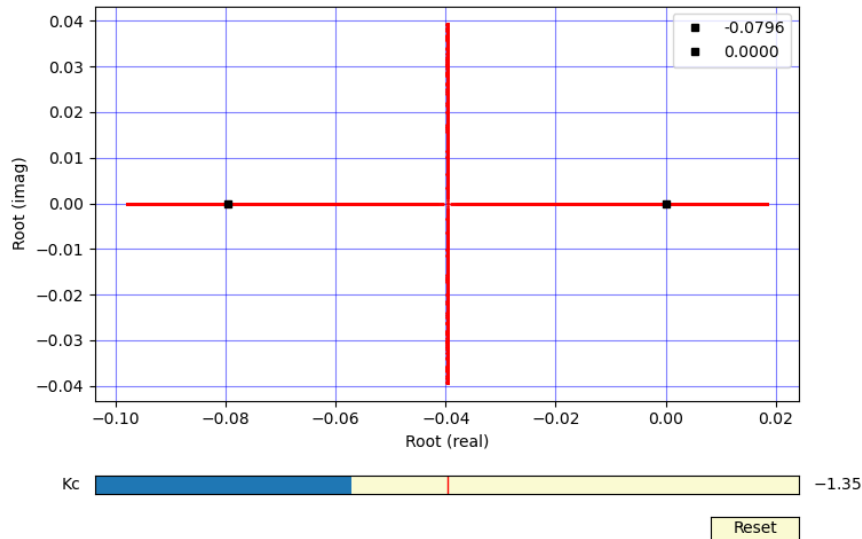


Figura 54. Inestabilidad en $K_c < -1.35$.

Si se sigue aumentando el valor de K_c se observa que en $K_c > 1.88$ hay raíces imaginarias y habrá oscilaciones (Fig.55). También puede verse que las partes reales de las raíces son negativas, por lo tanto, el sistema es estable y converge.

La ganancia del controlador P será en estable en el rango: $-1.35 < K_c < 1.88$

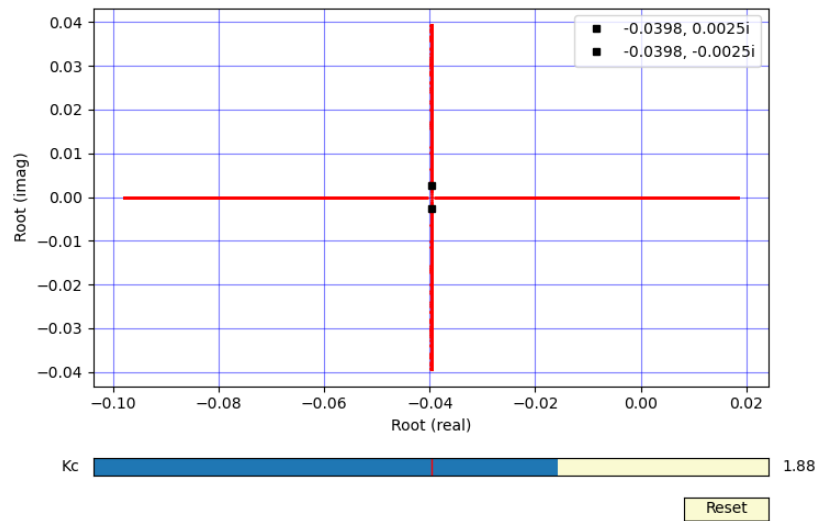


Figura 55. Raíces imaginarias en $K_c > 1.88$

5.21 Control en cascada

Sintonización de control en cascada

Para la sintonización se utilizan los parámetros obtenidos con la regresión de 2do orden del capítulo anterior (Tabla 5) y K_d ganancia de perturbación (ver capítulo 4.2.11)

Primero se hizo uso del simulador de control PI estándar para obtener un estimado inicial de K_c y τ_d , podemos ver que el IAE en esta simulación sin que se presente sobre impulso es IAE:1663. Después se utiliza el simulador de control en cascada que tendrá 4 barras deslizantes donde se ve claramente que el IAE bajó a IAE:1254, (Fig.56), dando a entender que un controlador en cascada mejora en rendimiento. La clave para utilizar los simuladores es la disminución del IAEie, además de observar que el gráfico no presenta grandes oscilaciones.

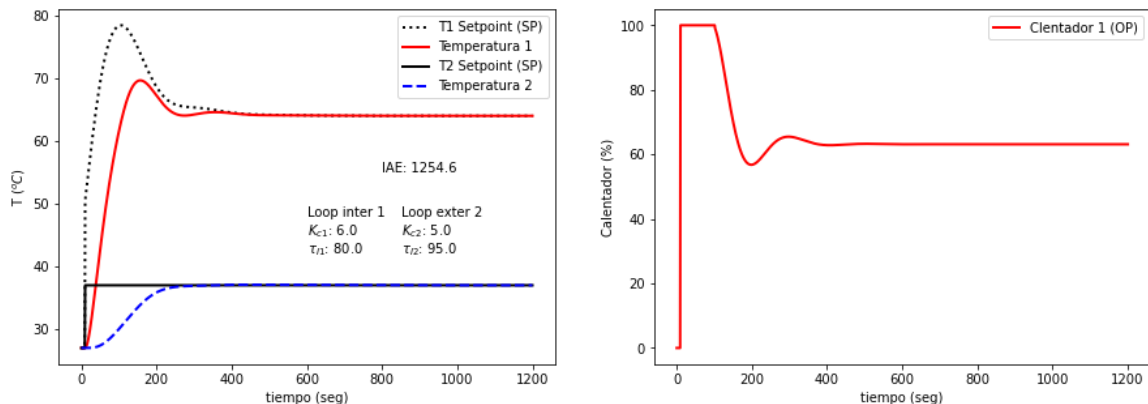


Figura 56. Sintonización con simulador de control en cascada

Los parámetros óptimos obtenidos para este controlador son:

Tabla 6. Parámetros para controlador en cascada

PARAMETROS PI	
K_{c1}	6
τ_{I1}	80
K_{c2}	5
τ_{I2}	95

Prueba de control en cascada con TCL

La última celda dentro de P21 contiene una prueba realizada en TCL para el control en cascada. Los parámetros PI utilizados son los obtenidos en la simulación de control en cascada (tabla 5).

El control PID es similar al utilizado en programas anteriores con la diferencia de incluir un condicional *if* para elegir con que parámetros realizar el control dependiendo del lazo de control como se muestra en el código siguiente.

```

47  if cid==1:
48      # controlador 1
49      KP = Kc1
50      KI = Kc1/tauI1
51      ophi = 100
52      oplo = 0
53  else:
54      # controlador 2
55      KP = Kc2
56      KI = Kc2/tauI2
57      ophi = 85
58      oplo = 27
    
```

Para llamar al controlador indicándole que se utilice el parámetro necesario se le pone 1 cuando se requiera usar el controlador 1 y se utiliza un 2 para usar el controlador 2, como se muestra en las siguientes líneas de código:

```

183  [Tsp1[i],P,ierr2] = pid(Tsp2[i],T2[i],T2[i-1],ierr2,dt,d1,2)
184  [Q1[i],P,ierr1] = pid(Tsp1[i],T1[i],T1[i-1],ierr1,dt,d2,1) #
    
```

Notar que el controlador del loop externo (línea 183) devuelve un valor de temperatura Tsp1, y el controlador del loop interno (línea 184) devuelve un valor de calentador Q1.

La prueba de 15 minutos da un error porcentual del IAE de 3.03%. Con esto se puede inferir que la simulación es precisa.

$$\left| \frac{IAE_{simulación} - IAE_{experimental}}{IAE_{simulación}} \right| * 100\% = \left| \frac{1254 - 1292}{1254} \right| * 100\% = 3.03\% \quad (89)$$

La gráfica fig.57 muestra que se controla a la variable T_2 en el valor de T_{2SP} deseado, siendo el calentador 1 el actuador que genera todo el calor durante la prueba, y tomando

un comportamiento muy parecido al del simulador (Fig.57), con la diferencia que el comportamiento real muestra más oscilaciones que en el simulado. En el gráfico del calentador seguimos viendo un poco de ruido durante todo el proceso, teniendo los picos más significativos al final del proceso (entre 600 y 800 s), en donde el porcentaje del calentador bajo a cero de forma precipitada.

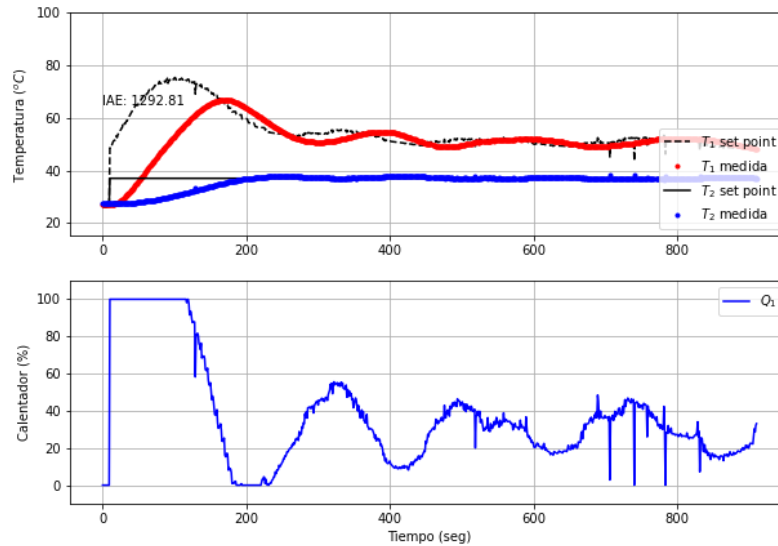


Figura 57. Prueba de control en cascada en TCL

6 Conclusión

Se ha implementado el uso de TCL en problemas de dinámica y control de temperatura en una serie de programas de los cuales se obtuvieron datos y gráficos que permiten un análisis y el estudio de dinámica y control de procesos de forma práctica. Esto impulsa al usuario a aplicar los conocimientos teóricos en un sistema real y finalmente lo lleva a aumentar el conocimiento adquirido durante el proceso de aprendizaje.

Es importante mencionar que para la utilización de TCL es necesario tener un conocimiento básico de Python, por lo que se recomienda dar una introducción de este lenguaje antes de utilizar los programas. Además, cabe mencionar que el alcance del curso teórico puede ser menor al que se expone en este trabajo, por lo que se recomienda una selección de los programas que se adecúen mejor con el programa de estudio.

7 Apéndice

A. Lista de carpeta de programas

PROGRAMA	TÍTULO DEL PROGRAMA
P1	Tclab Simulación de respuesta de paso ODE.
P2	Tclab Transferencia de calor por convección.
P3	Tclab Transferencia de calor radiactivo.
P4	Tclab Balance de energía linealizado.
P5	Tclab FOPDT Ajuste Grafico.
P6	Tclab FOPDT Regresión.
P7	Tclab Control solo proporcional.
P8	Tclab Control Proporcional Integral.
P9	Tclab Control PID.
P10	Tclab Sintonización de control PI.
P11	Tclab Sintonización de control PID.
P12	Tclab PID con retroalimentación.
P13	Tclab Actuadores de control.
P14	Tclab Sensores para automatización.
P15	Tclab Diagrama de bloques.
P16	Tclab Espacio de estado.
P17	Tclab Control On-Off.
P18	Tclab Tclab regresión en tiempo real de 2do orden.
P19	Tclab Regresión de 3er orden.
P20	Tclab Estabilidad y oscilaciones.
P21	Tclab Control en Cascada.

Link de carpeta de programas en dropbox:

https://www.dropbox.com/sh/fnwuw7o0uctig51/AACalwWpdC4we_ATOOu8HVrMa?dl=0

Link de carpeta de programas en drive:

https://drive.google.com/file/d/1-Gktuq_n5lanXKAdl6xy_4Z-dLr73Yak/view?usp=sharing

8 Bibliografía

Babatunde. (1994). *Process dynamics modeling and control*. New York: Oxford University Press.

Coughanowr. (2009). Controller Tuning and Process Identification. En Coughanowr, *Process Systems Analysis and Control* (pág. 391). New York: McGraw-Hill.

Coughanowr. (2009). Proportional Integral (PI) Control. En Coughanowr, *Process Systems Analysis an Control* (págs. 194-195). New York: McGraw-Hill's.

- Coughanowr. (2009). Proportional Integral Derivative (PID) control. En Coughanowr, *Process Systems Analysis and Control* (pág. 197). New York: McGraw-Hill.
- Coughanowr. (2009). State Space Representation of Physical Systems. En Coughanowr, *Process Systems Analysis and Control* (págs. 477-480). New York: McGraw-Hill.
- Hedengren. (26 de Septiembre de 2020). *TCLab Heater Actuators*. Obtenido de apmonitor.com: <https://apmonitor.com/pdc/index.php/Main/TCLabActuator>
- Hedengren. (07 de Diciembre de 2022). *apmonitor*. Obtenido de Process Dynamics and Control: <https://apmonitor.com/pdc/>
- Hedengren. (31 de 12 de 2022). *Proportional Integral Derivative (PID)*. Obtenido de apmonitor.com: <https://apmonitor.com/pdc/index.php/Main/ProportionalIntegralDerivative>
- Keesman. (2011). System Response Methods. En Keesman, *System Identification* (págs. 17-18). London: Springer.
- Roffel. (2006). Controller Equations. En Roffel, *Process Dynamics and Control Modeling for control and prediction* (pág. 457). England: Wiley.
- Stephanopoulos. (1984). Control Systems with Multiple Loops. En Stephanopoulos, *Chemical Process Control* (págs. 395-398). New Jersey: Prentice Hall.
- Stephanopoulos. (1984). Dynamic Behavior of second-Order Systems. En Stephanopoulos, *Chemical Process Control* (pág. 187). New Jersey: Prentice Hall .
- Stephanopoulos. (1984). Feedforward and Ratio Control. En Stephanopoulos, *Chemical Process Control* (pág. 411). New Jersey: Prentice Hall.
- Stephanopoulos. (1984). Linearization of systems with one variable. En Stephanopoulos, *Chemical Process Control* (pág. 116). New Jersey: Prentice Hall.
- Stephanopoulos. (1984). Transfer Functions and Input-Output Models. En Stephanopoulos, *Chemical Process Control* (págs. 159-160). New Jersey: Prentice Hall.
- Stephanopoulos. (1984). Types of feedback controllers. En Stephanopoulos, *Chemical Process Control* (pág. 245). New Jersey: Prentice Hall.

ÍNDICE DE FIGURAS

Figura 1. Sistema TCL.....	2
Figura 2. Dispositivo TCL conexiones con ordenador y a la corriente eléctrica. Extraída de: https://apmonitor.com/pds/notebooks/TCLab_Help.html	2
Figura 3. Calentador y diagrama de flujo de corriente.....	3

Figura 4.	Sistema de control de temperatura TCL.....	5
Figura 5.	Relación de primer orden utilizando el calentador Q1	5
Figura 6.	Relación de 4to orden utilizando dos calentadores.....	7
Figura 7.	Ajuste gráfico para sistemas de segundo orden.Extraída de: https://apmonitor.com/pdc/index.php/Main/TCLabOnOffControl	12
Figura 8.	Diagrama de bloques control hacia atrás.....	15
Figura 9.	Diagrama de bloques control hacia adelante.....	15
Figura 10.	Representación esquemática de un control en cascada.....	16
Figura 11.	Diagrama de bloques en cascada del TCL. Extraída de: https://apmonitor.com/pdc/index.php/Main/TCLabCascadeControl	16
Figura 12.	Funcionamiento del <i>antireset</i> en control PI.....	20
Figura 13.	Representación esquemática de un control en cascada.....	22
Figura 14.	Diagrama de bloques en cascada del TCL. Extraída de: https://apmonitor.com/pdc/index.php/Main/TCLabCascadeControl	23
Figura 15.	Curva de temperatura debida a un cambio de Q1 en TCL	27
Figura 16.	Curvas de temperatura simulada y medida en el TCL	29
Figura 17.	Simulación de modelo convectivo del TCL.....	30
Figura 18.	Simulación de modelo con transferencia por radiación y pérdidas de calor. 31	
Figura 19.	Simulación con modelo linealizado.....	33
Figura 20.	Prueba de paso con Q1=70.	34
Figura 21.	Método gráfico para determinar parámetros FOPDT.....	34
Figura 22.	Simulación FOPDT para verificar parámetros.....	35
Figura 23.	Diagrama de flujo para la obtención de parámetros fopdt optimizados	36
Figura 24.	Gráfica de optimización de parámetros FOPDT	38
Figura 25.	Simulador de control P.....	39
Figura 26.	Aplicación de un controlador P en TCL.....	40
Figura 27.	Control PI en TCL.....	41
Figura 28.	Respuesta a un controlador PID.....	43
Figura 29.	Simulador de sintonización PI.....	44
Figura 30.	Sintonización con IMC agresivo.....	45
Figura 31.	Sintonización con ITAE	45
Figura 32.	Sintonización con simulador PID	47
Figura 33.	Sintonización PID del TCL.....	48
Figura 34.	Prueba escalonada para determinar Kd	48
Figura 35.	Simulador PID feedforward	49

Figura 36.	Control PID feedforward del TCL.....	50
Figura 37.	Simulación y comprobación del modelo FOPDT de Q2	51
Figura 38.	Prueba de paso múltiple para Q1 y Q2.....	52
Figura 39.	Diagrama de bloques con funciones de transferencia. Extraída de: https://apmonitor.com/pdc/index.php/Main/TCLabBlockDiagram	53
Figura 40.	Diagrama de bloques con una perturbación.....	53
Figura 41.	Predicción de T_1 debido a un cambio de 20°C en T_2	55
Figura 42.	Diagrama de bloques con un cambio del SP	55
Figura 43.	Prueba de control PI con solución analítica de un cambio en SP.....	56
Figura 44.	Simulación de modelos SS y basado en la física	59
Figura 45.	Control On-Off en TCL	60
Figura 46.	Ajuste gráfico del sistema de segundo orden del TCL.....	62
Figura 47.	Diagrama de Flujo Retrogresión de 2do Orden Con Scipy	64
Figura 48.	Optimización de 2do Orden con Scipy.....	65
Figura 49.	Optimización de 2do Orden con Gekko	66
Figura 50.	Modelo de 3er Orden con TCL.....	67
Figura 51.	Optimización de modelo basado en la física.	69
Figura 52.	Simulación de modelos, espacio de estado y series de tiempo.....	71
Figura 53.	Obtención de parámetros de sistema de 2do orden con Gekko.....	73
Figura 54.	Inestabilidad en $K_c < -1.35$	74
Figura 55.	Raíces imaginarias en $K_c > 1.88$	75
Figura 56.	Sintonización con simulador de control en cascada.....	76
Figura 57.	Prueba de control en cascada en TCL	77

ÍNDICE DE TABLAS

Tabla 1.	Variables y constantes del balance de energía de orden superior del TCL.	8
Tabla 2.	Parámetros k_p , τ_p y θ_p , optimizados	37
Tabla 3.	Parámetros del modelo basado en la física.	59
Tabla 4.	Variables del modelo basado en la física	69
Tabla 5.	Parámetros del modelo de 2do Orden con optimización.	73
Tabla 6.	Parámetros para controlador en cascada	76