Nombre: Camila Millan

Semestre 2024-1

Practica: algoritmo genético

Los Algoritmos Genéticos son métodos adaptativos utilizados en la búsqueda y optimización de parámetros. Se basan en principios de selección natural y reproducción, imitando el proceso evolutivo. Comienzan con una población aleatoria de posibles soluciones y, a lo largo de generaciones, estas soluciones evolucionan y se adaptan, seleccionando y refinando las mejores hasta alcanzar la solución óptima. La computación evolutiva, de la cual los Algoritmos Genéticos son parte, estudia técnicas inspiradas en la evolución natural para resolver problemas computacionales.

$$\text{Computación Evolutiva} = \text{Algoritmos Genéticos} + \text{Estrategias de Evolución} + \text{Programación Evolutiva}$$

La Programación Evolutiva, como parte del campo de la computación evolutiva, es un enfoque de Soft Computing. Se inspira en procesos naturales, como la evolución, para resolver problemas mediante algoritmos que imitan la selección natural y la reproducción de los individuos más aptos.

$$\textit{Soft Computing} = \text{Computación Evolutiva} + \text{Redes Neuronas Artificiales} + \text{Lógica Difusa}$$

Los Algoritmos Genéticos trabajan con una población de individuos, donde cada uno representa una posible solución al problema en cuestión. Cada individuo tiene un nivel de ajuste que refleja su idoneidad en relación con el problema, similar a la eficiencia de un individuo en la naturaleza.

El proceso de generación de una nueva población se lleva a cabo mediante dos tipos de operadores de reproducción:

1. Cruce: Implica la reproducción sexual, donde se generan descendientes a partir de un par de individuos de la generación anterior. Hay varios tipos de cruces que se detallan más adelante.

2. Copia: Se trata de una reproducción asexual, donde un cierto número de individuos pasa directamente a la siguiente generación sin cambios.

OPERADORES GENETICOS

Selección: Los algoritmos de selección determinan qué individuos tienen la oportunidad de reproducirse. Se prioriza a los individuos más aptos, pero se permite cierta reproducción de los menos aptos para mantener la diversidad. Hay dos tipos principales: probabilísticos y determinísticos. Los probabilísticos, como la selección por ruleta o por torneo, utilizan el azar para escoger individuos. Los determinísticos asignan la reproducción según el ajuste conocido de cada individuo, permitiendo controlar la presión de selección.

Cruce: El cruce es una estrategia de reproducción sexual donde se combinan genes de individuos seleccionados para producir descendencia. Los métodos comunes incluyen el cruce de 1 punto, el cruce de 2 puntos y el cruce uniforme. Cada método tiene ventajas y desventajas en términos de exploración del espacio de búsqueda y rendimiento computacional.

Mutación: La mutación introduce cambios aleatorios en los genes de un individuo, generalmente con una probabilidad baja. Puede ocurrir junto con el cruce para diversificar la población y evitar la convergencia prematura. Los métodos de mutación incluyen el reemplazo aleatorio de un gen y la modificación de un gen por una pequeña cantidad aleatoria.

Reemplazo: Cuando se trabaja con una sola población en lugar de una temporal, es necesario reemplazar individuos para insertar nuevos descendientes. Los métodos de reemplazo incluyen el reemplazo aleatorio, el reemplazo de padres, el reemplazo de similares y el reemplazo de los peores. Cada método tiene implicaciones en la diversidad y la convergencia de la población.

Copia: La copia es una estrategia de reproducción asexual donde un individuo se copia directamente en la siguiente generación. Se utiliza con menos frecuencia que el cruce para evitar la convergencia prematura hacia un único individuo. Se elige después del cruce y la mutación, y generalmente se seleccionan dos individuos para el cruce antes de decidir si se realiza la copia.

TRABAJO EN CLASE

En clase se profundizó en la teoría de los algoritmos genéticos, explorando su funcionamiento y los operadores genéticos que los componen. El profesor proporcionó un programa en Python para implementar seis controladores, tres de ellos para sistemas de primer orden (P, PI y PID) y otros tres para sistemas de segundo orden (P, PI y PID). Se utilizó una entrada tipo escalón para evaluar las respuestas de cada controlador.

CODIGO

1. Código para el controlador P primer orden

```
def funcion_objetivo(K):

   # Planta
   num = [2.7817]
   den = [314.25, 1]
   # Funciones de Transferencia
   sys = cl.tf(num, den)   # FT Planta
   sys_k = cl.tf(K,1)      # FT Ganacia Proporcional (Kc)
   sysf = cl.series(sys_k,sys) # FT [Planta - Controlador P]
   feed = cl.feedback(sysf,1) # Lazo Cerrado
   # Parametrosde funcion step_info
   step = cl.step_info(feed)
  # RiseTime       = Tiempo de subida:
  # SettlingTime    = Tiempo de estabilización:
  # SettlingMin     = Asentamiento mín
  # SettlingMax     = Asentamiento max
  # Overshoot       = Sobreimpulso
  # Undershoot      = Subimpulso
  # Peak           = Pico
  # PeakTime       = Tiempo de pico
  # SteadyStateValue = Valor en estado estacionario   Valor en estado estable
   # Error en Estado Estable
   y = step.get('SteadyStateValue')
   e = abs(1 - y)
   return(e)
poblacion = Poblacion(
            n_individuos = 200,
            n_variables  = 1,
            limites_inf  = [1],
            limites_sup  = [100],
            verbose     = False
            )

C = poblacion.optimizar(
       funcion_objetivo   = funcion_objetivo,
       optimizacion      = "minimizar",
       n_generaciones    = 50,
       metodo_seleccion   = "tournament",
       elitismo         = 0.1,
       prob_mut         = 0.1,
```

```
        distribucion      = "uniforme",
        media_distribucion = 1,
        sd_distribucion   = 1,
        min_distribucion  = -1,
        max_distribucion  = 1,
        parada_temprana   = True,
        rondas_parada     = 10,
        tolerancia_parada  = 10**-16,
        verbose          = True
        )
K  = C[0] # la poblacion optimizada con la ganancia K
# Planta
num = [2.7817]
den = [314.25, 1]
sys = cl.tf(num, den)     # FT Planta
feeed = cl.feedback(sys,1) # lazo cerrdado de la planta
# Controlador
sys0 = cl.tf(K,1)     # FT Ganancia Propocinal
sysc = sys0 # FT Controlador P
sysf = cl.series(sysc,sys) # Series
feed = cl.feedback(sysf,1) # Lazo Cerado
# Grafica
t = np.linspace(0,500)
t,ye = cl.step_response(feed) # Respuesta Esacalon
t,y1 = cl.step_response(sys)
t,y2 = cl.step_response(feeed)
S = cl.step_info(feed) # Informacion
for i in S:
   print(f"{i}: {S[i]}") # imprimir la informacion del step de la plnata
# Para graficar
plt.grid()
plt.title('Entrenamiento: Datos vs Modelo')
plt.plot(t, ye, 'r-', linewidth=3)  # Control
plt.plot(t, y1, 'g-', linewidth=3)  #lazo abierto
plt.plot(t, y2, 'b-', linewidth=3)  #lazo cerrado
plt.legend(['Control P','lazo abiero','lazo cerrado'], loc="lower right")
plt.ylabel('Y(K)', fontsize=14)
plt.xlabel('Tiempo (s)', fontsize=14)
plt.show()
```

2. Codigo para controlador PI primer orden

```
def funcion_objetivo(K,Ti):

   # Planta
   num = [2.7817]
   den = [314.25, 1]
   # Funciones de Transferencia
   sys = cl.tf(num, den)   # FT Planta
   sys0 = cl.tf(K,1)      # FT Ganacia Proporcional (Kc)
   sys1 = cl.tf(1, Ti)  # FT Ganacia Integral (1/Ti*S)
      sysc = sys0 + sys1      # FT Controlador PI
   sysf = cl.series(sysc,sys) # FT [Planta - Controlador PI]
   feed = cl.feedback(sysf,1) # Lazo Cerrado
   # Parametrosde funcion step_info
   step = cl.step_info(feed)
  # RiseTime       = Tiempo de subida:
  # SettlingTime    = Tiempo de estabilización:
  # SettlingMin     = Asentamiento mín
  # SettlingMax     = Asentamiento max
  # Overshoot       = Sobreimpulso
  # Undershoot      = Subimpulso
  # Peak           = Pico
  # PeakTime       = Tiempo de pico
```

```
  # SteadyStateValue = Valor en estado estacionario   Valor en estado estable
   # Error en Estado Estable
   y = step.get('SteadyStateValue')
   e = 1 - y
    ##tiempo de estabilizacion
   t_i = step.get('SettlingMax')
   ta = t_i/5
   r = e + ta
      return(r)
poblacion = Poblacion(
            n_individuos = 800,
            n_variables  = 2,
            limites_inf  = [1,0.01],
            limites_sup  = [100,1],
            verbose     = False
            )
C = poblacion.optimizar(
       funcion_objetivo   = funcion_objetivo,
       optimizacion      = "minimizar",
       n_generaciones    = 1000,
       metodo_seleccion   = "tournament",
       elitismo         = 0.1,
       prob_mut         = 0.1,
       distribucion      = "uniforme",
       media_distribucion = 1,
       sd_distribucion    = 1,
       min_distribucion   = -1,
       max_distribucion   = 1,
       parada_temprana    = True,
       rondas_parada     = 10,
       tolerancia_parada  = 10**-16,
       verbose          = True
        )
# optimizacion de la parte K y Ti de la planta
K  = C[0]
Ti = C[1]
# Planta
num = [2.7817]
den = [314.25, 1]
sys = cl.tf(num, den) # FT Planta
feeed = cl.feedback(sys,1) # lazo cerrado
K_1 = K           # Ganancia Proporcional
den_Ti = [Ti,0]      # Tiempo Integral
# Controlador
sys0 = cl.tf(K,1)     # FT Ganancia Propocinal
sys1 = cl.tf(1, den_Ti) # FT Ganancia Integral
sysc = sys0 + sys1 # FT Controlador PID
sysf = cl.series(sysc,sys) # Series
feed = cl.feedback(sysf,1) # Lazo Cerado para el control
# Grafica
t = np.linspace(0,500)
t,ye = cl.step_response(feed) # Respuesta Esacalon
t,y1 = cl.step_response(sys)
t,y2 = cl.step_response(feeed)
S = cl.step_info(feed) # Informacion
for i in S:
   print(f"{i}: {S[i]}")
# Para graficar
plt.grid()
plt.title('Entrenamiento: Datos vs Modelo')
plt.plot(t, ye, 'r-', linewidth=3)  # Control
plt.plot(t, y1, 'g-', linewidth=3)  #lazo abierto
plt.plot(t, y2, 'b-', linewidth=3)  #lazo cerrado
plt.legend(['Control PI','lazo abiero','lazo cerrado'], loc="lower right")
plt.ylabel('Y(K)', fontsize=14)
plt.xlabel('Tiempo (s)', fontsize=14)
```

```
plt.show()
```

3. Código para controlador PID primer orden

```
def funcion_objetivo(K,Ti,Td):

  # Planta
  num = [1.6222]
  den = [0.3699, 1]
  # Funciones de Transferencia
  sys = cl.tf(num, den)   # FT Planta
  sys0 = cl.tf(K,1)      # FT Ganacia Proporcional (Kc)
  sys1 = cl.tf(1, Ti)  # FT Ganacia Integral (1/Ti*S)
  sys2 = cl.tf(Td,1)   # FT Ganacia Dervaritva (Td*S)
  sysc = sys0 + sys1 + sys2 # FT Controlador PID
  sysf = cl.series(sysc,sys) # FT [Planta - Controlador PID]
  feed = cl.feedback(sysf,1) # Lazo Cerrado
  # Parametrosde funcion step_info
  step = cl.step_info(feed)
 # RiseTime       = Tiempo de subida:
 # SettlingTime    = Tiempo de estabilización:
 # SettlingMin     = Asentamiento mín
 # SettlingMax     = Asentamiento max
 # Overshoot       = Sobreimpulso
 # Undershoot      = Subimpulso
 # Peak           = Pico
 # PeakTime       = Tiempo de pico
 # SteadyStateValue = Valor en estado estacionario   Valor en estado estable
  ## Error en Estado Estable
  y = step.get('SteadyStateValue')
  e = 1 - y
  ##tiempo de estabilizacion
  t_i = step.get('SettlingMax')
  ta = t_i/5
  ## sobreimpulso
  sp = step.get('Overshoot')
  ff = sp/10
 r = e + ta + ff
  return(r)
poblacion = Poblacion(
            n_individuos = 200,
            n_variables  = 3,
            limites_inf  = [1,0.01,0.001],
            limites_sup  = [100,1,0.1],
            verbose     = False
            )
C = poblacion.optimizar(
      funcion_objetivo   = funcion_objetivo,
      optimizacion      = "minimizar",
      n_generaciones    = 5000,
      metodo_seleccion   = "tournament",
      elitismo         = 0.1,
      prob_mut         = 0.1,
      distribucion      = "uniforme",
      media_distribucion = 1,
      sd_distribucion    = 1,
```

```
        min_distribucion   = -1,
        max_distribucion   = 1,
        parada_temprana    = True,
        rondas_parada      = 10,
        tolerancia_parada  = 10**-16,
        verbose          = True
         )
    # Grafica
    K  = C[0]
    Ti = C[1]
    Td = C[2]
    # Planta
    num = [2]
    den = [288.4, 1]
    sys = cl.tf(num, den) # FT Planta
    feeed = cl.feedback(sys,1) # lazo cerrado
    K_1 = K            # Ganancia Proporcional
    den_1 = [Ti,0]      # Tiempo Integral
    den_2 = [Td,0]     # Tiempo Derivativo
    # Controlador
    sys0 = cl.tf(K,1)     # FT Ganancia Propocinal
    sys1 = cl.tf(1, den_1) # FT Ganancia Integral
    sys2 = cl.tf(den_2,1)  # FT Ganancia Derivativo
    sysc = sys0 + sys1 + sys2 # FT Controlador PID
    sysf = cl.series(sysc,sys) # Series
    feed = cl.feedback(sysf,1) # Lazo Cerado
    # Grafica
    t = np.linspace(0,500)
    t,ye = cl.step_response(feed) # Respuesta Esacalon
    t,y1 = cl.step_response(sys)
    t,y2 = cl.step_response(feeed)
    S = cl.step_info(feed) # Informacion
    for i in S:
       print(f"{i}: {S[i]}")
    # Para graficar
    plt.grid()
    plt.title('Entrenamiento: Datos vs Modelo')
    plt.plot(t, ye, 'r-', linewidth=3)  # Control
    plt.plot(t, y1, 'g-', linewidth=3)  #lazo abierto
    plt.plot(t, y2, 'b-', linewidth=3)  #lazo cerrado
    plt.legend(['Control PID','lazo abiero','lazo cerrado'], loc="lower right")
    plt.ylabel('Y(K)', fontsize=14)
    plt.xlabel('Tiempo (s)', fontsize=14)
    plt.show()
```

4. Código para controlador P segundo orden

```
def funcion_objetivo(K):

    # Planta
    # Definir los parámetros de la función de transferencia
    k = 1.6222
    wn = 0.238
    zeta = 0.8
    # Numerador y denominador de la función de transferencia
    num = [k * wn**2]
    den = [1, 2*zeta*wn, wn**2]
    # Funciones de Transferencia
```

```
    sys = cl.tf(num, den)   # FT Planta
    sys_k = cl.tf(K,1)      # FT Ganacia Proporcional (Kc)
    sysf = cl.series(sys_k,sys) # FT [Planta - Controlador P]
    feed = cl.feedback(sysf,1) # Lazo Cerrado
    # Parametrosde funcion step_info
    step = cl.step_info(feed)
   # RiseTime        = Tiempo de subida:
   # SettlingTime    = Tiempo de estabilización:
   # SettlingMin     = Asentamiento mín
   # SettlingMax     = Asentamiento max
   # Overshoot       = Sobreimpulso
   # Undershoot      = Subimpulso
   # Peak            = Pico
   # PeakTime        = Tiempo de pico
   # SteadyStateValue = Valor en estado estacionario   Valor en estado estable
    # Error en Estado Estable
    y = step.get('SteadyStateValue')
    e = abs(1 - y)
    return(e)
poblacion = Poblacion(
            n_individuos = 200,
            n_variables  = 1,
            limites_inf  = [1],
            limites_sup  = [100],
            verbose     = False
            )
C = poblacion.optimizar(
        funcion_objetivo   = funcion_objetivo,
        optimizacion       = "minimizar",
        n_generaciones     = 50,
        metodo_seleccion   = "tournament",
        elitismo           = 0.1,
        prob_mut           = 0.1,
        distribucion       = "uniforme",
        media_distribucion = 1,
        sd_distribucion    = 1,
        min_distribucion   = -1,
        max_distribucion   = 1,
        parada_temprana    = True,
        rondas_parada      = 10,
        tolerancia_parada  = 10**-16,
        verbose            = True
        )
K  = C[0] # la poblacion optimizada con la ganancia K
# Planta
# Definir los parámetros de la función de transferencia
k = 1.6222
wn = 0.238
zeta = 0.8
# Numerador y denominador de la función de transferencia
num = [k * wn**2]
den = [1, 2*zeta*wn, wn**2]
sys = cl.tf(num, den)     # FT Planta
feeed = cl.feedback(sys,1) # lazo cerrdado de la planta
# Controlador
sys0 = cl.tf(K,1)     # FT Ganancia Propocinal
sysc = sys0 # FT Controlador P
sysf = cl.series(sysc,sys) # Series
feed = cl.feedback(sysf,1) # Lazo Cerado
# Grafica
t = np.linspace(0, 30, 500)
#t = np.linspace(0,500)
t,ye = cl.step_response(feed, T=t) # Respuesta Esacalon
t,y1 = cl.step_response(sys, T=t)
t,y2 = cl.step_response(feeed, T=t)
S = cl.step_info(feed) # Informacion
```

```
for i in S:
   print(f"{i}: {S[i]}") # imprimir la informacion del step de la plnata
# Para graficar
plt.grid()
plt.title('Entrenamiento: Datos vs Modelo')
plt.plot(t, ye, 'r-', linewidth=3)  # Control
plt.plot(t, y1, 'g-', linewidth=3)  #lazo abierto
plt.plot(t, y2, 'b-', linewidth=3)  #lazo cerrado
plt.legend(['Control P','lazo abiero','lazo cerrado'], loc="lower right")
plt.ylabel('Y(K)', fontsize=14)
plt.xlabel('Tiempo (s)', fontsize=14)
plt.show()
```

5. Código para controlador PI segundo orden

```
def funcion_objetivo(K,Ti):


   # Planta

   # Definir los parámetros de la función de transferencia

   k = 2

   wn = 0.2

   zeta = 0.8

   # Numerador y denominador de la función de transferencia

   num = [k * wn**2]

   den = [1, 2*zeta*wn, wn**2]

   # Funciones de Transferencia

   sys = cl.tf(num, den)   # FT Planta

   sys0 = cl.tf(K,1)      # FT Ganacia Proporcional (Kc)

   sys1 = cl.tf(1, Ti)  # FT Ganacia Integral (1/Ti*S)

   sysc = sys0 + sys1      # FT Controlador PI

   sysf = cl.series(sysc,sys) # FT [Planta - Controlador PI]

   feed = cl.feedback(sysf,1) # Lazo Cerrado

   # Parametrosde funcion step_info

   step = cl.step_info(feed)

  # RiseTime       = Tiempo de subida:

  # SettlingTime    = Tiempo de estabilización:

  # SettlingMin     = Asentamiento mín

  # SettlingMax     = Asentamiento max

  # Overshoot       = Sobreimpulso

  # Undershoot      = Subimpulso

  # Peak           = Pico
```

```
  # PeakTime       = Tiempo de pico
  # SteadyStateValue = Valor en estado estacionario    Valor en estado estable
   # Error en Estado Estable
      ##subimpulso
   ti = step.get(' Undershoot ')
   ta = ti*0.8
    return(ta)
poblacion = Poblacion(
            n_individuos = 800,
            n_variables  = 2,
            limites_inf  = [1,0.01],
            limites_sup  = [100,1],
            verbose     = False
            )
C = poblacion.optimizar(
        funcion_objetivo   = funcion_objetivo,
        optimizacion      = "minimizar",
        n_generaciones    = 1000,
        metodo_seleccion   = "tournament",
        elitismo         = 0.1,
        prob_mut         = 0.1,
        distribucion      = "uniforme",
        media_distribucion = 1,
        sd_distribucion    = 1,
        min_distribucion   = -1,
        max_distribucion   = 1,
        parada_temprana    = True,
        rondas_parada     = 10,
        tolerancia_parada  = 10**-16,
        verbose          = True
         )
# optimizacion de la parte K y Ti de la planta
K  = C[0]
Ti = C[1]
# Planta
# Definir los parámetros de la función de transferencia
k = 1.6222
```

```
wn = 0.238

zeta = 0.8

# Numerador y denominador de la función de transferencia

num = [k * wn**2]

den = [1, 2*zeta*wn, wn**2]

sys = cl.tf(num, den) # FT Planta

feeed = cl.feedback(sys,1) # lazo cerrado

K_1 = K            # Ganancia Proporcional

den_Ti = [Ti,0]      # Tiempo Integral

# Controlador

sys0 = cl.tf(K,1)     # FT Ganancia Propocinal

sys1 = cl.tf(1, den_Ti) # FT Ganancia Integral

sysc = sys0 + sys1 # FT Controlador PI

sysf = cl.series(sysc,sys) # Series

feed = cl.feedback(sysf,1) # Lazo Cerado para el control

# Grafica

t = np.linspace(0,50,500)

t,ye = cl.step_response(feed, T=t) # Respuesta Esacalon

t,y1 = cl.step_response(sys, T=t)

t,y2 = cl.step_response(feeed, T=t)

S = cl.step_info(feed) # Informacion

for i in S:

   print(f"{i}: {S[i]}")

# Para graficar

plt.grid()

plt.title('Entrenamiento: Datos vs Modelo')

plt.plot(t, ye, 'r-', linewidth=3)  # Control

plt.plot(t, y1, 'g-', linewidth=3)  #lazo abierto

plt.plot(t, y2, 'b-', linewidth=3)  #lazo cerrado

plt.legend(['Control PI','lazo abiero','lazo cerrado'], loc="lower right")

plt.ylabel('Y(K)', fontsize=14)

plt.xlabel('Tiempo (s)', fontsize=14)

plt.show()
```

6. Código para controlador PID segundo orden

```
def funcion_objetivo(K,Ti,Td):
```

```
# Planta
# Definir los parámetros de la función de transferencia
k = 2
wn = 0.2
zeta = 0.8
# Numerador y denominador de la función de transferencia
num = [k * wn**2]
den = [1, 2*zeta*wn, wn**2]
# Funciones de Transferencia
sys = cl.tf(num, den)   # FT Planta
sys0 = cl.tf(K,1)      # FT Ganacia Proporcional (Kc)
sys1 = cl.tf(1, Ti)  # FT Ganacia Integral (1/Ti*S)
sys2 = cl.tf(Td,1)   # FT Ganacia Dervaritva (Td*S)
sysc = sys0 + sys1 + sys2      # FT Controlador PI
sysf = cl.series(sysc,sys) # FT [Planta - Controlador PI]
feed = cl.feedback(sysf,1) # Lazo Cerrado
# Parametrosde funcion step_info
step = cl.step_info(feed)
# RiseTime       = Tiempo de subida:
# SettlingTime    = Tiempo de estabilización:
# SettlingMin     = Asentamiento mín
# SettlingMax     = Asentamiento max
# Overshoot       = Sobreimpulso
# Undershoot      = Subimpulso
# Peak           = Pico
# PeakTime       = Tiempo de pico
# SteadyStateValue = Valor en estado estacionario    Valor en estado estable
# Error en Estado Estable
y = step.get('SteadyStateValue')
e = 1 - y
   ##tiempo de estabilizacion
t_i = step.get('SettlingMax')
ta = t_i/5
   ##sobreimpulso
sp_1 = step.get('Overshoot')
sp = sp_1*5
```

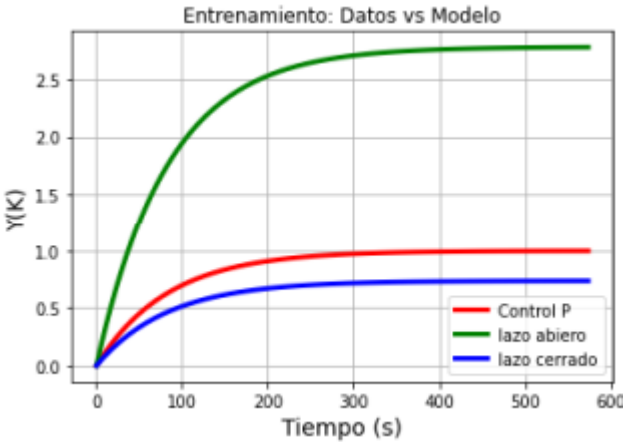
```
    r = e + ta + sp
    return(r)
poblacion = Poblacion(
            n_individuos = 800,
            n_variables  = 3,
            limites_inf  = [1,0.01,0.01],
            limites_sup  = [100,1,0.1],
            verbose     = False
            )
C = poblacion.optimizar(
        funcion_objetivo   = funcion_objetivo,
        optimizacion      = "minimizar",
        n_generaciones    = 1000,
        metodo_seleccion   = "tournament",
        elitismo         = 0.1,
        prob_mut         = 0.1,
        distribucion      = "uniforme",
        media_distribucion = 1,
        sd_distribucion    = 1,
        min_distribucion   = -1,
        max_distribucion   = 1,
        parada_temprana    = True,
        rondas_parada     = 10,
        tolerancia_parada  = 10**-16,
        verbose          = True
         )
# optimizacion de la parte K y Ti de la planta
K  = C[0]
Ti = C[1]
Td = C[2]
# Planta
# Definir los parámetros de la función de transferencia
k = 1.6222
wn = 0.238
zeta = 0.8
# Numerador y denominador de la función de transferencia
num = [k * wn**2]
```

```
den = [1, 2*zeta*wn, wn**2]
sys = cl.tf(num, den) # FT Planta
feeed = cl.feedback(sys,1) # lazo cerrado
K_1 = K           # Ganancia Proporcional
den_Ti = [Ti,0]      # Tiempo Integral
den_Td = [Td,0]
# Controlador
sys0 = cl.tf(K,1)     # FT Ganancia Propocinal
sys1 = cl.tf(1, den_Ti) # FT Ganancia Integral
sys2 = cl.tf(den_Td,1)  # FT Ganancia Derivativo
sysc = sys0 + sys1 + sys2 # FT Controlador PID
sysf = cl.series(sysc,sys) # Series
feed = cl.feedback(sysf,1) # Lazo Cerado para el control
# Grafica
t = np.linspace(0,60)
t,ye = cl.step_response(feed, T=t) # Respuesta Esacalon
t,y1 = cl.step_response(sys, T=t)
t,y2 = cl.step_response(feeed, T=t)
S = cl.step_info(feed) # Informacion
for i in S:
   print(f"{i}: {S[i]}")
# Para graficar
plt.grid()
plt.title('Entrenamiento: Datos vs Modelo')
plt.plot(t, ye, 'r-', linewidth=3)  # Control
plt.plot(t, y1, 'g-', linewidth=3)  #lazo abierto
plt.plot(t, y2, 'b-', linewidth=3)  #lazo cerrado
plt.legend(['Control PID','lazo abiero','lazo cerrado'], loc="lower right")
plt.ylabel('Y(K)', fontsize=14)
plt.xlabel('Tiempo (s)', fontsize=14)
plt.show()
```

NOTA: Los códigos anteriores se basaron en el código, CODIGO OPTIMIZACION CON ALGORTIMO GENETICO by Joaquín Amat Rodrigo, proporcionado por el profesor.

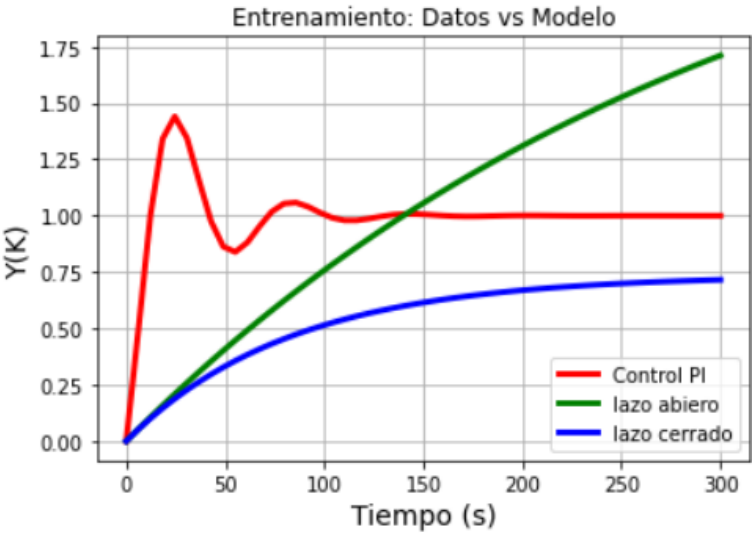
RESULTADOS

1. Para el controlador P, de primer orden

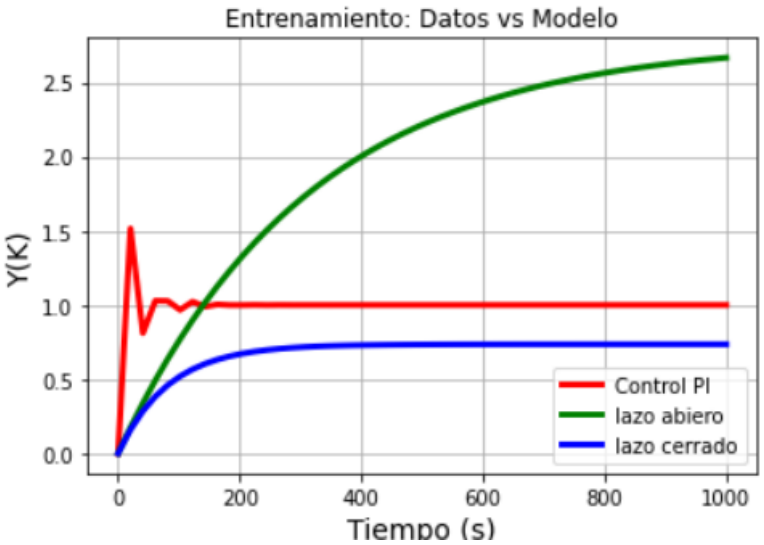


Este controlador proporcional (P) de primer orden ajusta su salida en proporción directa al error de entrada. Es ideal para sistemas que requieren una respuesta rápida sin necesidad de ajuste fino, siendo adecuado para aplicaciones simples donde una corrección proporcional es suficiente.

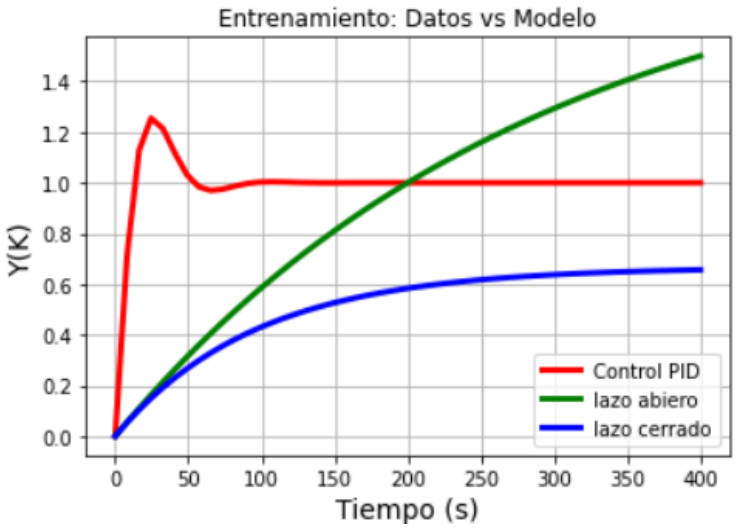
2. Para el controlador PI, de primer orden



Este controlador proporcional-integral (PI) de primer orden combina la acción proporcional con la integral. La parte proporcional responde al error presente, mientras que la integral corrige errores acumulados a lo largo del tiempo, eliminando el offset y mejorando la precisión del sistema en aplicaciones donde una respuesta más refinada es necesaria.

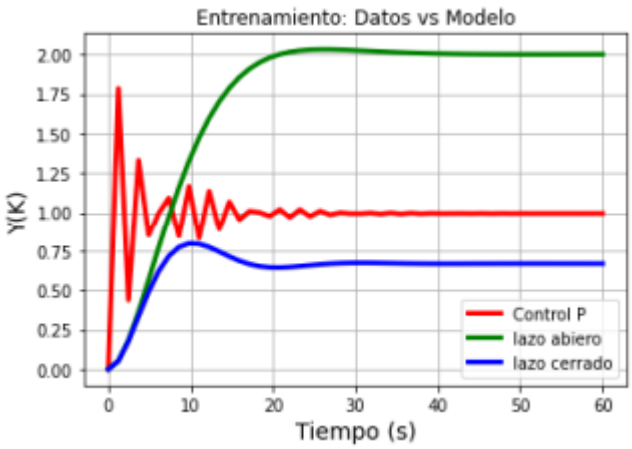


3. Para el controlador PID, primer orden



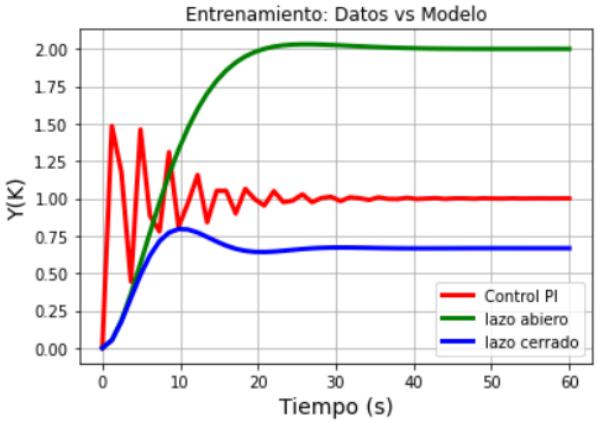
Este controlador proporcional-integral-derivativo (PID) de primer orden incluye las acciones proporcional, integral y derivativa. Proporciona una respuesta rápida y precisa ajustando la salida no solo en función del error presente y pasado, sino también anticipando futuros errores. Es adecuado para sistemas que requieren una alta precisión y estabilidad.

4. Para el controlador P, segundo orden



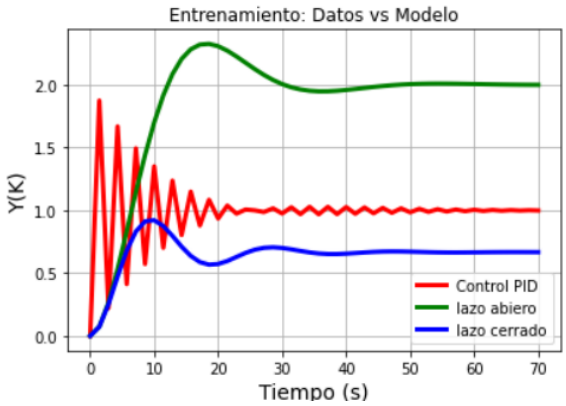
Este controlador proporcional (P) de segundo orden se aplica a sistemas que requieren una corrección proporcional, pero con una dinámica más compleja que un sistema de primer orden. Es útil en aplicaciones donde se debe tener en cuenta una mayor inercia o retraso en la respuesta del sistema.

5. Para el controlador PI, segundo orden



Este controlador proporcional-integral (PI) de segundo orden maneja sistemas con una dinámica más compleja, combinando acciones proporcionales e integrales para eliminar errores a largo plazo y ajustar de manera efectiva en sistemas con mayor inercia. Es ideal para aplicaciones donde la eliminación del offset es crucial y la respuesta del sistema es más compleja.

6. Para el controlador PID, segundo orden



Este controlador proporcional-integral-derivativo (PID) de segundo orden está diseñado para sistemas con una dinámica aún más compleja, ajustando la salida de manera precisa y rápida al considerar el error presente, pasado y futuro. Es adecuado para aplicaciones que requieren una precisión extrema y una estabilidad robusta, incluso en sistemas con inercia significativa y retrasos.

CONCLUSION

En conclusión, el laboratorio permitió explorar cómo los controladores P, PI y PID responden a distintas funciones objetivo, lo que proporcionó una comprensión más profunda de los algoritmos genéticos en el contexto de la ingeniería de control

Se pudo observar que para los sistemas de primer orden para controladores P, la acción proporcional proporciona corrección inmediata al error presente, pero puede no ser suficiente para sistemas con requisitos de precisión más elevados; para la de segundo orden ofrece una corrección proporcional efectiva, aunque puede no eliminar completamente los errores acumulados a lo largo del tiempo, resulta es para útil sistemas con una dinámica más compleja, incluyendo una mayor inercia o retraso.

Para los controladores PI de primer orden y segundo orden la acción integral elimina el offset al considerar los errores acumulados, mejorando la precisión global del sistema

Para los controladores PID de primer y segundo orden Proporciona una respuesta rápida y precisa al combinar las acciones proporcional, integral y derivativa, al considerar el error presente, pasado y anticipar futuros errores, ofrece una corrección

extremadamente precisa y estable, ideal para aplicaciones industriales críticas donde la exactitud y la estabilidad son primordiales