UNIVERSIDAD NACIONAL DE MISIONES

FACULTAD DE INGENIERÍA

Carrera:

INGENIERIA ELECTRÓNICA

Asignatura:

Inteligencia Artificial

(Dpto. Electrónica)

Informe de Trabajo Final.

Tema:

Modelado de sistemas del tipo caja gris mediante algoritmo genético.

Autor:

Berger, Juan José.

Docentes y/o equipo de cátedra responsables:

- Ing. Krujoski, Matías Gabriel.
- Ing. Skrauba, Axel Alfredo.
- Sr. Rodriguez, Mariano David

Oberá-Misiones, 22/12/2023

Contenido

Introducción.

El presente informe es un requerimiento del curso de posgrado “Inteligencia artificial aplicada a ingeniera con Python”, dictada en la Facultad de Ingeniería Oberá (FIO) de la Universidad Nacional de Misiones (UNaM), orientado a profundizar los conocimientos de una de las unidades temáticas del curso. En este caso, dicha unidad es la de algoritmos genéticos (A.G.) donde se busca modelar un sistema del tipo caja gris. Para ello: Primero se realiza una introducción teórica que abarca la deducción matemática del modelo del sistema a hallar, un motor de corriente continua (CC). En segundo lugar, se procede a explicar el ensayo realizado al motor de CC para adquirir las señales necesarias para la tercera parte de este trabajo y además se explica cómo modelar el mismo sistema con un método clásico. En tercer lugar, se diseña un A.G. con ayuda de la librería DEAP y se procesan las señales adquiridas con el mismo. Para finalizar, se realiza al análisis de los resultados obtenidos con el A.G. y el método clásico sacando conclusiones al respecto.

Introducción teórica.

En esta sección se proceden a explicar todos los fundamentos teóricos necesarios para entender qué es y cómo se modela un sistema del tipo caja gris.

Modelo del sistema-Motor de CC.

Un motor de CC se puede definir como una máquina que transforma energía eléctrica en energía mecánica provocando un movimiento rotatorio, gracias a la acción de un campo eléctrico rotatorio [1]. Un esquema básico del mismo es el que se observa en la Figura 1, en dicho esquema se observan varios parámetros que se describen en la Tabla 1.

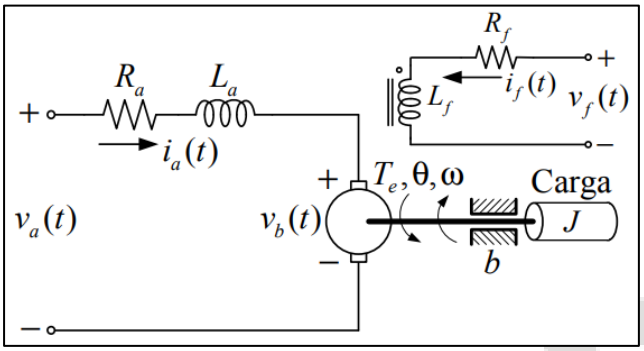

Figura 1: Esquema básico de un Motor CC [2].

Tabla 1: Resumen de datos y variables del Motor CC.

Resumen de Datos y Variables			
Simbolo	**Descripción**	**Valor**	**Unidades**
$v_a(t)$	Tensión de armadura (Entrada del sistema)	12	V
R_a	Resistencia de armadura	Desconocido	Ω
L_a	Inductancia de armadura	Desconocido	H
$i_a(t)$	Corriente de armadura	Desconocido	A
$v_b(t)$	Voltaje de campo	Desconocido	V
T_e	Torque del motor	Desconocido	Nm
J	Inercia del motor	Desconocido	Kgm^2
θ	Posición angular del eje del motor	Desconocido	rad
ω	Velocidad angular del eje del motor	Desconocido	rad/s
b	Rozamiento del eje del motor	Desconocido	Nms/rad
K_b	Constante de velocidad	Desconocido	rad/sV
K_t	Constante de torque	Desconocido	Nm/A

Modelar dicho motor es poder describir y predecir el comportamiento del mismo a través una función de transferencia. Para obtener esta última, primero se parte de ecuaciones diferenciales obtenidas a través de leyes físicas. Luego se llevan dichas ecuaciones al dominio de la frecuencia compleja para obtener dicha función de una forma más sencilla que en el dominio temporal.

En función de esto último se procede a analizar el sistema en dos partes, eléctrica y mecánica. Para la parte eléctrica se utilizará la ley de voltajes de Kirchhoff obteniendo una ecuación diferencial **(1)**.

$$v_{a(t)} = v_{R_a(t)} + v_{L_a(t)} + v_{b(t)}$$
$$v_{a(t)} = R_a i_{a(t)} + L_a \frac{di_{a(t)}}{dt} + v_{b(t)} \qquad \textbf{(1)}$$

Para relacionar la parte eléctrica con la parte mecánica se utilizará la constante de torque que relacionará el torque del motor con la corriente de armadura. Debido a esto es necesario cambiar un poco el enfoque y ver al sistema de la parte eléctrica como si su entrada fuera la diferencia de potencial entre la tensión de armadura y el voltaje de campo, además la salida del mismo será la corriente de armadura. Aplicando la transformada de Laplace a la **(1)** se halló la función de transferencia que representa esta última relación, la misma se presenta en la **(2)**.

$$V_{a(s)} = R_a I_{a(s)} + sL_a I_{a(s)} + V_{b(s)} \Longrightarrow V_{a(s)} - V_{b(s)} = I_{a(s)}(R_a + sL_a)$$
$$\frac{I_{a(s)}}{V_{a(s)} - V_{b(s)}} = \frac{1}{sL_a + R_a} = \frac{1}{R_a\left(\frac{L_a}{R_a}s + 1\right)} \qquad \textbf{(2)}$$

Siendo más específicos, la constante de torque relaciona a dicho torque y corriente de manera que:

$$i_{a(t)}K_t = T_{e(t)} \Longrightarrow I_{a(s)} = \frac{T_{e(s)}}{K_t} \Rightarrow \frac{T_{e(s)}}{I_{a(s)}} = K_t \qquad \textbf{(3)}$$

Con lo que a través de **(2)** y **(3)** se puede expresar una función de transferencia de la parte eléctrica cuya entrada sea la tensión de armadura y su salida sea el torque del motor:

$$\frac{I_{a(s)}}{V_{a(s)} - V_{b(s)}} = \frac{\frac{T_{e(s)}}{K_t}}{V_{a(s)} - V_{b(s)}} = \frac{1}{R_a\left(\frac{L_a}{R_a}s + 1\right)}$$
$$\frac{T_{e(s)}}{V_{a(s)} - V_{b(s)}} = \frac{K_t}{R_a\left(\frac{L_a}{R_a}s + 1\right)} \left[\frac{Nm}{V}\right] \qquad \textbf{(4)}$$

Otra relación entre la parte mecánica y eléctrica es la constante de velocidad que relaciona el voltaje de campo con la velocidad angular del eje del motor, dicha relación es:

$$\omega_{(t)}K_b = v_{b(t)} \Longrightarrow V_{b(s)} = \Omega_{(s)}K_b \qquad \textbf{(5)}$$

Dejaremos en ese punto el análisis eléctrico y se dará comienzo al análisis de la parte mecánica. Para empezar, podemos definir al Torque del motor como la **(6)** a partir de la segunda ley de Newton para la rotación.

$$T_{e(t)} = J\frac{d\omega_{(t)}}{dt} + b\omega_{(t)} \qquad \textbf{(6)}$$

Aplicando la transformada de Laplace, podemos relacionar al torque del motor, la velocidad angular del eje del motor y al torque del primer engranaje como:

$$T_{e(s)} = (sJ + b)\Omega_{(s)} \qquad \textbf{(7)}$$

Entonces, reemplazando la **(5)** y la **(7)** en la **(4)** se puede obtener una función de transferencia que describa la velocidad de giro del eje del motor en función de la tensión de armadura del motor, dicha expresión es la que se presenta en la (8).

$$\frac{T_{e(s)}}{V_{a(s)} - V_{b(s)}} = \frac{(sJ + b)\Omega_{(s)}}{V_{a(s)} - \Omega_{(s)}K_b} = \frac{K_t}{R_a\left(\frac{L_a}{R_a}s + 1\right)}$$
$$(sJ + b)\Omega_{(s)} = \frac{K_t}{R_a\left(\frac{L_a}{R_a}s + 1\right)}\left(V_{a(s)} - \Omega_{(s)}K_b\right)$$
$$(sJ + b)\Omega_{(s)} = \frac{K_t}{R_a\left(\frac{L_a}{R_a}s + 1\right)}V_{a(s)} - \frac{K_tK_b}{R_a\left(\frac{L_a}{R_a}s + 1\right)}\Omega_{(s)}$$

$$\frac{K_t}{R_a\left(\frac{L_a}{R_a}s+1\right)}V_{a(s)}=\left[\frac{K_tK_b}{R_a\left(\frac{L_a}{R_a}s+1\right)}+sJ+b\right]\Omega_{(s)}$$

$$\frac{K_t}{R_a\left(\frac{L_a}{R_a}s+1\right)}V_{a(s)}=\left[\frac{K_tK_b+R_a\left(\frac{L_a}{R_a}s+1\right)Js+bR_a\left(\frac{L_a}{R_a}s+1\right)}{R_a\left(\frac{L_a}{R_a}s+1\right)}\right]\Omega_{(s)}$$

$$\frac{K_t}{R_a\left(\frac{L_a}{R_a}s+1\right)}V_{a(s)}=\left[\frac{K_tK_b+L_aJs^2+R_aJs+bL_as+bR_a}{R_a\left(\frac{L_a}{R_a}s+1\right)}\right]\Omega_{(s)}$$

$$\frac{K_t}{R_a\left(\frac{L_a}{R_a}s+1\right)}V_{a(s)}=\left[\frac{(L_aJ)s^2+(R_aJ+bL_a)s+(bR_a+K_tK_b)}{R_a\left(\frac{L_a}{R_a}s+1\right)}\right]\Omega_{(s)}$$

$$\frac{\Omega_{(s)}}{V_{a(s)}}=\frac{K_t}{\cancel{R_a}\left(\frac{\cancel{L_a}}{\cancel{R_a}}s+1\right)}\frac{\cancel{R_a}\left(\frac{\cancel{L_a}}{\cancel{R_a}}s+1\right)}{(L_aJ)s^2+(R_aJ+bL_a)s+(bR_a+K_tK_b)}$$

$$G_p(s)=\frac{\Omega_{(s)}}{V_{a(s)}}=\frac{K_t}{(L_aJ)s^2+(R_aJ+bL_a)s+(bR_a+K_tK_b)} \qquad \textbf{(8)}$$

Si bien esta es la función de transferencia que describe al motor, la misma está en el tiempo continuo. El modelado de un sistema con un algoritmo implica trabajar en el dominio del tiempo discreto, motivo por el cual es necesario discretizar la (8) pasando del plano s al plano z mediante una aproximación bilineal de Tustin.

La aproximación Tustin consiste en partir de la ley de transformación de z a s (9) para pasar a la ley de transformación inversa, es decir, transformar de s a z. A esta última transformación se la aproxima al primer orden del logaritmo mediante la serie del argumento de la tangente hiperbólica [3]. Dicha aproximación puede observarse en la (10) y tomando solo el primer orden de la misma se llega finalmente a la (11) que es la aproximación de Tustin. Cabe aclarar que T_m es el período de muestreo.

$$z=e^{sT_m}\rightarrow s=\frac{\ln(z)}{T_m} \qquad \textbf{(9)}$$

$$s=\frac{2}{T_m}\left[\frac{z-1}{z+1}+\frac{1}{3}\left(\frac{z-1}{z+1}\right)^3+\frac{1}{5}\left(\frac{z-1}{z+1}\right)^5+\cdots\right] \qquad \textbf{(10)}$$

$$s\cong\frac{2}{T_m}\frac{z-1}{z+1} \qquad \textbf{(11)}$$

Entonces, reemplazando la (11) en la (8) se obtiene finalmente la función de transferencia de la planta en tiempo discreto o en el plano z. La misma es la que se presenta en la (12).

$$G_p(z) = \frac{\Omega_{(z)}}{V_{a(z)}} = \frac{K_t}{(L_aJ)\left(\frac{2}{T_m}\frac{z-1}{z+1}\right)^2 + (R_aJ + bL_a)\left(\frac{2}{T_m}\frac{z-1}{z+1}\right) + (bR_a + K_tK_b)}$$

$$G_p(z) = \frac{\Omega_{(z)}}{V_{a(z)}} = \frac{K_t}{\frac{4}{T_m{}^2}(L_aJ)\left(\frac{z-1}{z+1}\right)^2 + \frac{2}{T_m}(R_aJ + bL_a)\left(\frac{z-1}{z+1}\right) + (bR_a + K_tK_b)}$$

$$G_p(z) = \frac{\Omega_{(z)}}{V_{a(z)}} = \frac{K_t(z+1)^2}{\frac{4}{T_m{}^2}(L_aJ)(z-1)^2 + \frac{2}{T_m}(R_aJ + bL_a)(z-1)(z+1) + (bR_a + K_tK_b)(z+1)^2}$$

$$G_p(z) = \frac{K_t(z^2+2z+1)}{\frac{4}{T_m{}^2}(L_aJ)(z^2-2z+1) + \frac{2}{T_m}(R_aJ + bL_a)(z^2-1) + (bR_a + K_tK_b)(z^2+2z+1)}$$

$$G_p(z) = \frac{\Omega_{(z)}}{V_{a(z)}} = \frac{a_2z^2 + a_1z + a_0}{b_2z^2 + b_1z + b_0} \quad \textbf{(12)}$$

Donde:

$$a_2 = K_t;\ a_1 = 2K_t;\ a_0 = K_t$$

$$b_2 = \frac{4}{T_m{}^2}(L_aJ) + \frac{2}{T_m}(R_aJ + bL_a) + (bR_a + K_tK_b)$$

$$b_1 = -\frac{8}{T_m{}^2}(L_aJ) + 2(bR_a + K_tK_b)$$

$$b_0 = \frac{4}{T_m{}^2}(L_aJ) - \frac{2}{T_m}(R_aJ + bL_a) + (bR_a + K_tK_b)$$

Se acostumbra presentar la (12) de forma que el parámetro b_2 quede normalizado, es decir, que su valor sea unitario. Para ello se debe dividir a todos los coeficientes por dicho parámetro obteniendo la (13). Notar también que se aprovecha las relaciones de los coeficientes del numerador para simplificarlos a uno solo.

$$G_p(z) = \frac{\Omega_{(z)}}{V_{a(z)}} = \frac{nz^2 + 2nz + n}{z^2 + d_1z + d_0} \quad \textbf{(13)}$$

Donde:

$$n = \frac{a_2}{b_2}\ ;\ d_1 = \frac{b_1}{b_2}; d_0 = \frac{b_0}{b_2} \quad \textbf{(14)}$$

Por último, reordenado y aplicando la antitransformada z a la (13) puede obtenerse la ecuación recursiva que modela al motor de CC. La misma es la que se presenta en la (15).

$$\Omega_{(z)}(z^2 + d_1 z + d_0)z^{-2} = V_{a(z)}(n_2 z^2 + n_1 z + n_0)z^{-2}$$
$$\Omega_{(z)} + d_1 z^{-1}\Omega_{(z)} + d_0 z^{-2}\Omega_{(z)} = n_2 V_{a(z)} + n_1 z^{-1} V_{a(z)} + n_0 z^{-2} V_{a(z)}$$
$$\Omega_{(z)} = n_2 V_{a(z)} + n_1 z^{-1} V_{a(z)} + n_0 z^{-2} V_{a(z)} - d_1 z^{-1}\Omega_{(z)} - d_0 z^{-2}\Omega_{(z)}$$

$$\omega(k) = nv_a(k) + 2nv_a(k-1) + nv_a(k-2) - d_1\omega(k-1) - d_0\omega(k-2) \quad \textbf{(15)}$$

A partir de este punto ya queda definido como se puede modelar el motor de CC, lo que resta ahora es obtener algún mecanismo o proceso para estimar los parámetros que aparecen en la (15).

Antes de continuar, es necesaria presentar el gran problema que presenta la transformada z inversa que es la no unicidad de la misma. Esto quiere decir que va a haber infinitos valores de n, d_1 y d_0 que pueden representar muy bien la (15) pero no pueden representar bien la (13). Desde otro punto de vista, al aplicar a la (13) la antitransformada se pueden generar infinitas (15).

Por esto, es necesario hallar un procedimiento que permita hallar los coeficientes que modelen bien la relación entrada salida (15) pero que represente también un sistema útil (13). ¿Qué sería un sistema útil? Uno que como mínimo sea estable, es decir, que los polos generados por d_1 y d_0 se encuentren dentro del circulo unitario. Para lograr modelar bien la relación entrada-salida se buscará reducir el error cuadrático entre la salida real del sistema y la salida estimada.

Ensayo.

En esta sección se explica cómo se obtienen las señales requeridas para la identificación del sistema junto con la fundamentación del método clásico aplicado para lograr este fin.

Laboratorio

Las dos señales necesarias para el proceso, el escalón y la respuesta del mismo, se obtuvieron de un laboratorio realizado en la asignatura “Sistemas de Control 1” donde el objetivo del mismo era modelar sistemas a partir de métodos clásicos cuyo procedimiento era más que nada gráfico.

El laboratorio se realizó a el motor CC Remssi SR42-12200, que se muestra en la Figura 2 y equipamiento utilizado fue:

- Fuente de alimentación conmutada Maeni SW-12-20, 12V/10A.
- Fuente de alimentación ATTEN TPR3005T-3C.
- Llave electrónica con MOSFET, marca NFB.
- Cables para las conexiones.
- Un multímetro con las correspondientes puntas de medición.

- Osciloscopio de almacenamiento digital con puntas de medición.
- Generador de funciones con cable de señal BNC _ cocodrilos.
- Pendrive para almacenar las formas de ondas adquiridas en el osciloscopio.

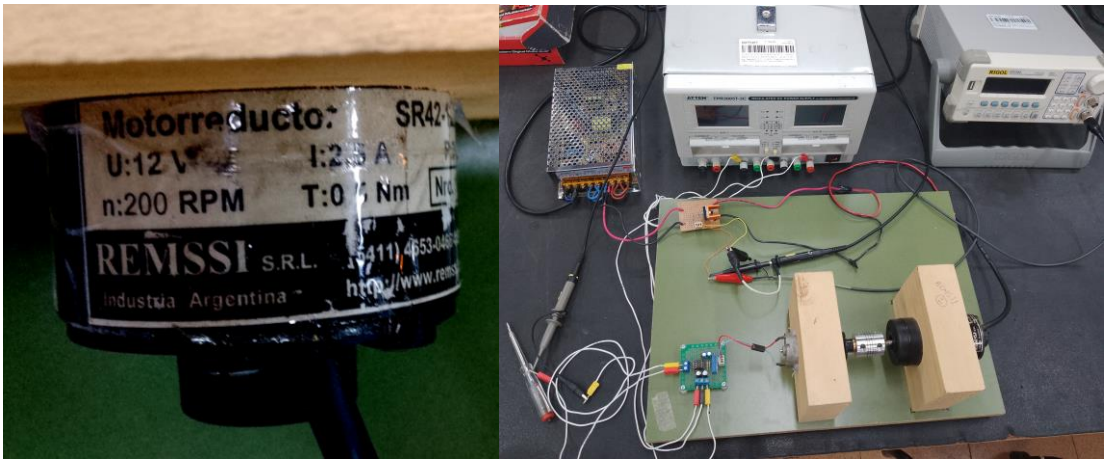

Figura 2: Motor CC a modelar e imagen del ensayo.

Para la obtención experimental de la respuesta (velocidad angular del eje) al escalón del motor utilizado, se debió implementar el circuito descrito en la Figura 3, realizando los siguientes procedimientos:

1) Encender el osciloscopio y ajustar la base de tiempo en el modo "Roll".

2) Configurar el generador de funciones (GF) para obtener una señal cuadrada con amplitud de 9 V y una frecuencia aproximada de 0,1Hz. Para esto, el cable de señal BNC deberá conectarse a la salida CMOS del GF, verificando la señal de salida con el canal 1 (CH1) del osciloscopio. La señal obtenida es la que permitirá aplicar el escalón de tensión a la planta.

3) Ajustar la escala vertical (sensibilidad vertical) y la base de tiempo del osciloscopio, a los efectos de visualizar correctamente un pulso completo en pantalla. Tratar de ocupar toda la pantalla con la visualización.

4) Encender la fuente de alimentación "*GW*" y regularla para obtener 12V a su salida. Verificar el voltaje con el multímetro.

5) Sin energía, realizar todas las conexiones indicadas en la Figura 3. El canal 1 del osciloscopio debe utilizarse para la señal que permite aplicar la referencia (salida del GF), y el canal 2 para la salida del acondicionador de señal (velocidad).

6) Energizar los elementos de medición, luego el GF, finalmente la fuente *Maeni*.

7) Visualizar en pantalla las señales inyectadas por el GF (CH1) y de salida en el acondicionador de señal del módulo (CH2).

8) Mediante el botón "Run/Stop", capturar en pantalla ambas señales. Finalmente, guardar la imagen adquirida, en el *pendrive*.

9) Desenergizar la fuente *Maeni*, luego el GF, la fuente *GW* y finalmente el osciloscopio.

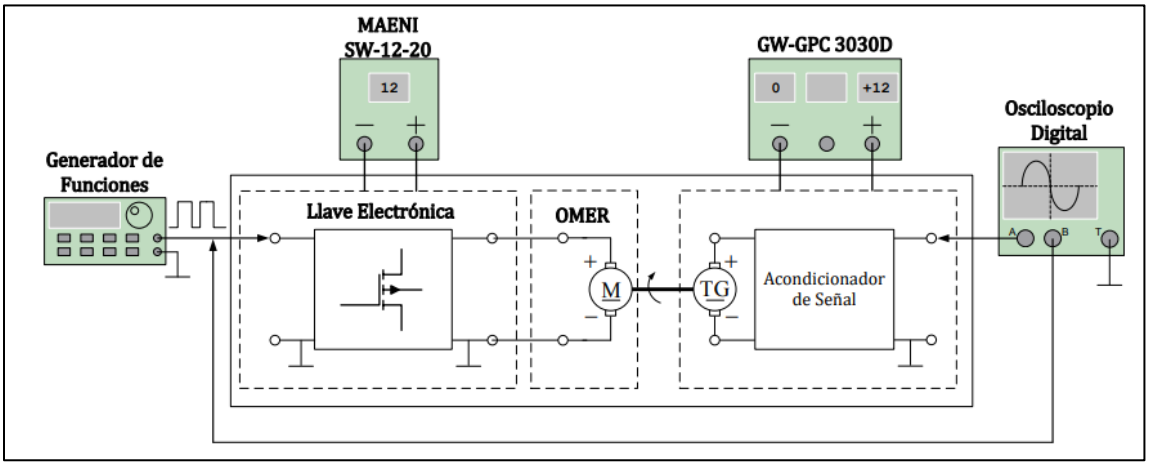

Figura 3: Diagrama de bloques del ensayo experimental para obtener la curva de respuesta al escalón del motor CC Remssi en lazo abierto.

La velocidad del motor es sensada a través de un tacogenerador (con su respectivo circuito acondicionador de señal), que se presenta en la, el cual entrega una tensión v_{tg} relacionada con la velocidad angular ω en [rad/s]. La tensión de salida del tacogenerador está calibrada a 4 V para una entrada de tensión de armadura nominal de Va = 12 V. Es decir, a su velocidad nominal de 200 rpm el taco generador debe entregar una señal de 4 V de amplitud, por lo que se puede definir que el mismo tiene una ganancia de:

$$\frac{\Omega(s)}{V_{tg}(s)} = K_{tg} = \frac{200\,rpm}{4\,V}\frac{2\pi\,rad}{revolucion}\frac{minuto}{60\,segundos} = \frac{5\pi}{3}\,[rad/Vs] \qquad \textbf{(16)}$$

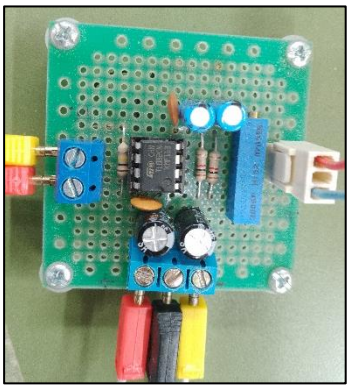

Figura 4: Circuito acondicionador de señal.

Mediante este ensayo se pudieron obtener las señales que se prestan en la Figura 5 de la misma forma en la que se observan en el osciloscopio. Además de dicha imagen se almacenó en archivo de extensión csv las amplitudes de ambas señales registradas por el osciloscopio junto con un vector de tiempo. Es con este último archivo que se realizara el procesamiento de la señal.

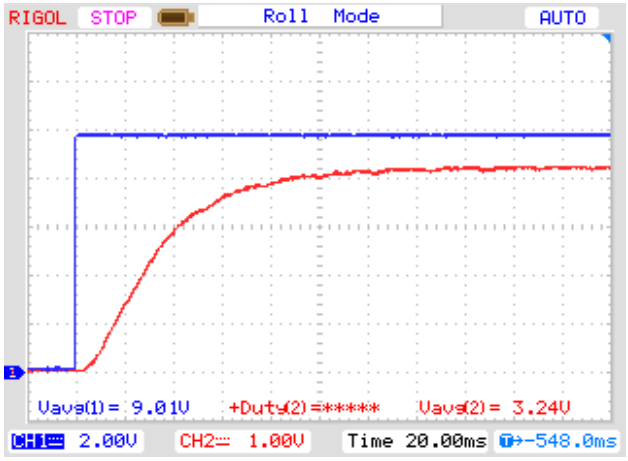

Figura 5: Señales obtenidas con el ensayo. CH1: Escalón de referencia, CH2: Respuesta del sistema.

Método de tres puntos de Stark – Mollenkamp

Como se mencionó anteriormente, el objetivo de este laboratorio era el modelar el motor con métodos clásicos. De todos los métodos que se probó, el que dio mejores resultados fue el de Stark – Mollenkamp. A continuación, se resume el mismo junto con los resultados que se obtuvieron.

Los instantes seleccionados en este método, son los tiempos requeridos para que la respuesta al escalón alcance el 15% (t_{15}), el 45% (t_{45}), y el 75% (t_{75}) del valor final y_{ss}. Basado en estos instantes de tiempo, los parámetros del modelo son calculados por las siguientes expresiones:

$$x = \frac{t_{45} - t_{15}}{t_{75} - t_{15}} \quad \textbf{(17)}$$

$$\xi = \frac{0{,}0805 - 5{,}547\,(0{,}475 - x)^2}{x - 0{,}356} \quad \textbf{(18)}$$

$$f_2(\xi) = 0{,}708\ (2{,}811)^{\xi}\ \ para\ \xi < 1 \quad (19)$$

$$f_2(\xi) = 2{,}6\ \xi - 0{,}60\ \ para\ \xi \geq 1 \quad (20)$$

$$\omega_n = \frac{f_2(\xi)}{t_{75} - t_{15}} \quad (21)$$

$$f_3(\xi) = 0{,}922\ (1{,}66)^{\xi} \quad (22)$$

$$\theta = t_{45} - \frac{f_3(\xi)}{\omega_n} \quad (23)$$

$$\tau_{1,2} = \frac{\xi \pm \sqrt{\xi^2 - 1}}{\omega_n}\ \ si, y\ solo\ si\ \xi \geq 1 \quad (24)$$

$$K_p = \frac{\Delta y}{\Delta u} \quad (25)$$

Todos estos parámetros son calculados mediante el Script. 1.

```
yss=np.mean(w[N-100:N])
Kp=yss/np.mean(u[N-100:N])
t_15=0
t_45=0
t_75=0
for i in range(N):
  if (w[i]>=(yss*0.151)) & (t_15==0):
    t_15=t[i]
  if (w[i]>=(yss*0.451)) & (t_45==0):
    t_45=t[i]
  if (w[i]>=(yss*0.751)) & (t_75==0):
    t_75=t[i]
x=(t_45-t_15)/(t_75-t_15)
epsilon=(0.0805-5.547*(((0.475-x))**2))/(x-0.356)
f2=2.6*epsilon-0.6
wn=f2/(t_75-t_15)
f3=0.992*((1.66)**(epsilon))
t1=(epsilon+np.sqrt((epsilon**2)-1))/wn
t2=(epsilon-np.sqrt((epsilon**2)-1))/wn
```

Script. 1: Algoritmo para determinar los parámetros según el método Stark – Mollenkamp [4].

Obteniendo finalmente la aproximación del modelo del motor expresado en la (26).

$$G_{PSM}(s) = \frac{K_p e^{-\theta s}}{(\tau_1 s + 1)(\tau_2 s + 1)} \quad (26)$$

$$K_p = 1{,}849\ \frac{rad}{sV}; \tau_1 = 0{,}0221\ s; \tau_2 = 0{,}012\ s; \theta_{PM} = 0\ s$$

Comparando la (8) con la (26) es posible establecer las relaciones expresadas en la (27).

$$L_a J = \tau_1 \tau_2;\ R_a J + b L_a = \tau_1 + \tau_2;\ b R_a + K_t K_b = 1;\ K_t = K_p \quad (27)$$

Teniendo en cuenta estas últimas relaciones, es posible discretizar la planta modelada con el método clásico a partir de las siguientes relaciones:

$$a_2 = K_p;\ a_1 = 2K_p;\ a_0 = K_p$$
$$b_2 = \frac{4}{{T_m}^2}(\tau_1\tau_2) + \frac{2}{T_m}(\tau_1 + \tau_2) + (1)$$
$$b_1 = -\frac{8}{{T_m}^2}(\tau_1\tau_2) + 2(1) \qquad \textbf{(28)}$$
$$b_0 = \frac{4}{{T_m}^2}(\tau_1\tau_2) - \frac{2}{T_m}(\tau_1 + \tau_2) + (1)$$

Algoritmo genético.

En función de lo desarrollado en la introducción teórica, es necesario un mecanismo o proceso para hallar los 3 coeficientes que participan en la ecuación recursiva. Los objetivos que deben satisfacer dichos coeficientes son:

- Modelar la relación entrada salida obteniendo un valor relativamente bajo de la suma del error cuadrático.
- Que la soluciones representen sistemas estables.

Debido a la no unicidad de la transformada z inversa no es posible resolver este problema planteándolo con enfoque de optimización con una solución analítica. El hecho de que existan infinitas soluciones al problema implica que de forma analítica sea muy sencillo obtener una solución que cumpla con el primer objetivo, pero no tiene por qué cumplir con el segundo. Este es el principal motivo en querer resolver este problema con un enfoque de optimización, pero con un proceso estocástico con un algoritmo genético. Con este último, es posible ajustar este proceso estocástico para obtener una solución que cumpla con los dos objetivos presentados. Para hacer esto, se implementó un algoritmo genético en el lenguaje de programación Python desarrollado en el entorno de CodeColab utilizando la librería *DEAP* y a continuación se presenta paso a paso como se lo implementó.

Antes de comenzar con el algoritmo es necesario instalar la librería e importar algunos métodos de la misma. Esto es lo que se realiza en el Script. 1.

```
#Instalación de DEAP
!pip install deap
#Librerias
from deap import algorithms #Para poner en marcha el algoritmo.
from deap import base       #Para importar clases para el algoritmo
from deap import creator    #Para crear nuevas clases
from deap import tools      #Para configurar el algoritmo
import random               #Para generar números aleatorios
```

Script. 2: Instalación de DEAP, importación de métodos y clases [4].

Definición del problema y generación de la población inicial.

Creación del problema.

Lo primero a realizar es definir el tipo de problema que se desea resolver, en este caso se tienen dos objetivos por lo que es un problema multiobjetivo. El primer objetivo será disminuir la raíz de la sumatoria del error cuadrático medio y el segundo objetivo será disminuir la distancia de los polos de la (13) al origen del plano z con el propósito de que los mismos se encuentren dentro del círculo unitario. Esto es lo que se realiza en el Script. 3 donde se utiliza el método *créate()* de la clase *creator* para crear una clase que representa el problema. La misma se bautiza como *"FitnesMulti"* haciendo alusión al enfoque de multiobjetivo y hereda la clase *base.Fitness* que tiene el atributo *weights*, el cual es un tupla donde cada uno de sus elementos definen que se buscara hacer con cada objetivo. En este caso son todos valores flotantes negativos para que el algoritmo busque minimizar los mismos.

```
creator.create("FitnessMulti", base.Fitness, weights=(-1.0, -1.0, -1.0))
```

Script. 3: Creación del problema [4].

Creación de la plantilla del individuo.

Lo segundo a realizar es definir cómo será encapsulado el individuo. Para ello se utiliza el Script. 4 donde se vuelve a utilizar el método *créate()* de la clase *creator* pero esta vez se crea una clase llamada *"individual"* que hereda la clase *list* y contiene el parámetro *fitness* que se inicializa con el objeto *FtinessMulti*. Es decir, la clase *Individual* es una lista que contienen el desempeño de un individuo.

```
creator.create("Individual", list, fitness=creator.FitnessMulti)
```

Script. 4: Tipo del individuo [4].

Creación de individuos aleatorios y población inicial.

En un algoritmo genético un individuo es una propuesta de solución del problema que en nuestro caso sería el valor de los coeficientes a, d_1 y d_0. Una población no es más que un conjunto de estos individuos, un conjunto de propuestas de solución. Para poder generar individuos aleatorios y en consecuencia una población inicial se utiliza el Script. 5.

En dicho Script se realizan las siguientes acciones:

1. Se crea un objeto *toolbox* utilizando de la clase *base* el método *Toolbox()*. El mismo permite registrar funciones que utilizara el algoritmo durante su ejecución. Este objeto recibe 3 atributos que son: El nombre con el que se registra la función en la caja de herramientas, la función a registrar y por último los parámetros que necesita la función.

2. Se define una función *gen_val()* que crea 3 valores flotantes aleatorios con una distribución de probabilidad uniforme en función de límites superiores e inferiores. En este caso, dichos límites van de -2 a 2 debido que a base de prueba y error se sabe que la mejor solución se encuentra dentro de este intervalo.
3. Con ayuda del objeto *toolbox* se registra la función anterior con el alias "*attr_uniform"* pasándole también los límites anteriormente mencionados.
4. Para generar un individuo se registra la función "*individual"* que realmente es la función *tolos.initIterate* que recibe como argumento la forma en la que se encapsula un individuo, es decir, el objeto *creator.Individual* y la función con la que debe generar los valores del individuo, es decir, *toolbox.attr_uniform*.
5. Por último, se registra la función con la que se genera la población inicial con el alias *"population"* que realmente es la función *tools.initRepeat* que recibe como argumento el tipo de objeto de la población, en este caso es una lista, y la función con la que debe crear a cada integrante de la población que es la función anteriormente registrada. Esta función también puede recibir un último parámetro que la cantidad de individuos dentro de la población.

```
toolbox=base.Toolbox() #1
def gen_val(lim_i_n, lim_s_n, lim_i_d, lim_s_d): #2
  n=random.uniform(lim_i_n, lim_s_n)
  d1=random.uniform(lim_i_d, lim_s_d)
  d0=random.uniform(lim_i_d, lim_s_d)
  return [n, d0, d1]
lim_i_n, lim_s_n, lim_i_d, lim_s_d= -2, 2, -2, 2  #Limites
toolbox.register("attr_uniform", gen_val, lim_i_n, lim_s_n, lim_i_d, lim_s_d) #3
toolbox.register("individual", tools.initIterate, creator.Individual, toolbox.attr_uniform) #4
toolbox.register("population", tools.initRepeat, list, toolbox.individual) #5
```

Script. 5: Creación de individuos aleatorios y población inicial [4].

Función objetivo.

La función objetivo será el aspecto más importante a definir ya que será de forma explícita el problema a resolver y de forma implícita la manera de lograr los objetivos propuestos. Para definir dicha función hay que calcular un error a partir de una salida real y una salida estimada donde para calcular esta última es necesario tener muestras atrasadas de la señal real y la entrada real del sistema. Para obtener estas dos últimas señales se realizó lo descrito en las sección de Ensayo. En el mismo se registraron estas señales en un archivo de formato csv y a continuación se explica cómo acceder al mismo.

Lo primero a realizar es levantar el archivo.csv y para ello es necesario descargarlo de un enlace de google drive y descomprimirlo. Esto es lo que se realiza con el Script. 6.

```
# archivo.zip con las señales
!wget -c --no-check-certificate
"https://drive.google.com/uc?export=download&id=1i1kMdcNw6xbYT1MTcqnLn4G9aeKUAm75&confirm=t" -O archivo.zip
# Descomprimir dataset
!unzip -qq archivo.zip
```

Script. 6: Descarga y descompresión del archivo [4].

Lo segundo a realizar es leer archivo.csv mediante el método *read_csv()* de la librería *pandas* como un dataframe y del mismo extraer las tres señales que capturo el osciloscopio. Luego se pueden graficar estas señales con ayuda de la librería *matplotlib* obteniendo la Figura 6. Esto es lo que se realiza en el Script. 7.

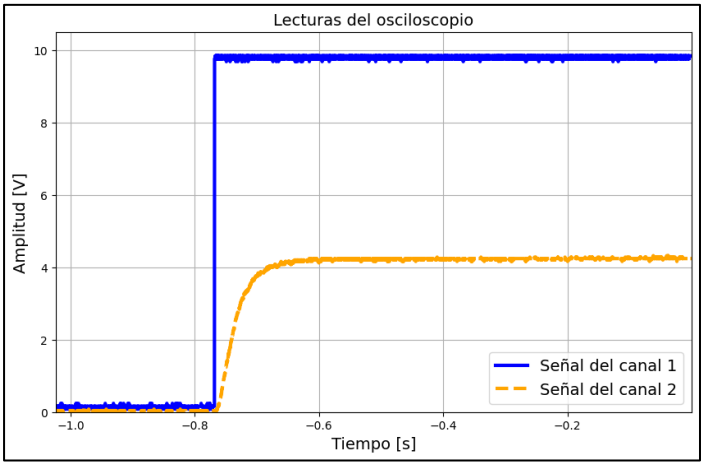

Figura 6: Lecturas del osciloscopio [4].

```
import pandas as pd            #Libreria para leer el archivo xlsx
import matplotlib.pyplot as plt #Libreria para graficar señales
# Lee el archivo csv como un dataframe
df=pd.read_csv('archivo.csv', header=[0, 1])
t    =    df[('Time', 'Second')]   #Extraigo el vector de tiempo
CH1  =    df[('X(CH1)', 'Volt')]   #Extraigo el vector del canal 1
CH2  =    df[('X(CH2)', 'Volt')]   #Extraigo el vector del canal 1
# Visualiza las señales
plt.figure(figsize=(10, 6))
plt.plot(t, CH1, label='Señal del canal 1', color='blue',linewidth=3)
plt.plot(t, CH2, label='Señal del canal 2', color='orange', linestyle="--",linewidth=3)
plt.title('Lecturas del osciloscopio', fontsize=14)
plt.grid(True)
plt.xlim(min(t),max(t))
plt.ylim(0,10.5)
plt.xlabel('Tiempo [s]', fontsize=14)
plt.ylabel('Amplitud [V]', fontsize=14)
plt.legend(loc='lower right', fontsize=14)
plt.show()
```

Script. 7: Lectura del archivo y grafica de las señales que capturo el osciloscopio [4].

En la última figura se pueden observar la señal del canal 1 no tiene la amplitud de los 12 V y esto es porque simplemente era una señal de referencia para la llave electrónica. Motivo por el cual se debe amplificar la misma para que tenga los 12 V. Por otro lado, la señal del canal 2 es levemente superior a 4 V por lo que el motor estaría girando un poco más rápido de su velocidad nominal lo

cual tiene sentido dado que el motor está prácticamente en vacío. Para que esta señal represente la velocidad del motor, se debe amplificar la misma por la ganancia del tacogenerador (16). Otro aspecto a analizar es el vector de tiempo toma valores negativos por lo que será necesario reconstruir el mismo a partir del período de muestreo y la cantidad de muestras que tomo el osciloscopio para cada señal. Para finalizar, se observa que la mitad de la señal no contiene información por lo que será necesario recortar las señales. Todo esto se realiza en el Script. 8 obteniendo la Figura 7, notar que se utiliza la librería *numpy* para definir y trabajar con los vectores.

```
import numpy as np
offset_inicial=0
offset_final=0
for i in range(len(t)):
  if CH1[i]>9:                #Busco el indice donde empieza el escalon
    offset_inicial=i
    break
Ts=t[offset_inicial+1]-t[offset_inicial]
N=len(t)-offset_inicial          #Cantidad de muestras
t=np.linspace(0,N*Ts-Ts,N)       #Redefino el vector de tiempo
for i in range(len(t)):
  if t[i]>0.2:                #Cuando la señal ya se estabilizo. Para no analizar de mas
    offset_final=i
    break
N=offset_final           #Cantidad de muestras final
u = (12/max(CH1))*np.array(CH1[offset_inicial:N+offset_inicial])      #Recorto las señales
omega  = ((5*np.pi)/3)*np.array(CH2[offset_inicial:N+offset_inicial])  #Recorto las señales
t = t[:N]                    #Recorto las señales
# Visualiza las señales
plt.figure(1,figsize=(10, 12))
plt.subplot(3, 1, 1)
plt.plot(t, u, label='Escalon', color='blue',linewidth=3)
plt.title('Señales para el proceso', fontsize=14)
plt.grid(True)
plt.xlim(0,max(t))
plt.ylim(0,12.5)
plt.xlabel('Tiempo [s]', fontsize=14)
plt.ylabel('Amplitud [V]', fontsize=14)
plt.legend(loc='lower right', fontsize=14)
plt.subplot(3, 1, 2)
plt.plot(t, omega, label='Velocidad angular', color='orange',linewidth=3)
plt.title('Señales para el proceso', fontsize=14)
plt.grid(True)
plt.xlim(0,max(t))
plt.xlabel('Tiempo [s]', fontsize=14)
plt.ylabel('Amplitud [rad/s]', fontsize=14)
plt.legend(loc='lower right', fontsize=14)
plt.tight_layout()
plt.show()
```

Script. 8: Ajuste de las señales [4].

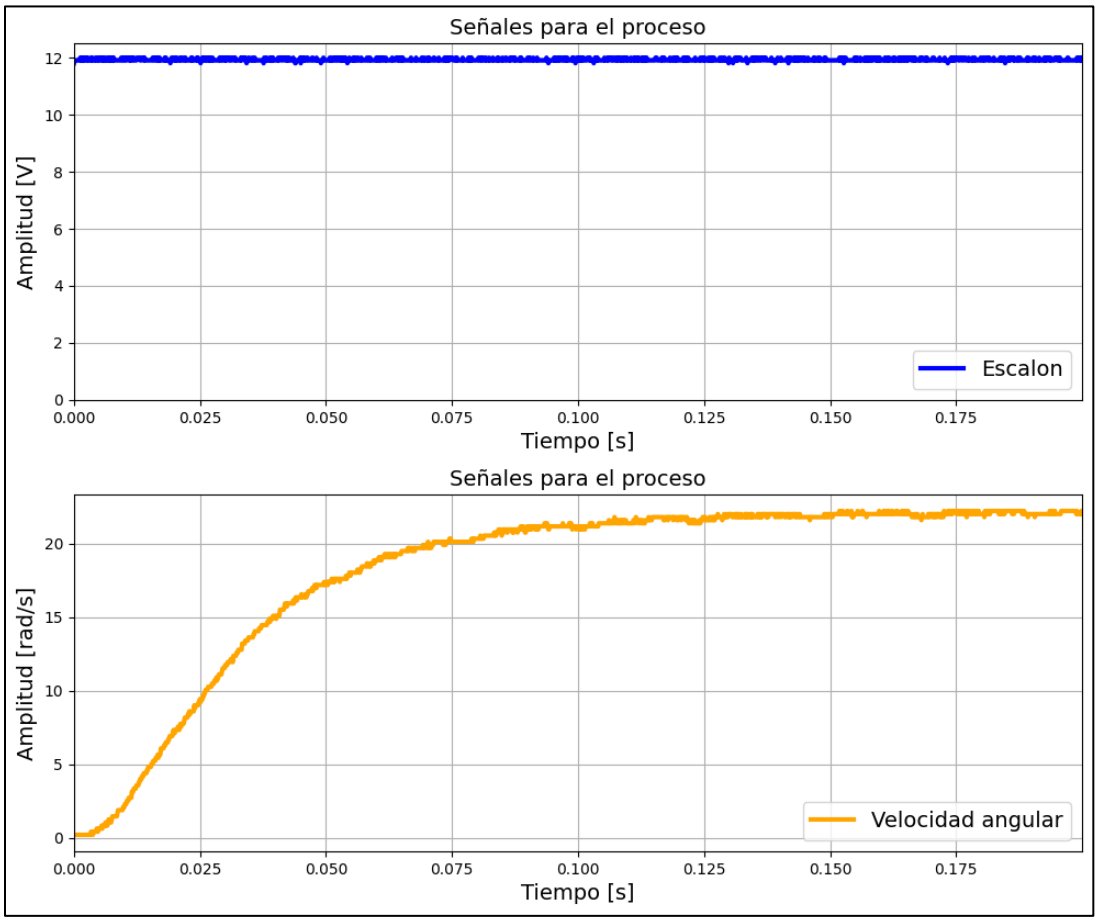

Figura 7: Ajuste de las señales [4].

Una vez listas las señales se procede a definir la función objetivo que se presenta en el Script. 9 donde se realiza lo siguiente:

1. Se declaran e inicializan los coeficientes y vectores necesarios.
2. Se calcula la señal estimada en función de la (15) para luego calcular y acumular el error cuadrático definido como: $e^2 = \left(\omega - \omega_{ag}\right)^2$. A este último se le calcula su raíz cuadrada y se lo divide por la cantidad de muestras de la señal.
3. En función de los coeficientes se calculan los polos con el método *roots()* de la clase *numpy* y se verifica si los mismos están fuera del circulo unitario. En caso de que se cumpla esto último se aplica una pena de muerte asignado una salida de 10000, es decir, un nefasto valor de desempeño por lo que es poco probable que este individuo participe en la reproducción. En caso de que no se cumpla esta condición se devuelve la distancia del polo al origen del plano z.
4. La función en general devuelve 3 parámetros, el resultado del error y las distancias de los polos al origen del plano z.

Todo esto en función de los coeficientes que tenga cada individuo, por ello la función recibe como argumento un individuo.

```
def funcion_objetivo(individuo):
  n2 = individuo[0]
  n1=2*n2
  n0=n2
  d1=individuo[2]
  d0=individuo[1]
  e_2=0
  w_e=np.zeros(N)
  for i in range(N):
    if i==0:
      w_e[i]=n2*u[i]
    if i>0:
      w_e[i]=n2*u[i]+n1*u[i-1]-d1*w[i-1]
    if i>1:
      w_e[i]=n2*u[i]+n1*u[i-1]+n0*u[i-2]-d1*w[i-1]-d0*w[i-2]
    e_2=e_2+(w[i]-w_e[i])**2
  e_rms=np.sqrt(e_2)/N
  #Definir los coeficientes del sistema discreto
  den = np.array([1, d1, d0])
  polos=np.roots(den)
  if abs(polos[0])>=1:
    if abs(polos[1])>=1:
      return e_rms, 10000, 10000,
    else:
      return e_rms, 10000, (abs(polos[1])),
  if (abs(polos[1])>=1 and abs(polos[0])<=1):
    return e_rms, (abs(polos[0])), 10000,
  return e_rms, (abs(polos[0])), (abs(polos[1])),
```

Script. 9: Función objetivo [4].

Operadores genéticos.

Reproducción o cruce.

La operación de cruce es una operación probabilística que permite que dos individuos seleccionados crucen o intercambien su información genética para crear dos nuevos individuos. Es importante indicar que la operación de cruce es probabilística; esto quiere decir que, aunque dos individuos sean seleccionados, puede que no sean modificados. La probabilidad de cruce es un hiperparámetro de los algoritmos genéticos que se va a definir más adelante [5].

Como los coeficientes son números flotantes se creó una función para realizar la reproducción que toma dichos coeficientes y crea otros haciendo un promedio entre los mismos, pero afectándolos por un factor de parentesco. Esto intenta emular el parentesco que tiene por ejemplo un hijo humano con su padre o su madre y está inspirado en el método *cxBlend* de la clase *tools* pero es relativamente más sencillo. El mismo se puede observar en el Script. 10 y devuelve dos individuos donde el primero se parece más al primer padre y el segundo se parece más al segundo padre. Esta función se registra con el alias *"mate"* y se pasa como parámetro el factor de parentesco que es igual a 1,3.

```
def arithmetic_crossover(ind1, ind2,p):
    for i, (x1, x2) in enumerate(zip(ind1, ind2)):
        ind1[i] = (p*x1+(p-1)*x2)/2
        ind2[i] = ((p-1)*x1+p*x2)/2
    return ind1, ind2
toolbox.register("mate", arithmetic_crossover, p=1.3)
```

Script. 10: Función para la reproducción [4].

Mutación.

Para la mutación se optó por el método *"mutGaussian"* de la clase *tools* que como su nombre indica a un individuo se le puede sumar un cierto valor con una densidad de probabilidad normal con una media de 0 y un desvió de 0.1. Esto pasa para cada gen del individuo, para cada coeficiente, con cierta probabilidad *indpb*. Esto es lo que se realiza en el Script. 11 y la función se registra con el alias *"mutate"* con los parámetros ya comentados.

```
toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=0.1, indpb=0.2)
```

Script. 11: Función para la mutación [4].

Selección.

Como se trata de un problema multiobjetivo, la función que seleccionará los individuos para la reproducción será el método *selNSGA2* del objeto *tools* dado que el mismo está muy optimizado y es el más usado para obtener el frente de Pareto. A continuación se presenta un resumen de la [5] respecto a este tema.

El frente de Pareto se define como el lugar geométrico que forman todas las soluciones de nuestro problema que no están dominadas. Hay que preguntarse, pues, qué significa que una solución esté dominada por otra. Para entender mejor la dominancia de Pareto, lo mejor es verla con un ejemplo práctico. La Figura 8 muestra cuatro soluciones S_i para un problema con dos objetivos, F_1 y F_2. En este caso, deseamos minimizar ambos objetivos. Podemos considerar que esas cuatro soluciones representan cuatro individuos de la población de un algoritmo genético. No estamos indicando cuántas variables tiene nuestro problema, pero en general, podemos considerar que tenemos n variables. Así, $F_1 = f(x_n)$ e, igualmente, $F_2 = f(x_n)$. En la Figura 8, cada S_i corresponde con unos valores concretos de las variables x_n. En este caso, no nos vamos a centrar en la representación de las variables, ya que estamos interesados en comparar soluciones en función de los objetivos del problema.

Para saber qué soluciones están dominadas, debemos comparar de dos en dos las cuatro soluciones.

- Empecemos comparando s_1 y s_2; s_1 es mejor que s_2 en F_2, ya que s_1 tiene un valor de 10 en F_2, mientras que s_2 tiene un valor de 20. Sin embargo, con respecto a F_1, ambas soluciones tienen el mismo valor; por lo tanto, no podemos decir que ninguna de ellas sea estrictamente mejor que la otra en F_1. Por este motivo, no podemos decir que s_1 domine a s_2. A su vez, s_2 no domina a s_1.

- Continuamos ahora comparando s_1 y s_3; de nuevo s_1 es mejor que s_3 con respecto al objetivo F_1, pero no ocurre lo mismo con respecto a F_2, ya que ambas soluciones obtienen el mismo valor. Por lo tanto, tampoco podemos decir que s_1 domine a s_3, ni que s_3 domine a s_1.
- Podemos seguir haciendo comparaciones, pero pasemos a realizar la comparación más interesante para nuestro objetivo de obtener el frente de Pareto. Esto es, pasemos a comparar s_1 y s_4. En este caso, sí podemos decir que s1 es estrictamente mejor que s4 para los dos objetivos, F_1 y F_2. Por lo tanto, podemos decir que s_1 domina a s_4 o, lo que es lo mismo, que s_4 está dominada por s_1.
- Si seguimos haciendo comparaciones, llegaremos a la conclusión de que el caso de s_1 y s_4 es el único en el que podemos encontrar una solución que domina a la otra.

Llegado a este punto, se puede afirmar que el frente de Pareto para nuestro ejemplo (según las cuatro soluciones que estamos considerando) está formado por las soluciones , s_1, s_2 y s_3.

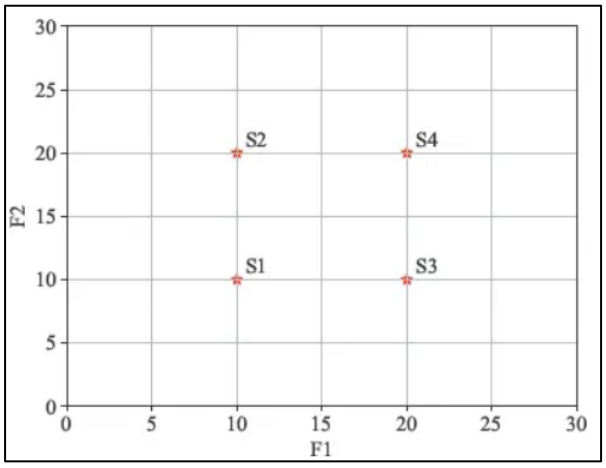

Figura 8: Ejemplo de soluciones para un problema con dos objetivos, F1 y F2 [5].

Entonces, el resultado que se espera del algoritmo genético es que logre encontrar el frente de Pareto y el algoritmo *NSGA-II* es de los más utilizados para esto en la etapa de selección. En el Script. 12 se observa cómo se registra la función *selNSGA2* bajo el alias de *"select"* en la caja de herramientas.

```
toolbox.register("select", tools.selNSGA2)
```

Script. 12: Registro de la función de selección [4].

Algoritmo genético como caja negra.

Con todo lo anterior solo falta ejecutar el algoritmo genético y para ello se desarrolló el Script. 13 donde:

1. Se define una semilla para poder repetir los resultados.

2. Se definen todos los hiperparámetros. Mediante todas las pruebas realizadas, esa combinación fue la mejor.
3. Se genera la población inicial en el objeto *pop.*
4. Se define una función *multi_stats* y se la registra para calcular métricas respecto del desempeño de cada generación.
5. Se crea un objeto *logbook* con el metodo *Logbook()* de la clase *tools* para almacenar la evolución de los valores del desempeño de los individuos durante la ejecución del algoritmo.
6. Se define un objeto *pareto* con el metodo *ParetoFront()* de la clase *tools* para almacenar todas las soluciones del frente de pareto.
7. Finalmente, se ejecuta el A.G. con el método *eaMuPlusLambda()* de la clase *algorithms*. En este tipo de algoritmo, en la selección participan tanto padres como hijos para ver quienes integran la próxima generación. El mismo, recibe los siguientes parámetros:
   7.1. *population*: Población inicial del algoritmo.
   7.2. *toolbox*: Objeto caja de herramientas con las siguientes funciones registradas: función objetivo, mecanismos de selección, cruce y mutación.
   7.3. *mu*: Número de individuos a seleccionar para cada generación.
   7.4. *lambda*: Número de hijos a producir en cada generación.
   7.5. *cxpb*: Probabilidad de cruce en tanto por uno.
   7.6. *mutpb*: Probabilidad de mutación en tanto por uno.
   7.7. *ngen*: Número de generaciones del algoritmo.
   7.8. *stats*: Objeto estadístico con las funciones registradas.
   7.9. *halloffame*: Objeto de tipo HallOfFame para almacenar la mejor solución encontrada.
   7.10. *verbose*: Atributo que indica si se deben mostrar las estadísticas en cada iteración o no.

```
random.seed(48)                                  #1_Se fija una semilla para poder repetir los resultados.
CXPB, MUTP, NGEN, N_p=0.5, 0.2, 320, 32  #2_Hiperparametros 50% de cruce, 20% de mutación,
                                                 #320 generaciones, 32 individuos en una población
pop=toolbox.population(N_p)                      #3_Población inicial
def multi_stats(pop):                            #4_Definir la función para calcular estadísticas
  media=np.zeros([3])
  desvio=np.zeros([3])
  minimo=np.zeros([3])
  maximo=np.zeros([3])
  e_2=np.zeros(N_p)
  polo1=np.zeros(N_p)
  polo2=np.zeros(N_p)
  for i in range(N_p):
    e_2[i], polo1[i], polo2[i]=toolbox.evaluate(pop[i])
  media=np.mean(e_2), np.mean(polo1), np.mean(polo2)
  desvio=np.std(e_2), np.std(polo1), np.std(polo2)
  minimo=np.min(e_2), np.min(polo1), np.min(polo2)
  maximo=np.max(e_2), np.max(polo1), np.max(polo2)
  return media, desvio, minimo, maximo
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("multi_stats", multi_stats)
logbook=tools.Logbook()                          #5_Se crea un objeto logbook
pareto=tools.ParetoFront()                       #6_Se crea un objeto pareto
pop, logbook = algorithms.eaMuPlusLambda(pop,    #7_Ejecución del algortimo
                                toolbox,
                                mu=N_p,
                                lambda_=N_p,
                                cxpb=CXPB,
                                mutpb=MUTPB,
                                ngen=NGEN,
                                stats=stats,
                                halloffame=pareto,
                                verbose=True,
                                   )
```

Script. 13: Ejecución del A.G [4].

Resultados del algoritmo genético.

Evolución del fitness.

Mediante el objeto *logbook* es posible visualizar la evolución de desempeño de los individuos a lo largo de las generaciones. Esto es lo que se realiza en el Script. 14 y se obtienes la Figura 9 y la Figura 10.

```
# Extrae la información del logbook
gen = [entry['gen'] for entry in logbook]
avg_fitness_e = [entry['multi_stats'][0][0] for entry in logbook]
avg_fitness_p1 = [entry['multi_stats'][0][1] for entry in logbook]
avg_fitness_p2 = [entry['multi_stats'][0][2] for entry in logbook]
min_fitness_e = [entry['multi_stats'][2][0] for entry in logbook]
min_fitness_p1 = [entry['multi_stats'][2][1] for entry in logbook]
min_fitness_p2 = [entry['multi_stats'][2][2] for entry in logbook]
max_fitness_e = [entry['multi_stats'][3][0] for entry in logbook]
max_fitness_p1 = [entry['multi_stats'][3][1] for entry in logbook]
max_fitness_p2 = [entry['multi_stats'][3][2] for entry in logbook]
# Evolución del error
plt.figure(1,figsize=(8, 6))
plt.plot(gen, avg_fitness_e, label='Media del error', linewidth=4)
plt.plot(gen, min_fitness_e, label='Mínimo del error', linewidth=4)
plt.plot(gen, max_fitness_e, label='Maximo del error', linewidth=4)
plt.xlabel('N° Generación', fontsize=14)
plt.ylabel('Velocidad Angular [rad/s]', fontsize=14)
plt.title('Evolucion de los parametros', fontsize=14)
#plt.ylim(0, max(min_fitness_e))
plt.legend(loc='center right', fontsize=14)
plt.grid(True)
plt.show()
# Evolución de la distancia al origen del polo1
plt.figure(2,figsize=(8, 6))
plt.plot(gen, avg_fitness_p1, label='Media de la distancia del polo 1', linewidth=4)
plt.plot(gen, min_fitness_p1, label='Mínimo de la distancia del polo 1', linewidth=4)
plt.plot(gen, max_fitness_p1, label='Maximo de la distancia del polo 1', linewidth=4)
plt.xlabel('N° Generación', fontsize=14)
plt.ylabel('Distancia al origen del plano z', fontsize=14)
plt.title('Evolucion de los parametros', fontsize=14)
#plt.ylim(0, max(min_fitness_p1))
plt.legend(loc='center right', fontsize=14)
plt.grid(True)
plt.show()
# Evolución de la distancia al origen del polo1
plt.figure(3,figsize=(8, 6))
plt.plot(gen, avg_fitness_p2, label='Media de la distancia del polo 2', linewidth=4)
plt.plot(gen, min_fitness_p2, label='Mínimo de la distancia del polo 2', linewidth=4)
plt.plot(gen, max_fitness_p2, label='Maximo de la distancia del polo 2', linewidth=4)
plt.xlabel('N° Generación', fontsize=14)
plt.ylabel('Distancia al origen del plano z', fontsize=14)
plt.title('Evolucion de los parametros', fontsize=14)
#plt.ylim(0, max(min_fitness_p2))
plt.legend(loc='center right', fontsize=14)
plt.grid(True)
plt.show()
```

Script. 14: Evolución del desempeño de los individuos [4].

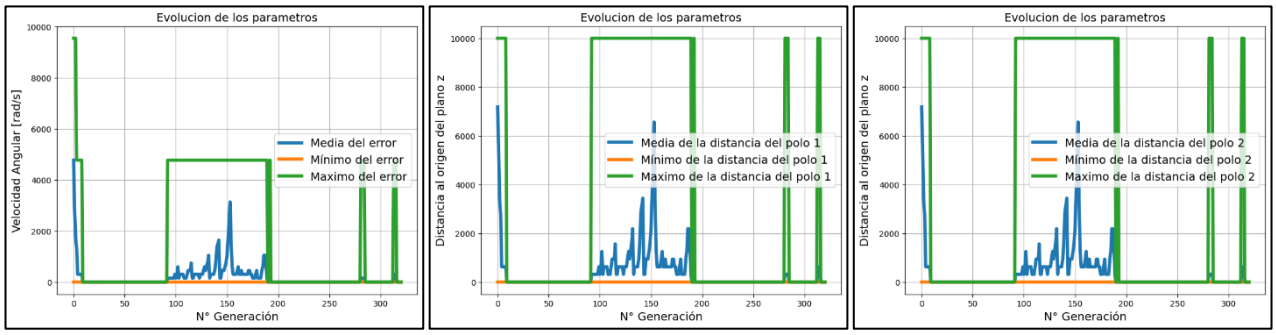

Figura 9: Evolución de los resultados en la ejecución del A.G. [4].

En la última figura se puede observar una estrecha relación entre el error estimado y la ubicación de los polos. El error estimado alcanza valores de unos 5000 rad/s cuando los polos no caen dentro del círculo unitario. Por otro lado, observando los valores medios se podría decir que existe una relación lineal entre el error calculado y la distancia de los polos al origen. Por último, se observa que la evolución de la distancia de ambos polos al origen es idéntica por lo que se puede suponer que siempre los polos son complejos conjugados.

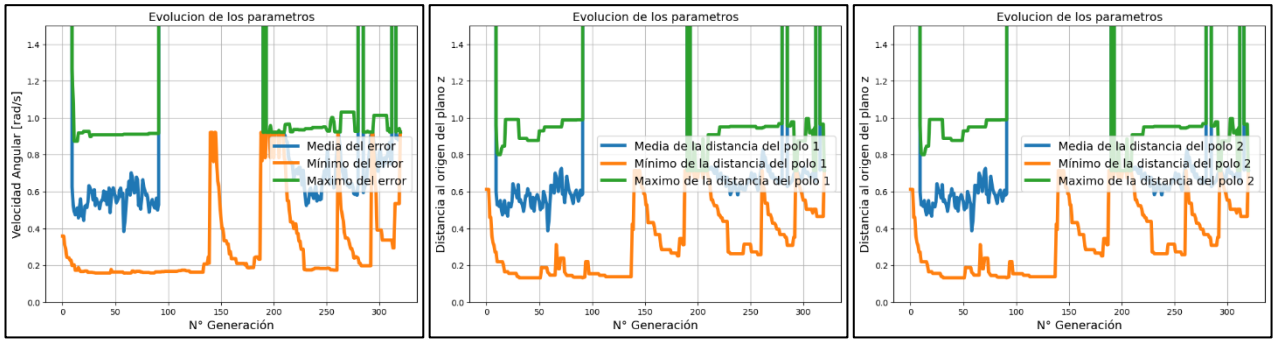

Figura 10: Evolución de los resultados en la ejecución del A.G. Zoom de los mínimos [4].

En la última figura se observan algunas diferencias respecto a los mínimos que toma el error y la distancia a los polos lo que da indicio a que la relación entre estos no es perfectamente lineal y confirma que los polos son complejos conjugados. Por otro lado, se observa un aumento de la distancia mínima de los polos a partir de la generación 140.

Frente de Pareto.

A partir de objeto *pareto* es posible visualizar el frente de Pareto mediante el Script. 15 obteniendo la Figura 11, la Figura 12 y la Figura 13.

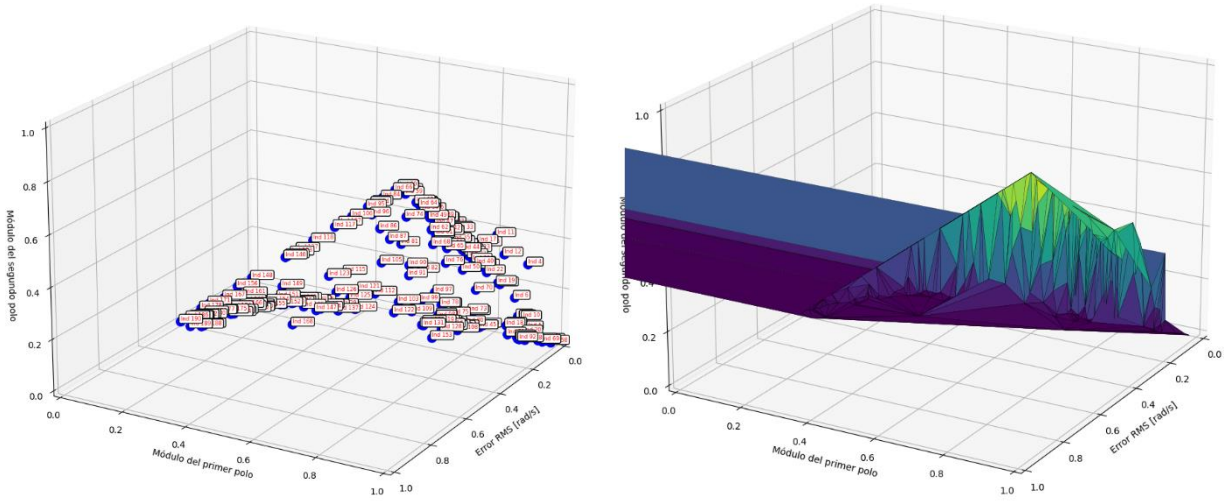

Figura 11: Frente de Pareto a) [4].

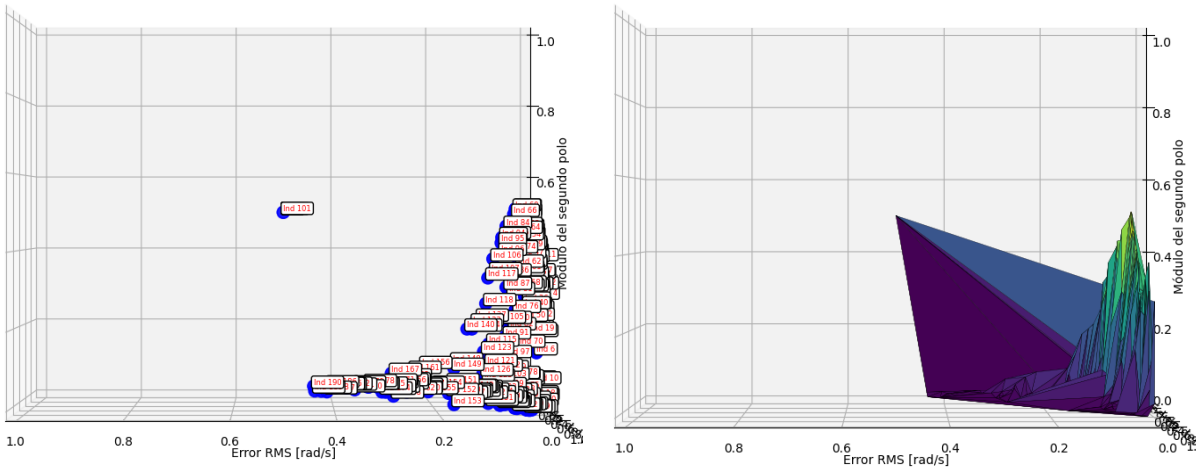

Figura 12: Frente de Pareto b) [4].

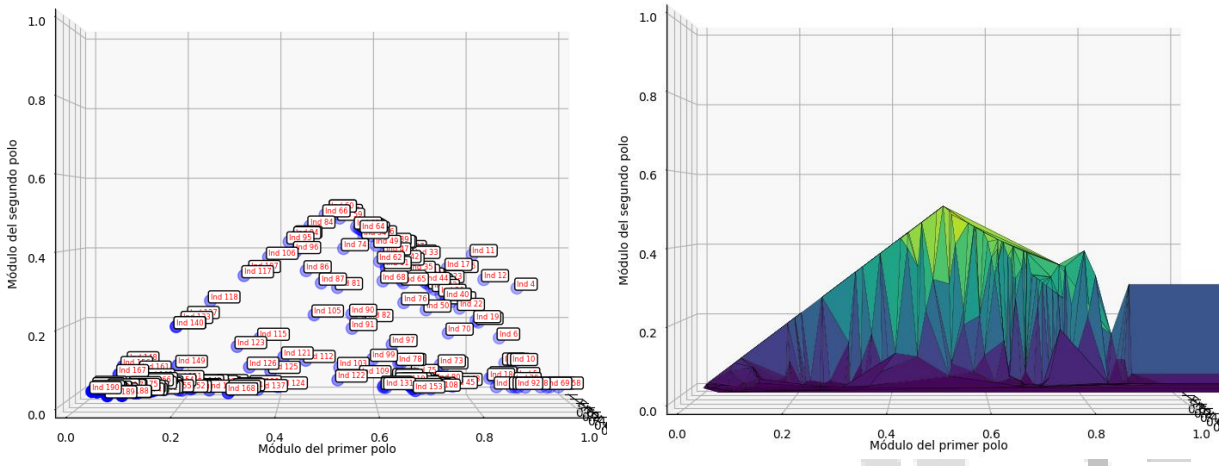

Figura 13: Frente de Pareto C [4].

Las ultimas figuras presentan un frente de Pareto difícil de interpretar, por ello se desarrolló el Script. 16 donde se lo presenta de a parejas. Obteniendo la Figura 14, la Figura 15 y la Figura 16. En la primera de estas pude observarse que el error parece ser inversamente proporcional al módulo del

polo 1 pero no parece ser lo mismo para el polo 2 dado que se observa una compleja relación entre este y el Error. Por último, la relación entre los módulos de los polos también parece ser compleja pero algunos puntos dejan entrever una relación lineal de polos complejos conjugados.

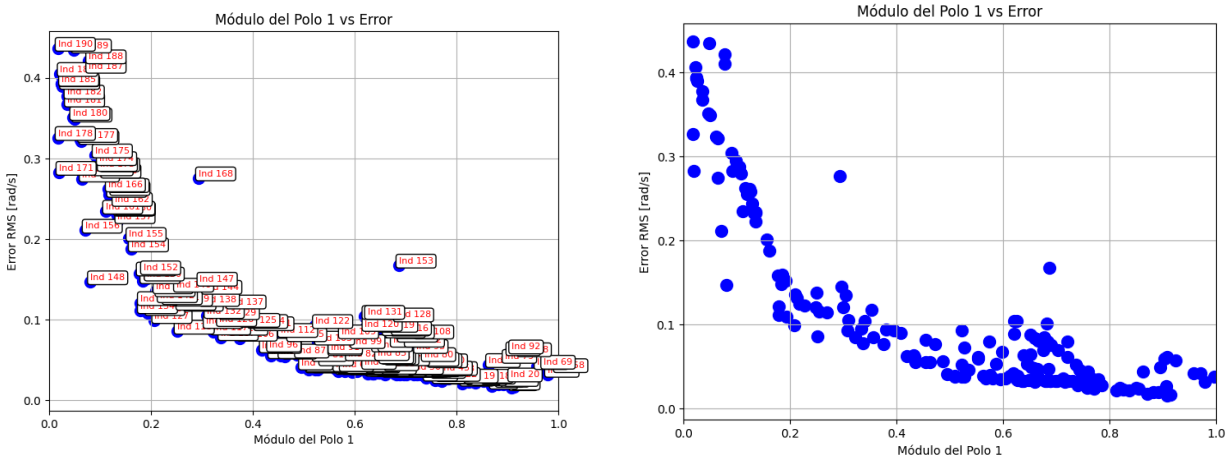

Figura 14: Modulo del Polo 1 vs Error [4].

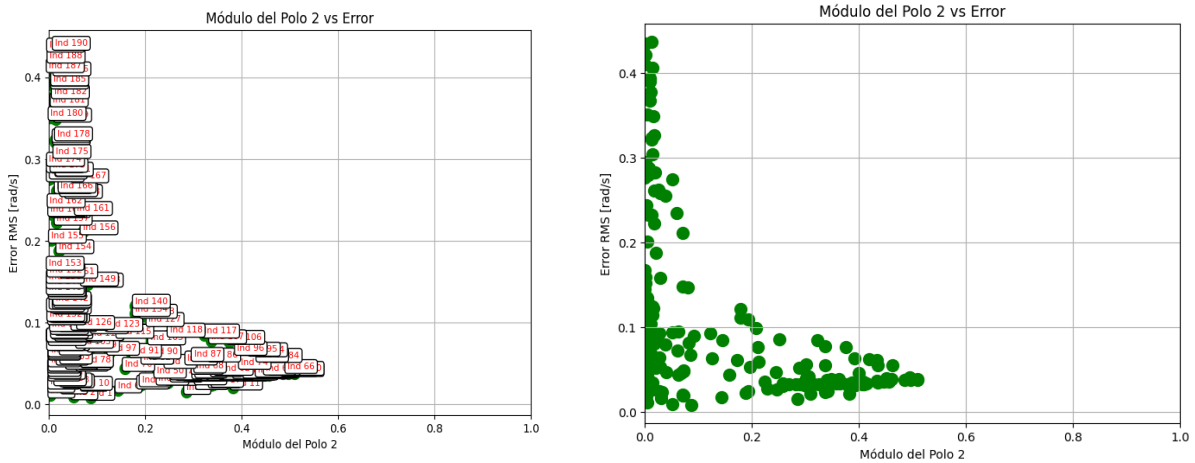

Figura 15: Modulo del Polo2 vs Error [4].

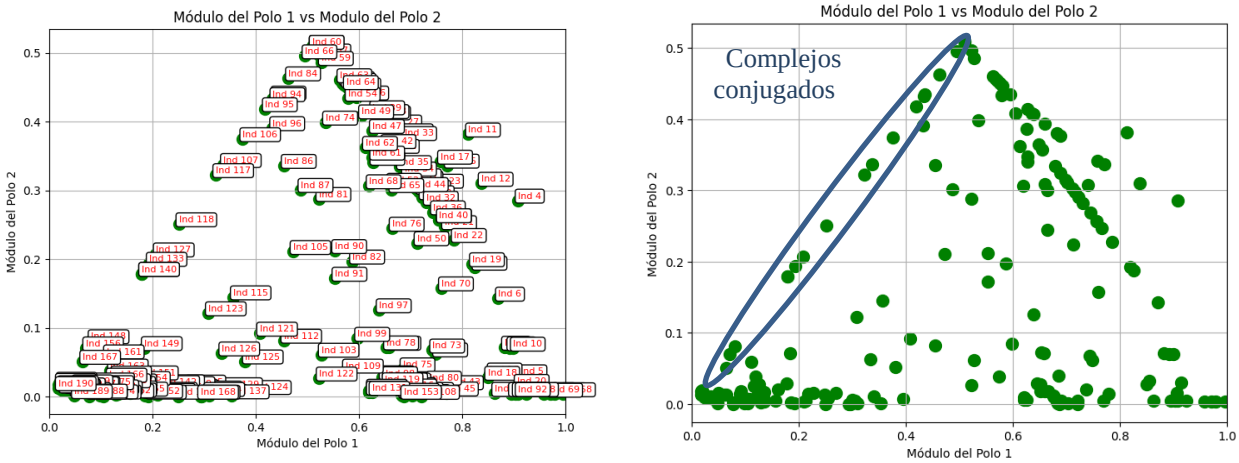

Figura 16: Polo1 vs Polo2 [4].

```
from mpl_toolkits.mplot3d import Axes3D
fitness_values = [ind.fitness.values for ind in pareto]# Obtener las funciones objetivo del frente de Pareto
fitness_1 = [values[0] for values in fitness_values]# Separar las funciones objetivo en listas separadas
fitness_2 = [values[1] for values in fitness_values]
fitness_3 = [values[2] for values in fitness_values]
elev=(20,0,0)
azim=(30,90,0)
for j in range(3):
  fig = plt.figure(i,figsize=(16, 12))# Crear el diagrama de Pareto tridimensional
  ax = fig.add_subplot(111, projection='3d')
  for i, (x, y, z) in enumerate(zip(fitness_1, fitness_2, fitness_3)):# Etiquetar los puntos con números de identificación
    ax.text(x, y, z, f'Ind {i + 1}', color='red', fontsize=6, ha='left', va='bottom', bbox=dict(boxstyle='round,pad=0.3',
edgecolor='black', facecolor='white'))
  scatter = ax.scatter(fitness_1, fitness_2, fitness_3, s=100, c='blue', marker='o', label='Frente de Pareto')
  ax.set_xlabel('Error RMS [rad/s]')
  ax.set_ylabel('Módulo del primer polo')
  ax.set_zlabel('Módulo del segundo polo')
  ax.set_title('Diagrama de Pareto')
  ax.set_xlim([0, 1])# Establecer límites en los ejes
  ax.set_ylim([0, 1])
  ax.set_zlim([0, 1])
  elev_i=elev[j]
  azim_i=azim[j]
  ax.view_init(elev_i, azim_i)# Ajustar el ángulo de visualización (elevación, azimut)
  ax.legend()# Agregar leyenda
  plt.show()# Mostrar el gráfico
```

Script. 15: Frente de Pareto [4].

```
  etiquetas = [f'Ind {i + 1}' for i in range(len(pareto))]
  cont=0
  def mostrar_grafica(fitness_x, fitness_y, color, titulo,i):
      plt.figure(figsize=(8, 6))
      plt.scatter(fitness_x, fitness_y, c=color, marker='o', s=100)
      for etiqueta, x, y in zip(etiquetas, fitness_x, fitness_y):
          plt.text(x, y, etiqueta, color='red', fontsize=8, ha='left', va='bottom', bbox=dict(boxstyle='round,pad=0.3',
edgecolor='black', facecolor='white'))
      if i==0:
        plt.xlabel('Módulo del Polo 1')
        plt.ylabel('Error RMS [rad/s]')
      if i==1:
        plt.xlabel('Módulo del Polo 2')
        plt.ylabel('Error RMS [rad/s]')
      if i==2:
        plt.ylabel('Módulo del Polo 2')
        plt.xlabel('Módulo del Polo 1')
      plt.title(titulo)
      plt.grid(True)
      plt.xlim([0, 1])
      plt.savefig(f'{titulo.replace(" ", "_").lower()}.png')
  mostrar_grafica(fitness_2, fitness_1, 'blue', 'Módulo del Polo 1 vs Error',cont)
  cont=cont+1
  mostrar_grafica(fitness_3, fitness_1, 'green', 'Módulo del Polo 2 vs Error',cont)
  cont=cont+1
  mostrar_grafica(fitness_2, fitness_3, 'green', 'Módulo del Polo 1 vs Modulo del Polo 2',cont)
```

Script. 16: Frente de Pareto de a parejas [4].

Mejor individuo

Dada las complejas relaciones entre los objetivos se buscó el individuo que presentara el menor valor de error, con sus polos dentro del círculo unitario y con un margen de fase positivo. Esto también se lo realizo con un script, pero dado lo largo que es el mismo y como simplemente es una búsqueda, no se adjunta al documento de forma explícita. De todas formas, se brinda acceso a todo el código donde si aparece esta parte. Dicho esto, pasamos directamente a los resultados obtenidos con el mejor individuo del frente de Pareto, estos son los que se presentan en la Figura 17.

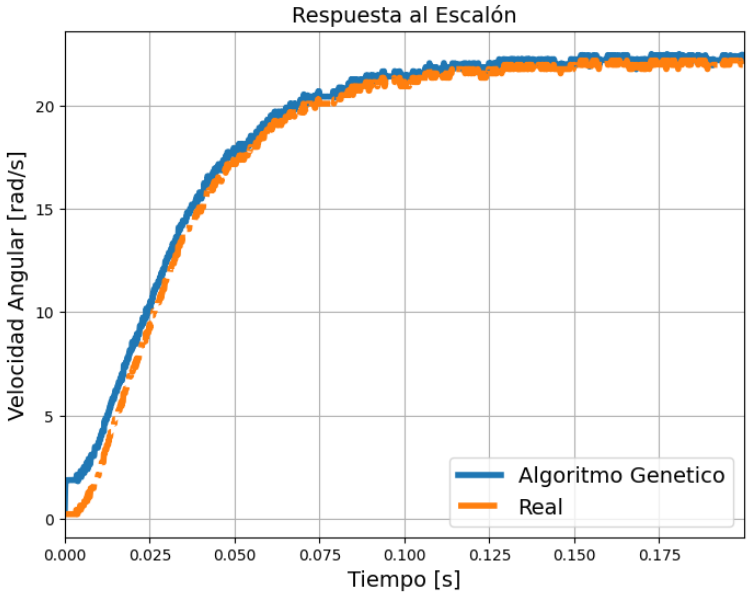

Figura 17: Comparación de salidas [4].

En la última figura podemos observar como la señal estimada con los coeficientes del algoritmo genético es muy similar a la salida real del sistema. Aunque al comienzo se puede observar como la salida estimada con el A.G. es levemente superior a la real. Los valores de los coeficientes son los que se presentan en la (29).

	$n_{2AG} = 0{,}03487;\ n_{1AG} = 0{,}06973;\ n_{0AG} = 0{,}03487$ $d_{1AG} = -1{,}19362; d_{0AG} = 0{,}25924$	**(29)**

Análisis de resultados.

Método clásico

Los valores presentados en la (29) se pueden obtener también del método clásico que se utilizó en el laboratorio teniendo en cuenta la (14) y la (28). Los resultados obtenidos del mismo son los que se presentan en la (30), todos ellos fueron obtenidos mediante el Script. 17.

$$n_{2SM} = 0{,}000027;\ n_{1SM} = 0{,}000054;\ n_{0SM} = 0{,}000027; \qquad \textbf{(30)}$$
$$d_{1SM} = -1{,}98; d_{0SM} = -0.98$$

```
#---------- Metodo Stark – Mollenkamp----------------------------------
#----------Calculo de coeficientes discretos-----------------------------
a2=Kp
a1=2*Kp
a0=Kp
b2=(4/(Ts**2))*(t1*t2)+(2/Ts)*(t1+t2)+(1)
b1=-(8/(Ts**2))*(t1*t2)+2*(1)
b0=(4/(Ts**2))*(t1*t2)-(2/Ts)*(t1+t2)+(1)
#--------Calculo de parametros de la ecuación recursiva----------------------
n2sm=a2/b2
n1sm=a1/b2
n0sm=a0/b2
d1sm=b1/b2
d0sm=b0/b2
print("El coeficiente n2sm es: ", n2sm)
print("El coeficiente n1sm es: ", n1sm)
print("El coeficiente n0sm es: ", n0sm)
print("El coeficiente d1sm es: ", d1sm)
print("El coeficiente d0sm es: ", d0sm)
#-------Calculo de la salida-----------------------------------------------
w_sm=np.zeros(N)
e_sm=np.zeros(N)
for k in range(N):
  if k==0:
    w_sm[k]=n2sm*u[k]
  if k==1:
    w_sm[k]=n2sm*u[k]+n1sm*u[k-1]-d1sm*w_sm[k-1]
  if k>1:
    w_sm[k]=n2sm*u[k]+n1sm*u[k-1]+n0sm*u[k-2]-d1sm*w_sm[k-1]-d0sm*w_sm[k-2]
  e_sm[k]=w[k]-w_sm[k]
```

Script. 17: Salida estimada con el método Stark – Mollenkamp [4].

Observando la (30) respecto (29) se puede notar una gran diferencia. Por ello y ante la duda de haber implementado mal el método clásico, se desarrolló el Script. 18 para observar en una misma grafica las diferentes salidas del sistema. Esto es lo que se observa en la Figura 18.

```
# Graficar la respuesta al escalón de la función de transferencia estimada
plt.figure(figsize=(8, 6))
plt.plot(t, w_ag, label='Algoritmo Genético', linewidth=4)
plt.plot(t, w, label='Real', linewidth=4,linestyle='-.')
plt.plot(t,w_sm, label='Stark – Mollenkamp', linewidth=4,linestyle='--')
plt.xlabel('Tiempo [s]', fontsize=14)
plt.ylabel('Velocidad Angular [rad/s]', fontsize=14)
plt.title('Respuesta al Escalón', fontsize=14)
plt.xlim(0, max(t))
plt.legend(loc='lower right', fontsize=14)
plt.grid(True)
plt.show()
```

Script. 18: Grafica de las salidas [4].

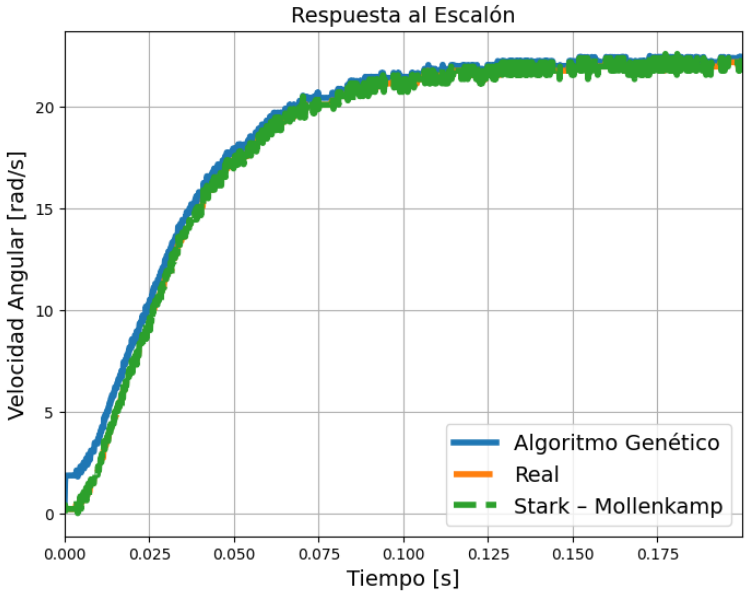

Figura 18: Comparación de las salidas [4].

En la última figura se puede observar como a pesar de que los parámetros tengan valores muy diferentes logran representar muy bien la salida real del motor de CC.

Correlaciones y graficas de dispersión

Se procedió a realizar los cálculos de las correlaciones de la salida estimada por el A.G. respecto de la salida real y de forma análoga para la salida estimada con el método clásico. Los resultados de esto se presentan de forma ordenada en la Tabla 2. Además, se realizan las respectivas gráficas de dispersión que se presentan en la Figura 19. Todo esto es obtenido mediante el Script. 19.

```
from scipy.stats import pearsonr
# Calcular el coeficiente de correlación de Pearson
cor_w_w_ag, _ = pearsonr(w, w_ag)
cor_w_w_sm, _ = pearsonr(w, w_sm)
# Graficas de dispersión lado a lado
fig, axs = plt.subplots(1, 2, figsize=(16, 6))  # 1 fila, 2 columnas
# Primer gráfico de dispersión
axs[0].scatter(w, w_ag, s=20, color='blue', marker='.')
axs[0].set_xlabel('Repuesta real [rad/s]', fontsize=14)
axs[0].set_ylabel('Respuesta estimada [rad/s]', fontsize=14)
axs[0].set_title('Dispersión entre la salida real y la estimada por el A.G.', fontsize=14)
axs[0].grid(True)
# Segundo gráfico de dispersión
axs[1].scatter(w, w_sm, s=20, color='blue', marker='.')
axs[1].set_xlabel('Repuesta real [rad/s]', fontsize=14)
axs[1].set_ylabel('Respuesta estimada [rad/s]', fontsize=14)
axs[1].set_title('Dispersión entre la salida real y la estimada por el método clásico', fontsize=14)
axs[1].grid(True)
plt.show()
```

Script. 19: Correlación y gráficos de dispersión [4].

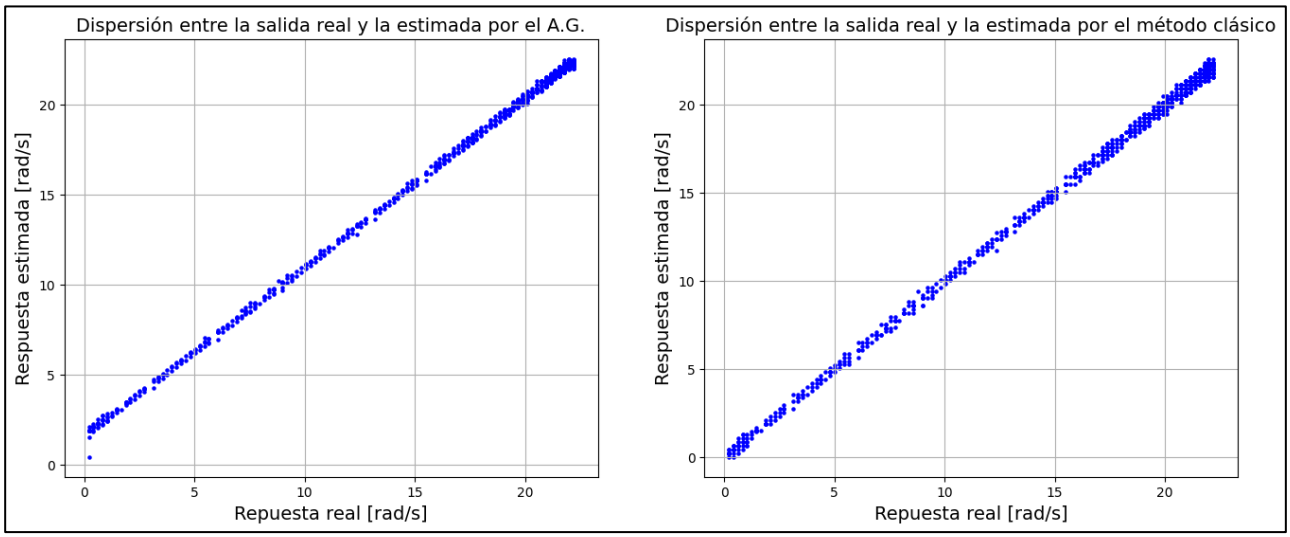

Figura 19: Comparación de las salidas [4].

En la última figura se observa como el método clásico logra dibujar una línea recta de pendiente unitaria lo que indica una correlación prácticamente perfecta entre ambas señales. Aunque, la del algoritmo genético también dibuja una línea recta, pero tiene una pendiente menor a 1 y parece tener un offset.

Con la ayuda de la librería *control* se realizaron los diagramas de bode de ambos sistemas que es lo que se puede observar en la Figura 20.

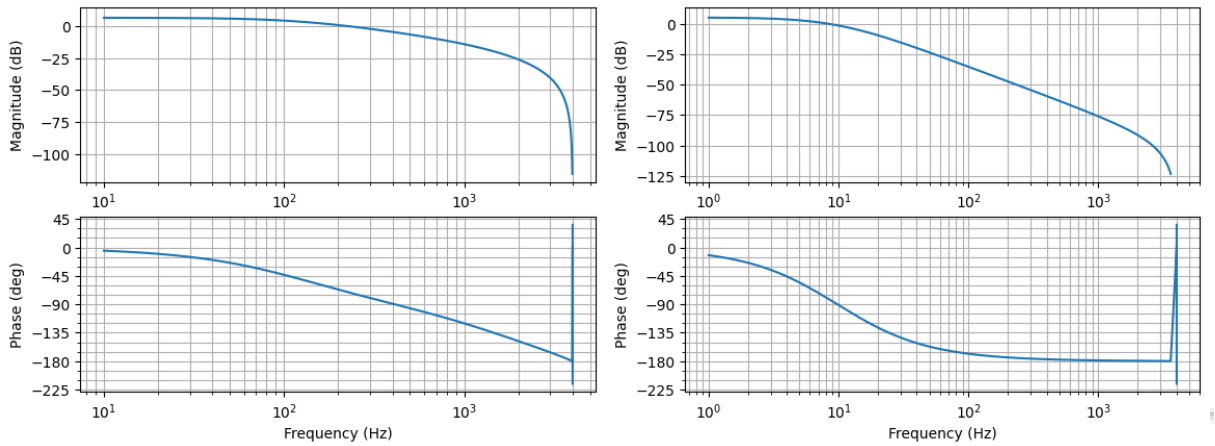

Figura 20: Diagramas de bode. Izquierda A.G. Derecha Método clásico [4].

Con el método *margin* de la última librería es posible obtener directamente los márgenes de ganancia de fase de los sistemas. El primero de estos se puede observar en la gráfica que son prácticamente infinitos, en cambio, el margen de fase del A.G. es superior al del método clásico. El método clásico tiene un margen de fase de 97,28 ° y el del algoritmo genético es de 109,38 °.

Errores y comparativa final.

Para concluir con el análisis, se calcularon las raíces de los errores cuadráticos medios de las señales de salida estimadas y la señal de salida real junto con los errores relativos porcentuales entre los coeficientes obtenidos entre los dos métodos considerando que los del método clásico eran los valores reales. Todos estos se calculan en el Script. 20 y se presentan de forma ordenada en la Tabla 2.

```
#Cálculo de los errores RMS
E_ag=e_rms_pareto_best
E_sm=np.sqrt(np.sum(e_sm**2))/N
#Errores de los coeficientes
num = np.array([n2_ag_best, n1_ag_best, n0_ag_best])
den = np.array([1, d1_ag_best, d0_ag_best])
e_n2=((abs(abs(n2sm)-abs(n2_ag_best)))/(abs(n2sm)))*100
e_n1=((abs(abs(n1sm)-abs(n1_ag_best)))/(abs(n1sm)))*100
e_n0=((abs(abs(n0sm)-abs(n0_ag_best)))/(abs(n0sm)))*100
e_d1=((abs(abs(d1sm)-abs(d1_ag_best)))/(abs(d1sm)))*100
e_d0=((abs(abs(d0sm)-abs(d0_ag_best)))/(abs(d0sm)))*100
# Se crea un dataframe
e_df=pd.DataFrame({
    'Correlación: Real-Método Clásico': [cor_w_w_sm],
    'Correlación: Real-A.G.': [cor_w_w_ag],
    'Error RMS con el Método Clásico [rad/s]': [E_sm],
    'Error RMS con el Filtro Adaptativo [rad/s]': [E_ag],
    'Error en el coeficiente n2 [%]' : [e_n2],
    'Error en el coeficiente n1 [%]': [e_n1],
    'Error en el coeficiente n0 [%]': [e_n0],
    'Error en el coeficiente d1 [%]': [e_d1],
    'Error en el coeficiente d0 [%]': [e_d0]
     })
e_df.head()
# Para descargarlo como una tabla de excel
#e_df.to_excel('e_df.xlsx', index=False)
```

Script. 20: Resumen de resultados [4].

Tabla 2: Resumen de resultados [4].

Correlación: Real-Método Clásico	Correlación: A.G.	Error RMS con el Método Clásico [rad/s]	Error RMS con el A.G. [rad/s]	Error en el coeficiente n2 [%]	Error en el coeficiente n1 [%]	Error en el coeficiente n0 [%]	Error en el coeficiente d1 [%]	Error en el coeficiente d0 [%]
0,9994	0,9997	0,005190	0,015515	129392,71	129392,71	129392,71	39,84	73,57

En la última tabla se observan varias cuestiones. Primero, ambas señales estimadas son prácticamente idénticas a la señal real dado los coeficientes de correlación que se obtuvieron. Segundo y en línea con lo anterior, los errores RMS obtenidos son despreciables. Por último, existe una enorme diferencia entre los coeficientes obtenidos con los distintos métodos, pero ambos lograr modelar bien la salida del sistema.

Conclusión.

Con la realización de este trabajo he llegado a las siguientes conclusiones:

- Se logró diseñar un algoritmo genético para modelar un sistema del tipo caja gris dado que la salida real del sistema y la estimada por los coeficientes del A.G. tienen una correlación prácticamente perfecta y un error despreciable.
- A pesar de la gran diferencia entre los valores de los coeficientes del A.G. y del método clásico, ambos métodos logran una correlación prácticamente perfecta con la salida lo que confirma que no existe solo una combinación de valores de los 5 coeficientes que permiten representar al sistema. Es más, debido a la propiedad de la no unicidad de la transformada z inversa se sabe que hay infinitas soluciones.
- Con el A.G. fue posible hallar una solución con un mayor margen de fase respecto al del método clásico por lo que se podría decir que el mismo converge a una de las mejores soluciones de las infinitas que existen.

Bibliografía

[1] «Wikipedia,» 09 12 2023. [En línea]. Available: https://es.wikipedia.org/wiki/Motor_de_corriente_continua.

[2] D. I. F. Botterón, *Obtención del modelo de un motor de CC con exitación independiente en base a ensayos experimentales*, OBERÁ, 2011.

[3] «Wikipedia,» 05 1 2023. [En línea]. Available: https://es.wikipedia.org/wiki/Transformaci%C3%B3n_bilineal. [Último acceso: 09 12 2023].

[4] B. J. Jose, «IA_Final_Berger_Juan_Jose,» 22 12 2023. [En línea]. Available: https://colab.research.google.com/drive/167IO_iiyQ3IgitR4N0C1oeWpIh4jgush?usp=sharing. [Último acceso: 22 12 2023].

[5] D. G. Reina, A. T. Córdoba y Á. Rodríguez del Nozal, Algoritmos Genéticos con PYTHON, 2020.