Optimización de un Controlador PID con un Algoritmo Bioinspirado: Evolución Diferencial

Sophia Upegui Robledo
Ingeniería Mecatrónica
Escuela de Ingeniería
Universidad EIA
Envigado, Colombia
sophia.upegui@eia.edu.co

***Resumen*—El presente informe describe la optimización de un controlador PID discreto mediante el algoritmo bioinspirado de Evolución Diferencial (DE), una metaheurística basada en principios evolutivos como mutación, cruce y selección. Se modela un sistema dinámico con una función de transferencia de primer orden con retardo, discretizándolo con un tiempo de muestreo adecuado para garantizar estabilidad y precisión. El algoritmo DE se implementa en MATLAB para sintonizar los parámetros del controlador (q_0, q_1, q_2), minimizando una función objetivo que pondera el tiempo de establecimiento y el sobrepaso. Los resultados muestran una mejora significativa en el desempeño del controlador optimizado respecto al controlador inicial. La convergencia rápida del algoritmo evidencia su eficacia para encontrar soluciones óptimas en problemas de control, equilibrando exploración y explotación del espacio de búsqueda.**

I. Introducción

Los controladores Proporcional-Integral-Derivativo (PID) han prevalecido como una de las estrategias de control más implementadas en entornos industriales debido a su simplicidad conceptual y robustez operativa; sin embargo, su desempeño depende en gran medida de la adecuada sintonización de sus parámetros. El ajuste de q_0, q_1 y q_2 constituye un problema de optimización complejo al cual la metaheurística se propone a dar solución.

Mediante la implementación de algoritmos bioinspirados, estas técnicas logran explorar espacios de búsqueda extensos. Esto a su vez les permite obtener suficientemente soluciones satisfactorias en tiempos de cómputo razonables sin la necesidad de garantizar la obtención del óptimo global.

En este contexto, la Evolución Diferencial (DE) ha emergido como una de las metaheurísticas más versátiles dentro del paradigma de la computación evolutiva, demostrando particular eficacia en la optimización de controladores PID para sistemas dinámicos.

II. Fundamentos del algoritmo bioinspirado

El algoritmo de evolución diferencial (ED) fue desarrollado en 1995 por los investigadores Kenneth Price, un matemático e ingeniero especializado en inteligencia artificial y optimización computacional, y Rainer Storn, un ingeniero alemán experto en optimización y procesamiento de señales.

Mientras trabajaban en problemas de optimización en los Laboratorios Bell, Storn y Price experimentaron con operadores de mutación basados en combinaciones vectoriales. Su trabajo conjunto surgió de la búsqueda de métodos eficaces para encontrar el mejor valor en funciones altamente no lineales con múltiples mínimos y máximos locales. Inspirado en las teorías de Darwin y Lamark sobre la evolución y la supervivencia del más apto, DE surgió como un método alternativo más eficiente y simple que los algoritmos genéticos tradicionales, los cuales requerían ajustes complejos y a menudo convergían lentamente. Su propuesta de mutar soluciones mediante la diferencia ponderada entre individuos de la población permitió obtener una exploración más efectiva del espacio de búsqueda sin depender excesivamente del cruce o la selección tradicional.

Asimismo, el algoritmo ED traslada la variación genética y los principios evolutivos para replicar esta dinámica al ámbito computacional. Al igual que en los mecanismos biológicos de la selección natural y la evolución de una especia, el algoritmo replica procesos fundamentales como mutación, cruce y selección. La mutación simula la modificación genética, donde las soluciones iniciales experimentan pequeñas alteraciones para explorar nuevas posibilidades. El cruce representa la mezcla de características de diferentes soluciones para crear nuevas combinaciones, análogo a la reproducción genética donde los genes de dos individuos se combinan para generar descendencia con características mixtas. Finalmente, se puede observar el proceso de la selección. En la naturaleza, los organismos más adaptados tienen más probabilidades de sobrevivir y generar descendencia; en ED, las mejores soluciones (las que minimizan la función de costo) son seleccionadas para formar la nueva generación.

El algoritmo de evolución diferencial fue publicado por primera vez por Storn y Price en 1997 en un artículo titulado: "Differential Evolution – A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces."

III. Explicación de la metaheurística

III-A. Procedimiento

El proceso comienza con una fase de modelado del sistema mediante una función de transferencia.

$$G(s) = \frac{0{,}8e^{-2{,}1s}}{125s + 1} \tag{1}$$

Con base a este, se determina el tiempo de muestreo idóneo para discretizar el modelo. Se determina que se quiere emplear el menor tiempo de muestreo posible, que no inestabilice el sistema y permita una representación fiel del modelo. Para esto, se comparan los tiempos de muestreo obtenidos mediante el uso del criterio del tau equivalente y el criterio de tiempo de establecimiento. Entre estos dos, se selecciona el menor y se compara con el valor del retraso (Θ). Si el valor obtenido por estos criterios es menor, se utiliza como el tiempo de muestreo; si es mayor, se determina el tiempo de muestreo como la mitad del retraso, puesto a que el tiempo de muestreo tiene que poder representar con precisión el retraso del sistema.

Teniendo esto, se inicializan los parámetros característicos del algoritmo ED y se genera una población inicial aleatoria dentro del rango de LB y UB. Esta representa los vectores de soluciones posibles, en este caso, combinaciones posibles de q_0, q_1 y q_2. Cada individuo de esta población es analizado empleando la función objetivo para determinar las soluciones de menor costo y así generar el controlador PID inicial.

Seguidamente, se procede al mecanismo iterativo que representa el ciclo evolutivo del algoritmo. Primero, se selecciona de forma ordenada por iteración el individuo de la población que se va a mutar (X_i). En adición a este, se seleccionan aleatoriamente tres otros individuos ($R1$, $R2$, $R3$) que formen parte de la población y sean distintos entre ellos y al individuo de referencia. Estos serán los valores que serán utilizados para crear la mutación del individuo de referencia. La mutación se realiza aplicando el factor de escala (F) a la diferencia entre dos de estos individuos, creando un vector mutado V:

$$V = R1 + F * (R2–R3) \tag{2}$$

Posteriormente, se aplica el cruce, donde cada componente de la solución original (X_i) es combinada con la solución mutada (V) para generar una nueva solución candidata (U). La tasa de cruce (CR) actúa como un parámetro probabilístico que determina la proporción de componentes que serán heredados del vector mutado, mientras que un índice (j_{rand}) aleatorio predeterminado en el rango [1,N] garantiza la introducción de al menos una modificación en el vector de prueba. Se hace uso de un aleatorio que este uniformemente distribuido y este se compara con CR tal que:

$$U = \begin{cases} V & \text{si } (rand_j[0,1] \leq CR) \text{ o } (j = j_{rand}), \\ X_i & \text{en otro caso;} \end{cases} \tag{3}$$

El índice j representa la posición específica de cada parámetro dentro del vector de solución. Para un problema de sintonización de controlador PID, j adquiere un significado concreto al identificar cada uno de los parámetros del controlador: cuando j es 1, corresponde a la ganancia proporcional (q_0); cuando j es 2, representa la ganancia integral (q_1); y cuando j es 3, indica la ganancia derivativa (q_2). Este índice permite al algoritmo realizar un proceso de selección preciso y sistemático, donde cada componente del vector puede ser potencialmente modificado según una probabilidad predefinida, asegurando así una exploración metódica del espacio de soluciones. La estrategia garantiza que, durante cada iteración, se introduzca la suficiente variabilidad para permitir la evolución del algoritmo.

Después del cruce, la nueva solución U es evaluada con la misma función objetivo utilizada para la población inicial. Si el costo de la nueva solución es menor que el del individuo original, se reemplaza en la población. Este proceso de selección garantiza que las soluciones evolucionen progresivamente hacia un mejor desempeño, evitando degradaciones en la calidad de la población. A lo largo de varias iteraciones, el algoritmo converge hacia una solución óptima, minimizando la función de costo y mejorando el desempeño del controlador.

III-B. Parámetros

El algoritmo opera mediante un conjunto de parámetros críticos que determinan su comportamiento. Los parámetros de control utilizados en el DE están estrechamente vinculados al caso específico que se está analizando, lo cual afecta directamente su desempeño.

- La dimensión (D) representa el número de variables a optimizar, definiendo la complejidad del problema. Para un controlador PID discreto, D será típicamente 3, correspondiente a los parámetros q_0, q_1 y q_2.
- El tamaño de población (N) representa el número de individuos (soluciones candidatas) en la población determinada. Un tamaño poblacional mayor permite una exploración más comprehensiva del espacio de búsqueda - reduciendo la probabilidad de quedar atrapado en mínimos locales - mientras que un tamaño de población menor puede llevar a una convergencia prematura. Para problemas sencillos, se recomienda N $\approx$ 5D, mientras que para problemas complejos, N puede aproximarse a 10D. En este caso, se utiliza un tamaño de población de 20.
- La tasa de cruce (CR) controla la tasa de mezcla entre el vector mutante y el vector objetivo. Este parámetro determina la probabilidad de que una solución hija herede componentes del vector mutado. Valores altos ($CR >$ 0,9) promueven la exploración (mayor influencia del vector mutante), mientras que valores bajos ($CR <$ 0,5) favorecen la explotación (se preserva más el vector original) y se recomienda si el caso particular requiere más estabilidad y menos cambios bruscos. Un valor típico recomendado es aproximadamente 0.9, aunque problemas altamente no lineales pueden beneficiarse de valores entre 0.7 y 0.8.
- El factor de escala (F) controla la magnitud de cambios durante la mutación. Se usa para escalar la diferencia entre dos individuos y generar variabilidad en la población. Oscilando entre 0 y 2, valores recomendados se encuentran entre 0.9 y 1.2. Un F alto genera cambios significativos que favorecen la exploración, mientras que un F bajo produce modificaciones más discretas, privilegiando la explotación local. Una selección adecuada de este factor mejor la precisión de la solución y facilita

la evasión de óptimos locales. Para esto, se encuentran dos maneras principales: F puede ser un valor fijo, con el objetivo de equilibrar explotación y exploración; sin embargo, para promover la diversificación, F se define como un aleatorio con una distribución de probabilidades, generalmente una distribución uniforme. Se determinan los valores F_{min} y F_{max} para restringir este valor.

- Los límites inferior y superior de búsqueda (UB, LB) acotan el espacio de búsqueda para cada variable (LB $\leq$ q_i $\leq$ UB). Estos representan las restricciones del problema para evitar que el algoritmo explore valores no realistas o físicamente imposibles y mantienen la búsqueda dentro de un dominio válido. Límites muy estrechos pueden restringir la exploración, mientras que límites amplios exigen más iteraciones para converger.

III-C. Función objetivo

La función objetivo para optimizar el controlador PID evalúa su desempeño mediante métricas de estabilidad y respuesta del sistema. Su propósito fundamental es minimizar el tiempo de establecimiento y el sobrepaso, penalizando simultáneamente soluciones que generen inestabilidad sistémica.

La función objetivo evalua el desempeño del controlador PID mediante un costo que cuantifica su estabilidad y respuesta dinámica. Recibe como entrada los parámetros de sintonización (q_0, q_1 y q_2), el modelo discretizado de la planta (Gz), un peso para priorizar el efecto del tiempo de asentamiento y el tiempo de muestreo (Ts). Primero, construye el controlador PID discreto y calcula la respuesta en lazo cerrado. Luego, extrae métricas clave como el tiempo de asentamiento (SettlingTime) y el sobrepaso (Overshoot). Si el sistema es inestable, asigna un costo infinito para descartar esa solución. En caso contrario, combina ambas métricas en un único valor de costo donde:

$$costo = peso * SettlingTime + Overshoot \tag{4}$$

El objetivo de la optimización es minimizar este costo, equilibrando rapidez (tiempo de asentamiento bajo) y estabilidad (sobrepaso reducido). Así, el algoritmo evolutivo busca los valores óptimos de q_0, q_1 y q_2 que logren un controlador eficiente y robusto.

III-D. Optimización

El algoritmo ED requiere la manipulación estratégica un conjunto de parámetros que gobiernan su comportamiento de búsqueda para poder converger hacia soluciones óptimas. Para esto, se hace uso de:

- Intensificación: La intensificación representa un mecanismo de refinamiento selectivo que busca optimizar la calidad de las soluciones mediante una exploración sistemática de los entornos más prometedores. Este proceso se implementa en el algoritmo ED realiza la intensificación mediante la comparación directa entre los individuos de la población actual ("padres") y los vectores generados U ("hijos"), donde se evalúa meticulosamente el desempeño de cada solución candidata. La selección actúa como un filtro que conserva únicamente las soluciones con mejor desempeño en cada generación, garantizando que cada iteración del algoritmo converja hacia configuraciones cada vez más eficientes. Este proceso de selección competitiva garantiza que la búsqueda se refine progresivamente en regiones prometedoras del espacio de soluciones, manteniendo y mejorando las soluciones óptimas locales encontradas. Al concentrarse en estas regiones con potencial de optimización, el algoritmo permite un refinamiento progresivo y preciso de los resultados.
- Diversificación: La diversificación es un mecanismo clave para evitar el estancamiento del algoritmo y ampliar la búsqueda de soluciones. Se implementa principalmente mediante la mutación, que genera vectores mutantes posicionados deliberadamente fuera de la región actual de individuos seleccionados, introduciendo variabilidad controlada en el espacio de soluciones. Además, la aleatorización del factor de escala (F) permite ajustar dinámicamente la magnitud de estos cambios, garantizando el balance entre exploración (regiones nuevas) y explotación (refinamiento local).
- Aleatorización: La aleatoriedad introduce variabilidad y previene la convergencia prematura hacia soluciones subóptimas. Durante el proceso de generación del vector mutante, el algoritmo selecciona tres individuos de la población de manera completamente aleatoria, garantizando que cada solución candidata tenga exactamente la misma probabilidad de ser elegida y contribuir a la exploración. Este proceso de selección uniforme permite que el algoritmo explore diversas regiones del espacio de soluciones sin privilegiar sistemáticamente ninguna zona, reduciendo significativamente el riesgo de estancamiento en configuraciones no óptimas.
- Probabilidad: El algoritmo emplea modelos probabilísticos para guiar su exploración, destacando la probabilidad de cruce (CR). Esta taza actua como un umbral probabilístico que determina la probabilidad de la herencia de componentes del vector mutante hacia la nueva solución. Este proceso probabilístico permite una exploración sistemática pero no determinista del espacio de soluciones, introduciendo un equilibrio entre la preservación de características parentales y la generación de variantes potencialmente más eficientes.

III-E. Estrategia para escapar de mínimos locales

Uno de los desafíos comunes en este método es la convergencia prematura hacia óptimos locales, especialmente cuando se trabaja con funciones complejas y de muchas dimensiones.

La estrategia para evadir mínimos locales en el Algoritmo de Evolución Diferencial se fundamenta principalmente en la manipulación dinámica del factor de escala (F), un parámetro que controla la magnitud de las modificaciones durante la mutación. Cuando F adquiere valores grandes, el algoritmo genera soluciones significativamente distantes de los individuos parentales, promoviendo una exploración amplia y diversa del

espacio de búsqueda. Conversamente, valores pequeños de F producen soluciones más próximas a los padres, facilitando una explotación más precisa de regiones prometedoras. Esta variabilidad dinámica impide que el algoritmo se estanque en una única región, garantizando una búsqueda flexible y adaptativa que puede escapar eficientemente de configuraciones localmente óptimas pero globalmente subóptimas.

Cuando F toma valores elevados, los vectores mutantes se ubican a mayor distancia de los originales, favoreciendo la exploración de nuevas regiones. Por el contrario, valores más bajos de F permiten un refinamiento local, equilibrando exploración y explotación. Este enfoque asegura que el algoritmo no quede confinado a óptimos locales, manteniendo su capacidad adaptativa para descubrir soluciones óptimas en áreas previamente inexploradas.

IV. Implementación y simulaciones

A continuación, se muestra la implementación del algoritmo en MATLAB para la optimización de los parámetros PID:

```
%% Evolucion Diferencial (DE) para Sintonizacion de un Controlador PID Discreto

clc
close all
clear

%% Modelo del sistema

Tau = 125;
K = 0.8;
Theta = 2.1;

Num = [K];
Den = [Tau 1];
Gs = tf(Num, Den, 'inputdelay', Theta);

% Criterio de tau equivalente
Tau_eq = Tau + Theta/2;
Ts1 = Tau_eq / 10;

% Criterio de tiempo de establecimiento
Gs_lc = feedback(Gs,1)
ts = stepinfo(Gs_lc).SettlingTime; % Step de lazo cerrado
rango = [0.05*ts 0.15*ts] % 0.05*ts < Ts < 0.15*ts
Ts2 = 0.05*ts; % Valor elegido

% Definir tiempo de muestreo
Ts_ponderado = min([Ts1 Ts2]);

% Comparaci n con retardo
if Ts_ponderado < Theta
    Ts = Ts_ponderado;
else
    Ts = Theta/2;
end

% Discretizar modelo
Gz = c2d(Gs, Ts, 'zoh');

%% Inicializaci n de par metros de Evoluci n Diferencial

N = 20;          % Tama o de la poblaci n
D = 3;           % Numero de par metros (q0, q1, q2)
CR = 0.9;        % Tasa de cruce (0 - 1)
F_min = 0.9;     % Minimo valor de factor de escala (min: 0)
F_max = 1.2;     % Maximo valor de factor de escala (max: 2)

% L mites de b squeda para q0, q1, q2
lb = [0.1, 0.01, 0.01];
ub = [10, 5, 5];

% Inicializar poblaci n aleatoria dentro de los l mites
pob = lb + rand(N, D) .* (ub - lb);
pob_inicial = pob;
writematrix(pob_inicial, 'population_results.txt');
% pob = load('population_results.txt') % Se usa para comparar parametros
fitness = zeros(N, 1);

%% Controlador inicial

% Evaluar funci n de costo para la poblaci n inicial
peso = 0.5; % a criterio mio
for i = 1:N
    fitness(i) = objective_function(pob(i, :), Gz, peso, Ts);
end

% Guardar el primer controlador generado y su costo
[first_fitness, first_idx] = min(fitness);
first_solution = pob(first_idx, :); % q0, q1, q2 iniciales

% Discretizar el controlador inicial
Cs_inicial_d = pid(first_solution(1), first_solution(2), first_solution(3), Ts);
Cs_inicial_d = c2d(Cs_inicial_d, Ts, 'zoh');
Control_inicial_d = feedback(Cs_inicial_d * Gz, 1);

% Grafica comparacion de controladores discretos
figure(1)
step(Control_inicial_d)
hold on
title(['Respuesta del controlador PID inicial (Costo = ', num2str(first_fitness), ')'])
grid on

% Obtener datos de la respuesta al escal n
[y_init, t_init] = step(Control_inicial_d);
info_init = stepinfo(Control_inicial_d);
ts_init = info_init.SettlingTime;
os_init = info_init.Overshoot;

% Marcar tiempo de establecimiento y sobreimpulso
plot([ts_init ts_init], [0 1.2*max(y_init)], '--', Color='#A2142F')
plot([0 t_init(end)], [1+os_init/100 1+os_init/100], '--', Color='#D95319')
legend('Controlador inicial', 'Tiempo de establecimiento (ts)','Sobreimpulso (Mp)', Location='southeast')
hold off


%% Evoluci n Diferencial (Iteraciones)

```

```
% Inicializacion
MaxIt = 50; % N mero  m ximo de iteraciones

% Proceso iterativo ED
for iter = 1:MaxIt

    for i = 1:N

        % Seleccionar 3 individuos aleatorios distintos
        idx = randperm(N, D+1);
        idx(idx == i) = [];           % Asegurar que i no est  en la selecci n
        r1 = pob(idx(1), :);
        r2 = pob(idx(2), :);
        r3 = pob(idx(3), :);

        % Factor de escala (mutaci n)
        F = F_min + (F_max - F_min) * rand;

        % Mutaci n
        V = r1 + F * (r2 - r3);
        V = max(min(V, ub), lb);  % Restringir rango

        % Cruce
        j_rand = randi(D); % Asegurar que al menos un valor cambie

        for j = 1:D
            if rand <= CR || j == j_rand
                U(j) = V(j);
            else
                U(j) = pob(i, j);
            end
        end

        % Asegurar que U est  dentro de los l mites
        U = max(min(U, ub), lb); % Restringuir limites

        % Evaluar el nuevo vector de prueba
        new_fitness = objective_function(U, Gz, peso, Ts);

        % Selecci n: Reemplazar si la nueva soluci n es mejor
        if new_fitness < fitness(i)
            pob(i, :) = U;
            fitness(i) = new_fitness;
        end

    end

    % Guardar la mejor soluci n de esta iteraci n
    [best_fitness, best_idx] = min(fitness);
    best_solution = pob(best_idx, :);

    % Guardar el mejor fitness en el historial
    fitness_history(iter) = best_fitness;

    % Mostrar progreso
    fprintf('Iteraci n %d: Mejor fitness = %.5f\n', iter, best_fitness);

end

%% Resultados Finales

fprintf('\nMejor soluci n encontrada: q0 = %.5f, q1 = %.5f, q2 = %.5f', best_solution);
fprintf('\nCosto del mejor controlador: %.5f\n', best_fitness);

% Comparaci n con el primer controlador generado
fprintf('\nPrimer controlador generado:\nq0 = %.5f, q1 = %.5f, q2 = %.5f', first_solution);
fprintf('\nCosto del primer controlador: %.5f\n', first_fitness);

% Discretizar el controlador inicial
Cs_inicial_d = pid(first_solution(1), first_solution(2), first_solution(3), Ts);
Cs_inicial_d = c2d(Cs_inicial_d, Ts, 'zoh');
Control_inicial_d = feedback(Cs_inicial_d * Gz, 1);

% Discretizar el controlador final
Cs_final_d = pid(best_solution(1), best_solution(2), best_solution(3), Ts);
Cs_final_d = c2d(Cs_final_d, Ts, 'zoh');
Control_final_d = feedback(Cs_final_d * Gz, 1);

% Grafica comparacion de controladores discretos
figure(2)
step(Control_final_d)
hold on
title('Controlador inicial vs optimizado (Discreto)')
step(Control_inicial_d)
legend('Controlador final','Controlador inicial', Location='southeast')
hold off

% Gr fica de minimizaci n
figure(3)
plot(1:MaxIt, fitness_history, 'LineWidth', 1.5)
xlabel('Iteraciones');
ylabel('Valor de la funci n de costo');
title('Evoluci n de la funci n de costo');
grid on;

%% Funci n Objetivo (Evaluaci n del Controlador)
function costo = objective_function(q, Gz, peso, Ts)
    q0 = q(1);
    q1 = q(2);
    q2 = q(3);

    % Crear y evaluar el controlador PID
    Cs =  pid(q0, q1, q2, Ts);
    Cs = c2d(Cs, Ts, 'zoh');
    Control = feedback(Cs * Gz, 1);

    % Obtener m tricas de desempe o
    C_aux = stepinfo(Control);

    if isnan(C_aux.SettlingTime) || isnan(C_aux.Overshoot)
        costo = inf; % Sistema es inestable
    else
        costo = peso * C_aux.SettlingTime + C_aux.Overshoot;
    end

end
```

V. Análisis de resultados

V-A. Comparación de la respuesta del controlador

Con base a la población aleatoria generada inicialmente, se hayan las condiciones más optimas para q_0, q_1 y q_2 y se obtiene:

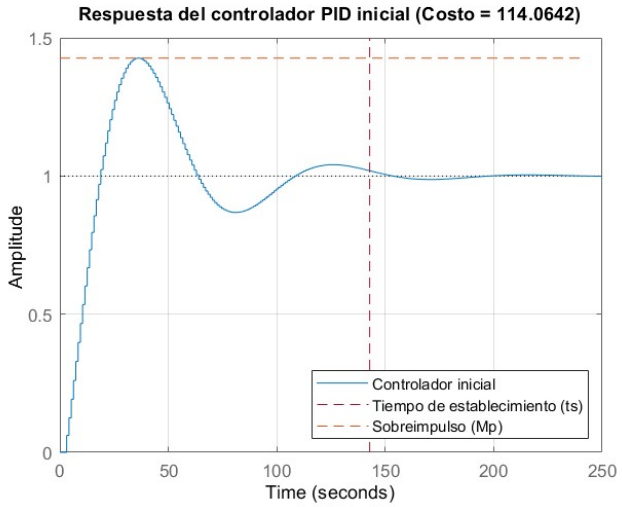

Figura 1: Respuesta al escalón del controlador PID inicial

Con este controlador se obtiene un tiempo de establecimiento de 142.8 segundos y un sobrepico significativo de 42.6642 %. Se puede observar un comportamiento oscilatorio al inicio hasta estabilizarse y eliminar el error de estado estable. El controlador demuestra muestra una respuesta dinámica inestable, con sobrecorrección y oscilaciones que indican una sintonización poco eficiente.

Tras realizar el proceso de optimización, se obtiene el siguiente resultado:

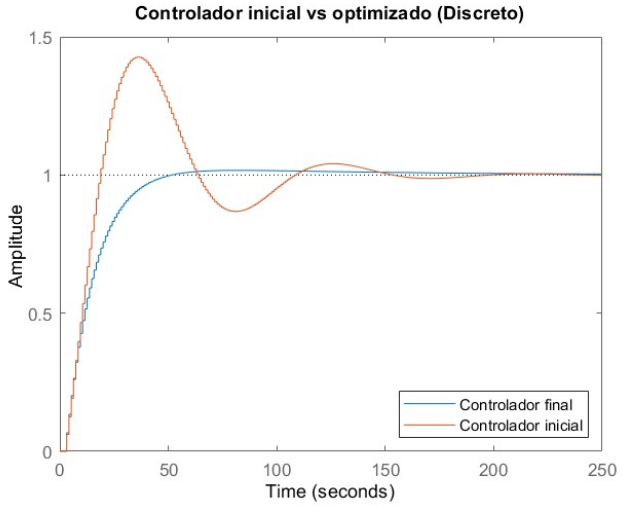

Figura 2: Respuesta al escalón del controlador PID inicial y final

Inicialmente, se puede observar como la respuesta a un escalón unitario del controlador optimizado se da de manera más precisa y se estabiliza con mayor rapidez. Se puede denotar un comportamiento que se asemeja a aquel de un sistema de primer orden, a diferencia del controlador original el cual comparte mayor similitud con uno de segundo orden subamortiguado. Esto se recalca en la disminución significativa del sobreimpulso. En el controlador optimizado, se haya un sobreimpulso de 1.99 % y un tiempo de establecimiento de 42 segundos. Con base a esto, se haya una reducción del 95.34 % del sobrepico y del 70.59 % en el tiempo de establecimiento. Adicionalmente, existe un mejor seguimiento de la referencia (escalón unitario).

Además, se propone analizar la minimización de la función de costo a lo largo de las iteraciones.

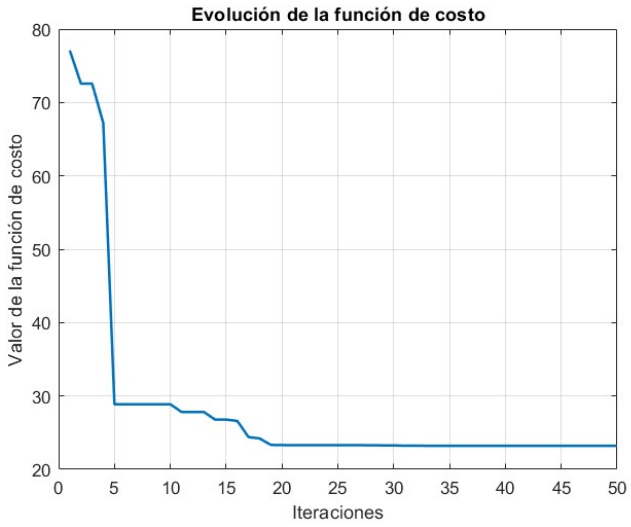

Figura 3: Minimización de la función de costo

La evolución de la función de costo en el tiempo muestra una reducción drástica en las primeras iteraciones, donde el valor del costo disminuye de aproximadamente 75 a 30 en las primeras cinco iteraciones y continúa reduciéndose hasta alcanzar un valor cercano a 5. Este comportamiento indica una rápida convergencia. Hacia las últimas iteraciones, la reducción del costo se estabiliza en un valor cercano a 25, formando una meseta que sugiere que el algoritmo ha encontrado una solución cercana al óptimo local. Este patrón indica que aumentar el número de iteraciones no aportaría mejoras significativas en el resultado, ya que la convergencia ocurre principalmente en las primeras 10-15 iteraciones.

V-B. Ajuste de parámetros

Se realiza el ajuste de los parámetros del algoritmo ED para analizar su comportamiento ante estas modificaciones, en busca de optimizar el desempeño del controlador generado. En cada prueba, se analizaron las respuestas al escalón obtenidas.

V-C. *Primer caso: modificación de CR*

El primer parámetro que se busca variar es la taza de cruce. Como se pudo evidenciar en 3, el algoritmo converge rápidamente, por lo cual se plantea la posibilidad de disminuir el parámetro CR buscando evitar una convergencia prematura hacia soluciones subóptimas. Un valor alto de CR tiende a generar soluciones que heredan demasiada información de los vectores mutados, lo que puede reducir la diversidad de la población y provocar que el algoritmo quede atrapado en óptimos locales. Al disminuir este parámetro, se logra que las soluciones propuestas sean más similares a las anteriores, permitiendo que la búsqueda mantenga una exploración más amplia y controlada del espacio de soluciones. Esta característica es especialmente útil en sistemas sensibles, como los controlados por PID, donde modificaciones bruscas de parámetros pueden generar respuestas inestables o con sobre-oscilaciones. Por tanto, reducir CR contribuye a mejorar la calidad de la búsqueda y la estabilidad de la respuesta final del sistema.

- N = 20
- D = 3
- **CR = 0.3**
- F_{min} = 0.9
- F_{max} = 1.2
- LB = [0.1, 0.01, 0.01]
- UB = [10, 5, 5]

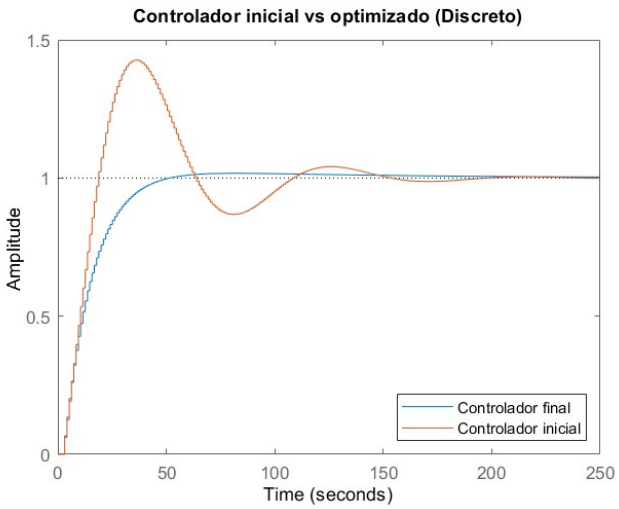

Figura 4: Comparación de controladores con disminución en CR

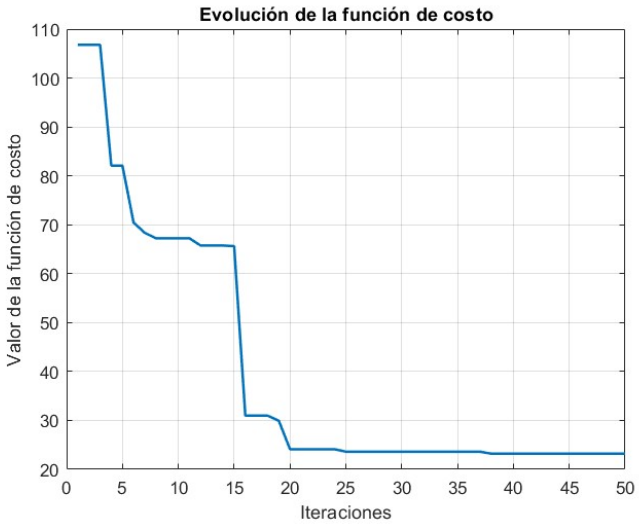

Figura 5: Minimización de la función de costo con disminución en CR

Al comparar la Figura 3 y la Figura 5, se observa que la gráfica original muestra una convergencia inicial muy rápida en las primeras iteraciones (alrededor de las primeras 5), logrando una importante reducción del valor de la función de costo. Sin embargo, después de esta rápida caída, el proceso de optimización se vuelve más lento y presenta pequeñas mejoras graduales, estabilizándose sin alcanzar un valor de costo tan bajo como en la gráfica seleccionada. obtenida tras disminuir el parámetro CR, logra no solo una buena velocidad de convergencia inicial, sino que mantiene una trayectoria más eficiente y sostenida hasta alcanzar un valor mínimo de la función de costo bajo, sin perder el equilibrio entre exploración y explotación, lo que permite evitar estancamientos prematuros.

V-D. *Segundo caso: modificación de F*

Otra estrategia recomendable es explorar una gama más amplia de valores para el parámetro F, tanto inferiores como superiores al rango inicialmente considerado. Esto se debe a que F controla directamente la amplitud de las perturbaciones generadas durante la creación de nuevos individuos, afectando así la capacidad del algoritmo para explorar o explotar el espacio de búsqueda.

Inicialmente, se evalua la respuesta con valores más altos de F. Estos podrían fomentar una exploración más agresiva, permitiendo escapar de óptimos locales y descubrir regiones más prometedoras de la solución.

- N = 20
- D = 3
- CR = 0.9
- **F_{min} = 0.3**
- **F_{max} = 0.7**
- LB = [0.1, 0.01, 0.01]
- UB = [10, 5, 5]

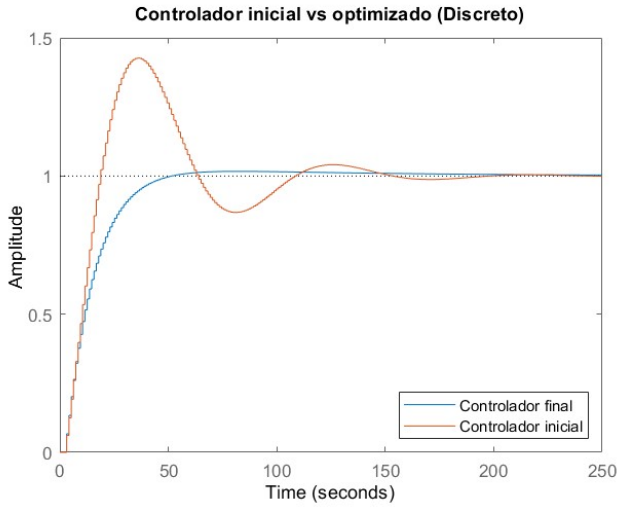

Figura 6: Comparación de controladores con disminución en F

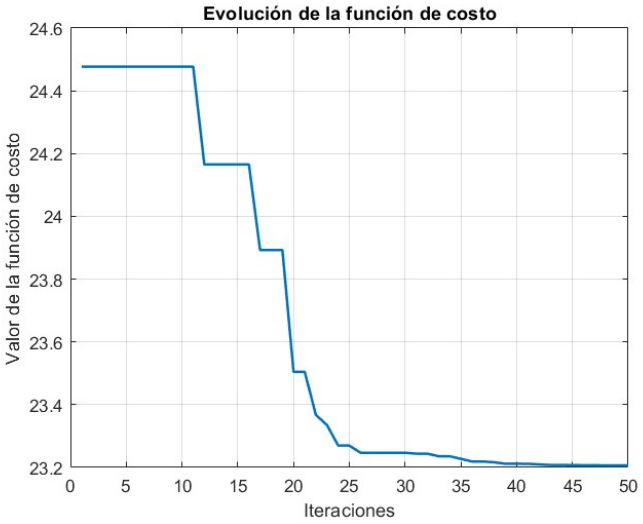

Figura 7: Minimización de la función de costo con disminución en F

Por otro lado, valores más bajos favorecerían una búsqueda más fina y controlada, útil cuando se busca refinar la solución en torno a un mínimo ya identificado.

Parámetros utilizados:

- N = 20
- D = 3
- CR = 0.9
- $\mathbf{F}_{min}$ **= 1.5**
- $\mathbf{F}_{max}$ **= 1.9**
- LB = [0.1, 0.01, 0.01]
- UB = [10, 5, 5]

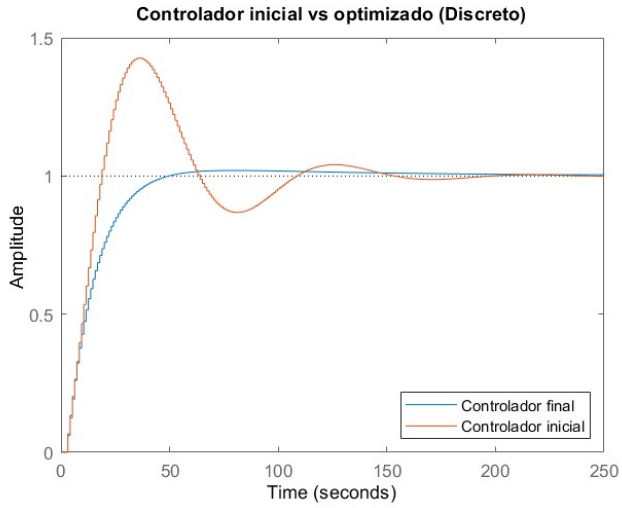

Figura 8: Comparación de controladores con aumento en F

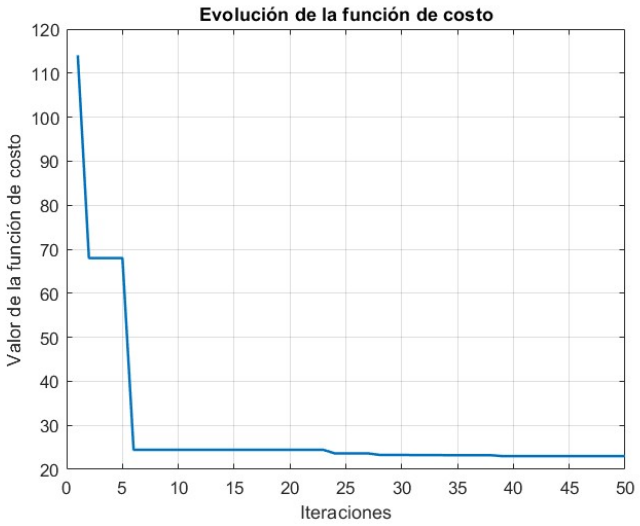

Figura 9: Minimización de la función de costo con aumento en F

Explorar ambos extremos permitiría balancear mejor la dinámica del algoritmo, ajustándose a la naturaleza específica del problema y, potencialmente, obteniendo mejores resultados en términos de velocidad y calidad de convergencia.

Cuando se incrementa F, la curva muestra un descenso más rápido y agresivo al inicio, pero también evidencia una alta variabilidad en las primeras iteraciones, con un costo inicial más alto (115) que desciende abruptamente. Sin embargo, aunque logra estabilizarse alrededor de 23, parece presentar pequeños saltos o irregularidades, lo que podría indicar que un F demasiado alto favorece una exploración muy amplia y genera inestabilidad antes de llegar a la convergencia. Esto sugiere que, aunque el algoritmo explora regiones amplias, carece de refinamiento en etapas avanzadas.

Por otro lado, cuando se disminuye F, se aprecia un descenso mucho más controlado y suave. Sin embargo, este

comportamiento también implica que el algoritmo tiene un progreso más lento en la reducción del costo, requiriendo más iteraciones para acercarse al mismo valor mínimo (23.2). Esta suavidad puede ser útil para una búsqueda más fina y estable, pero también indica que una explotación prematura que puede dejar atrapada la solución en un mínimo local si el valor de F es demasiado bajo.

V-E. Tercer caso caso: modificación de LB y UB

Otra estrategia que podría considerarse para mejorar el desempeño del algoritmo es la modificación de los valores de las cotas superiores (ub) e inferiores (lb) de las variables de decisión. Ajustar estas cotas de manera adecuada permite al algoritmo concentrarse en zonas más prometedoras del espacio de búsqueda y acelerar la convergencia hacia soluciones de menor costo.

Parámetros utilizados:

- N = 20
- D = 3
- CR = 0.9
- **F_{min} = 0.9**
- **F_{max} = 1.2**
- LB = [0.1, 0.01, 0.01]
- UB = [10, 5, 5]

El ajuste de los límites inferiores (LB) y superiores (UB) de los parámetros del controlador PID (q_0, q_1, q_2) se basa en el impacto que cada ganancia tiene en el desempeño del sistema. El aumento de q_0 (asociado a la ganancia proporcional) permite reducir el tiempo de establecimiento al incrementar la velocidad de respuesta del sistema, aunque esto puede generar un mayor sobreimpulso si no se compensa adecuadamente con las otras ganancias. Por ello, se recomienda expandir su límite superior hasta 20 para explorar soluciones más agresivas, manteniendo un límite inferior de 0.5 que evite ganancias demasiado bajas que ralenticen la respuesta. En el caso de q_1 (asociado a la ganancia integral), su incremento favorece la eliminación del error en estado estacionario, pero valores excesivos pueden introducir oscilaciones. Para equilibrar este efecto, se propone un límite inferior de 0.1 y un superior de 10, lo que permite una acción integral más intensa sin comprometer la estabilidad. Finalmente, el ajuste de q_2 (asociado a la ganancia derivativa) busca mitigar el sobreimpulso y mejorar la amortiguación del sistema. Aunque su límite inferior puede mantenerse en 0.01, el superior puede extenderse hasta 10 para potenciar el efecto de amortiguación, siempre que no se introduzca ruido en la señal de control.

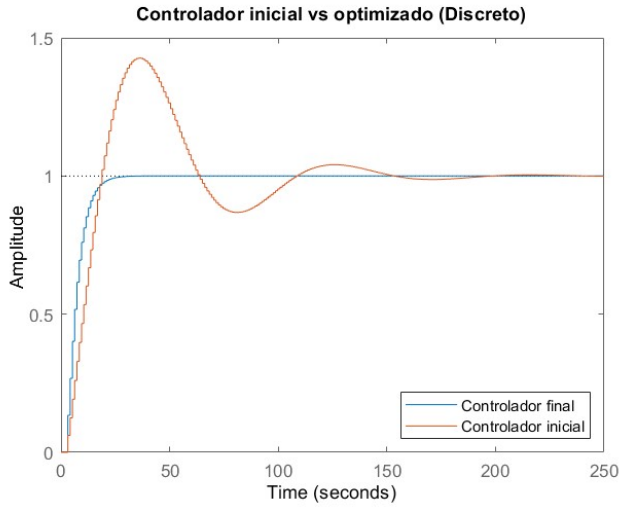

Figura 10: Comparación de controladores con ajuste en los limites

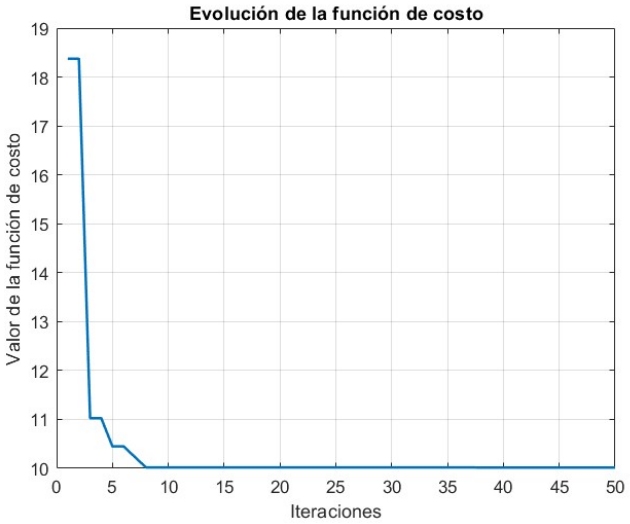

Figura 11: Minimización de la función de costo con ajuste en los limites

Estos ajustes permiten al algoritmo de explorar soluciones más rápidas y eficaces, como se observa en la mejora del controlador optimizado frente al inicial en la gráfica adjunta.

Al comparar las respuestas de ambos controladores, se observa que la modificación de los límites superiores e inferiores permitió obtener un controlador que presenta un desempeño dinámico mejorado. En la Figura 10, el controlador optimizado logra una menor sobreoscilación y un asentamiento más rápido en comparación con el controlador original de la Figura 2.

Adicionalmente, se puede observar una reducción en el tiempo de establecimiento, el cual se logra sin comprometer la estabilidad del sistema. El sobreimpulso se mantiene controlado en ambos casos, aunque con una ligera ventaja para el controlador que experimento modificaciones en UB y LB, puesto a que este presenta una amortiguación más eficiente. La curva del controlador ajustado muestra una pendiente inicial

más pronunciada, indicando una respuesta dinámica acelerada gracias a parámetros PID más agresivos. Aun así, se observa que ambos controladores logran estabilizarse finalmente en el valor deseado, el optimizado presenta un comportamiento más amortiguado y una respuesta transitoria más suave.

Esto sugiere que la adecuada elección de UB y LB puede ampliar la región de búsqueda del algoritmo, permitiendo encontrar soluciones más efectivas que mejoren la respuesta del sistema tanto en términos de estabilidad como de rapidez.

VI. Conclusiones

Podemos concluir acerca de los resultados obtenidos que:

1. La implementación del algoritmo de Evolución Diferencial demostró ser una herramienta efectiva para la sintonización de controladores PID discretos, logrando optimizar los parámetros q_0, q_1 y q_2 de manera que se minimizan tanto el tiempo de establecimiento como el sobrepaso, mejorando significativamente la respuesta dinámica del sistema.
2. El controlador optimizado exhibió una reducción del 95.34 % en el sobrepaso y del 70.59 % en el tiempo de establecimiento en comparación con el controlador inicial, pasando de un comportamiento oscilatorio subamortiguado a una respuesta más estable y similar a un sistema de primer orden, lo que resalta la capacidad del DE para encontrar soluciones robustas y eficientes.
3. La evolución de la función de costo mostró una convergencia rápida en las primeras 10-15 iteraciones, estabilizándose en un valor óptimo cercano a 25, lo que indica que el algoritmo equilibra adecuadamente la exploración y la explotación del espacio de soluciones, evitando la necesidad de un número excesivo de iteraciones.
4. La selección adecuada de parámetros como el tamaño de la población (N), la tasa de cruce (CR) y el factor de escala (F) resultó crucial para el éxito de la optimización. La variabilidad dinámica de F permitió escapar de óptimos locales, mientras que un CR alto favoreció la exploración inicial, contribuyendo a la calidad de la solución final.
5. Los resultados obtenidos sugieren que el enfoque basado en DE es una alternativa viable y eficiente para la sintonización de controladores PID en sistemas dinámicos, especialmente en aplicaciones industriales donde se requiere un balance entre rapidez de respuesta y estabilidad.

Referencias

[1] Dik, A. (2024, Abril 17). *Algoritmos de optimización de la población: Evolución diferencial (Differential Evolution, DE)*. MQL5 Community. https://www.mql5.com/es/articles/13781

[2] Novoa-Hernández, P., Corona, C. C., Pelta, D. A. (2014). *Un estudio comparativo sobre evolución diferencial auto-adaptativa en ambientes dinámicos.* Universidad de las Ciencias Informáticas. https://www.redalyc.org/journal/3783/378368201005/html

[3] Martínez, M.G. (2019, Febrero 20). *Cómputo Evolutivo.* Instituto Nacional De Astrofísica, Óptica Y Electrónica, 1–19. https://ccc.inaoep.mx/ a.morales/EC/pdf/ClaseED.pdf

[4] Sosa Toranzo, C., Leguizamón, G. (s. f.). Evolución Diferencial con Factor de Mutación Dinámico [Manuscrito no publicado]. Universidad Nacional de San Luis, Facultad de Ciencias Físico Matemáticas y Naturales, San Luis, Argentina. https://sedici.unlp.edu.ar/bitstream/handle/10915/23601/4859-DE$_{FD}inamico_{T}oranzo.pdf;jsessionid = 29E96235B4D3880DAAA916C667639BAA?sequence = 1$