Industrial Data
ISSN: 1560-9146
ISSN: 1810-9993
industrialdata@unmsm.edu.pe
Universidad Nacional Mayor de San Marcos
Perú

Tejada Muñoz, Guillermo
Controlador PID con algoritmos genéticos de números reales
Industrial Data, vol. 22, núm. 2, 2019, Julio-
Universidad Nacional Mayor de San Marcos
Perú

DOI: https://doi.org/10.15381/idata.v22i2.16489

Disponible en: https://www.redalyc.org/articulo.oa?id=81662532017

Revista Industrial Data 22(2): 213-234 (2019)
DOI: http://dx.doi.org/10.15381/idata.v22i2.16489

ISSN: 1560-9146 (Impreso) / ISSN: 1810-9993 (Electrónico)
Facultad de Ingeniería Industrial - UNMSM

Controlador PID con algoritmos genéticos de números reales

GUILLERMO TEJADA MUÑOZ [1]

RECIBIDO: 21/07/2019 ACEPTADO: 23/09/2019

RESUMEN

Un algoritmo genético o *genetic algorithm* (GA) es una técnica de la inteligencia artificial que es utilizada en cualquiera de las especialidades de ingeniería. En el presente estudio, el GA propuesto encuentra valores óptimos para los parámetros Kp, Ki y Kd de un controlador PID, utilizado ampliamente en la industria. Los cromosomas, formados con los genes Kp, Ki y Kd, representados por números reales, evolucionan y son evaluados mediante el error cuadrático medio (ECM) de la salida deseada. En ese sentido, la solución es el cromosoma con menor ECM, el cual produce menores transitorios en la salida. Además, el GA ha sido codificado en lenguaje M (MATLAB) y los resultados han sido comparados con otros trabajos.

Palabras-claves: Algoritmos genéticos; genes de números reales; PID; error cuadrático medio.

INTRODUCCIÓN

Un proceso industrial puede verse como un sistema cerrado (retroalimentado) constituido por un controlador y una planta controlada. Para el diseño del controlador, una de las técnicas más utilizadas, por su simplicidad y robustez, es la denominada proporcional integral derivativo (PID); sin embargo, no es una tarea fácil encontrar los valores óptimos de los parámetros del controlador denominados Kp, Ki y Kd. Por ese motivo, para el cálculo de esos parámetros, se ha propuesto como solución utilizar algoritmos de inteligencia artificial; por ejemplo, se tiene: optimización de enjambre de partículas (PSO) (Iwan *et al*., 2011), optimización de colonia de hormigas (ACO) (Lahlouh *et al*., 2019), lógica difusa (Akbari-Hasanjani *et al*., 2015), redes neuronales (Hernández *et al*., 2016), algoritmos genéticos (GA) (Feng *et al*., 2018), entre otros. En el presente estudio se describe la elaboración de un GA, codificado en lenguaje M de MATLAB, que calcula los parámetros Kp, Ki y Kd del controlador, de tal modo que la salida del sistema (controlador más planta) se estabilice al valor deseado, minimizando sus valores transitorios de sobreimpulso, tiempo de establecimiento y tiempo de subida.

Un algoritmo genético (GA) es un algoritmo de búsqueda que sigue el paradigma de la evolución. A partir de una población inicial, el algoritmo aplica operadores genéticos (selección, cruce y mutación) para producir descendientes (en la terminología de búsqueda local, corresponde a explorar el vecindario), que son claramente más aptos que sus antepasados. En cada generación (iteración), cada nuevo cromosoma corresponde a una solución (Pezzella *et al*., 2008). Los métodos para seleccionar los cromosomas que van a generar descendientes dan preferencia a aquellos con mayores valores de aptitud; entre estos métodos de selección se tiene a los siguientes: selección por ruleta, selección por torneo, *ranking* lineal, etc. Asimismo, en este estudio

1 Ingeniero electrónico y doctor en Ingeniería Industrial por la Universidad Nacional Mayor de San Marcos. Además, es magíster en Ingeniería Electrónica por la Universidad de São Paulo (Brasil). Actualmente, es profesor principal de la Facultad de Ingeniería Electrónica y Eléctrica de la UNMSM.
E-mail: gtejadam@unmsm.edu.pe

se ha utilizado la selección de *ranking* lineal porque con ella se logra asignar a los cromosomas, ranqueados de acuerdo a su valor de aptitud, distintos valores de probabilidad que se diferencian en una proporción lineal decreciente desde el mejor hasta el peor ranqueado; es decir, las probabilidades que tienen los cromosomas de ser seleccionados no se diferencian en forma desproporcionada.

Los métodos para realizar el cruce de los cromosomas son diversos y dependen de la naturaleza de los genes, los cuales puede ser binarios, permutados, de números reales, etc. Así, entre los métodos de cruce, se tiene a los siguientes: SPX (*single point exchange*), DPX (*double point crossover*), PMX (*partially matched crossover*), OX (*order crossover*), CX (*cycle crossover*), aritmético, etc. (Haupt y Haupt, 2004). Este último, el aritmético, ha sido utilizado en el presente trabajo, por ser adecuado para genes de números reales. Así, mientras que el cruce tiende a que la población genética converja, preferiblemente a valores óptimos, la mutación tiene como objetivo mantener un cierto nivel de diversidad en la población, es decir, evitar que la población converja rápidamente (Srinivas y Patnaik, 1994; Schaffer y Eshelman, 1991; Beck, 2000). La mutación utilizada con mayor frecuencia es aquella en que uno de los genes (variables) del cromosoma varía aleatoriamente (Gestal *et al*., 2010). Por último, el número de mutaciones se calcula mediante un porcentaje con respecto al tamaño de la población.

De esta forma, el problema por solucionar es multiobjetivo, porque se desea minimizar más de una variable, las que son: el sobreimpulso, el tiempo de establecimiento y el tiempo de subida de la respuesta del sistema. No obstante, para mayor sencillez, el problema ha sido tratado como mono-objetivo, es decir, persiguiendo un solo objetivo, que se correlacione con las variables de interés. En este sentido, se ha seleccionado como función objetivo, también denominado como función aptitud, función de ajuste, función *fitness* o función costo, a*l error cuadrático medio (ECM) de la salida.*

Por otro lado, el código del programa se ha escrito en lenguaje de MATLAB. El programa ha corrido controlando las plantas utilizados por otros investigadores, de esta forma se ha podido comparar resultados y demostrar que el algoritmo propuesto ha tenido un buen desempeño, ya que ha encontrado los valores adecuados de Kp, Ki y Kd del controlador PID con los cuales se han minimizado el sobreimpulso, el tiempo de establecimiento y el tiempo de subida de la respuesta del sistema.

METODOLOGÍA

La estructura del algoritmo implementado se describe en la figura 1. Después de creada una población genética inicial, se evalúa cada cromosoma de la población calculando e*l error cuadrático medio (ECM) de la salida que le corresponde. L*uego, se ordenan a los cromosomas de menor a mayor valor según su ECM y, posteriormente, entre los cromosomas más aptos se seleccionan a los que se van a cruzar con el objeto de generar descendencia, se mutan aleatoriamente a los cromosomas para luego realizar una nueva evaluación. El proceso se reitera tantas veces como lo permita el número de generaciones indicado por el usuario. Finalmente, se obtendrá la población óptima y el mejor cromosoma será el primero de la lista. En los párrafos siguientes se dan mayores precisiones sobre cada etapa.

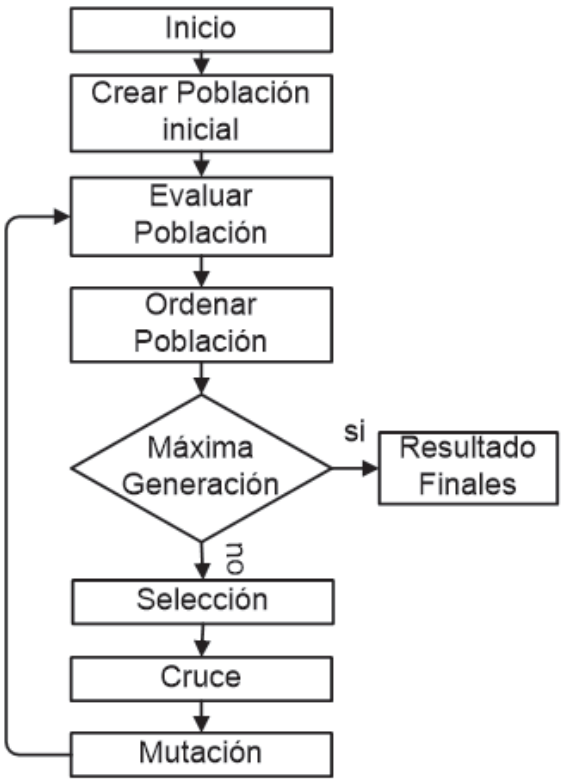

Figura 1. Estructura del algoritmo genético.
Fuente: elaboración propia.

Población

La población genética está constituida por un número de cromosomas, en donde cada uno contiene un número de genes. Por la naturaleza del problema, se definió que cada cromosoma estuviera constituido por tres genes o variables de números reales, lo

cual matemáticamente significa que un cromosoma es un vector de tres elementos, en donde cada elemento son las variables proporcional (KP), integral (Ki) y derivativa (Kd). De esta manera, cada vector (cromosoma) es una probable solución del problema, por lo que la población genética se ha construido como un arreglo de N filas por tres columnas, donde N es el número de cromosomas o tamaño de la población y cada una de las tres columnas representan respectivamente a las variables Kp, Ki y Kd. La población inicial ha sido creada con números reales aleatorios respetando los intervalos mínimo y máximo (rangos) de cada variable indicadas por el usuario. La figura 2 muestra parte del código que crea a la población genética.

Selección

En el tipo de selección de *ranking* lineal, los cromosomas son clasificados de acuerdo con su valor de ajuste y un rango $r_i \in \{1, ..., N\}$ se asigna a cada uno, donde N es el tamaño de la población. El mejor cromosoma obtiene el rango N mientras que el peor obtiene el rango 1. Luego: $P_i = (2r_i)/N(N+1)$ es la probabilidad de elegir el i-enésimo cromosoma en el *ranking* lineal (Pezzella *et al.*, 2008).

En este sentido, se escogen a los n primeros cromosomas para cruzarse, donde n es el producto de la tasa de cruce por el número total de la población. Luego, se le otorga mayor probabilidad de cruce a los mejores ranqueados, formando un arreglo denominado «ánfora», en donde los números que identifican a los cromosomas se repiten proporcionalmente a su importancia (Tejada, 2017). Así, por ejemplo, suponiendo que se ha escogido a los cuatro primeros cromosomas de la población (1,2,3,4), con ellos se genera una «ánfora», dando mayor prioridad de ser elegido al primer cromosoma porque es el mejor ranqueado de la población; esta se construye de la siguiente manera:

ánfora = (4 3 3 2 2 2 1 1 1 1)

Esto indica que, al seleccionar aleatoriamente un cromosoma de la «ánfora», el cromosoma 1 tiene la mayor probabilidad de ser elegido (4/10=40%), seguido del cromosoma 2 (3/10=30%), luego el cromosoma 3 (2/10=20%) y, finalmente, el cromosoma 4 (1/10=10%). Después, se selecciona de la «ánfora», mediante un sorteo, los padres y madres que se aparearán de acuerdo con el número de cruces especificado por el usuario. Se forman dos vectores, un vector de «padres» y un vector de «madres», así como se representan a continuación:

padres = (1 2 1 2 2 1 1 2)

madres = (2 1 2 4 1 3 3 2)

Hasta aquí, la etapa de selección termina su tarea. Luego, en la etapa de cruce, se van a cruzar los cromosomas: (1, 2), (2, 1), …, (1, 3), (2, 2). Los mismos padres pueden cruzarse varias veces y siempre van a generar hijos diferentes. Además, puede ocurrir que, tanto el padre como la madre, sean los mismos cromosomas, por lo que no es necesario evitarlo, esto no ha afectado el desempeño del algoritmo. La figura 3 muestra parte del código de la selección.

Cruce

El operador de cruce aritmético de números reales puede generar hasta dos descendientes que son una combinación lineal de dos cromosomas X e Y (Yalcinoz y Altun, 2001):

$$\text{hijo 1} = \alpha X + (1-\alpha)Y$$

$$\text{hijo 2} = (1-\alpha)X + \alpha Y$$

Donde α se escoge aleatoriamente entre 0 y 1.

```
% ¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬
% ¬¬¬¬¬¬ GENERACIÓN DE LA POBLACIÓN INICIAL ¬¬¬¬¬¬
% ¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬
MatrizRangos=[RangoP;RangoI;RangoD];
%Se generan valores de P, I y D aleatorios respetando Intervalos de usuario
P_aleatorio = MatrizRangos(1,1)+(MatrizRangos(1,2)-MatrizRangos(1,1))*rand(TamanoPob,1);
I_aleatorio = MatrizRangos(2,1)+(MatrizRangos(2,2)-MatrizRangos(2,1))*rand(TamanoPob,1);
D_aleatorio = MatrizRangos(3,1)+(MatrizRangos(3,2)-MatrizRangos(3,1))*rand(TamanoPob,1);
pob = [P_aleatorio I_aleatorio D_aleatorio];
```

Figura 2. Parte del código que genera la población genética.

Fuente: elaboración propia.

En esta investigación se ha preferido generar por cada cruce un solo hijo, por lo que el número de padres es igual al número de cruces. Esto se ha realizado con el objeto de dar oportunidad a que más padres generen descendencia obteniendo mayor diversidad en la población. Si se hubiera optado por generar dos hijos en cada cruce, solo se hubiera seleccionado para cruzarse la mitad del número de padres. La figura 4 muestra parte del código de cruce.

Mutación

Una vez seleccionado al azar un cromosoma de la población y la variable que será modificada (Kp, Ki o Kd), esta variable es remplazada por un número aleatorio:

pob(cromosoma, variable) ← valor aleatorio

Sin embargo, como cada variable tiene un valor dentro de un intervalo especificado por el usuario (Imin, Imax); entonces, para evitar que la mutación destruya un valor válido de la variable, el valor aleatorio tiene que estar dentro de ese rango especificado, por lo que el reemplazo debe ser (Seck Tuoh, *et al*., 2016):

pob(cromosoma, variable) ← Imin + (Imax-Imin)* valor aleatorio

```
% ¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬
% ¬¬¬¬¬¬¬¬ SELECCIONAR LISTA DE PADRES PARA CRUZAMIENTO ¬¬¬¬
% ¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬
Totalcupones = length(anfora);

sorteo1 = ceil (totalcupones*rand(1,NumeroCruces)); %ejem:(7 4 8 6 5 7 8 4)
sorteo2 = ceil (totalcupones*rand(1,NumeroCruces)); %ejem:(6 10 4 1 9 3 2 6)
listapadres = anfora (sorteo1); % listapadres1=(1 2 1 2 2 1 1 2)
listamadres = anfora (sorteo2); % listapadres2=(2 1 2 4 1 3 3 2)
% Se van aparear (1,2), (2,1). . .(1,3), (2,2). Cada apareamiento
%genera un nuevo hijo. Solo en (2,2) es inútil, esto en más probable
%que aparezca en poblaciones pequeñas y no es necesario corregir.
```

Figura 3. Parte del código de la selección.

Fuente: elaboración propia.

```
% ¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬
% ¬¬¬¬¬¬ CRUCE DE PADRES ¬¬¬¬¬¬
% ¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬

NumeroHijos = length(listapadres);
ListaHijos = zeros(NumeroHijos,3);

index = 1;

for i=1:NumeroHijos

  padre = listapadres(index);
  madre = listamadres(index);
  index = index + 1;

  alpha = rand;
  ListaHijos(i,:) = alpha*pob(padre,:) + (1-alpha)*pob(madre,:);
end
```

Figura 4. Parte del código de cruce.

Fuente: elaboración propia.

Otro importante punto que se ha tenido en cuenta es que nunca se elegirá al primer cromosoma para ser mutado, de tal manera que se conservará sin cambios al mejor rankeado, que pasará a la siguiente generación. La figura 5 muestra parte del código de mutación.

Evaluación

Se ha creado la función PID_ERROR_OBJETIVO, con la cual se evalúa a cada cromosoma de la población. De esta manera, con los valores Kp, Ki y Kd de cada cromosoma se calcula la función de transferencia del controlador PID. Luego, con la función de transferencia del controlador y con la función de transferencia de la planta del sistema se configura un sistema cerrado retroalimentado. Finalmente, con el sistema cerrado retroalimentado se calcula su respuesta frente a la entrada de un escalón de amplitud uno (1). En todo este proceso, para abreviar cálculos, se han utilizado las funciones de MATLAB: tf (la cual crea una función de transferencia de tiempo continuo), series (la que conecta dos modelos o dos funciones de transferencias) y *feedback* (encuentra la función de dos modelos, uno de los cuales pertenece al lazo de realimentación), un procedimiento similar se realiza en el estudio de Ibrahim Mohamed (2005).

Durante el transitorio que demora la función de salida hasta alcanzar el valor de 1, se hubiera podido encontrar por cada cromosoma los valores que toman las siguientes variables: tiempo de subida, sobreimpulso, tiempo de establecimiento, entre otros. De esta manera, se hubiera tenido que clasificar a los cromosomas que tengan a esas variables minimizadas, con lo cual el problema hubiera sido multiobjetivo. Sin embargo, todas las variables anteriores se pueden correlacionar con una sola variable, convirtiendo el problema a mono-objetivo. Es por eso que, en este estudio, se ha optado por calcular por cada cromosoma el error cuadrático medio (ECM) de la salida.

En ese sentido, el progreso del valor de la salida (Yi) hasta alcanzar el valor del escalón es leído durante n espacios de tiempo, calculando el error cuadrático medio (ECM) de la siguiente forma:

$$\mathrm{ECM} = \frac{1}{\mathrm{n}}\sum_{1}^{\mathrm{n}}(\mathrm{Yi}-1)^2$$

Donde:

n = número de medidas totales

Yi= valor de la salida del sistema en el instante i

$(\mathrm{Yi}-1)^2$ =error cuadrático de la salida en el instante i con respecto al escalón (1) de entrada.

Por cada cromosoma de la población, se tiene que realizar el cálculo de ECM de la manera antes descrita, con lo cual la función objetivo finaliza su tarea. Posteriormente, el algoritmo asocia a cada cromosoma un valor de ECM que lo identificará y los miembros de la población, de acuerdo con su ECM, son ordenados de menor a mayor valor, garantizando indirectamente un *ranking* ordenado de cromosomas de mejores características de tiempo de subida, sobreimpulso y tiempo de establecimiento. La figura 6 muestra la parte del código de evaluación.

El programa de *software* desarrollado se ha codificado en el lenguaje de MATLAB. Los resultados, tales como el valor de Kp, Ki, Kd y las características de la curva de salida, obtenidas con la función stepinfo, han sido exportados a una hoja de Excel mediante la función de MATLAB xlswrite, donde

```
% ¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬
% ¬¬¬¬¬¬¬ MUTACION ¬¬¬¬¬¬¬
% ¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬

for i = 1: NumeroMutaciones
ColVar = ceil(NumVariables*rand);
IndividuoPob = ceil((TamanoPob-1)*rand+1); % No muta nunca el primer cromosoma
pob (IndividuoPob,ColVar) = MatrizRangos(ColVar,1) + (MatrizRangos(ColVar,2) - ...
                    MatrizRangos(ColVar,1))*rand;
end
```

Figura 5. Parte del código de mutación.

Fuente: elaboración propia.

```
function [ICE] = PID_ERROR_OBJETIVO (pob, planta_sistema, tm)
% Se encuentra el número de miembros de la población
[cantidad_cromosomas, col] = size(pob);
for i = 1:cantidad_cromosomas
P = pob(i,1); % El primer gen del cromosomai es la constante P
I = pob(i,2); % El segundo gen del cromosomai es la cosntante I
D = pob(i,3); % El tercer gen del cromosomai es la constante D
% Se crea función de transferencia con P, I y D
controlador_pid = tf ([D P I],[1 0]);
% Se conecta planta y controlador
%(Equivale a:controlador_pid*planta_sistema)
planta_controlador = series (controlador_pid, planta_sistema);
%Conecta el modelo "planta_controlador" con el modelo ubicado
%en el lazo retrolimentado (en este caso es 1)
sistema_cerrado = feedback (planta_controlador,1);
% Para varios puntos del tiempo (muestras) se calcula la salida
tiempo =0:0.1:ceil (tm);
[salida,t] = step (sistema_cerrado, tiempo);
% Se cálcula el número de puntos muestreados
[muestras colu]=size(salida);
%¬¬¬ Se calcula el Error Cuadrático Medio(ICE)¬¬¬¬
for j = 1:muestras
% error^2= (salida deseada-salida obtenida)^2
error_cuadrado(j) = (1-salida(j))^2;
end
ICE(i,1) = sum (error_cuadrado) / muestras;
end
end
```

Figura 6. Código de evaluación.
Fuente: elaboración propia.

también se han generado los gráficos con las curvas de salida.

Los investigadores que han diseñado sus propios algoritmos genéticos o algoritmos genéticos difusos, entre otros, han contrastado sus resultados con los obtenidos con el método clásico de Ziegler-Nichols, exhibiendo buenos resultados. Por esta razón, se ha creído conveniente probar el algoritmo diseñado con los modelos de plantas utilizados por estos investigadores.

RESULTADOS

El programa ha sido ejecutado utilizando los modelos matemáticos de procesos (plantas) aplicados por otros estudiosos, lo cual ha permitido comparar resultados. Se ha seleccionado a los autores que han comparado sus datos con el método clásico de Ziegler-Nichols.

Para todos los casos, el programa se ha ejecutado con las siguientes especificaciones: tamaño de la población = 40, número de variables = 3, porcentaje de cruces = 70%, porcentaje de mutación = 10%, número de generaciones = 20.

Las tablas 1-4 muestran el modelo matemático de la planta controlada, los parámetros del controlador PID (Kp, Ki y Kd), los valores transitorios de la salida (sobre pico máximo, tiempo de subida, tiempo de establecimiento) y el error cuadrático medio. Además, se comparan los resultados con los obtenidos por otros investigadores, la columna «Algoritmo Genético Propuesto» corresponde al presente trabajo. Los intervalos de búsqueda para cada variable se muestran en la última fila de cada tabla.

Las figuras 7-10 muestran gráficamente los resultados de las tablas anteriores, la salida deseada del sistema se muestra con líneas punteadas y ha sido fijada para una amplitud de 1 (un escalón Step Res-

Tabla 1. *Comparaciones con resultados para Planta 1.*

Planta 1 $\frac{5S+2}{s^3+5s^2+8s+4}$	Algoritmo Genético Pareto (Acevedo *et al*., 2014) Multivariable	Algoritmo Genético PROPUESTO
Kp	37,6692	45,4248
Ki	18,9251	19,8057
Kd	13,0447	7,3211
Sobre pico máximo (%)	0,9995	1,2674
Tiempo de subida (s)	No indica	0,0545
Tiempo de establecimiento (s)	1,14	0,0847
Error cuadrático medio	----------	0,0100
Búsqueda: 0≤kp≥50; 0≤ki≥20; 0≤kd≥20		

Fuente: elaboración propia.

Tabla 2. *Comparaciones con resultados para Planta 2.*

Planta 2 $\frac{10}{s^2+2s+10}$	Algoritmo Genético Pareto (Acevedo *et al*., 2014) Multivariable	Algoritmo Genético PROPUESTO
Kp	8,4047	48,1812
Ki	17,1467	19,5887
Kd	12,0918	11,6125
Sobre pico máximo (%)	1,0137	1,1786
Tiempo de subida (s)	No indica	0,0180
Tiempo de establecimiento (s)	1,9484	0,0291
Error cuadrático medio	----------	0,0099
Búsqueda: 0≤kp≥50; 0≤ki≥20; 0≤kd≥20		

Fuente: elaboración propia.

Tabla 3. *Comparaciones con resultados para Planta 3.*

Planta 3. Control de nivel de tres tanques $\frac{1}{64s^3+9{,}6s^2+0{,}48s+0{,}008}$	Algoritmo Genético (Colorado *et al*., 2018)	Algoritmo Genético PROPUESTO
Kp	0,0361	0,03827
Ki	0,000731	0,00061
Kd	0,6999	0,63012
Sobre pico máximo (%)	18	12,3728
Tiempo de subida (seg.)	10	16,3166
Tiempo de establecimiento (seg.)	90	86,0968
Tiempo de pico	45	33,5201
Error cuadrático medio	----------	0,03280
Búsqueda: 0≤kp≥0.5; 0≤ki≥0.01; 0≤kd≥0.7		

Fuente: elaboración propia.

Tabla 4. *Comparaciones con resultados para Planta 4.*

Planta 4. Control de nivel de tanque $\frac{5(s+1)}{s(s+1)(s+6)} = \frac{5s+5}{s^3+7s^2+6s}$	Fuzzy PD+PID Controller (Souran *et al.*, 2013)	Algoritmo Genético PROPUESTO
Kp	30	39,10042
Ki	21,2	1,16204
Kd	9	6,86381
Sobre pico máximo (%)	0,72	0
Tiempo de subida (seg.)	No indica	0,06548
Tiempo de establecimiento (seg.)	23	0,12271
Error cuadrático medio		0,01236
Búsqueda: 0≤kp≥40; 0≤ki≥30; 0≤kd≥20		

Fuente: elaboración propia.

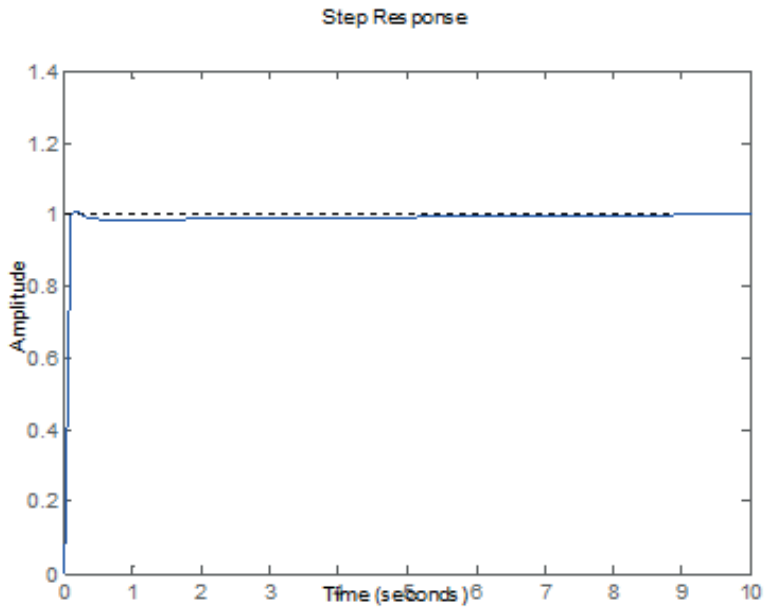

Figura 7. Respuesta controlada de la Planta 1 (tabla 1).

Fuente: elaboración propia.

ponse). Además, la línea azul es la salida del sistema que alcanza finalmente a la salida deseada.

DISCUSIÓN

Como se puede observar de los resultados de las tablas 1-4, en todos los casos los sobre picos máximos han sido similares, los tiempos de establecimientos han sido superiores en todos los casos y solo en la tabla 3 el tiempo de subida no ha sido el mejor. Estos resultados demuestran que el algoritmo propuesto ha tenido un buen desempeño en encontrar los valores adecuados de Kp, Ki y Kd del controlador PID. Además, para todos los casos se ha utilizado las mismas especificaciones de tamaño de población, porcentaje de cruce, porcentaje de mutación y número de generaciones lo cual demuestra que el algoritmo es bastante robusto.

CONCLUSIONES

Mediante algoritmos genéticos, se ha solucionado el problema de sintonizar los parámetros proporcional, derivativo e integral de un controlador PID,

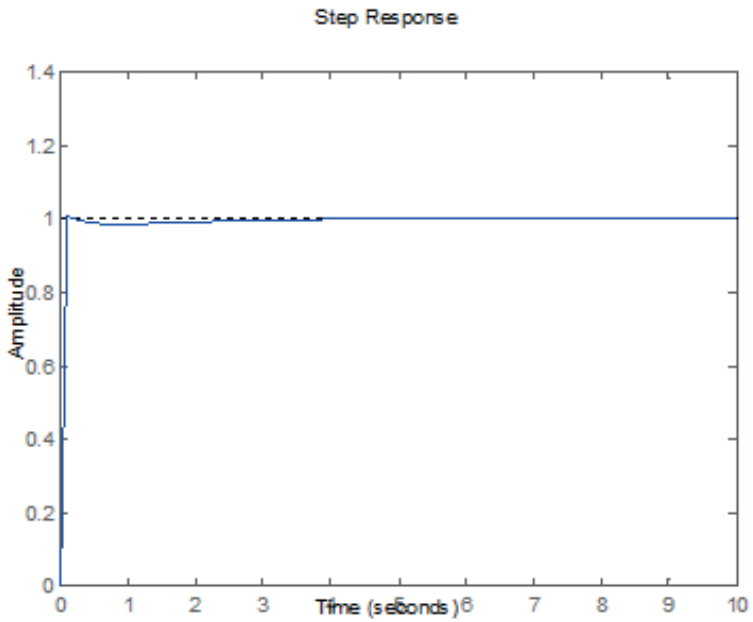

Figura 8. Respuesta controlada de la Planta 2 (tabla 2).

Fuente: elaboración propia.

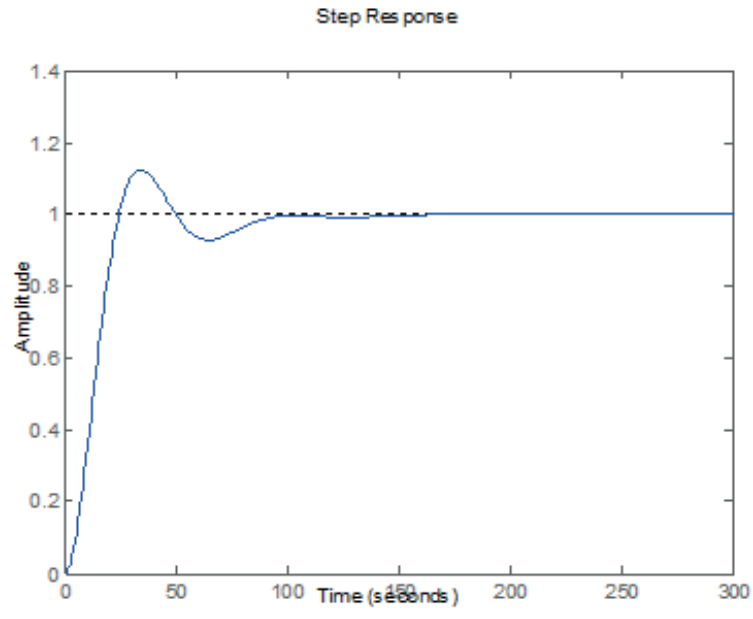

Figura 9. Respuesta controlada de la Planta 3 (tabla 3).

Fuente: elaboración propia.

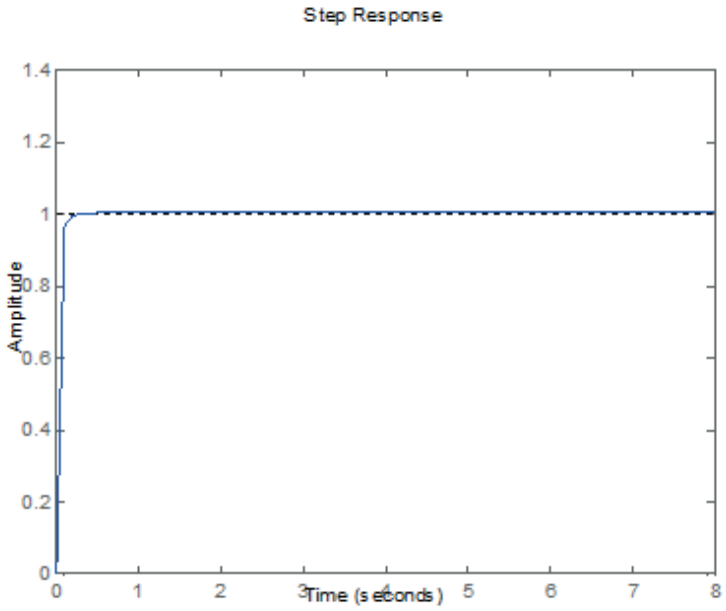

Figura 10. Respuesta controlada de la Planta 4 (tabla 4).
Fuente: elaboración propia.

minimizando el tiempo de establecimiento y el sobreimpulso de la respuesta en plantas no lineales. Asimismo, se ha probado que los resultados obtenidos han sido óptimos al compararlos con resultados de otras investigaciones, mostrados en las tablas 1-4.

Además, se ha probado que el algoritmo diseñado como mono-objetivo ha sido eficiente al minimizar, indirectamente, diversas variables (sobreimpulso, tiempo de establecimiento, tiempo de subida) mediante la minimización de una sola variable: *el error cuadrático medio, en contraposición con un algoritmo genético multiobjetivo basado en el enfoque de Pareto, como el presentado por Acevedo et al. (2014).*

REFERENCIAS BIBLIOGRÁFICAS

[1] Acevedo, B.; Fonseca, J. y Gómez, J. (2014). Desarrollo de una herramienta en Matlab para sintonización de controladores PID, utilizando algoritmos genéticos basado en técnicas de optimización multiobjetivo. *Revista SENNOVA*, *1*(1), 80-103.

[2] Akbari-Hasanjani, R.; Javadi, S. y Sabbaghi-Nadooshan, R. (2015). DC motor speed control by self-turning fuzzy PID algorithm. *Transactions of the Institute of Measurement and Control*, *37*(2), 164-176.

[3] Beck, F. (2000). *Escalonamiento de tarefas job-shop realistas utilizando algoritmos genéticos en Matlab*. (Tesis de maestría). Universidad Federal de Santa Catarina, Florianópolis, SC.

[4] Colorado, O.; Hernández, N.; Seck Tuoh, J. y Medina, J. (2018). Algoritmo genético aplicado a la sintonización de un controlador PID para un sistema acoplado de tanques. *PADI Boletín Científico de Ciencias Básicas e Ingenierías del ICBI*, *5*(10), 50-55.

[5] Feng, H.; Yin, C.; Weng, W.; Ma, W.; Zhou, J.; Jia, W. y Zhang, Z. (2018). Robotic excavator trajectory control using an improved GA based PID controller. *Mechanical Systems and Signal Processing*, *105*, 153-168.

[6] Gestal, M.; Rivero, D.; Rabuñal, J.; Dorado, J. y Pazos, A. (2010). *Introducción a los algoritmos genéticos y la programación genética*. La Coruña, España: Universidad de La Coruña.

[7] Haupt, R. y Haupt, S. (2004). *Practical genetic algorithms*. Nueva Jersey, EE. UU.: Wiley-Interscience.

[8] Hernández, R.; García, L.; Salgado, T.; Gómez, A. y Fonseca, F. (2016). Neural network-based self-turning PID control for underwater vehicles. *Sensors*, *16*(9). Recuperado de https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5038707/.

[9] Iwan, M.; Fook, L. y Leap, M. (2011). Tuning of PID controller using particle swarm optimization (PSO). *International Journal on Advanced Science, Engineering and Information Technology*, *1*(4), 458-461. Recuperado de http://insightsociety.org/ojaseit/index.php/ijaseit/article/viewFile/93/98.

[10] Lahlouh, I.; Elakkary, A. y Sefiani, N. (2019). PID controller of a MIMO system using ant colony algorithm and its application to a poultry house system. En H. Hachimi (Ed.), *2019 5th International Conference on Optimization and Application (ICOA)* llevado a cabo en Marruecos.

[11] Mohamed, I. (2005). *The PID controller design using genetic algorithm*. (Tesis de pregrado). University of Southern Queensland, Toowoomba, QLD.

[12] Pezzella, F.; Morganti, G. y Ciaschetti, G. (2008). A genetic algorithm for the flexible job-shop scheduling problema. *Computers & Operations Research*, *35*(10), 3202-3212.

[13] Schaffer, J. y Eshelman, L. (1991). On crossover as an evolutionarily viable strategy. *International Conference on Genetic Algorithms*, *4*, 61-68.

[14] Seck Tuoh, J.; Medina, J. y Hernández, N. (2016). *Introducción a los algoritmos genéticos con Matlab*. Recuperado de https://repository.uaeh.edu.mx/bitstream/bitstream/handle/123456789/16991/introduccion_a_los_algoritmos_geneticos_con_matlab.pdf.

[15] Souran, D.; Abbasi, S. y Shabaninia, F. (2013). Comparative study between tank's water level control using PID and fuzzy logic controller. En V. Balas, J. Fodor, A. Várkonyi-Kóczy, J. Dombi y L. Jain (Eds.), *Soft computing applications V. 195* (pp. 141-153). Berlín, Alemania: Springer-Verlag Berlin Heidelberg.

[16] Srinivas, M. y Patnaik, L. (1994). Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, *24*(4), 656-667.

[17] Tejada, G. (2017). *Enrutamiento y secuenciación óptimos en un flexible job shop multiobjetivo mediante algoritmos genéticos*. (Tesis doctoral). Universidad Nacional Mayor de San Marcos, Lima.

[18] Yalcinoz, T. y Altun, H. (2001). Power economic dispatch using a hybrid genetic algorithm. *IEEE Power Engineering Review*, *21*(3), 59-60.