UNIVERSIDAD DE CHILE

# Universidad de Chile

EL7012

Control Inteligente de Sistemas, Otoño

---

# Ejercicio N°2

---

*Autor:*
Elias Obreque
Gustavo Ceballo
Maibeth Sánchez

11 de julio de 2020

# Índice



# Índice de figuras

# Índice de tablas

# 1. Introducción

En este ejercicio se dará solución al Problema 1 y Problema 2 (a,b) con ayuda del material del curso [1, 2, 3, 4]. Los códigos utilizados para los modelos se encuentran en [5].

## 1.1. Problema 1

Considere la siguiente función:

$$f(x,y) = e^{-2\log(2)\left(\frac{\sqrt{x^2+y^2}-0{,}08}{0{,}854}\right)^2} \sin^2\left(5\pi((\sqrt{x^2+y^2})^{0{,}75} - 0{,}1)\right) \quad (1)$$
$$x, y \in [0,1]$$

El objetivo de este problema es maximizar la función objetivo anterior usando: un algoritmo convencional, algoritmos Genéticos (GA) y PSO.

Para ello se pide:

a) Grafique la función en el dominio indicado. (1 pto.)

b) Encuentre el máximo de f (x, y) utilizando algún algoritmo de optimización convencional. Indique el valor óptimo de f, punto óptimo $[x^{y]}$ y el tiempo de ejecución. (1 pto.)

c) Utilice PSO para resolver el problema de optimización. Encuentre una configuración óptima variando y analizando el efecto de los diversos parámetros del algoritmo. Varíe la cantidad de partículas y el número de iteraciones. Indique el valor óptimo de f, el punto óptimo $[x^*_{PSO}, y^*_{PSO}]$ y el tiempo de ejecución. (1.5 ptos)

d) Utilice GA para resolver el problema de optimización. Encuentre una configuración óptima variando y analizando el efecto de los diversos parámetros del algoritmo. Varíe la cantidad de partículas y el número de iteraciones. Indique el valor óptimo de f, el punto óptimo $[x^*_{GA}, y^*_{GA}]$ tiempo de ejecución. (1.5 ptos)

Compare los resultados obtenidos con los tres métodos de optimización (Convencional, PSO y GA) en términos del valor óptimo encontrado, el punto de operación óptimo de $[x^*, y^*]$ el esfuerzo computacional. (1 pto.)

## 1.2. Problema 2: Control del Estanque

En la industria existen una gran cantidad de procesos que requieren del control de nivel en estanques, siendo esto de vital importancia para garantizar tanto la calidad de los productos como la seguridad del personal y los equipos involucrados.

La dinámica de estos sistemas es un aspecto fundamental para el diseño de estrategias de control y depende fuertemente de la forma que presenten estos estanques. Los contenedores cuadrados y cilíndricos por un lado presentan una dinámica lineal, pero que impide un drenaje correcto de su contenido por lo que una de las alternativas más utilizadas son los estanques cónicos, los cuales solucionan este problema, pero presentan una dinámica no lineal.

El siguiente esquema muestra el estanque cónico presente en el Laboratorio de Automática del Departamento de Ingeniería Eléctrica de la Universidad de Chile:

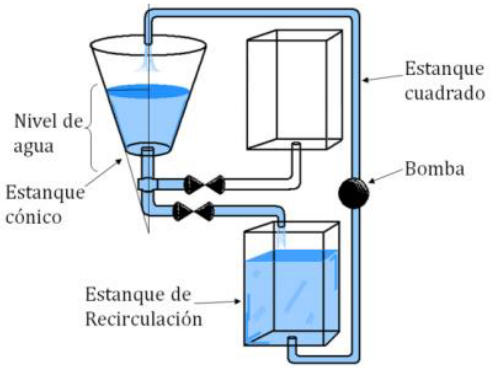



Figura 1: Estanque de laboratorio.

El agua es impulsada desde el estanque de recirculación mediante una bomba que es controlada por un variador de frecuencia (f) que opera desde 0 a 100 %, siendo el máximo los 50 Hz que proporciona la red eléctrica.

Para modelar la dinámica del estanque se utiliza la ley de conservación de la masa descrita como sigue:

$$\frac{dV(t)}{dt} = F_{in}(t) - F_{out}(t) \tag{2}$$

donde $V$ representa el volumen de agua en el estanque, medido en $cm^3$, $F_{in}$ y $F_{out}$

son el flujo de entrada y salida respectivamente los cuales son descritos como:

$$F_{in} = 5{,}53f + 78{,}23 \tag{3}$$

$$F_{out} = 20{,}21\sqrt{h} \tag{4}$$

siendo en este caso $f$ el porcentaje de frecuencia (0-100 %) aplicado a la bomba y $h$ el nivel de agua en el estanque cónico en $cm$.

Por otra parte, si bien un análisis geométrico permitiría deducir la dependencia del volumen de agua con el nivel en el estanque, existen ciertas situaciones particulares al estanque del Laboratorio que impiden que este modelo se ajuste con precisión a la realidad de la planta:

- En este análisis se considera un cono perfecto, no se toma en cuenta el volumen de agua contenido en tuberías.
- Existe una leve inclinación del montaje respecto a la vertical, lo cual afecta la forma en que se llena el estanque.

Dado esto, el modelo que mejor ajusta lo observado en la realidad viene dado por el siguiente ajuste polinomial:

$$V(h) = 0{,}21h^3 + 5{,}7h^2 + 17{,}1h + 290{,}7 \tag{5}$$

Dado esto, el modelo que mejor ajusta lo observado en la realidad viene dado por el siguiente ajuste polinomial:

$$\frac{dV}{dt} = \frac{dV}{dh}\frac{dh}{dt} = \frac{dh}{dt}(0{,}63h^2 + 11{,}4h + 17{,}1) \tag{6}$$

$$\frac{dh}{dt} = \frac{5{,}43f + 78{,}23 - 20{,}21\sqrt{h}}{(0{,}63h^2 + 11{,}4h + 17{,}1)} \tag{7}$$

Con estos antecedentes y considerando un tiempo de muestreo de 10 segundos y una condición inicial para la altura de $h = 20$ [cm]. Pruebe los controladores para las siguientes referencias $r_1$= 25, $r_2 = 35$, $r_3 = 30$ y $r_4 = 45$ (en ese orden de forma secuencial), teniendo en consideración que el tiempo de vaciado es significativamente más largo que el de llenado (defina un tiempo razonable de duración para los escalones). De esta forma, se pide lo siguiente:

a) Utilizando la aproximación de Euler para la derivada Temporal sobre el modelo de la altura:

$$\frac{dh}{dt} \approx \frac{h(k+1) - h(k)}{\Delta T} \tag{8}$$

Diseñe un controlador predictivo que minimice el siguiente funcional

$$J = A\sum_{i=1}^{N}(h(k+i) - r)^2 + B\sum_{j=1}^{M}(f(k+j))^2 \tag{9}$$

El problema de optimización debe ser resuelto utilizando uno de los algoritmos evolutivos visto en clases (GA o PSO). Tenga en consideración las restricciones sobre las variables $h \in$ [15,50] y $f \in$ [0,100] y pruébelo en el simulador "SimEstanque.slx". Justifique debidamente la elección de su controlador, haga énfasis en los parámetros $A$, $B$, $N$, y $M$ elegidos, así como los parámetros utilizados en el algoritmo elegido.

b) A partir de los Datos del archivo "DataEstanque.mat", derive un modelo difuso de Takagi Sugeno y un modelo basado en redes neuronales para la altura del estanque cónico. Especifique claramente los modelos obtenidos y grafique su desempeño.

c) Diseñe e implemente un controlador predictivo difuso a 1 y 5 pasos basándose en los dos modelos identificados en la parte anterior utilizando un algoritmo de optimización convencional (son 2 controladores en total). Puede utilizar fmincon en MATLAB para resolver el problema de optimización con restricciones.

d) Repita lo solicitado en c) utilizando GA y PSO para resolver el problema. Optimice los parámetros de estos algoritmos. Deje claramente establecidos los parámetros de los algoritmos utilizados y justifíquelos (son 4 controladores en total).

e) Compare el esfuerzo computacional y el desempeño (tiempo de establecimiento, sobrepaso y función objetivo) entre los 3 algoritmos utilizados sobre los modelos identificados (Neuronal y TS).

f) Compare el desempeño del controlador de la parte a) con el desempeño de los controladores de la parte d) que utilicen el mismo algoritmo de optimización, es decir, compare el controlador que utiliza el modelo discretizado con los que utilizan TS y Neuronal para el mismo algoritmo (PSO o GA). Concluya teniendo en consideración que la identificación fue realizada en lazo cerrado con un controlador PI simple.

# 2. Solución problema 1

a) La función a analizar es:

$$f(x,y) = e^{-2\log(2)\left(\frac{\sqrt{x^2+y^2}-0{,}08}{0{,}854}\right)^2} \sin^2\left(5\pi((\sqrt{x^2+y^2})^{0{,}75} - 0{,}1)\right) \qquad (10)$$
$$x, y \in [0,1]$$

cuyo dominio en el cual se desea analizar su comportamiento es $x, y \in [0,1]$. Una gráfica de la función se muestra en la Fig. 2

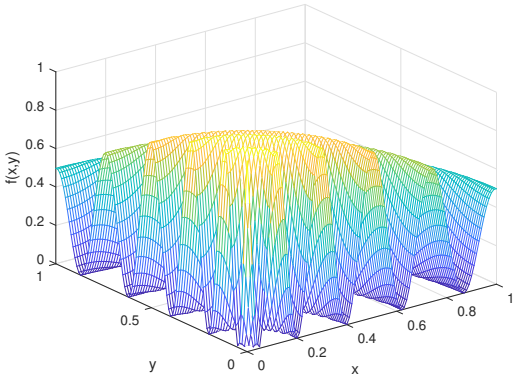


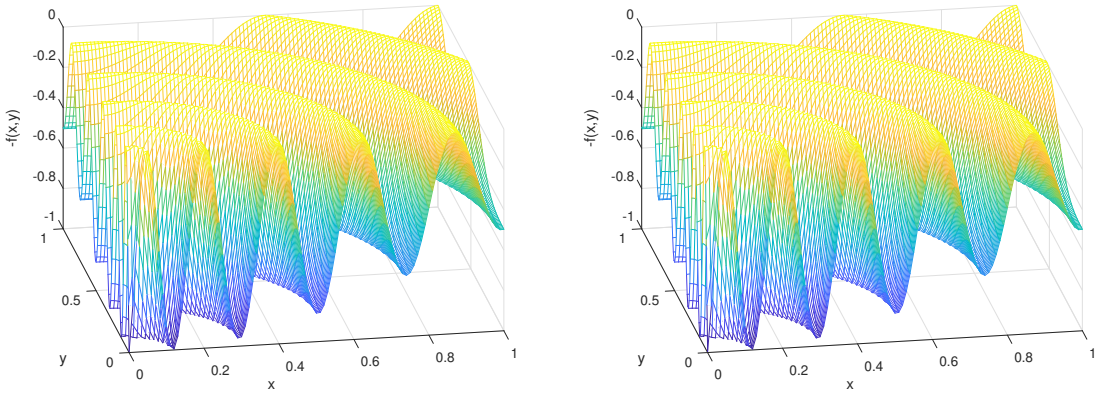

Figura 2: Gráfica de la función f(x,y).

Para llevar a cabo dicha gráfica, se usaron los comandos mesh y meshgrid de Matlab. La función meshgrid genera una grilla rectangular, en la cual, se usaron pasos de 0.01 para llenar el rango de valores del dominio de x,y.

b) Ahora bien, como el objetivo de este apartado es determinar el valor máximo de la función f(x,y) y en general las funciones de optimización solo minimizan, entonces, determinar el máximo esta función es equivalente a minimizar el negativo de dicha función, es decir, denotamos una nueva función a minimizar $f_{min}(x,y)$ tal que:

$$f_{min}(x,y) = -f(x,y) \qquad (11)$$

Así entonces, la nueva función a minimizar se muestra en la Fig. 3. Además, en (b) se muestra una vista rotada de la función $f_{min}(x,y)$ para apreciar de mejor manera, sus valores extremos.




(a) Gráfica de la función fmin(x,y)=-f(x,y). (b) Vista rotada

Figura 3: Representación de la función.

Para llevar a cabo el cálculo del punto o vector óptimox $\vec{x} = [x^*, y^*]$, el cual verifica la maximización de la función f(x,y), se utilizó la función **fmincon** de Matlab [6], la cual encuentra los valores que minimizan -f(x,y). Es decir, esta función encuentra el mínimo de f(x) de un problema especificado por:

$$\min_x f(x) \qquad tal\ que \begin{cases} c(x) \\ Ax \leq b \\ lb \leq x \leq ub \end{cases} \tag{12}$$

donde x es un vector de dimensión arbitraria (en lo sucesivo y en caso que no quede claro que x es un vector, éste se denotará con su notación clásica como $\vec{x}$). En nuestro caso, $\vec{x} = [x, y]$ . A es una matriz y b es un vector en que $Ax \leq b$ y $c(x) \leq 0$ son restricicoen de desigualdad. Además, lb y ub son vectores que dan cuenta de los rangos máximos y mínimos de cada componente del vector x.

La estructura de la función fmincon en su forma más general en Matlab, es la siguiente: [x*, fval*] = fmincon(fun, x0, A, b, Aeq, beq, lb, ub, nonlcon, options), donde:

x*: Vector óptimo.

fval*: Valor de la función "fun" definida por el usuario y evaluada en el vector óptimo x*.

fun: función a optimizar.

$x_0$: vector para iniciar la búsqueda del vector óptimo x*.

A, b: Matriz A y vector b que dan cuenta de restricciones de desigualdad.

Aeq, beq: Matriz Aeq y vector beq que dan cuenta de restricciones de igualdad.

lb, ub: Vector de limites inferior y superior del vector x.

nonlcon: Restricciones no lineales del tipo $c(x) \leq 0$ y $c_{eq}(x) = 0$

options: opciones de optimización

En caso que no existan restricciones, estos parámetros se ingresan como vectores o matrices vacías ([]).

Finalmente, como el problema no tiene mayores restricciones excepto para los rangos máximos (ub) y mínimos (lb) de x,y, se ingresaron parámetros vacíos a dicha función excepto para dichos rangos, es decir:

$[x*, fval*] = fmincon(@fun_min, x0, [], [], [], [], lb, ub)$

**Nota:** Notar que el valor de función evaluada en los vectores óptimos debe ser el negativo de fval.

Así entonces, se utilizaron diferentes valores para $\vec{x}_0 = [x_0 \quad y_0]$ como puntos de inicio de la búsqueda para probar si el algoritmo encontraba otros puntos (o vectores) óptimos dada que a simple vista, se aprecia que la función f(x,y) a analizar, presenta varios máximos y mínimos locales, lo que hace pensar que el algoritmo de optimización convencional, podría converger a máximos locales y no necesariamente al máximo global.

A continuación, en la Tabla 1, se presenta algunas simulaciones usando el algoritmo de optimización convencional con la función de Matlab fmincon.

Tabla 1: Valores óptimos locales y tiempos de ejecución para determinar el óptimo

| [x0 y0] | [x* y ] | f(x*,y*) | Tiempo de ejecución [seg] |
|---|---|---|---|
| [0. 5 0.5] | [0. 5228 0.5228] | 0.6972 | 0.0631 |
| [ 0 0 ] | [ 0 0 ] | 0.9947 | 0.2908 |
| [0.2 0.2] | [0.2079 0.2079] | 0.9628 | 0.1221 |
| [0.8 0.8] | [0.7033 0.7033] | 0.4993 | 0.0632 |

Es interesante notar que los valores extremos de la función son diferentes dependiendo del vector de búsqueda inicial $x_0$. Es decir, los valores de la función convergen a máximos locales. Más aún, los vectores óptimos, se encuentran

relativamente cercanos al vector de búsqueda inicial. Esto era de esperarse, dada la forma de la función, la cual como se mencionó anteriormente, presenta varios máximos (o mínimos) locales. Pareciera, sin embargo, que el máximo global es cercano a 1 puesto que 0.9947 es el valor más grande que aparece en la Tabla 1.

c) En este caso, para determinar el máximo de la función f(x,y) usaremos otra técnica denominada Optimización por Enjambre de Partículas (Particle Swarm Optimization del inglés o simplemente PSO). Esta es una técnica perteneciente a la familia de la inteligencia de enjambres, aunque últimamente, se ha podido establecer una estrecha relación con la categoría de los algoritmos evolutivos.

Para llevar a cabo esta técnica, utilizaremos la función particleswarm de Matlab, la cual está disponible a partir de la versión 2014b. Los fundamentos de este algoritmo se pueden encontrar en los apuntes del curso EL7012 Control de Sistemas Inteligentes [3, 7].

La forma general de esta función es:

$[x^*_{PSO}, fval^*_{PSO}]$ = particleswarm(fun,nvars,lb,ub,options)

Similar al caso de la función fmincon, los parámetros de la función y las salidas son las siguientes:

$x^*_{PSO}$: Partícula óptima (o vector óptimo).

$fval^*_{PSO}$: Valor de la función "fun" definida por el usuario y evaluada en la partícula óptima $x^*_{PSO}$.

fun: Función a optimizar o de fitness definida por el usuario

nvars= Número de componentes o variables de la partícula x (o vector x).

lb, ub: Vector de limites inferior y superior de las componentes de la partícula x (o vector x).

options: opciones de optimización.

Ahora bien, por defecto en Matlab, el algoritmo PSO, define el tamaño del enjambre (número de partículas que componen el enjambre) como el mínimo de 100 y 10*nvars. Así entonces, como en nuestro caso, cada partícula del enjambre tiene solo 2 componentes, a saber $\vec{x} = [x, y] => nvars = 2$, entonces, el tamaño del enjambre será de 20 partículas. Este parámetro se puede variar con el parámetro options. De igual modo, el número máximo de iteraciones para la búsqueda del óptimo, es por defecto 200*nvars=400 iteraciones, este número también puede ser modificado con el parámetro options.

En la Tabla 2, se muestran los valores de la partícula óptima $\vec{x}^*_{PSO}$ y el valor respectivo de la función para diversos tamaños del enjambre.

Tabla 2: Valores óptimos y tiempos de ejecución para determinar el óptimo usando PSO

| Tamaño del enjambre | Número de Iteraciones | [x* y ] | f(x*,y*) | Tiempo de ejecución [seg] |
|---|---|---|---|---|
| 10 | 21 | [0 0] | 0.9947 | 0.0113 |
| 10 | 23 | [0.03 0.113] | 0.9989 | 0.0074 |
| 10 | 37 | [0.1156 0.019] | 0.9989 | 0.0073 |
| **20** | **21** | **[ 0 0 ]** | **0.9947** | **0.0049** |
| 20 | 23 | [0.1169 0] | 0.9989 | 0.0103 |
| 20 | 37 | [0.0505 0.1054] | 0.9889 | 0.0125 |
| 50 | 21 | [0 0.117] | 0.9989 | 0.01 |
| 50 | 23 | [0 0.1169] | 0.9989 | 0.0122 |
| 50 | 37 | [0.1169 0] | 0.9989 | 0.0198 |
| 100 | 21 | [0.1169 0] | 0.9989 | 0.0355 |
| 100 | 23 | [0.0143 0.116] | 0.9989 | 0.0321 |
| 100 | 37 | [0 0.1169] | 0.9989 | 0.0407 |

Destacado está el caso con los parámetros por defecto que utiliza Matlab para llevar cabo el algoritmo PSO. Es evidente que el algoritmo convergió mucho antes (21 iteraciones) del número máximo de iteraciones que considera por defecto (400 iteraciones). Esto da una idea de lo robusto del algoritmo. Más aún, pareciera que se obtiene el óptimo global (0.99891) en prácticamente todas las combinaciones de la tabla, ya que para un gran número de partículas del enjambre (100 por ejemplo), se obtiene el mismo valor que para un tamaño menor (50 o 20 partículas).

También, se puede apreciar que no es tan decisivo el tamaño del enjambre, para determinar la partícula óptima ya que, por ejemplo, para 23 iteraciones, el óptimo de la función ó máximo de la función en nuestro caso, es igual a 0.9989 (aproximadamente igual a 1), no importando si el tamaño del enjambre es de 10, 20, 50 ó 100 partículas. En un caso más extremo, en el cual disminuyéramos el tamaño del enjambre a solo 5 partículas por ejemplo, manteniendo el número de iteraciones en 23, se tendría una leve baja de la función objetivo f(x,y) a 0.9628. Aun así, este valor es cercano a los valores óptimos anteriores.

Respecto de los tiempos de ejecución del algoritmo, estos son menores mientras más pequeño es el enjambre. Sin embargo, estos tiempos son relativamente pequeños, no importando el tamaño del enjambre e iteraciones del algoritmo, al menos para este problema. Por tanto, no debiera ser necesario insistir en reducir el número máximo de iteraciones o tamaño del enjambre (a menos de 50 partículas), ya que esta disminución a valores excesivamente pequeños, podría afectar el encontrar la partícula óptima. Es evidente que, si este tiempo fuera mayor al tiempo de muestreo, en caso que se necesite disponer del valor óptimo en tiempo real, entonces sí sería razonable probar con menores números de iteraciones, aunque no se logre obtener un valor tan óptimo para la partícula que maximiza el fitness.

d) En este caso, para determinar el máximo de la función f(x,y) usaremos la técnica denominada Algoritmos Genéticos (Genetic Algorithm del inglés o GA en diminutivo). Esta es una técnica perteneciente a la familia de los algoritmos evolutivos y probablemente, la principal representante de dicha familia.

Para llevar a cabo esta técnica, utilizaremos la función ga, la cual está disponible a partir de la versión 2012a de Matlab. Los fundamentos de este algoritmo se pueden encontrar en los apuntes del curso EL7012 Control de Sistemas Inteligentes [3]

La forma general de esta función es:

[$x^*_{GA}$,$fval^*_GA$,exitflag,output,population,scores]=

ga(fun,nvars,A,b,Aeq,beq,lb,ub,nonlcon,options)

Los parámetros de la función y las salidas son las siguientes.

$x^*_{GA}$: Cromosoma óptimo (o vector óptimo).

$fval^*_GA$: Valor de la función "fun" definida por el usuario y evaluada en el cromosoma óptimo $x^*_{GA}$.

output: Salida que indica algunos parámetros del proceso de optimización como el número de generaciones, por ejemplo.

populations: valores de los genes del cromosoma en cada generación.

scores: Es el puntaje de la población final.

fun: Función a optimizar (o función de fitness) definida por el usuario

nvars= Número de componentes o genes del cromosoma x (o vector x).

A, b: Matriz A y vector b que dan cuenta de restricciones de desigualdad.

Aeq, beq: Matriz Aeq y vector beq que dan cuenta de restricciones de igualdad.

lb, ub: Vector de limites inferior y superior de los genes del cromosoma x.

nonlcon: Restricciones no lineales del tipo $c(x) \leq 0$ y $c_{eq}(x) = 0$

options: opciones de optimización

Ahora bien, por defecto, en Matlab, el algoritmo GA define el tamaño de la población (número de cromosomas que componen la población). Este tamaño es 50 cuando el número de variables (o genes) es menor o igual que 5, y 200 en otro caso. Así entonces, como en nuestro caso, cada cromosoma $\vec{x}$ consta de 2 genes, es decir, solo 2 componentes, a saber $\vec{x} = [x, y] => nvars = 2$, entonces, el tamaño de la población será de 50 cromosomas. Este parámetro se puede variar con el parámetro options. De igual modo, el número máximo de iteraciones para la búsqueda del óptimo, es por defecto 100*nvars=200 iteraciones, este número también puede ser modificado con el parámetro options.

En la Tabla 3, se muestran los valores del cromosoma óptimo $\vec{x}_{GA}{}^*$ y el valor respectivo de la función de fitness f($x^*_{GA}$,$y^*_{GA}$) para diversos tamaños de la población.

Tabla 3: Valores óptimos y tiempos de ejecución para determinar el óptimo usando GA

| Tamaño del enjambre | Número de Iteraciones | [x* y ] | f(x*,y*) | Tiempo de ejecución [seg] |
|---|---|---|---|---|
| 10 | 21 | [0.1317 0.7276] | 0.6972 | 0.0224 |
| 10 | 23 | [0.2384x10-6 0] | 0.9947 | 0.0241 |
| 10 | 37 | [0.0919 0.2793] | 0.9628 | 0.0147 |
| 20 | 21 | [ 0.2829 0.0799] | 0.9628 | 0.0131 |
| 20 | 23 | [0.1344 0.2615] | 0.9628 | 0.0178 |
| 20 | 37 | [0.1192x10-6 0.0447x10-6] | 0.9947 | 0.0186 |
| 50 | 21 | [0.109 0.0421] | 0.9989 | 0.0366 |
| 50 | 23 | [0.0536 0.1039] | 0.9989 | 0.0316 |
| 50 | 37 | [ 0.0877 0.0773] | 0.9989 | 0.0342 |
| **50** | **64** | **[ 0.0412 0.1094]** | **0.9989** | **0.0462** |
| 100 | 21 | [0.0839 0.0813] | 0.9989 | 0.0297 |
| 100 | 23 | [0.1098 0.0399] | 0.9989 | 0.0318 |
| 100 | 37 | [0.0745x10-6 0.1043x10-6] | 0.9947 | 0.1713 |

Nuevamente se destaca, el caso con los parámetros por defecto que utiliza Matlab para llevar cabo el algoritmo GA. Sin embargo, es evidente el deterioro de la búsqueda del óptimo cuando el número de cromosomas de la población disminuye. También se aprecia que este óptimo se degrada, al disminuir el número de iteraciones (o generaciones) que se escojan. De todas maneras, a partir de una población de 50 o más cromosomas y 21 o más generaciones, el algoritmo es capaz de encontrar el máximo global en tiempos relativamente reducidos.

A modo de ejemplo, en el gráfico de la Fig. 4, se muestra cómo va convergiendo el fitness promedio de la población o función objetivo (-f(x,y) en nuestro caso) al óptimo global conforme van pasando las generaciones (iteraciones) de la población. Además, también se puede apreciar, el mejor cromosoma en cada generación (puntos negros) conforme avanza el número de generaciones. Por último, se debe considerar que los valores del gráfico anterior deben ser multiplicado por -1 ya que el algoritmo **ga** está minimizando la función f(x,y) y lo que se desea es maximizar, o dicho de otro modo, determinar el valor de x,y que hagan máximo la función f(x,y).

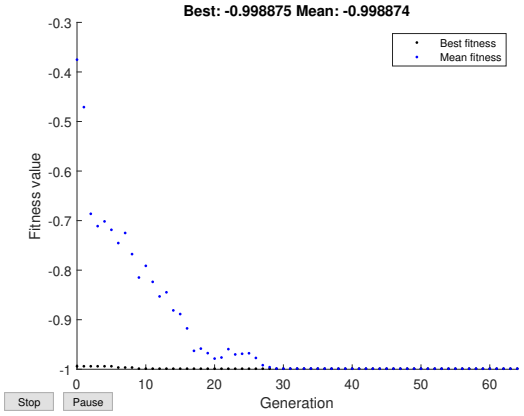



Figura 4: Gráfica del mejor fitness y fitness promedio de la población versus las generaciones

## 2.1. Conclusiones

Lo más relevante, a nuestro parecer es que el método convencional de optimización es muy dependiente del vector de búsqueda inicial, Más aún, este método no es capaz de encontrar el vector óptimo global a menos que el vector de búsqueda inicial este cercano a este. Esta característica, es muy distinta a la capacidad de búsqueda que presentan los algoritmos evolutivos vistos en

este apartado (PSO y GA), a saber, PSO y GA son capaces de identificar y de manera eficiente, es decir, con un costo computacional (tiempo de ejecución del algoritmo) similar e incluso menor, que el método de búsqueda tradicional o clásico, llevado a cabo con la función fmincon.

Por otro lado, si se comparan los algoritmos PSO versus GA, se nota una mejor eficiencia de PSO sobre GA, ya que para alcanzar el óptimo global (0.99891), este algoritmo requiere de un tamaño menor de población (tamaño de enjambre en el caso de PSO) que el caso de GA, lo que se traduce además, en un costo computacional menor, a saber, 20 partículas y 23 iteraciones con un tiempo de ejecución de 0.013 segundos para PSO versus 50 cromosomas y 21 iteraciones con un costo computacional mayor, de 0.0366 segundos para GA.

# 3. Solución problema 2

## 3.1. Contolador Predictivo

En la Fig. 5 se muestra el diagrama utilizado en Simulink para comprobar el funcionamiento del controlador predictivo, implementado en la función de Matlab **Controlador**.

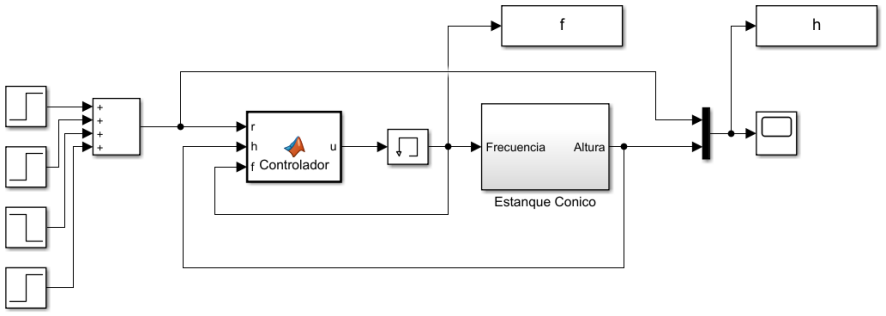



Figura 5: Esquema del controlador Predictivo

La función objetivo a minimizar es:

$$J = A\sum_{i=1}^{N}(h(k+i) - r)^2 + B\sum_{j=1}^{M}(f(k+j))^2 \tag{13}$$

donde utilizando la aproximación de Euler para la derivada temporal se obtiene que:

$$h(k+i) = \frac{5{,}43f(k+i-1) + 78{,}23 - 20{,}21\sqrt{h(k+i-1)}}{0{,}63h(k+i-1)^2 + 11{,}4h(k+i-1) + 17{,}1}\Delta T + h(k+i-1) \tag{14}$$

con $\Delta T = 10 \quad segundos$

resultando finalmente en

$$\min_{[f(k+1),\ldots,f(k+M)]} J \tag{15}$$

cumpliendo las restricciones $h(k+i) \in$ [15,50] y $f(k+j) \in$ [0,100].

### 3.1.1. Optimización por Enjambre de Partículas

Optimización por Enjambre de Partículas (Particle Swarm Optimization, PSO por sus siglas en Inglés) es una metaheurística basada en poblaciones, inspirada en la inteligencia de los enjambres. El mismo imita el comportamiento social de cierta población biológica (insectos, aves, peces, etc.) en la búsqueda de alimentos por ejemplo. Según sus creadores [8] cada individuo (partícula) interactúa con otro intercambiando experiencias, por lo que el grupo se moverá gradualmente dentro de mejores regiones del espacio de búsqueda.

En PSO, la búsqueda de soluciones para cada partícula es definida por la actualización de su velocidad y posición dentro del enjambre, que son establecidas como:

$$\begin{aligned} &V_i^{(j,t)} = \omega V_i^{(j,t-1)} + C_1 r_1 (Pb_i^j - X_i^{(j,t-1)}) + C_2 r_2 (Gb_i - X_i^{(j,t-1)}) \\ &j = 1, \ldots, N \qquad i = 1, \ldots, D \\ &X_i^{(j,t)} = X_i^{(j,t-1)} + V_i^{(j,t)} \end{aligned} \tag{16}$$

donde $i$, $j$ y $t$ son el número de variables a optimizar por el algoritmo, el número de partículas en el enjambre y el número de iteraciones, respectivamente. $V$ y $X$ son la velocidad y la posición de la partícula (solución del problema), $C_1$ y $C_2$ son los coeficientes de aceleración (individual y colectivo), $Pb_i$ es la mejor posición encontrada por la partícula en previas iteraciones, $Gb$ la mejor posición encontrada por la población, $r_1$ y $r_2$ son números aleatorios en el intervalo $[0, 1]$ usados para mantener a diversidad de la población y $\omega$ el peso inercial.

Un alto valor de $\omega$ implica mayor exploración del espacio de búsqueda y largo tiempo de convergencia, mientras que un pequeño valor reduce el tiempo de convergencia pero aumenta el riesgo de caer en un óptimo local. Este parámetro en la función **particleswarm** de Matlab se ajusta mediante la propiedad *InertiaRange* y por defecto está en un rango [0,1; 1,1] lo cual se mantiene en todas las simulaciones para lograr una mayor exploración en los inicios del algoritmo y luego disminuir su tiempo de convergencia. Los coeficientes $C_1$ y $C_2$ se ajustan mediante *SelfAdjustmentWeight* y *SocialAdjustmentWeight* y mantienen su valor por defecto 1.49, valor recomendado en varias publicaciones.

### 3.1.2. Ajuste de los A, B, N, M y parámetros del algoritmo.

En la función de optimización intervienen los parámetros A, B, N y M y a continuación será analizada su influencia en el desempeño del controlador predictivo.

El parámetro N constituye el horizonte máximo de predicción y M el horizonte de control, garantizando que $M \leqslant N$. En aquellos caso donde $M < N$ se establece que

no hay variación de la señal de control propuesta, esta estructuración de la ley de control produce una mejora en la robustez del sistema.

Por su parte A y B son factores de ponderación del error de la señal de salida y la acción de control respectivamente, que son los objetivos de control en este caso, y su ajuste depende de cuál resultado se quiere alcanzar.

Entre los parámetros del algoritmo se analizarán como influye el número de partículas del enjambre *SwarmSize* y el número de iteraciones que puede realizar el algoritmo *MaxIterations*.

1. Caso 1

   Suponiendo N=M=6, que corresponde a un intervalo de predicción de un minuto (dado que el tiempo de muestreo del sistema es 10 segundos), se establece A=B=1 para no potenciar ningún objetivo de control. En este caso se trabaja con 60 partículas (cumple con el valor recomendado de 10*Nvar) y un máximo de 100 iteraciones.

   Para la simulación se utilizaron las siguientes referencias $r_1 = 25$, $r_2 = 35$, $r_3 = 30$ y $r_4 = 45$ (en ese orden de forma secuencial), de tiempo igual a 180 segundos. Teniendo en consideración que el tiempo de vaciado es significativamente más largo que el de llenado, el tiempo de $r_3$ se establece igual a 360 segundos.

   En la Fig. 6 se puede comprobar que el controlador predictivo aún cuando intenta seguir la referencia no logra alcanzarla y la señal de control no llega a un 10 % de su valor, este comportamiento se debe a que el valor de la acción de control se encuentra dentro de la función objetivo a minimizar y por consiguiente el algoritmo de optimización intentará minimizarla.

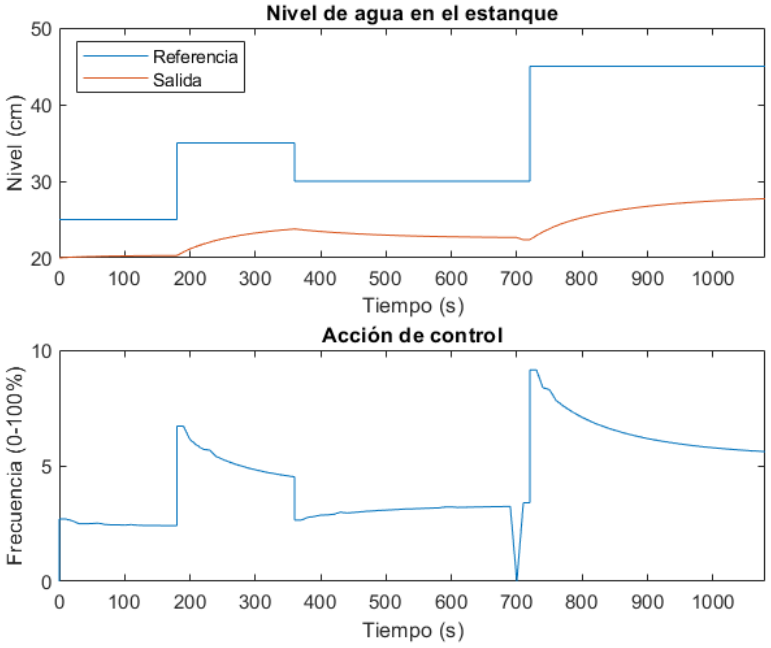



Figura 6: Nivel en el estanque y acción de control. Caso 1

La altura del nivel del agua en el estanque se mueve dentro de un rango de [15,50] cm, su diferencia respecto a la referencia no superará los 35 cm; y la frecuencia en el rango de [0,100] %; por lo tanto el algoritmo de optimización, dada su parametrización actual, penaliza más los valores altos de la frecuencia que errores en la referencia, lo cual no es el comportamiento deseado.

2. Caso 2

   Manteniendo los demás parámetros constantes se establece B=0.01, lo cual reduce el aporte del valor de la señal de control a la función objetivo y A=2, para incrementar en este caso la penalización del error en la salida del sistema.

   La Fig. 7 muestra como en este caso se alcanza la referencia, con bruscos incrementos de la frecuencia hasta encontrar el valor que logra que el flujo de entrada al estaque sea igual al de salida para mantener el nivel constante. Una modificación al control predictivo podría ser incluir dentro de la función objetivo las variaciones en la acción de control.

   En el caso del valor de referencia $r_3 = 30$ donde se debe disminuir el nivel del estanque se puede observar como la bomba se apaga hasta que el flujo de salida logra que el nivel baje para luego encederse nuevamente y regular el nivel deseado.

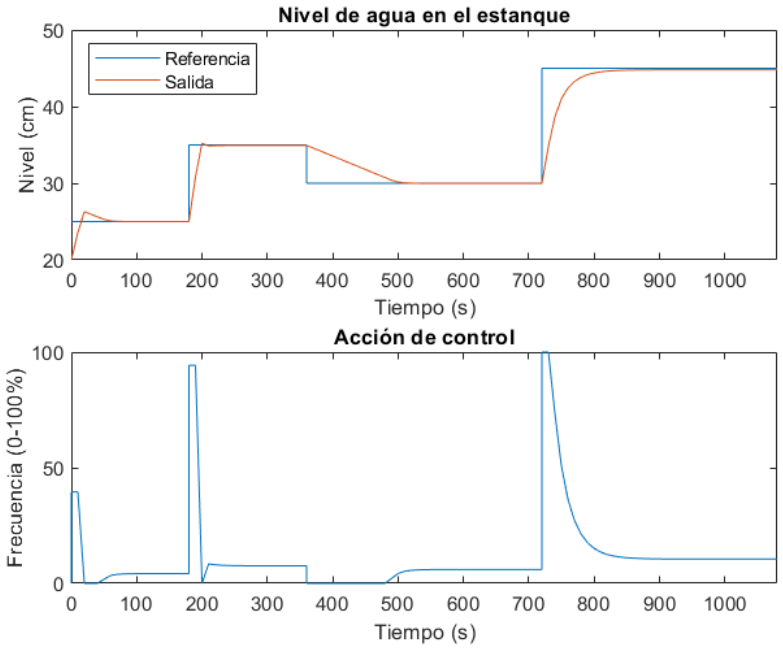



Figura 7: Nivel en el estanque y acción de control. Caso 2

En la Tabla 4 se muestra, para los casos analizados, la raíz del error cuadrático medio (RMSE) y el tiempo de ejecución, demostrando que al ajustar A y B no solo se mejora el desempeño sino que también disminuye el tiempo de ejecución.

$$RMSE = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(y(k) - y_{fuzzy}(k))^2} \tag{17}$$

Tabla 4: Comparación de diferentes parámetros

| Caso | N | M | A | B | Partículas | Iteraciones | RMSE | Tiempo simulación [segundos] |
|---|---|---|---|---|---|---|---|---|
| 1 | 6 | 6 | 1 | 1 | 60 | 100 | 10.8470 | 10.98 |
| 2 | 6 | 6 | 2 | 0.01 | 60 | 100 | 3.0085 | 8.13 |
| 3 | 6 | 2 | 2 | 0.01 | 60 | 100 | 2.9970 | 6.08 |
| 4 | 6 | 2 | 2 | 0.01 | 40 | 100 | 2.9970 | 3.48 |

3. Caso 3

Investigaciones han demostrado que una estructuración de la ley de control produce una mejora en la robustez del sistema. Esta estructura de la ley de control se basa en el uso del concepto de horizonte de control (M) que consiste en suponer que tras un cierto intervalo $M < N$ hay variación en las señales de control propuestas, por lo cual en ese caso se establece M=2.

La Fig. 8 muestra el desempeño ante estos nuevos parámetros y no solo se logra el desempeño anterior sino que se mejora el tiempo de ejecución. Esto se debe a que al ser M=2 se reduce la dimensión da la partícula de 6 a 2 con lo cual disminuye la complejidad del espacio de búsqueda y el algoritmo puede alcanzar de forma más rápida el valor óptimo.

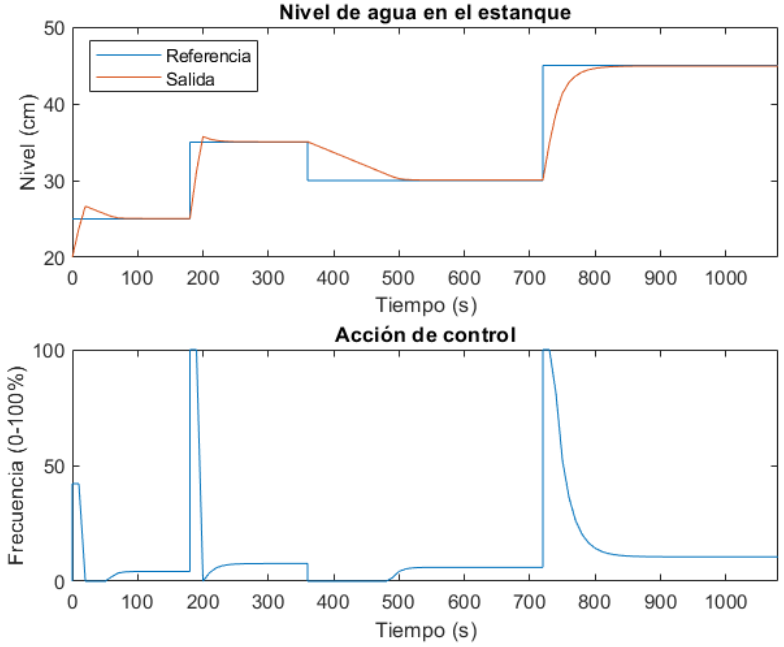



Figura 8: Nivel en el estanque y acción de control. Caso 3

4. Caso 4

En la anteriores simulaciones se ha trabajado con un número de 60 partículas lo cual es un alto número si la dimensión del espacio de búsqueda es 2, por lo cual se reduce a 40 para comprobar su desempeño. El número de iteraciones se mantiene en 100 pues ese no es el único criterio de término del algoritmo, también se define la tolerancia que implica que las iteraciones finalicen cuando la mejora en la solución es inferior al valor definido en *FunctionTolerance*, en este caso $10^{-6}$.

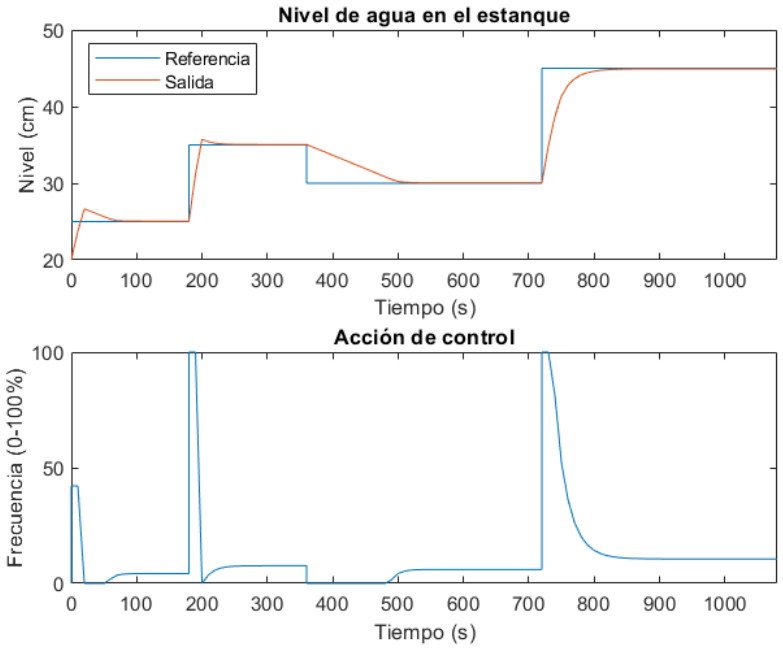



Figura 9: Nivel en el estanque y acción de control. Caso 4

El resultado se muestra en la Fig.9 donde se mantiene el desempeño y se reduce considerablemente el tiempo de simulación como se muestra en la Tabla 4. Si el número de partículas e iteraciones disminuye considerablemente el algoritmo se ve imposibilitado de alcanzar el óptimo y tanto en la salida de la planta como en la acción de control se aprecia un comportamiento oscilante.

# 4. Identificación del modelo

Los datos en el archivo *"DataEstanque.mat"* incorporan el tiempo de muestreo y se puede observar que algunas muestras no fueron tomadas cada 10 segundos (según se indica en la orden del ejercicio), por lo que se realiza un preprocesamiento para extraer del set de datos aquellos que cumplan con la indicación.

Una vez obtenidos los datos experimentales de entrada-salida, éstos son clasificados en tres conjuntos con distinta información: datos de entrenamiento, datos de validación y datos de prueba; esto con el fin de evaluar adecuadamente los modelos generados. El conjunto de entrenamiento se utiliza para determinar los parámetros del modelo. El conjunto de prueba permite comparar distintas estructuras de los modelos generados. Finalmente, el conjunto de validación permite verificar el sobreajuste del modelo óptimo obtenido, evaluándolo en un nuevo conjunto de datos

(distintos a los datos del conjunto de entrenamiento y prueba), analizando su capacidad de generalización. En este caso se utiliza una división de 60 % de los datos para entrenamiento, 20 % para prueba y 20 % validación. La función utilizada para separar los datos son *MuestreoConstante.m* y *createMatrixInput.m*.

## 4.1. Modelo Difuso

Los modelos difusos de Takagi-Sugeno son estructuras basadas en la lógica difusa que permiten representar procesos con dinámicas no lineales mediante la combinación de información otorgada por modelos locales. Estos tipos de modelos pueden ser expresados a partir de una base de reglas del tipo "Si-Entonces", de la forma

$$R_r : Si\ z_1(k)\ es\ MF_1^r\ y ... y\ z_p(k)\ es\ MF_p^r\ entonces\ y_r(z(k)) = f_r(z(k)) \tag{18}$$

donde $R_r$ denota la r-ésima regla del modelo difuso, con $r \in 1, ..., N_r$ y $N_r$ el número total de reglas; $y_r(z(k))$ es su consecuencia o modelo local; $z(k) = [z_1(k), ..., z_p(k)]$ es el vector de premisas en el tiempo $k$, las cuales por lo general son regresores de la entrada y/o salida del sistema; $f_r(z(k))$ es una función de las premisas del modelo; y $MF_i^r$ es el conjunto difuso (función de pertenencia) de la i-ésima premisa correspondientes a la r- ésima regla.

Sea $\mu_r(z_i(k))$ el grado de pertenencia de la i-ésima premisa $z_i(k)$ al conjunto difuso $MF_i^r$, donde $\mu_r(z_i(k)) \in [0, 1]$, siendo 0 cuando la premisa no pertenece en ningún grado al conjunto $MF_i^r$, y siendo 1 si pertenece completamente a dicho conjunto. Luego, se define el grado de activación de la r-ésima regla, $w_r(z(k))$, como

$$w_r(z(k)) = oper(\mu_r(z_1(k)), ..., \mu_r(z_p(k))) \tag{19}$$

donde $oper(.)$ puede ser el operador mínimo o el producto. Se denota $h_r(z(k))$ al grado de activación normalizado de la r-ésima regla, es decir,

$$h_r(z(k)) = \frac{w_r(z(k))}{\sum_{l=1}^{N_r} w_l(z(k))} \tag{20}$$

Ya definido el grado de activación de cada regla, la salida del modelo difuso, $y_{fuzzy}(k)$, está dada por una suma ponderada de cada modelo local por su grado de activación normalizado, de la forma

$$y_{fuzzy}(k) = \sum_{r=1}^{N_r} h_r(z(k)) * y_r(z(k)) \tag{21}$$

Los modelos difusos TS son una clase de sistemas no lineales, cuya formulación requiere definir una serie de variables que no se conocen a priori, por lo que una estructura adecuada para representar un sistema es desconocida y, como consecuencia, un proceso de identificación debe ser llevado a cabo para determinar la estructura y cada uno de los parámetros del modelo [9].

- **Selección de variables**: Para seleccionar las variables que actúan como entrada al sistema difuso, se realiza un análisis de sensibilidad. Suponiendo una estructura del modelo inicial difuso con 12 variables de entrada $y(k-1)$,..., $y(k-6)$, $u(k-1)$,...,$u(k-6)$. En la Fig. 10 se muestran los índices de las sensibilidades para las variables $y(k-1)$, $u(k-2)$, $y(k-3)$ y $u(k-1)$ que presentaron los mayores valores de sensibilidad en el modelo difuso. A pesar de que la sensibilidad de $u(k-1)$ es inferior se decidió utilizar ya que la misma corresponde a la señal de control, parámetro con el que se actúa sobre el modelo del estanque para alcanzar la referencia deseada.

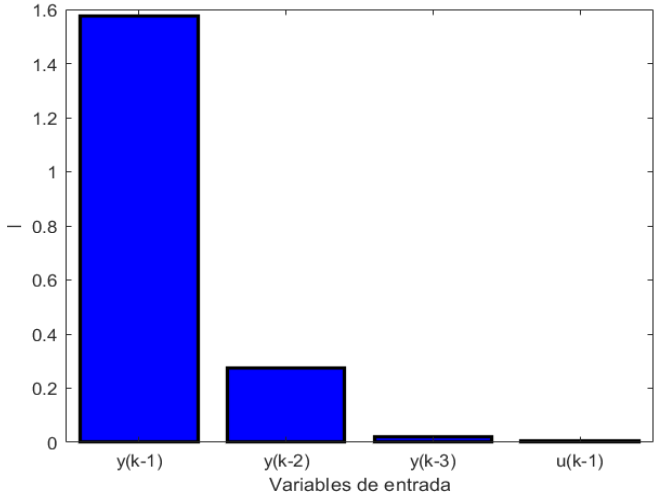



Figura 10: Índice de Sensibilidades.

La Tabla 5 indica el valor de la Raíz del Error Cuadrático Medio (RMSE) para los modelos con 12 y 4 regresores, en el conjunto de prueba, y se puede notar que son muy semejantes, y por consiguiente se selecciona el modelo más sencillo. Recordar que RMSE se define,

$$RMSE = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(y(k) - y_{fuzzy}(k))^2} \tag{22}$$

donde $N$ es la cantidad total de datos, $y(k)$ es la salida de la planta real en

el instante $k$ , e $y_{fuzzy}(k)$ es la predicción realizada por el modelo difuso en el instante $k$.

Tabla 5: Índices de Error para el Análisis de Sensibilidades.

| Modelo | Variables de entrada | RMSE |
|---|---|---|
| 1 | $y(k-1)...,y(k-6)$<br>$u(k-1),...,u(k-6)$ | 0.1345 |
| 2 | $y(k-1),y(k-2),y(k-3,u(k-1)$ | 0.1311 |

- **Optimización de la estructura**: La optimización de la estructura del modelo difuso consiste principalmente en determinar el número óptimo de reglas del modelo difuso. En este caso se definió un número máximo de 15 clusters y se entrenó el modelo para cada una de las posibles valores de clusters, utilizando como algoritmo de clustering el Fuzzy C-Means.

  La Fig. 11 muestra el RMSE para los conjuntos de entrenamiento y prueba. Si no importa la complejidad, el mejor modelo es aquel que tiene menor RMSE. Sin embargo, es posible que un modelo con peor índice de desempeño, pero menos complejo que el modelo óptimo, pueda obtener resultados aceptables bajo un estándar de rendimiento definido preliminarmente. En este problema se escoge como número de clusters 12, por lo que el modelo difuso contará con 12 reglas.

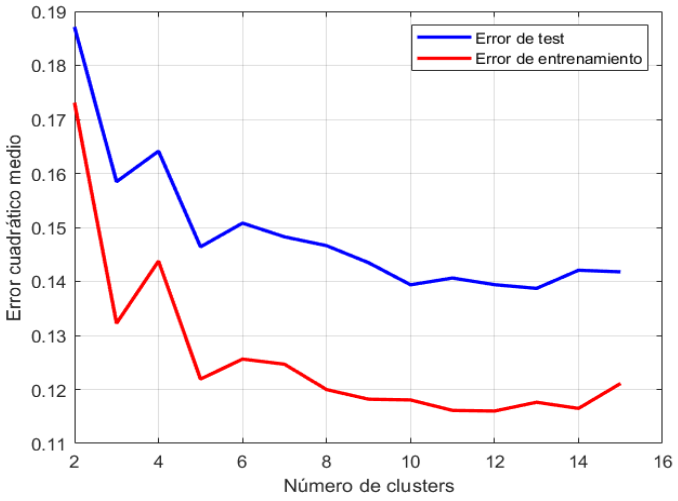



Figura 11: Índice de Sensibilidades.

Una vez realizado el clustering es posible proyectar las agrupaciones en el espacio de entrada y ajustar funciones paramétricas para describir los conjuntos difusos $MF_i^r$. En particular, se consideraron funciones de pertenencia gaussianas dadas por

$$MF_r^i(z_i(k)) = exp(-0{,}5(a_{r,i} * (z_i(k) - b_{r,i}))^2) \tag{23}$$

donde $MF_r^i(z_i(k))$ es la función de pertenencia de la i-ésima premisa $z_i(k)$ al r-ésimo cluster, $a_{r,i}$ es el inverso de la desviación estándar de los datos ajustados por la gaussiana, $b_{r,i}$ representa su media. Para este modelo difuso se tiene:

$$a = \begin{pmatrix} 1{,}6552 & 1{,}6174 & 1{,}5548 & 1{,}4028 \\ 1{,}0192 & 0{,}9643 & 0{,}8938 & 0{,}5231 \\ 1{,}1299 & 1{,}0526 & 0{,}9170 & 0{,}2586 \\ 0{,}7576 & 0{,}7010 & 0{,}6418 & 0{,}1316 \\ 0{,}8773 & 0{,}8212 & 0{,}7544 & 0{,}2725 \\ 1{,}1833 & 1{,}0973 & 0{,}9164 & 0{,}1288 \\ 1{,}0853 & 1{,}0148 & 0{,}9054 & 0{,}3852 \\ 0{,}9801 & 0{,}7794 & 0{,}6142 & 0{,}0659 \\ 1{,}3379 & 1{,}2847 & 1{,}1733 & 0{,}2301 \\ 0{,}9734 & 0{,}9075 & 0{,}8040 & 0{,}3595 \\ 0{,}4629 & 0{,}3511 & 0{,}3024 & 0{,}0448 \\ 1{,}0954 & 0{,}9108 & 0{,}7759 & 0{,}1096 \end{pmatrix} \tag{24}$$

$$b = \begin{pmatrix} 15{,}2880 & 15{,}2969 & 15{,}3094 & 0{,}1097 \\ 18{,}9198 & 18{,}9645 & 19{,}0137 & 1{,}1600 \\ 32{,}1746 & 32{,}1814 & 32{,}1853 & 6{,}4555 \\ 28{,}8128 & 28{,}8150 & 28{,}8137 & 5{,}4593 \\ 35{,}4410 & 35{,}4495 & 35{,}4530 & 7{,}3477 \\ 41{,}0286 & 41{,}0293 & 41{,}0245 & 9{,}6677 \\ 25{,}7435 & 25{,}7730 & 25{,}8019 & 3{,}4380 \\ 48{,}6460 & 48{,}6491 & 48{,}6461 & 11{,}3508 \\ 43{,}3266 & 43{,}3304 & 43{,}3282 & 9{,}92078 \\ 22{,}4641 & 22{,}4854 & 22{,}5068 & 2{,}7554 \\ 37{,}6550 & 37{,}6527 & 37{,}6445 & 9{,}3637 \\ 45{,}9550 & 45{,}9574 & 45{,}9557 & 10{,}6818 \end{pmatrix} \tag{25}$$

Además del RMSE, se pueden definir el Error Porcentual Absoluto Medio (MAPE, Mean Absolute Percentage Error) y Error Absoluto Medio (MAE, Mean Absolute Error) para comprobar el modelo difuso obtenido.

$$MAPE = \frac{1}{N}\sum_{k=1}^{N}\frac{y(k) - y_{fuzzy}(k)}{y(k)} * 100 \tag{26}$$

$$MAE = \frac{1}{N}\sum_{k=1}^{N}|y(k) - y_{fuzzy}(k)| \tag{27}$$

Tabla 6: Índices de Error para el Modelo Difuso

| Métricas | Conjuntos | | |
|---|---|---|---|
| | Entrenamiento | Prueba | Validación |
| RMSE | 0.1160 | 0.1394 | 0.1311 |
| MAPE ( %) | 0.1285 | 0.1426 | 0.1479 |
| MAE | 0.0403 | 0.0441 | 0.0428 |

En la Tabla 6 se puede ver como el RMSE aunque aumenta ligeramente no deteriora el desempeño del modelo difuso para los conjuntos de prueba y validación lo que indica la capacidad de generalización del modelo. Por su parte el índice MAPE, mide el tamaño del error (absoluto) en términos porcentuales, indicando que el error porcentual promedio del modelo se encuentra alrededor del 0.15 %, lo cual da una medida de la certeza de la predicción del modelo, este valor inferior a un 1 % se considera un modelo válido. El MAE tampoco presenta variaciones entre los diferentes conjuntos de datos.

- **Optimización de los parámetros**: Se utiliza el algortimo de clustering difuso Fuzzy C-Means obteniéndose siguientes parámetros de los consecuentes $\theta^T = [\theta_{1,1}, ..., \theta_{N_r,1}, ..., \theta_{1,n}, ..., \theta_{N_r,1}]$

$$\theta^T = \begin{pmatrix} 0{,}8448 & 1{,}2886 & -0{,}5079 & 0{,}1632 & 0{,}1531 \\ 0{,}5988 & 1{,}0827 & -0{,}0933 & -0{,}02922 & 0{,}1102 \\ 1{,}6975 & 1{,}5325 & -0{,}6616 & 0{,}0697 & 0{,}0298 \\ 0{,}5810 & 0{,}8073 & 0{,}1910 & -0{,}0319 & 0{,}0701 \\ -0{,}3188 & 3{,}8166 & -2{,}6874 & -0{,}1287 & 0{,}0128 \\ 4{,}4026 & 1{,}5923 & -0{,}5386 & -0{,}1592 & 0{,}0012 \\ 0{,}2338 & 1{,}2464 & 0{,}2135 & -0{,}4718 & 0{,}0164 \\ 0{,}1921 & 0{,}7150 & 0{,}2929 & -0{,}0176 & 0{,}0235 \\ 1{,}1283 & 0{,}9624 & 0{,}01826 & -0{,}0136 & 0{,}0305 \\ 0{,}1505 & 1{,}0226 & -0{,}0155 & -0{,}0249 & 0{,}0837 \\ 5{,}0904 & 0{,}7690 & 0{,}0439 & 0{,}0429 & 0{,}0411 \\ 0{,}1121 & 0{,}7247 & 0{,}2657 & -0{,}0016 & 0{,}0371 \end{pmatrix} \tag{28}$$

Considerando un compromiso entre complejidad y desempeño se determina que el modelo propuesto con 4 regresores y 12 reglas es suficiente. La Fig. 12 muestra la salida del modelo difuso para una muestra del conjunto de validación.

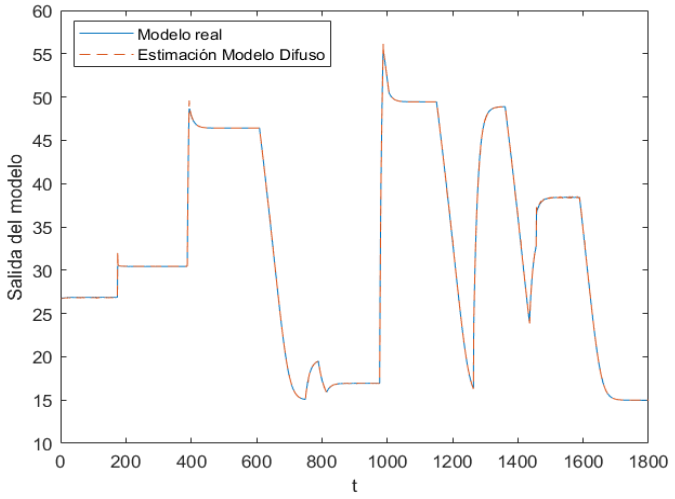


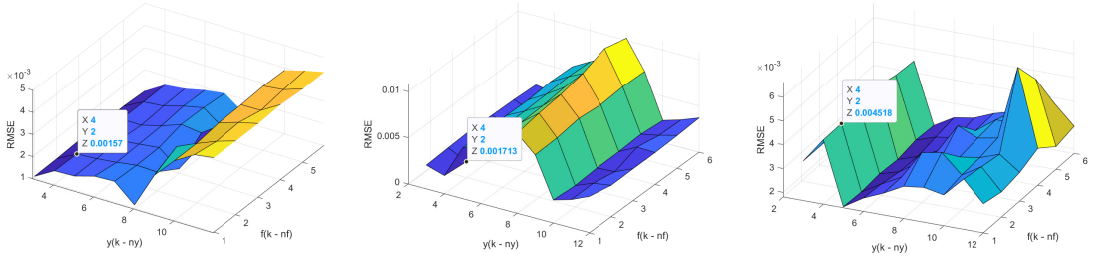

Figura 12: Salida del modelo.

## 4.2. Modelo Neuronal

El proceso de modelado con redes neuronales esta establecido por la identificación de la estructura misma de la red. Lo primer es hacer una inspección de los datos para considerar solo aquellos que fueron tomados cada 10 segundos, cuya amplitud de nivel están entre 15-50 cm de altura y cuya frecuencia esta entre 0-100 %.

Posteriormente, se debe clasificar cuantos auto-regresores $y(k-i)$ y regresores $f(k-i)$ se deben considerar para mejorar el resultado final del modelo. Para esto se inicia el modelo con un rango de auto-regresores y regresores en el cual se itera con un número de neuronas constante en la función llamada *RegressorsSelection.m*. Luego, la combinación que de menor RMSE es guardado en un archivo llamado *best_regressor.mat* para evitar volver a calcular nuevamente los parámetros de la estructura inicial. En la Figura 13 se muestran los valores RMSE para diferente combinación de auto-regresores y regresores donde el menor se obtiene para 4 auto-regresores del modelo y 2 regresores de control con un RMSE de 0.0017.

Con la combinación de entradas que mejora el valor RMSE en el conjunto de prueba, se procede a encontrar el numero óptimo de neuronas en un rango establecido. Esto se




(a) Entrenamiento (b) Prueba (c) Validación

Figura 13: RMSE para diferente combinación de variables de entradas en los 3 conjuntos.

hace mediante la función *Identification.m*, la cual itera de $2-21$ neuronas calculando en cada iteración el índice de sensibilidad para una red con un capa oculta de función de activación tanh. Luego, se fija el numero de neuronas en la capa oculta igual a 6.

Como ya se tiene un número de neuronas óptimas en base a los primeros regresores calculados, se puede volver a iniciar un análisis de selección de regresores con el fin de buscar el menor valor RMSE en el modelo ya que el primero fue un análisis con un numero de neuronas cualquiera, en este caso igual a la media entre los rangos de neuronas definidos. Sin embargo, este proceso se iteró varias veces mostrando que, si bien el numero de regresores cambiada desde 4 a 10 entradas, el valor RMSE se mantiene casi constante y depende en mayor medida del número de neuronas y los parámetros de entrenamiento. Los resultados de sensibilidad para el numero de neuronas óptimo encontrado se muestra en la Figura 14 y el modelo obtenido con el conjunto de validación se muestra en la Figura 15. En la Tabla 7 se reagrupan algunos índices de error para hacer posible la comparación con el modelo de T&S. Como se puede ver, la diferencia es leve y ambos modelo permiten representar el comportamiento de la planta. Conclusiones más relevantes respecto a controladores se verán más adelante.

Tabla 7: Índices de Error para el Modelo Neuronal

| Métricas | Conjuntos | | |
|---|---|---|---|
| | Entrenamiento | Prueba | Validación |
| RMSE | 0.0017 | 0.0022 | 0.0047 |
| MAPE (%) | 0.1591 | 0.1774 | 0.2258 |
| MAE | 0.0414 | 0.0486 | 0.0560 |

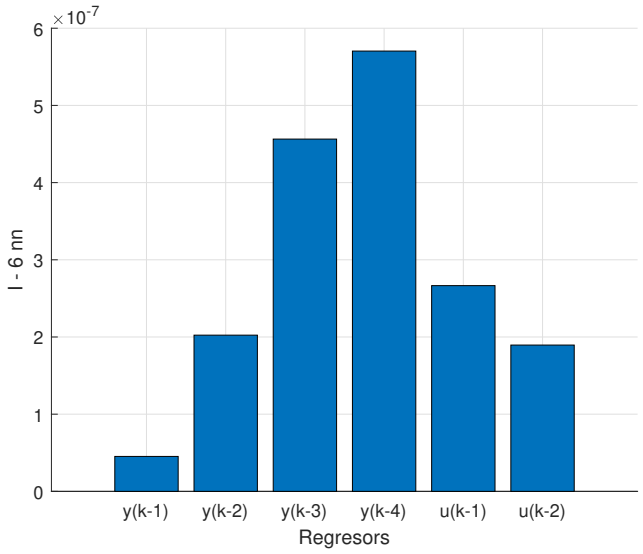



Figura 14: Índice de sensibilidad de cada regresor para el número de neurona óptima.

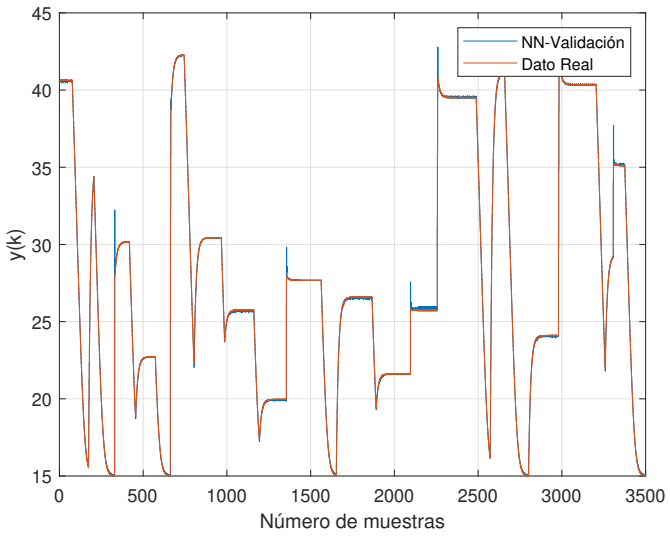



Figura 15: Resultado del modelo neuronal en el conjunto de validación.

# Referencias

[1] D. S. H., "Apuntes el7012 – control inteligente de sistemas, modelación difusa," 2016.

[2] ——, "Apuntes el7012 – control inteligente de sistemas, modelación neuronal," 2016.

[3] ——, "Apuntes el7012 – control inteligente de sistemas, algoritmos evolutivos," 2016.

[4] ——, "Apuntes el7012 – control inteligente de sistemas, control difuso," 2016.

[5] E. Obreque, "El7012-tarea2." [Online]. Available: https://github.com/EliasObreque/EL7012-Tarea2

[6] Matlab, "fmincon." [Online]. Available: https://www.mathworks.com/help/optim/ug/fmincon.html.

[7] ——, "particleswarm, particle swarm optimization." [Online]. Available: https://www.mathworks.com/help/gads/particleswarm.html#d120e51319

[8] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," in *IEEE International Conference on Neural Networks*, Perth, 1995, pp. 1942–1948.

[9] G. Alvarez, "Metodología de Identificación Difusa Basada en el Estudio de Controlabilidad de Sistemas Dinámicos," p. 172.

# Anexo

### ControladorOpt.m

```
function [u] = ControladorOpt(r,h,f)
tm=10;
A=1;
B=1;
N=10;
M=10; %Se debe cumplir M<=N

%Optimizaci n
lb=zeros(1,M);
ub=100 * ones(1, M);%**Vectores fila
nvars = N;
fun = @(x)myfun(x,r,h,f,tm,A,B,N,M);
options = optimoptions('particleswarm','SwarmSize',200,'
    MaxIterations',1000);
[F,fval,exitflag,output]= particleswarm(fun, nvars, lb, ub,
    options);
u=F(1);
end

function [J] = myfun(F,r,h,f,tm,A,B,N,M)

H(1)=(5.43*f+78.23-20.21*sqrt(h))/(0.63*h^2+11.4*h+17.1)*tm
    + h;

J=0;
for i=1:N
    J = J + A*(H(i)-r)^2;
    H(i+1)=(5.43*F(i)+78.23-20.21*sqrt(H(i)))/(0.63*(H(i))
        ^2+11.4*H(i)+17.1)*tm + H(i);
    if i <= M
        J = J + B*F(i)^2;
    end
end
end
```

**createData.m**

```
%Script para el preprocesamiento de los datos y creaci n de los conjuntos
%de Entrenamiento, Prueba y Validaci n

clear all, clc
load('DataEstanque.mat')
%
%%%%%----------Contrucci n del vector de datos ----------
[Ref,Entrada,Salida,Tiempo] = MuestreoConstante(Ref,Entrada,Salida,Tiempo);
%60% para entrenamiento
%20% test y 20% validaci n
Dt=69990;           %Tama o del vector
%y=normalizar(Salida,[15 50],[0 1]);  %Normalizar la altura de [15 55] a [0 1]
%u=normalizar(Entrada,[0 100],[0 1]); %Normalizar le frecuencia de [0 100] a [0 1]
y=Salida;
u=Entrada;
ry=8;          %Numero de regresores en y
ru=2;          %Numero de regresores en u
[X, Y] = createMatrixInput(Dt, ry, ru, y, u);

le=Dt*0.6;        %Limites de los conjuntos de entrenamiento y prueba
lp=Dt*(0.6+0.2);

Xent = X(1:le,:);
Yent = Y(1:le,:);
Xtest = X(le+1:lp, :);
Ytest = Y(le+1:lp, :);
Xval = X(lp+1:Dt, :);
Yval = Y(lp+1:Dt, :);

savefile = 'DatosProblema2_r10.mat';
save(savefile, 'Xent', 'Xtest','Xval', 'Yent', 'Ytest', 'Yval');
```

**fun_min.m**

```
function fxy_min = fun_min(x)

 fxy_min=-exp(-2*log10(2)*(((sqrt(x(1)^2+x(2)^2)-0.08)
    /0.854)^2)).....
    *sin(5*pi*(sqrt(x(1)^2+x(2)^2)^0.75-0.1))^2;

end
```

**P1_Ejercicio2_final.m**

```
clear all;clc;
x=0:0.01:1; y=0:0.01:1;
[X,Y]=meshgrid(x,y);
fxy=exp(-2*log10(2)*(((sqrt(X.^2+Y.^2)-0.08)/0.854).^2)).....
    .*sin(5*pi*(sqrt(X.^2+Y.^2).^0.75-0.1)).^2;

%Determinar el maximo de f(x,y)
figure(1)
mesh(X,Y,fxy)
xlabel('x')
ylabel('y')
zlabel('f(x,y)')

%Es equivalente a determinar el minimo de -f(x,y)
fxy_min=-fxy;
figure(2)
mesh(X,Y,fxy_min)
xlabel('x')
ylabel('y')
zlabel('-f(x,y)')

lb=[0,0];ub=[1,1];%**Vectores fila
x0 = (lb + ub)/2;%X0=[0.5  0.5]

%x = fmincon(fun_min,x0,A,b,Aeq,beq,lb,ub)
tic
[x,fval_1] = fmincon(@fun_min,x0,[],[],[],[],lb,ub);
t1_ejecucion=toc%    t1_computo=0.7070 seg
%fval_1=-0.6972 para [x*,y*]=[0.5228, 0.5228]
%Puedo evaluar fun(x) con x como vector fila o columna
```

```
x
f_max1=fun_max([0.5228;0.5228])
fmx1=fun_max(x)
fmax1=-fval_1 %Todas estas funciones son equivalentes

x0=[0  0];
%options = optimoptions(@fmincon,'Display','iter','Algorithm
    ','sqp');
% tic
% [x,fval_2] = fmincon(@fun_min,x0,[],[],[],[],lb,ub,[],
    options);
% t2_ejecucion=toc %    t2_computo=0.5648 seg
[x,fval_2] = fmincon(@fun_min,x0,[],[],[],[],lb,ub);
x
fmax2=-fval_2

x0=[0.2  0.2];
tic
[x,fval_3] = fmincon(@fun_min,x0,[],[],[],[],lb,ub);
t3_ejecucion=toc %    t1_computo
x
fmax3=-fval_3


x0=[0.8  0.8]; %Son puntos iniciales No sim tricos
tic
[x,fval_4] = fmincon(@fun_min,x0,[],[],[],[],lb,ub);
t4_ejecucion=toc %
x
fmax4=-fval_4

%Conclusion: Dependiendo del punto de busqueda inicial X0
    vector
%se obtienen diferentes optimos ya que f(x,y) tiene
%muchos m ximos locales

%***PSO***
%Por defecto, el numero de particulas del enjambre es el
    minimo
%de 100 y 10*nvar, es decir: SwarmSize =min(100, 10*nvars)
```

```
67 %y por defecto, el maximo de iteraciones es 200*nvars
68 lb=[0,0];ub=[1,1];%**Vectores fila
69 nvars = 2;
70 % tic
71 %x = particleswarm(@fun_min,nvars,lb,ub)
72 %%**En una corrida me dio [x* y*]=[0 0] !!!
73 % t_pso1=toc
74
75 %Lo mismo anterior pero con mas informacion de la salida
76 tic
77 [x,fval,exitflag,output]=particleswarm(@fun_min,nvars,lb,ub)
       ;
78 t_pso1=toc
79 x
80 fmax_pso1=-fval
81
82 options1 = optimoptions('particleswarm','SwarmSize',100);
83 tic
84 [x,fval,exitflag,output]=particleswarm(@fun_min,nvars,lb,ub,
       options1);
85 t_pso2=toc
86 x
87 fmax_pso2=-fval
88
89 %Otra con maximo numero de iteraciones (apenas 50): default
       es 200*nvars
90 tic
91 options2 = optimoptions('particleswarm','SwarmSize',5,'
       MaxIterations',23);
92 [x,fval,exitflag,output]=particleswarm(@fun_min,nvars,lb,ub,
       options2);
93 t_pso3=toc
94 x
95 fmax_pso3=-fval
96 output
97
98
99 %****Genetic Algorithm**********************
100 nvars = 2;
101
```

```
tic
[x,fval,exitflag,output,population,scores] = ga(@fun_min,
    nvars);
t_ga1=toc
x
fmax_ga1=-fval

%**Otra con cotas inferiores y superiores
lb=[0,0];ub=[1,1];%**Vectores fila
tic
[x,fval,exitflag,output,population,scores] = ga(@fun_min,
    nvars,[],[],[],[],lb,ub);
t_ga2=toc
x
fmax_ga2=-fval

%**Otra con despliegue
%options = optimoptions('ga','PlotFcn', @gaplotbestf,'
    PopulationSize',50,'MaxGenerations',64);

options = optimoptions('ga','PopulationSize',20,'
    MaxGenerations',37);
lb=[0,0];ub=[1,1];%**Vectores fila
tic
[x,fval,exitflag,output,population,scores] = ga(@fun_min,
    nvars,[],[],[],[],lb,ub,[],[],options);
t_ga3=toc
x
fmax_ga3=-fval
output
```

**NeuralNetwork.m**

```
function [net_trained, tr] = NeuralNetwork(num_neu, x_train, y_train,...
    x_test, y_test, x_val, y_val)

% Elias Obreque
% els.obrq@gmail.com

global train_prc test_prc val_prc

x_data = [x_train; x_test; x_val];
y_data = [y_train; y_test; y_val];

total_length = length(y_data);

% Crea una capa oculta en la red
net_fit = fitnet(num_neu,'trainlm');

[trainInd,valInd,testInd] = divideblock(total_length, train_prc, val_prc, test_prc);

% Funcion de activacion
net_fit.layers{1}.transferFcn = 'tansig'; % tansig = tanh
net_fit.divideFcn = 'divideind';
net_fit.divideParam.trainInd = trainInd;
net_fit.divideParam.testInd = testInd;
net_fit.divideParam.valInd = valInd;


net_fit.trainParam.max_fail = 50;

%TRAINING PARAMETERS
net_fit.trainParam.show = 200;      %# of ephocs in display
net_fit.trainParam.lr = 0.05;       %learning rate
net_fit.trainParam.epochs = 5000;   %max epochs
%net_fit.trainParam.goal=0.05^2;    %training goal

%Name of a network performance function %type help nnperformance
net_fit.performFcn = 'mse';
```

```
37
38 [net_trained, tr] = train(net_fit, x_data', y_data');
39 return
```

**CreateMatrix.m**

```
1  function [X, Y] = createMatrixInput(Dt, ry, ru, y_model, u,
       range_y_m, range_u_m, normalize)
2  %CREATEMATRIXINPUT Summary of this function goes here
3  %-------------Contrucci n del vector de datos
4  min_amp_y_m = range_y_m(1);
5  max_amp_y_m = range_y_m(2);
6
7  min_amp_u_m = range_u_m(1);
8  max_amp_u_m = range_u_m(2);
9
10 g_lower = y_model(y_model>min_amp_y_m);
11 u_lower = u(y_model>min_amp_y_m);
12 new_y_m = g_lower(g_lower<max_amp_y_m);
13 new_u   = u_lower(g_lower<max_amp_y_m);
14 if normalize == 1
15     new_y_m = new_y_m/max_amp_y_m;
16     new_u = new_u/max_amp_u_m;
17 end
18
19
20 f  = length(new_y_m);
21 Y  = zeros(Dt,1);
22 X  = zeros(Dt, ry + ru);
23 for i = f: -1 :f - Dt + 1
24     Y(Dt, 1) = new_y_m(i);
25     %Regresores de y
26         for j = 1:ry
27             X(Dt, j) = new_y_m(i - j);
28         end
29     %Regresores de u
30         for j = 1:ru
31             X(Dt, j + ry) = new_u(i - j);
32         end
33     Dt = Dt-1;
34 end
```

```
35 end
```

**MuestreoConstante.m**

```
1  function [r1,f1,h1,t1] = MuestreoConstante(r,f,h,t)
2  %Extaer del vector de datos los valores tomados con tiempo
       de muestreo 10 s
3  r1 = r(mod(t, 10)==0);
4  f1 = f(mod(t, 10)==0);
5  h1 = h(mod(t, 10)==0);
6  t1 = t(mod(t, 10)==0);
7  end
```

**Identification.m**

```
1  function [best_hidden_neurons] = Identification(
       best_auto_reg, best_reg, x_train, y_train, x_test,...
2          y_test, x_val, y_val, num_neu, normalize)
3  % Elias Obreque
4  % els.obrq@gmail.com
5
6  num_neu_min = num_neu(1);
7  num_neu_max = num_neu(2);
8
9  max_row = 5; max_col = 4;
10
11 I = zeros(num_neu_max - num_neu_min, best_auto_reg +
       best_reg);
12
13 rmse_train = zeros(1, num_neu_max - num_neu_min);
14 rmse_test  = zeros(1, num_neu_max - num_neu_min);
15 rmse_val   = zeros(1, num_neu_max - num_neu_min);
16
17 for num_neu = num_neu_min: 1: num_neu_max
18     %NEURALNETWORK
19     [net_trained, tr] = NeuralNetwork(num_neu, x_train,
           y_train, x_test,...
20         y_test, x_val, y_val);
21
22     y_train_nn = net_trained(x_train');
23     y_test_nn = net_trained(x_test');
```

```
24      y_val_nn = net_trained(x_val');
25
26      %Sensitivity analysis
27      %Datos de entrenamiento
28      I(num_neu, :) = SensitivityCalc('tanh', best_auto_reg +
            best_reg, x_train, net_trained);
29
30      %RMSE
31      rmse_train(num_neu - num_neu_min + 1) = RMSE(y_train,
            y_train_nn');
32      rmse_test(num_neu - num_neu_min + 1)  = RMSE(y_test,
            y_test_nn');
33      rmse_val(num_neu - num_neu_min + 1)   = RMSE(y_val,
            y_val_nn');
34  end
35  %%
36  fig1 = figure('Name', 'Indicator by number of neurons', '
        Position', [100 50 600 600]);
37  if (num_neu_max > max_row*max_col + 1)
38      fig2 = figure('Name', 'Indicator by number of neurons');
39  end
40  for num_neu = num_neu_min: num_neu_max
41      if num_neu <=21
42          figure(fig1)
43          if (num_neu - num_neu_min + 1) == max_col
44              ax11 = subplot(max_row, max_col, num_neu -
                    num_neu_min + 1);
45          elseif (num_neu - num_neu_min + 1) == (1 + max_col*(
                max_row - 1))
46              ax12 = subplot(max_row, max_col, num_neu -
                    num_neu_min + 1);
47          end
48          subplot(max_row, max_col, num_neu - num_neu_min + 1)
49      else
50          figure(fig2)
51          if (num_neu - num_neu_min + 1) == max_col + max_col*
                max_row
52              ax21 = subplot(max_row, max_col,  num_neu -
                    max_row*max_col - 1);
53          elseif (num_neu - num_neu_min + 1) == (1 + max_col*(
```

```
            max_row - 1)) + max_col*max_row
54              ax22 = subplot(max_row, max_col, num_neu -
                    max_row*max_col - 1);
55          end
56          subplot(max_row, max_col, num_neu - max_row*max_col
                - 1)
57      end
58      hold on
59      grid on
60      xlabel('Regresors')
61      bar(I(num_neu, :))
62      ylabel(join(['I-', num2str(num_neu), 'N_h']))
63  end
64  %%
65  fig_rmse = figure('Renderer', 'painters', 'Units', '
        centimeters', 'Position', [4 4 18 10])
66  axes_rmse = axes('Parent', fig_rmse);
67  hold on
68  grid on
69  %set(gca, 'YScale', 'log')
70  ylabel('RMSE')
71  xlabel('N mero de neuronas')
72  [n1, m1] = min(rmse_train);
73  plot(m1 + num_neu_min - 1, n1, '*b', 'DisplayName', 'min-
        Train')
74  [n2, m2] = min(rmse_test);
75  best_hidden_neurons = m2 + num_neu_min - 1;
76  plot(best_hidden_neurons, n2, '*r', 'DisplayName', 'min-Test
        ')
77  [n3, m3] = min(rmse_val);
78  plot(m3 + num_neu_min - 1, n3, '*k', 'DisplayName', 'min-
        Validation')
79  plot(num_neu_min:num_neu_max, rmse_train, 'b','DisplayName',
        'Train')
80  plot(num_neu_min:num_neu_max, rmse_test, 'r','DisplayName','
        Test')
81  plot(num_neu_min:num_neu_max, rmse_val, 'k','DisplayName','
        Validation')
82  legend()
83  %InSet_rmse = get(axes_rmse, 'TightInset');
```

```
84  %set(gca(fig_rmse), 'Position', [InSet_rmse(1:2), 1-
        InSet_rmse(1)-InSet_rmse(3), 1-InSet_rmse(2)-InSet_rmse
        (4)]);
85  %create a new pair of axes inside current figure
86  %axes('position',[.23 .5 .5 .42])
87  %box on %put box around new pair of axes
88  %plot(num_neu_min:num_neu_max, rmse_test, 'r')
89  %hold on
90  %grid on
91  %plot(m2 + num_neu_min - 1, n2, '*r')
92  %ylim([2.5e-3, 3e-3])
93  %xlim([2, 41])
94  %%PLOT
95
96  %plots error vs. epoch for the training, validation, and
        test performances of the training record TR
97  figure()
98  plotperform(tr)
99  %Plot training state values
100 figure()
101 plottrainstate(tr)
102
103 best_hidden_neurons_mat = 'best_hidden_neurons.mat';
104 if normalize == 1
105     best_hidden_neurons_mat = sprintf('%s%s%s', '
          best_hidden_neurons','_normalized','.mat');
106 end
107 save(best_hidden_neurons_mat, 'best_hidden_neurons')
108 return
```

**RegressorsSelection.m**

```
1  function [best_auto_reg, best_reg, best_X, best_Y] =
       RegressorsSelection(num_auto_reg,...
2      num_reg, y_m, u_m, range_y_m, range_u_m, num_neu,
           normalize)
3
4  global train_prc test_prc val_prc
5
6  min_auto_reg = num_auto_reg(1);
7  max_auto_reg = num_auto_reg(2);
```

```
8
9  min_reg = num_reg(1);
10 max_reg = num_reg(2);
11
12 num_neu_min = num_neu(1);
13 num_neu_max = num_neu(2);
14 num_neu = floor((num_neu_max + num_neu_min)/2);
15
16 best_test_rmse = 100;
17 best_auto_reg = 0;
18 best_reg = 0;
19 RMSE_train = zeros(max_reg - min_reg, max_auto_reg -
      min_auto_reg);
20 RMSE_test = zeros(max_reg - min_reg, max_auto_reg -
      min_auto_reg);
21 RMSE_val = zeros(max_reg - min_reg, max_auto_reg -
      min_auto_reg);
22 best_X = struct('x_train', 0, 'x_test', 0, 'x_val', 0);
23 best_Y = struct('y_train', 0, 'y_test', 0, 'y_val', 0);
24 for reg_y = min_auto_reg:max_auto_reg
25     for reg_u = min_reg:max_reg
26         total_data = length(y_m);
27         Dt = floor(total_data/reg_y);
28         [X, Y] = createMatrixInput(Dt, reg_y, reg_u, y_m,
              u_m, range_y_m, range_u_m, normalize);
29
30         [trainInd, testInd, valInd] = divideblock(Dt,
              train_prc, test_prc, val_prc);
31
32         x_train = X(trainInd, :);
33         y_train = Y(trainInd);
34         x_test  = X(testInd, :);
35         y_test  = Y(testInd);
36         x_val   = X(valInd, :);
37         y_val   = Y(valInd);
38
39                 %NEURALNETWORK
40         [net_trained, tr] = NeuralNetwork(num_neu, x_train,
              y_train,...
41             x_test, y_test, x_val, y_val);
```

```
42
43          y_train_nn = net_trained(x_train');
44          y_test_nn = net_trained(x_test');
45          y_val_nn = net_trained(x_val');
46
47          %RMSE
48          rmse_train = RMSE(y_train, y_train_nn');
49          rmse_test  = RMSE(y_test, y_test_nn')
50          rmse_val   = RMSE(y_val, y_val_nn');
51          RMSE_train(reg_u-min_reg + 1, reg_y - min_auto_reg +
                1) = rmse_train;
52          RMSE_test(reg_u-min_reg + 1, reg_y - min_auto_reg +
               1) = rmse_test;
53          RMSE_val(reg_u-min_reg + 1, reg_y - min_auto_reg +
               1) = rmse_val;
54
55          if rmse_test < best_test_rmse
56              disp(['RMSE train: ', num2str(rmse_train), ',
                   RMSE test: ',...
57                  num2str(rmse_test), ', RMSE: ', num2str(
                       rmse_val)])
58              best_test_rmse = rmse_test;
59              best_auto_reg = reg_y;
60              best_reg = reg_u;
61              best_X.x_train = x_train;
62              best_X.x_test = x_test;
63              best_X.x_val = x_val;
64
65              best_Y.y_train = y_train;
66              best_Y.y_test = y_test;
67              best_Y.y_val = y_val;
68
69              disp(['Best auto-reg: ', num2str(best_auto_reg)
                    ,...
70                  ', Best reg: ', num2str(best_reg)])
71          end
72      end
73  end
74
75  fig_1 = figure('Name', 'RMSE for Train');
```

```
76  hold on
77  grid on
78  xlabel('y(k - ny)')
79  ylabel('f(k - nf)')
80  zlabel('RMSE')
81  [X,Y] = meshgrid(min_auto_reg:max_auto_reg, min_reg:max_reg)
        ;
82  surf(X, Y, RMSE_train)
83
84  fig_2 = figure('Name', 'RMSE for Test');
85  hold on
86  grid on
87  xlabel('y(k - ny)')
88  ylabel('f(k - nf)')
89  zlabel('RMSE')
90  [X,Y] = meshgrid(min_auto_reg:max_auto_reg, min_reg:max_reg)
        ;
91  surf(X, Y, RMSE_test)
92
93  fig_3 = figure('Name', 'RMSE for Validation');
94  hold on
95  grid on
96  xlabel('y(k - ny)')
97  ylabel('f(k - nf)')
98  zlabel('RMSE')
99  [X,Y] = meshgrid(min_auto_reg:max_auto_reg, min_reg:max_reg)
        ;
100 surf(X, Y, RMSE_val)
101
102
103 best_regressor_mat = 'best_regressor.mat';
104 if normalize == 1
105     best_regressor_mat = sprintf('%s%s%s', 'best_regressor','
            _normalized','.mat');
106 end
107 save(best_regressor_mat,'best_test_rmse','best_auto_reg', '
        best_reg',...
108      'num_neu', 'best_Y', 'best_X')
109 end
```

**ModelbasedonNeuralNetworks.m**

```
% Elias Obreque
% els.obrq@gmail.com
clc
clear all
close all

%-------------Generar Datos del modelo-------------
global train_prc test_prc val_prc
loaded_data = load('DataEstanque.mat');

y_m = loaded_data.Salida;
u_m = loaded_data.Entrada;
ref_m = loaded_data.Ref;
time_m = loaded_data.Tiempo;

normalize = 0;

best_regressor_mat = 'best_regressor.mat';
best_hidden_neurons_mat = 'best_hidden_neurons.mat';
NN_model_mat = 'NN_model.mat';

if normalize == 1
    best_regressor_mat = sprintf('%s%s%s', 'best_regressor','
        _normalized','.mat');
    best_hidden_neurons_mat = sprintf('%s%s%s', '
        best_hidden_neurons','_normalized','.mat');
    NN_model_mat = sprintf('%s%s%s', 'NN_model','_normalized'
        ,'.mat');
end

[y_m, u_m, ref_m, time_m] = MuestreoConstante(y_m, u_m,
    ref_m, time_m);

figure ()
hold on
plot(y_m)
plot(u_m)
grid on
xlim([0 2000])
```

```
36 xlabel('N mero de muestras')
37 ylabel('Salida del modelo')
38 legend('h(k)', 'u(k)')
39
40 %minimum and maximum value of auto-regresors
41 num_auto_reg = [3 12];
42 %minimum and maximum value of regresors
43 num_reg = [1, 6];
44 %range of hidden neurons
45 num_neurons = [2, 21];
46 %range of amplitude of y_model
47 range_y_m = [15, 50];
48 %range of amplitude of u_model
49 range_u_m = [0, 100];
50
51 train_prc = 0.6;
52 test_prc = 0.2;
53 val_prc = 0.2;
54
55 %% Regressors Selection
56
57 if exist(best_regressor_mat, 'file') == 0
58     [best_auto_reg, best_reg, best_X, best_Y] =
          RegressorsSelection(num_auto_reg,...
59          num_reg, y_m, u_m, range_y_m, range_u_m, num_neurons
               , normalize);
60 else
61     load(best_regressor_mat)
62 end
63
64 x_train = best_X.x_train;
65 x_test = best_X.x_test;
66 x_val = best_X.x_val;
67
68 y_train = best_Y.y_train;
69 y_test = best_Y.y_test;
70 y_val = best_Y.y_val;
71
72 clear best_X best_Y num_neu best_train_rmse
73
```

```
74 %% Optimal hidden neurons
75 if exist(best_hidden_neurons_mat, 'file')==0
76     [best_hidden_neurons] = Identification(best_auto_reg,
       best_reg, x_train, y_train, x_test,...
77         y_test, x_val, y_val, num_neurons, normalize);
78 else
79     load(best_hidden_neurons_mat)
80 end
81 NUM_OPT_NEU = best_hidden_neurons;
82
83 %% NEURAL NETWORK
84
85 [net_trained, tr] = NeuralNetwork(NUM_OPT_NEU, x_train,
     y_train, x_test, y_test, x_val, y_val);
86 save(NN_model_mat, 'net_trained', 'tr')
87
88 %% Sensitivity analysis
89 I = SensitivityCalc('tanh', best_auto_reg + best_reg, x_test
     , net_trained);
90 regressors_name = cell(1, best_auto_reg + best_reg);
91 y_k = 'y(k-';
92 y_fi = ')';
93 for i=1:best_auto_reg
94     regressors_name{i} =  sprintf('%s%d%s',y_k, i, y_fi);
95 end
96 u_k = 'u(k-';
97 u_fi = ')';
98 for i=1:best_reg
99     regressors_name{i + best_auto_reg} =  sprintf('%s%d%s',
       u_k, i, u_fi);
100 end
101 fig_i = figure('Name', 'Indicator by number of neurons');
102 hold on
103 grid on
104 xlabel('Regresors')
105 bar(I)
106 ylabel(join(['I - ', num2str(NUM_OPT_NEU), ' nn']))
107 set(gca(fig_i), 'XTick', [1:best_auto_reg + best_reg], '
     xticklabel', regressors_name);
108
```

```
109 %%
110 y_train_nn = net_trained(x_train');
111 y_test_nn = net_trained(x_test');
112 y_val_nn = net_trained(x_val');
113
114 error_train = y_train - y_train_nn';
115 error_test  = y_test - y_test_nn';
116 error_val   = y_val- y_val_nn';
117
118 figure()
119 grid on
120 hold on
121 plot(error_train)
122 plot(error_test)
123 plot(error_val)
124 legend('train','test','val')
125
126 %%
127 rmse_train = RMSE(y_train, y_train_nn')
128 rmse_test = RMSE(y_test, y_test_nn')
129 rmse_val = RMSE(y_val, y_val_nn')
130 mse_train = MSE(y_train, y_train_nn')
131 mse_test = MSE(y_test, y_test_nn')
132 mse_val = MSE(y_val, y_val_nn')
133 mae_train = MAE(y_train, y_train_nn')
134 mae_test = MAE(y_test, y_test_nn')
135 mae_val = MAE(y_val, y_val_nn')
136 mape_train = MAPE(y_train, y_train_nn')
137 mape_test = MAPE(y_test, y_test_nn')
138 mape_val = MAPE(y_val, y_val_nn')
139
140 %%
141 % Plot Histogram of Error Values
142 figure()
143 ploterrhist(error_train,'Train', error_test, 'Test',
      error_val, 'Validation')
144
145 % plots error vs. epoch for the training, validation, and
      test performances of the training record TR
146 figure()
```

```
147 plotperform(tr)
148
149
150 %%
151 figure()
152 stairs(y_val_nn)
153 hold on
154 grid on
155 ylabel('y(k)')
156 xlabel('N mero de muestras')
157 stairs(y_val)
158 legend('NN-Validaci n', 'Dato Real')
159
160 figure()
161 stairs(y_test_nn)
162 hold on
163 grid on
164 ylabel('y(k)')
165 xlabel('N mero de muestras')
166 stairs(y_test)
167 legend('NN-Prueba', 'Dato Real')
168
169 figure()
170 stairs(y_train_nn)
171 hold on
172 grid on
173 ylabel('y(k)')
174 xlabel('N mero de muestras')
175 stairs(y_train)
176 legend('NN-Entrenamiento', 'Dato Real')
```