

PRÁCTICA 1: Identificación y control neuronal

Sistemas de Control Inteligente



Alba Calvo Herrero
María Sanz Espeja

Grupo A3 lunes
Grado Ingeniería Informática

Índice

Introducción	3
Parte I	3
Ejercicio 1. Perceptrón	3
Ejercicio 2 .Aproximación de funciones	5
Ejercicio 3. Aproximación de funciones (II)	12
Ejercicio 4. Clasificación.	20
Parte II	25
Ejercicio 1. Objetivo y descripción del sistema.	25
Ejercicio 2. Desarrollo de la práctica	27
Problemas encontrados y soluciones	33

Introducción

En este trabajo, se explorarán dos áreas clave: en la primera parte, se estudiarán las redes neuronales y su capacidad para aproximaciones de funciones. En la segunda parte, se abordará el diseño de un control de posición para robots móviles utilizando redes neuronales no recursivas en MATLAB y Simulink. El objetivo final es evaluar el rendimiento y aplicabilidad de estas redes en el control de robots móviles.

Parte I

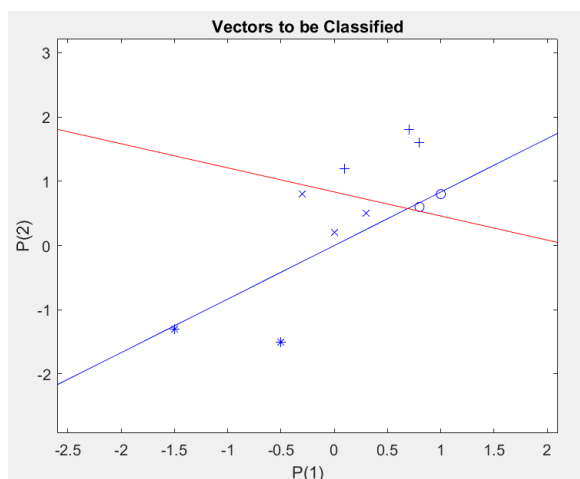
Ejercicio 1. Perceptrón

1.1 ¿Consigue la red separar los datos?, ¿cuántas neuronas tiene la capa de salida?, ¿por qué?

Creamos los vectores con los datos de entrada dados en el enunciado. P será x_1 y x_0 , mientras que T contiene las clases en binario.

```
% Datos de entrada (x1 y x0) y sus etiquetas de clase
P = [0.1 0.7 0.8 0.8 1.0 0.3 0.0 -0.3 -0.5 -1.5;
1.2 1.8 1.6 0.6 0.8 0.5 0.2 0.8 -1.5 -1.3];
T = [1 1 1 0 0 1 1 1 0 0;
0 0 0 0 0 1 1 1 1 1];
% Crear una red neuronal simple
net = newp(P, T);
% Entrenar la red neuronal con los datos de entrada y etiquetas
net = train(net, P, T);
% Visualizar los datos de entrada y las etiquetas de clase
plotpv(P, T);
% Visualizar la frontera de decisión aprendida por la red
plotpc(net.iw{1, 1}, net.b{1});
```

Se genera la siguiente gráfica:



Training Results

Training finished: Met performance criterion ✓

Training Progress

Unit	Initial Value	Stopped Value	Target Value	
Epoch	0	8	1000	
Elapsed Time	-	00:00:08	-	
Performance	0.45	0	0	

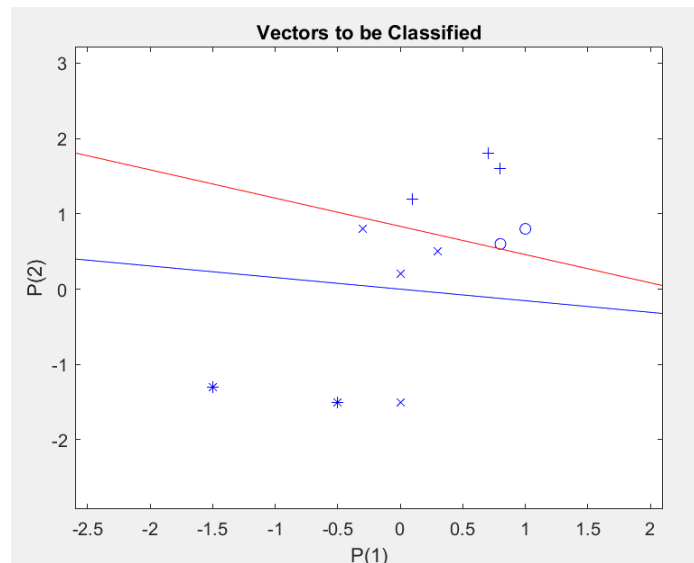
Al generar la gráfica, comprobamos que la red consigue clasificar los datos correctamente, ya que, los separa en cuatro cuadrantes, acorde a las cuatro clases (0,1,2,3). Por tanto, se cumple la igualdad que vimos en la teoría: $N^{\circ}cuadrantes = 2^n$

También podemos comprobar que el algoritmo hace 8 iteraciones (Stopped value = 8 epochs).

1.2. ¿Qué ocurre si se incorpora al conjunto un nuevo dato: [0.0 -1.5] de la clase 3?

```
% Datos de entrada (x1 y x0) y sus etiquetas de clase
P = [0.1 0.7 0.8 0.8 1.0 0.3 0.0 -0.3 -0.5 -1.5 0.0;
1.2 1.8 1.6 0.6 0.8 0.5 0.2 0.8 -1.5 -1.3 -1.5];
T = [1 1 1 0 0 1 1 1 0 0 1;
0 0 0 0 0 1 1 1 1 1 1];
% Crear una red neuronal simple
net = newp(P, T);
% Entrenar la red neuronal con los datos de entrada y etiquetas
net = train(net, P, T);
% Visualizar los datos de entrada y las etiquetas de clase
plotpv(P, T);
% Visualizar la frontera de decisión aprendida por la red
plotpc(net.iw{1, 1}, net.b{1});
```

Si añadimos el nuevo dato y generamos la gráfica, observamos que no se consigue clasificar en cuatro cuadrantes los elementos como antes. En su lugar los elementos de cada clase se están en varios cuadrantes a la vez. Esta es una de las limitaciones del perceptrón.



Además observando los resultados del entrenamiento vemos como llega al límite establecido (target value) que son 1000 iteraciones, sin conseguir una clasificación correcta.

Training Results

Training finished: Reached maximum number of epochs ✓

Training Progress

Unit	Initial Value	Stopped Value	Target Value	
Epoch	0	1000	1000	▲
Elapsed Time	-	00:00:04	-	
Performance	0.409	0.136	0	▼

Ejercicio 2 .Aproximación de funciones

Estudie los efectos sobre la solución final de modificar el método de entrenamiento (consulte la ayuda de Matlab y pruebe 4 métodos diferentes) y el número de neuronas de la capa oculta.

En este ejercicio se va a utilizar cuatro distintos métodos de entrenamiento: `traingd`, `trainrp`, `trainlm`, `trainbr` y `traingd` que se explicarán posteriormente. El código es el siguiente:

```
% Ejercicio 2. Aproximación de funciones
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% APROXIMACIÓN DE FUNCIONES
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all; close all;
% DEFINICIÓN DE LOS VECTORES DE ENTRADA-SALIDA
% =====
t = -3:.1:3; % eje de tiempo
F=sinc(t)+.001*randn(size(t)); % función que se desea aproximar
plot(t,F,'+');
title('Vectores de entrenamiento');
xlabel('Vector de entrada P');
ylabel('Vector Target T');
% DISEÑO DE LA RED
% =====
%hiddenLayerSize = 4;
hiddenLayerSize = input('Introduce el numero de neuronas de la capa oculta:');
algoritmo = elegir_algoritmo();
net = fitnet(hiddenLayerSize, algoritmo);
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
% Entrenamos la red
net = train(net,t,F);
Y=net(t);
% Mostramos graficas
plot(t,F,'+'); hold on;
plot(t,Y,'-r'); hold off;
```

```
title('Vectores de entrenamiento');
xlabel('Vector de entrada P');
ylabel('Vector Target T');
```

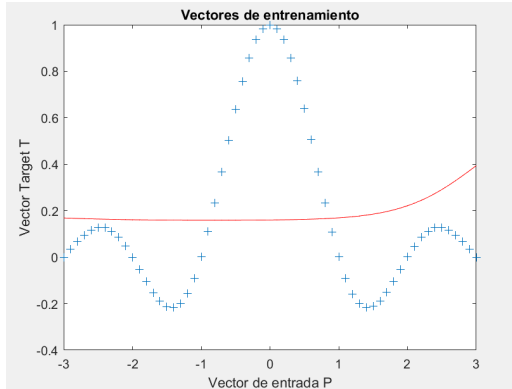
Damos a elegir al usuario el algoritmo mediante la función elegir_algoritmo:

```
function algoritmo = elegir_algoritmo()
    algoritmo = "";
    while true
        disp('1. Gradient Descent. ');
        disp('2. Resilient Backpropagation. ');
        disp('3. Levenberg-Marquardt ');
        disp('4. Bayesian regularization ');
        num_algoritmo = input('Selecciona el algoritmo de entrenamiento: ');
        switch num_algoritmo
            case 1
                algoritmo = 'traingd';
                break;
            case 2
                algoritmo = 'trainrp';
                break;
            case 3
                algoritmo = 'trainlm';
                break;
            case 4
                algoritmo = 'trainbr';
                break;
            otherwise
                disp('Error. Opcion no valida. ');
                break;
        end
    end
end
```

Vamos a entrenar las redes con 2, 10 y 20 neuronas en los 4 modos de entrenamiento. Primero, observaremos los resultados y después sacaremos una conclusión para cada algoritmo.

- **2 neuronas:**

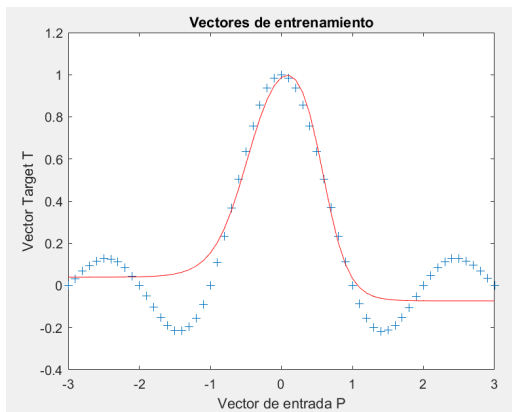
1. Gradient Descent:



Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	6	1000
Elapsed Time	-	00:00:02	-
Performance	0.153	0.153	0
Gradient	0.0464	0.0447	1e-05
Validation Checks	0	6	6

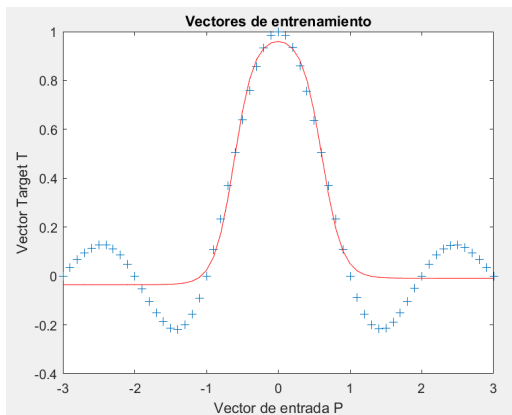
2. Resilient Backpropagation



Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	27	1000
Elapsed Time	-	00:00:02	-
Performance	0.629	0.0106	0
Gradient	1.39	0.0251	1e-05
Validation Checks	0	6	6

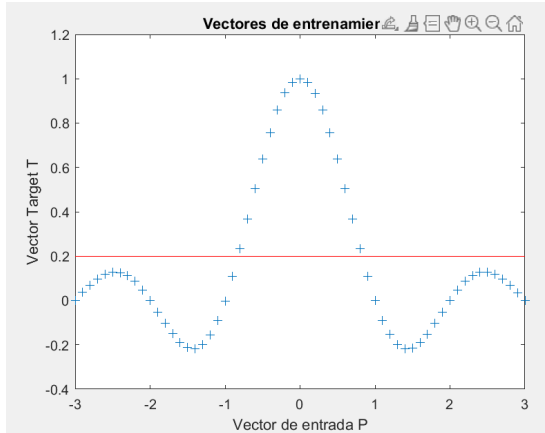
3. Levenberg-Marquardt



Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	25	1000
Elapsed Time	-	00:00:01	-
Performance	0.358	0.0112	0
Gradient	0.956	8.22e-05	1e-07
Mu	0.001	1e-12	1e+10
Validation Checks	0	6	6

4. Bayesian Regularization



Training Results

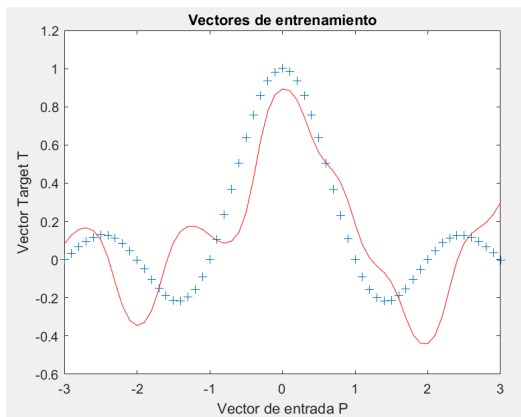
Training finished: Reached maximum mu ✔

Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	20	1000
Elapsed Time	-	00:00:02	-
Performance	1.19	0.129	0
Gradient	1.99	0.015	1e-07
Mu	0.005	1e+10	1e+10
Effective # Param	7	0.94	0
Sum Squared P...	33.2	0.1	0

• 10 neuronas:

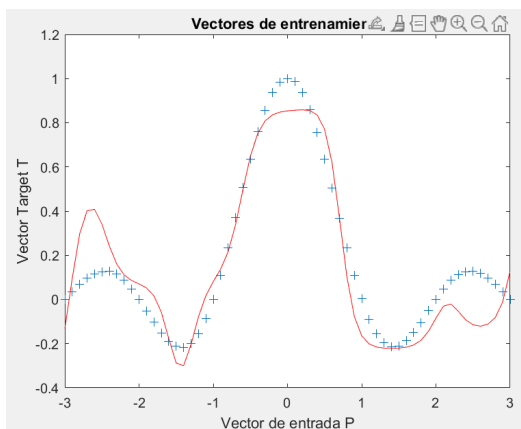
1. Gradient Descent:



Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	1000	1000
Elapsed Time	-	00:00:02	-
Performance	0.424	0.0413	0
Gradient	1.24	0.0637	1e-05
Validation Checks	0	0	6

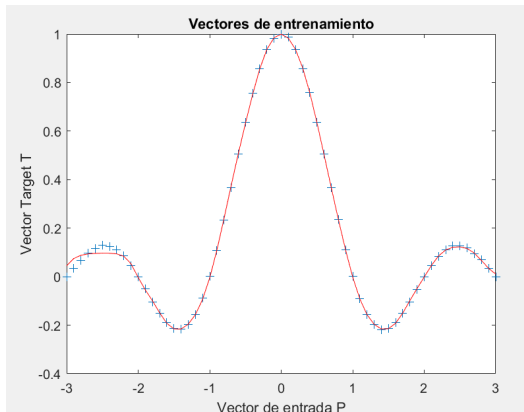
2. Resilient Backpropagation



Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	13	1000
Elapsed Time	-	00:00:01	-
Performance	0.471	0.00599	0
Gradient	1.32	0.0352	1e-05
Validation Checks	0	6	6

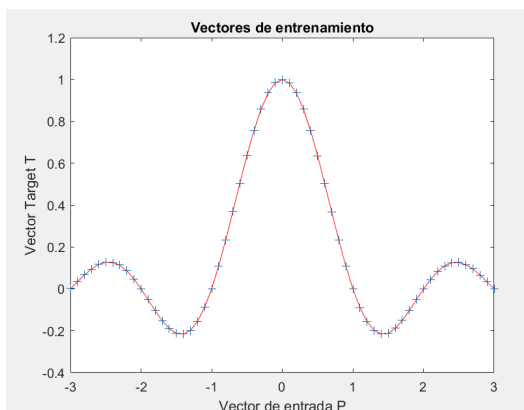
3. Levenberg-Marquardt



Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	18	1000
Elapsed Time	-	00:00:01	-
Performance	0.327	1.32e-05	0
Gradient	1.05	1.75e-05	1e-07
Mu	0.001	1e-06	1e+10
Validation Checks	0	6	6

4. Bayesian Regularization

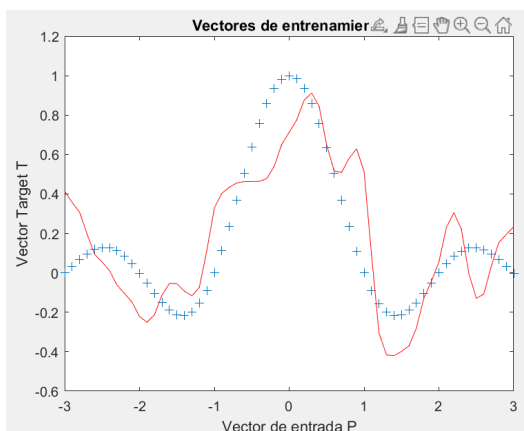


Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	1000	1000
Elapsed Time	-	00:00:03	-
Performance	1.4	3.65e-07	0
Gradient	2.49	4.78e-07	1e-07
Mu	0.005	5	1e+10
Effective # Param	31	19.2	0
Sum Squared P...	2.76e+03	165	0

• 20 neuronas:

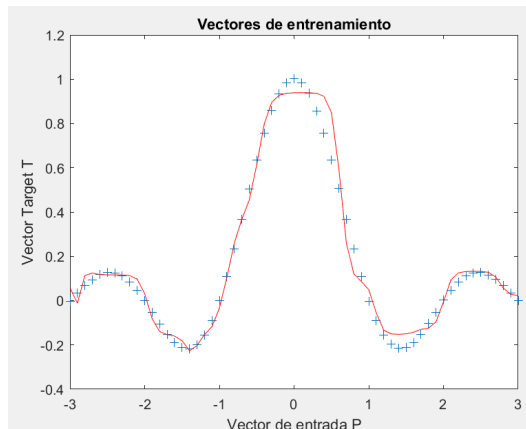
1. Gradient Descent



Training Progress

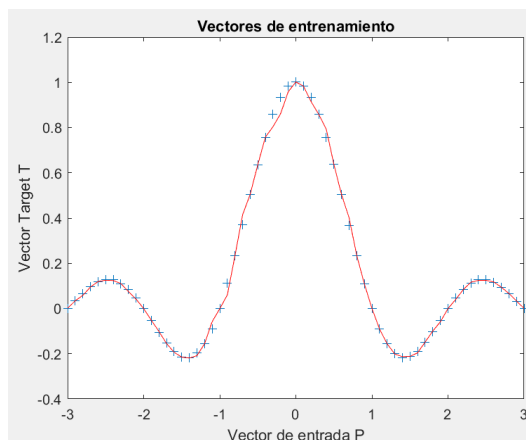
Unit	Initial Value	Stopped Value	Target Value
Epoch	0	1000	1000
Elapsed Time	-	00:00:02	-
Performance	1.09	0.0391	0
Gradient	2.24	0.0817	1e-05
Validation Checks	0	0	6

2. Resilient Backpropagation



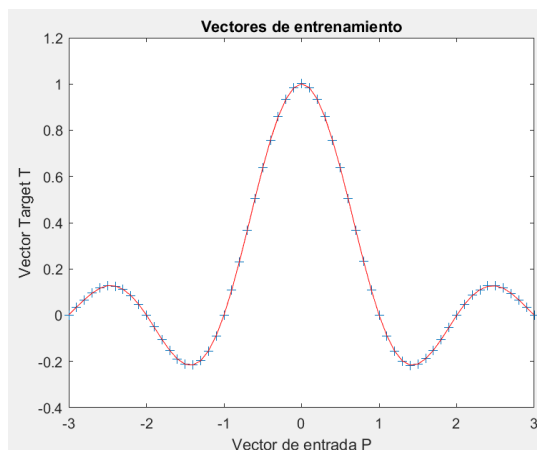
Training Progress			
Unit	Initial Value	Stopped Value	Target Value
Epoch	0	52	1000
Elapsed Time	-	00:00:01	-
Performance	3.47	0.00165	0
Gradient	5.99	0.00819	1e-05
Validation Checks	0	6	6

3. Levenberg-Marquardt



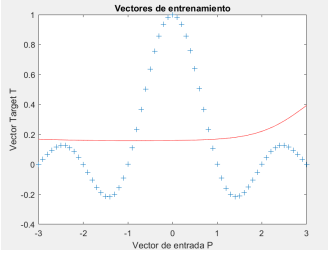
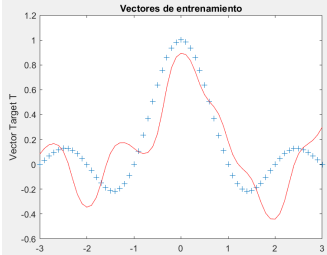
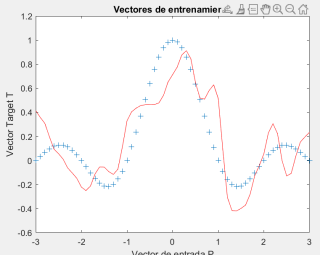
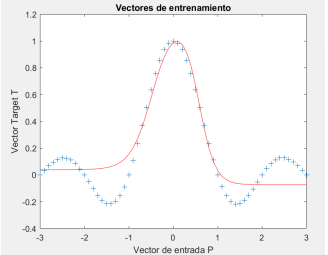
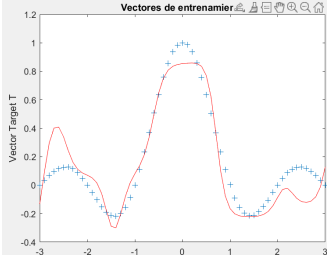
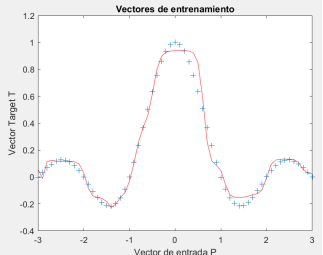
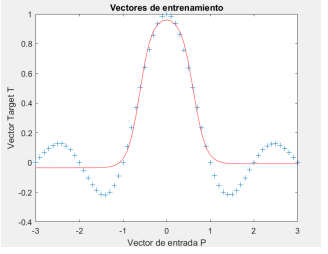
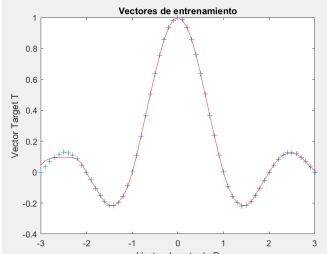
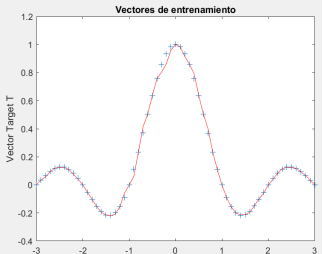
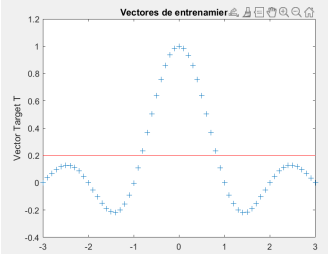
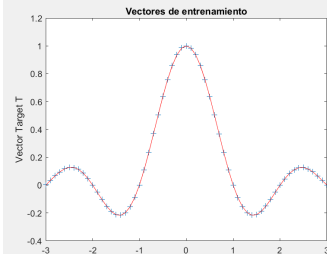
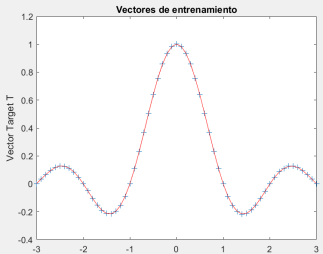
Training Progress			
Unit	Initial Value	Stopped Value	Target Value
Epoch	0	15	1000
Elapsed Time	-	00:00:01	-
Performance	4.39	6.23e-07	0
Gradient	7.2	4.8e-05	1e-07
Mu	0.001	1e-09	1e+10
Validation Checks	0	6	6

4. Bayesian Regularization



Training Progress			
Unit	Initial Value	Stopped Value	Target Value
Epoch	0	1000	1000
Elapsed Time	-	00:00:04	-
Performance	1.45	4.66e-07	0
Gradient	4.08	0.000251	1e-07
Mu	0.005	0.5	1e+10
Effective # Param	61	20.7	0
Sum Squared P...	2.15e+04	199	0

Hemos realizado una tabla para ayudarnos a sacar conclusiones. Contiene el número de iteraciones/épocas y de neuronas. Debemos tener en cuenta que el límite introducido son 1000 iteraciones.

Neuronas / Algoritmo	2	10	20
Gradient Descent	 <p>6 epochs</p>	 <p>1000 epochs</p>	 <p>1000 epochs</p>
Resilient Backpropag ation	 <p>27 epochs</p>	 <p>13 epochs</p>	 <p>52 epochs</p>
Levenberg- Marquardt	 <p>25 epochs</p>	 <p>18 epochs</p>	 <p>15 epochs</p>
Bayesian Regularizati on	 <p>20 epochs</p>	 <p>1000 epochs</p>	 <p>1000 epochs</p>

Para cada método de entrenamiento, además, hemos observado lo siguiente en cuanto al rendimiento para los valores óptimos (número de iteraciones agotadas, performance y tiempo medio):

Gradient Descent

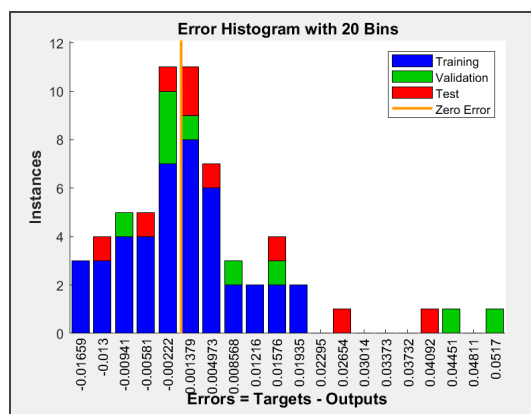
Notamos que al aumentar la cantidad de capas, se acerca mejor a puntos específicos, pero al mismo tiempo, aumenta la cantidad de picos en la función aproximada. A pesar de esto, no logra ajustarse de manera precisa a la función original.

Resilient Backpropagation

La elección del número de neuronas en la capa oculta debe equilibrarse cuidadosamente. Un número demasiado bajo conduce a un rendimiento deficiente, mientras que un número excesivamente alto puede provocar sobreajuste.

Levenberg-Marquadt

El aumento en el número de neuronas en la capa oculta conlleva una mejora gradual en el rendimiento de la red neuronal, pero también puede aumentar la complejidad y el riesgo de sobreajuste. La elección del número óptimo de neuronas dependerá de encontrar un equilibrio adecuado entre la capacidad de aprendizaje y la generalización. Vemos como está muy cerca del valor 0 en el error.



Bayesian Regularization

Finalmente, empleando este algoritmo concluimos que realiza una aproximación perfecta una vez se tienen las suficientes capas intermedias, lo cual es congruente ya que hace aproximaciones más precisas a cambio de consumir mayor cantidad de recursos y cálculos más complejos.

En resumen, podemos afirmar que el método de entrenamiento más efectivo en términos de número de iteraciones, rendimiento y tiempo es **Levenberg-Marquadt** (trainlm). Sin embargo, es importante destacar que **Bayesian Regularization** (trainbr) ha demostrado ser el que más se ha acercado a la función de manera notable y con la máxima precisión.

Ejercicio 3. Aproximación de funciones (II)

En este ejercicio, se estudiarán en detalle las herramientas que facilita Matlab para el diseño y prueba de redes neuronales ejecutando el siguiente código de ejemplo:

```
% Carga de datos de ejemplo disponibles en la toolbox
[inputs,targets] = simplefit_dataset;
% Creación de la red
hiddenLayerSize = 10;
net = fitnet(hiddenLayerSize);
% División del conjunto de datos para entrenamiento, validación y test
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
```

```

net.divideParam.testRatio = 15/100;
% Entrenamiento de la red
[net,tr] = train(net,inputs,targets);
% Prueba
outputs = net(inputs);
errors = gsubtract(outputs,targets);
performance = perform(net,targets,outputs)
% Visualización de la red
view(net)

```

Explore las gráficas disponibles:

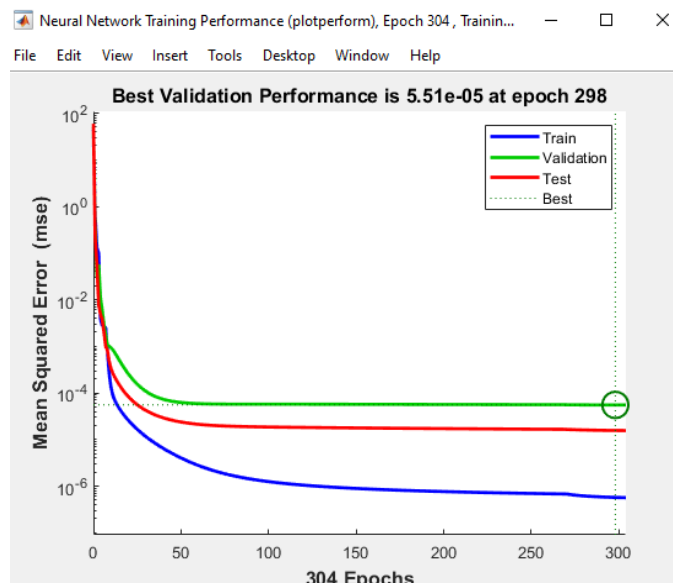
- *Performance*: gráfica que representa el error en función del número de épocas para los datos de entrenamiento, validación y test.
- *Training State*: evolución del entrenamiento.
- *Error Histogram*: histograma del error.
- *Regression y Fit*: ajuste de los datos de entrenamiento, validación y test.

Pruebe este mismo script con el conjunto de datos *bodyfat_dataset*, y evalúe sus resultados. Estudie la mejora que supone utilizar distintos métodos de entrenamiento y una división diferente de los datos (entrenamiento, validación y test).

Comenzamos analizando el código de ejemplo con la base de datos *simplefit_dataset*:

1. Plotperform:

Esta gráfica visualiza el error en función del número de épocas en para los datos. Se observa que el error disminuye progresivamente según avanza el entrenamiento para todos los conjuntos de datos.

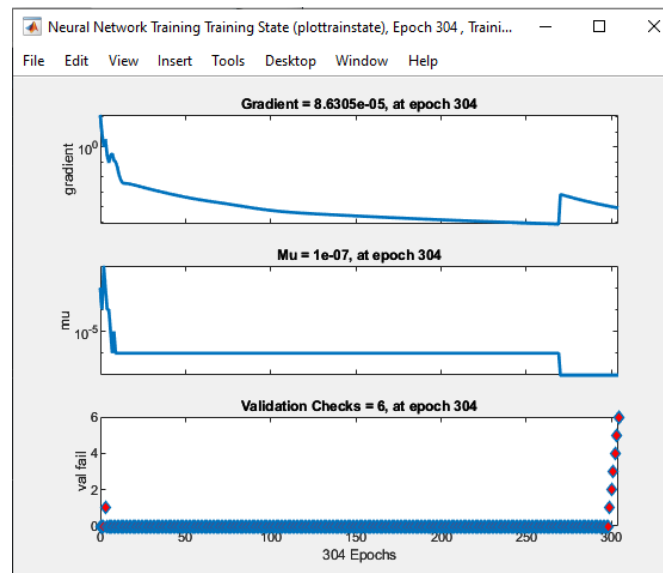


2. Plottrainstate:

En esta parte se obtienen tres gráficas donde se puede ver la evolución del entrenamiento.

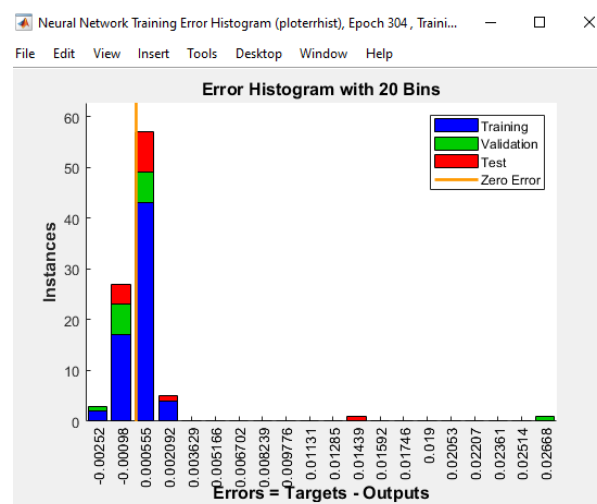
En la primera, se observa el gradiente que nos informa sobre lo rápido o lento que son los cambios que se realicen en el proceso de backpropagation de la siguiente interacción. En la segunda gráfica se obtiene los datos sobre la ganancia del entrenamiento y en la última gráfica los incrementos de error consecutivos en los datos de validación.

En esta última se observa que el entrenamiento termina al llegar siete incrementos seguidos.



3. Plottrainstate:

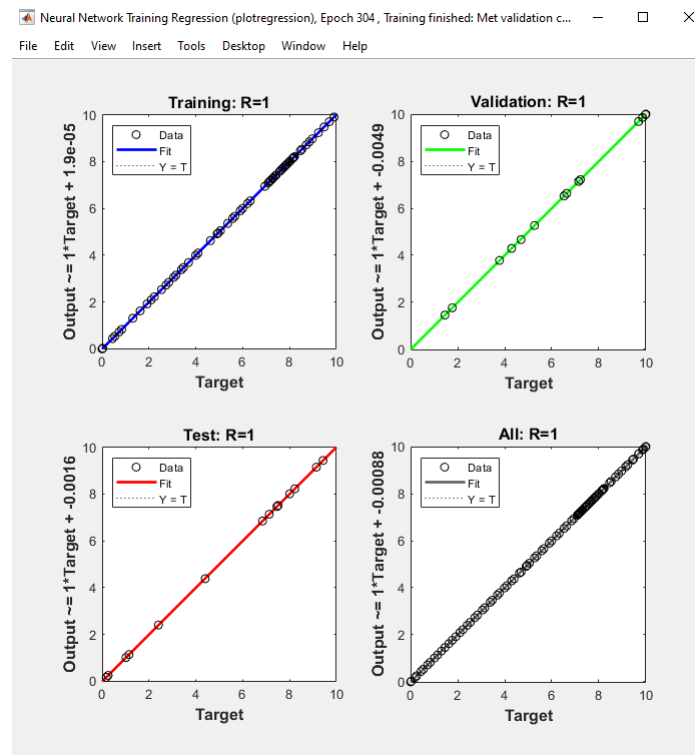
El histograma de error nos indica la cantidad de error ocurrida para cada dato de los datos de entrada de la red. Se observa en la gráfica que la cantidad de errores es baja.



4. Plotregression

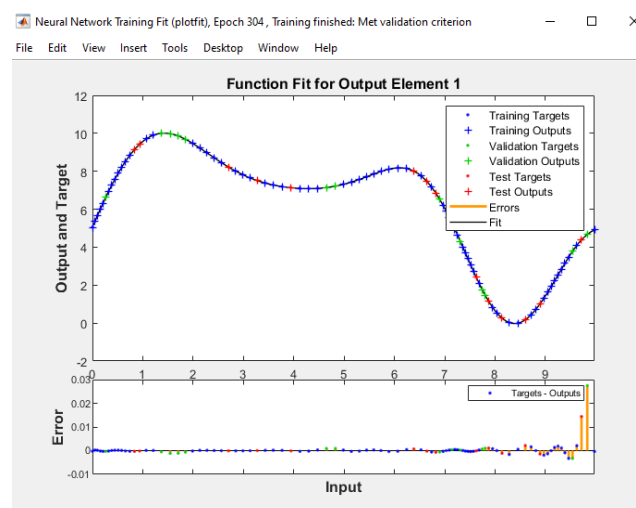
En esta gráfica, a diferencia de las otras, se puede ver el estado final de la red tras haber sido entrenada.

La inclinación y posición de la recta de regresión obtenida sobre los puntos de la gráfica representará la desviación media entre los resultados obtenidos y los esperados.

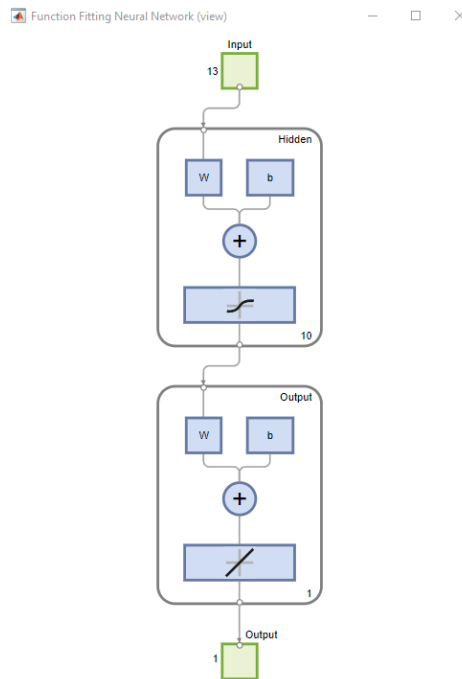


5. Plotfit

Como podemos ver la salida de la red se ajusta a la función esperada. Cuanto más se aproxime la función generada respecto a la función ideal mejor habrá sido el entrenamiento y menor será el error.

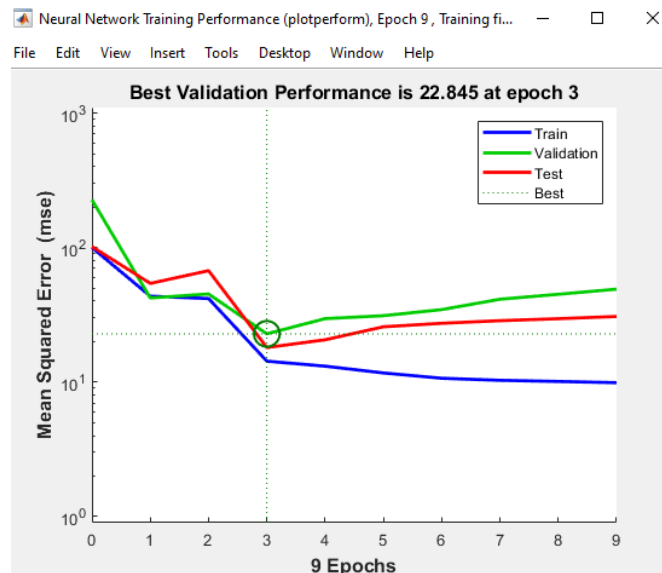


Continuamos con la base de datos `bodyfat_dataset`. Para este caso se utilizarán 13 neuronas de entrada y una de salida debido a los datos y la gran cantidad de argumentos que hay.



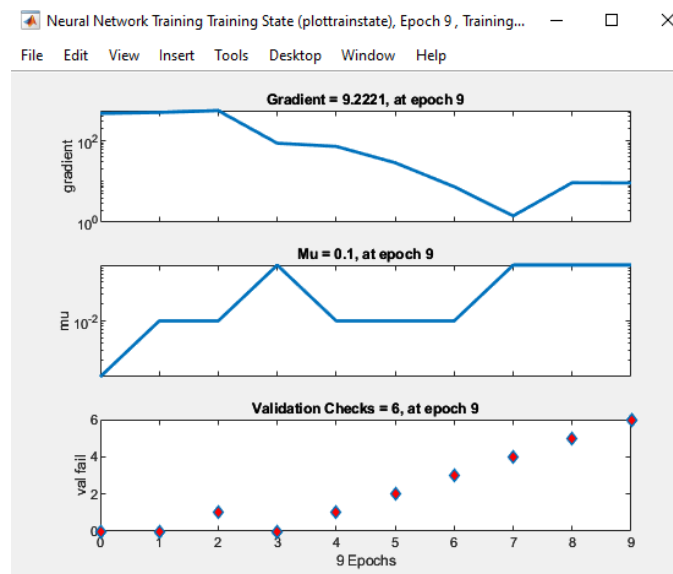
1. Plotperform

Se observa que el error en la primera época es grande, por lo que la red no está aprendiendo. También mencionar que en el punto 3, comienza a estabilizarse y después continúa un poco más constante.



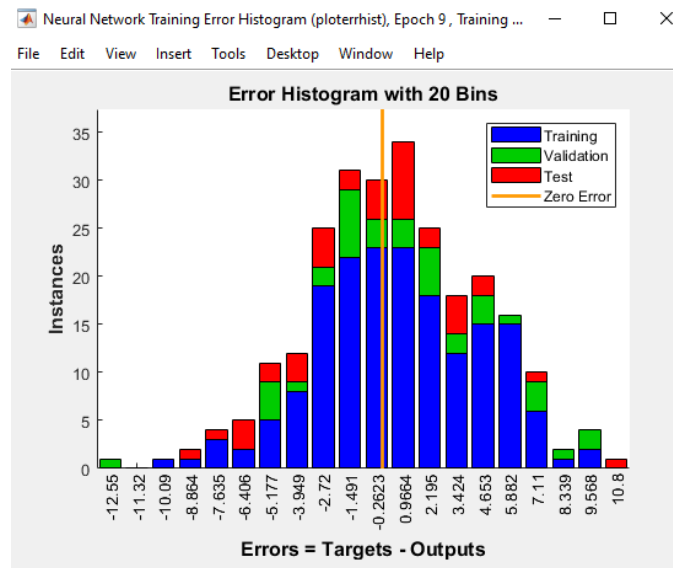
2. Plottrainstate

Como se menciona anteriormente, hasta la época 3 la red no empieza a aprender, por lo que los errores eran mayores. También se observa que el entrenamiento finaliza en la época 9.



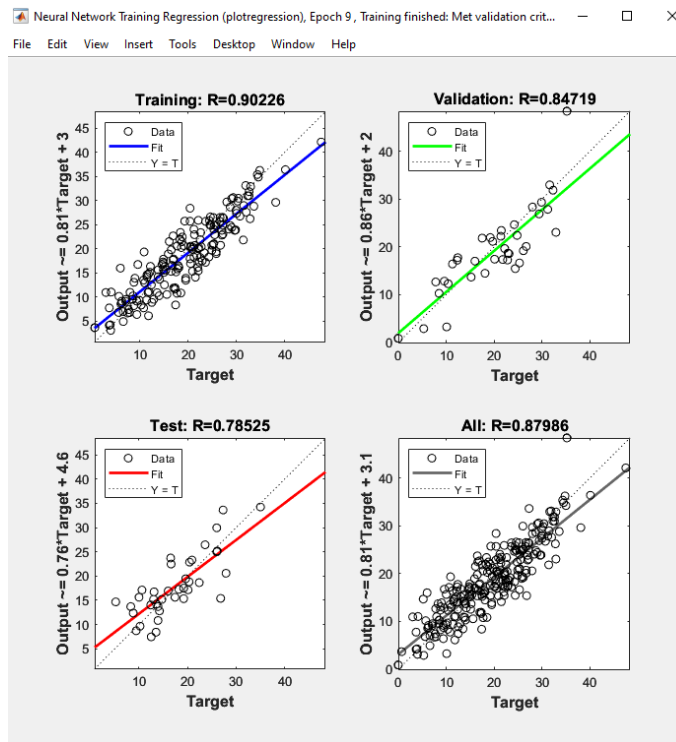
3. Ploterrhist

En este caso, la cantidad de errores es mayor pero se observa en la gráfica que los errores de datos, test y validación son muy similares, es decir, ninguno de ellos es más propenso que el resto a acumular errores.



4. Plotregression

En este caso se observa que el entrenamiento no ha sido perfecto.



5. **Plotfit**: no genera gráfica.

Ejercicio 4. Clasificación.

La clasificación de patrones es una de las aplicaciones que dieron origen a las redes neuronales artificiales. Como en el caso anterior, la toolbox de redes neuronales de Matlab dispone de una red optimizada para la clasificación, *patternnet*, que analizaremos en este ejemplo.

```
% Carga de datos de ejemplo disponibles en la toolbox
[inputs,targets] = simpleclass_dataset;
% Creación de una red neuronal para el reconocimiento de patrones
hiddenLayerSize = 10;
net = patternnet(hiddenLayerSize);
% División del conjunto de datos para entrenamiento, validación y test
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
% Entrenamiento de la red
[net,tr] = train(net,inputs,targets);
% Prueba
outputs = net(inputs);
```

```
errors = gsubtract(targets, outputs);
performance = perform(net, targets, outputs);
% Visualización
view(net)
```

En lugar de las gráficas específicamente relacionadas con la aproximación de una función, en el caso de una tarea de clasificación, se ofrecen:

- Confusion: matrices de confusión de los resultados.
- Receiver Operating Characteristic: curvas ROC (característica operativa del receptor).

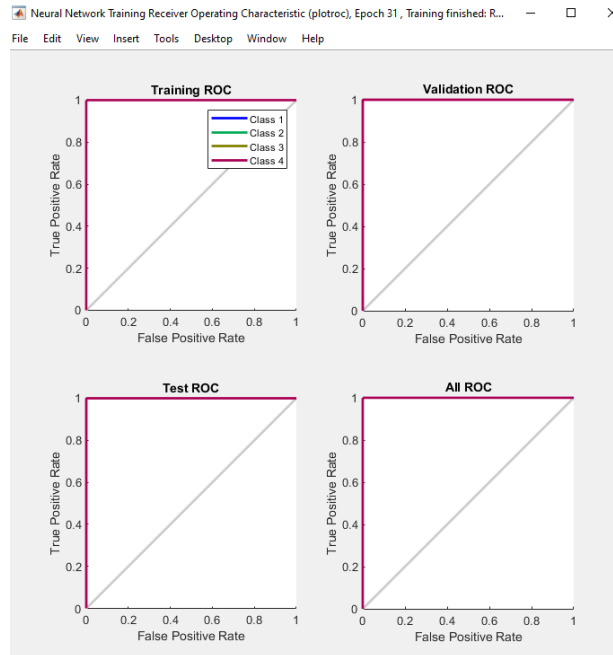
Pruebe este mismo script con el conjunto de datos **cancer_dataset**, y evalúe sus resultados. Estudie de nuevo la mejora que supone utilizar distintos métodos de entrenamiento y una división diferente de los datos (entrenamiento, validación y test).

Se comienza analizando la base de datos **simplecass_dataset**, cuyos valores predeterminados son 70% entrenamiento, 15% validación y 15% test.

Al ejecutar, se obtiene los siguientes resultados donde se analizarán las gráficas confusion y performance:



No se obtiene ningún error a la hora de predecir las clases por lo que los resultados son los mejores posibles ya que en la gráfica se puede ver como no deja de descender hasta llegar al mínimo. Además para comprobar que la predicción es de los mejores resultados, se observa la gráfica ROC:



Continuamos con el conjunto de datos de **cancer_dataset** cuyos valores predeterminados son 70% entrenamiento, 15% validación y 15% test.

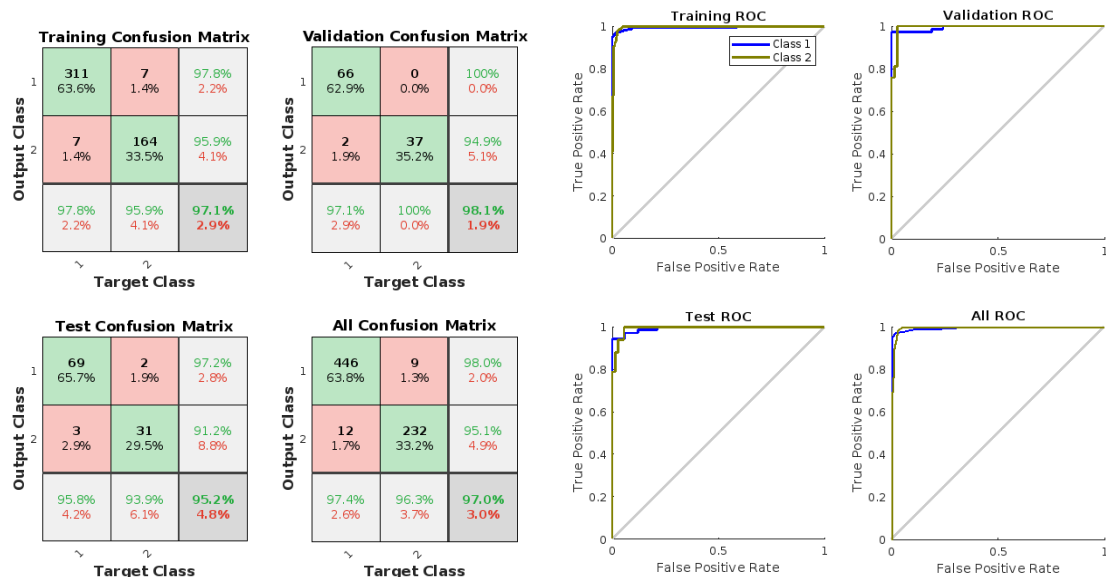
- **Gradient Descent (traingd)**

Algoritmo muy efectivo, pues solo hay un 4,5% de datos mal clasificados. Además, fijándonos en la gráfica del ROC, vemos que los verdaderos positivos son mucho mayores que los falsos positivos.



- Resilient Backpropagation (trainrp)

Podemos observar que sólo hay un 2,9% de datos mal clasificados, por lo que la clasificación es bastante buena. Esto se debe a que el conjunto de datos es más difícil encontrar patrones por la complejidad de los datos. En cuanto a la gráfica ROC (Receiver Operating Characteristic) se comprueba que los verdaderos positivos son mayores que los falsos.



- Levenberg-Marquardt (trainlm)

En este caso, el error total es solo de un 1%. Por otro lado, vemos que los falsos positivos aumentan.



- Bayesian regularization (trainbr)

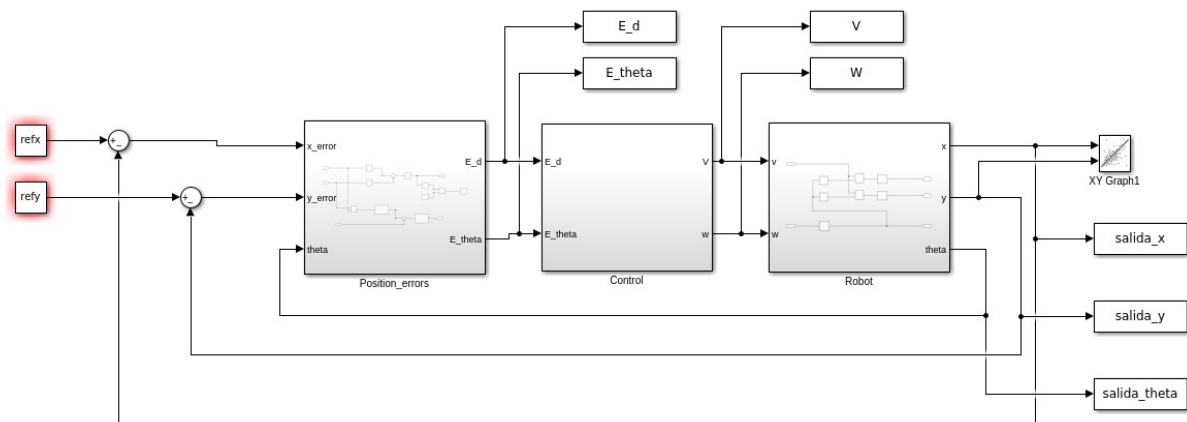
Por último, para este conjunto de datos el algoritmo bayesiano nos da un error total del 0%. El mejor dato de los cuatro algoritmos. Sin embargo, en la gráfica podemos ver que el número de falsos positivos es mayor que el de verdaderos positivos.



Parte II

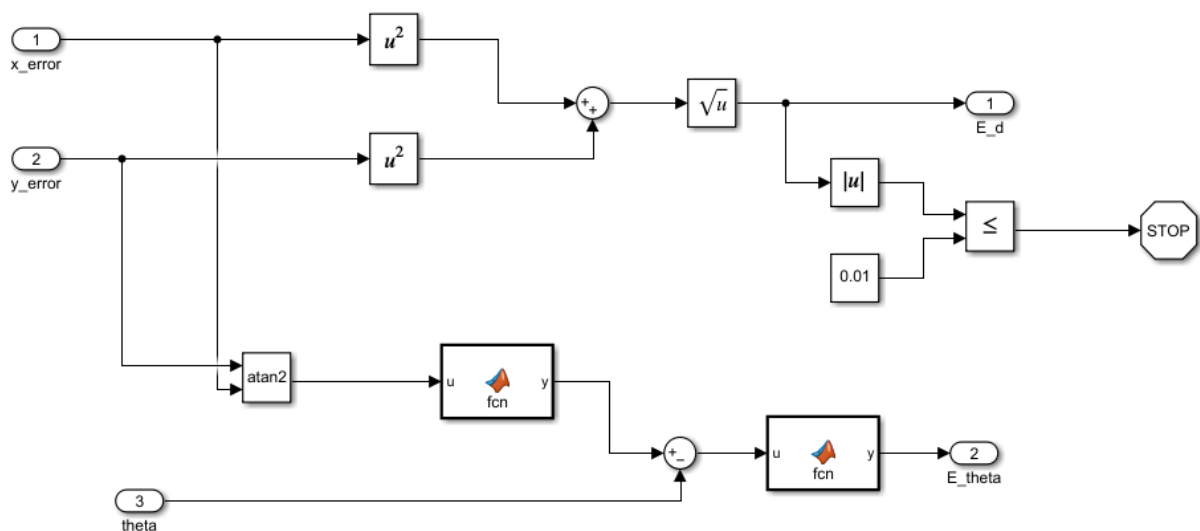
Ejercicio 1. Objetivo y descripción del sistema.

En esta primera sección de la práctica, se nos solicita construir el circuito en Simulink siguiendo las instrucciones proporcionadas en el enunciado. La descripción general del sistema creado es la siguiente:



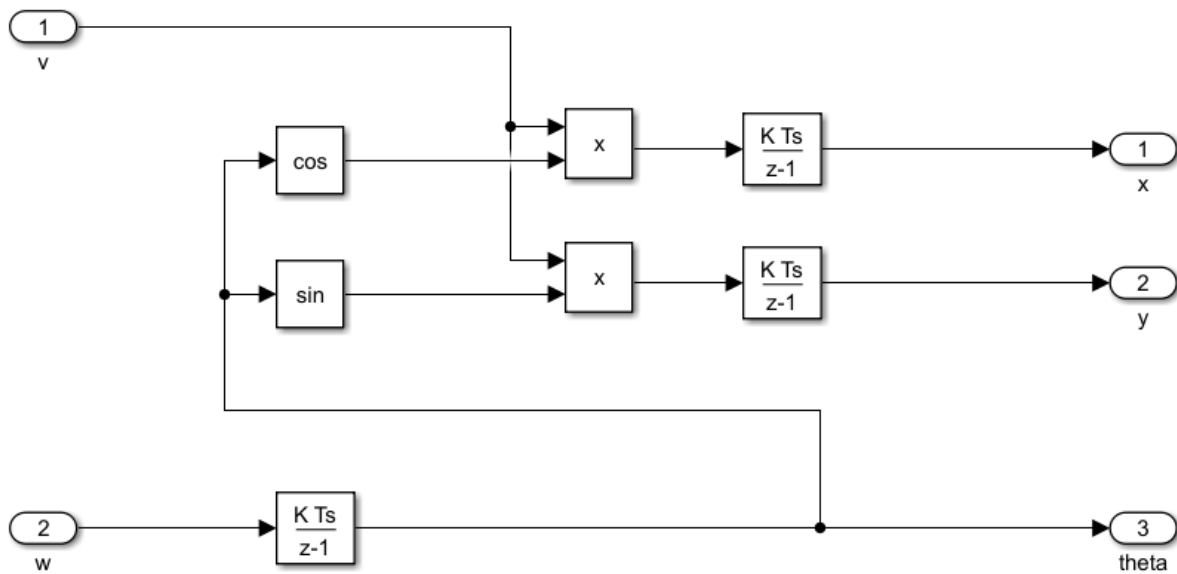
Las salidas del sistema se envían al workspace de Matlab para su posterior procesamiento. Las señales de salida x e y también se redirigen a un bloque de función encargado de mostrar estas dos señales. El sistema consta de tres subsistemas.

- El bloque de "Position_errors" contiene el siguiente circuito interno:

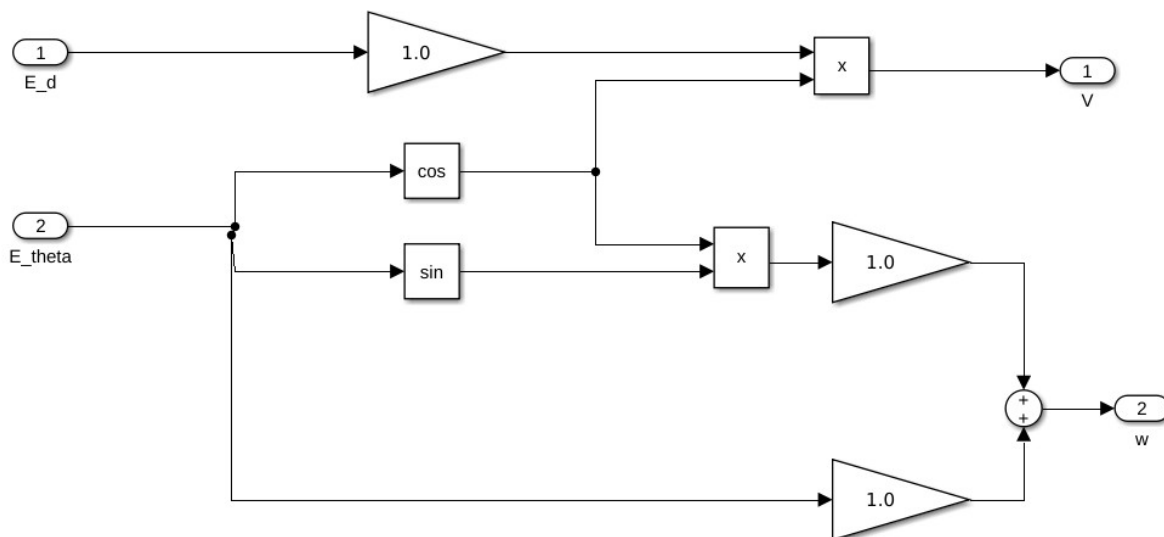


En la figura, se muestra la condición de parada del sistema, que establece que si E_d es menor que 0.01 en cualquier momento, la ejecución se detendrá.

- El subsistema "Robot" es el mismo que se utilizó en la práctica 0 y tiene el siguiente circuito interno:



- Por último, está el subsistema "Control" proporcionado por el enunciado, que contiene el siguiente circuito interno:



Lo único que queda por hacer es configurar las variables de salida para que funcionen como "Structures with Time." Además, se limita el tiempo de simulación a 100 segundos y se define el tiempo de muestreo como una variable "Ts" que se definirá en el archivo de script de Matlab. Para que estas señales se

exporten de acuerdo con el código que se utilizará, es necesario desactivar la opción “single simulation output” dentro de las opciones de Model Settings.

Ejercicio 2. Desarrollo de la práctica

a) Implemente el esquema de la Figura 1, incluyendo todos los bloques principales. Utilice como controlador el proporcionado en “controlblackbox.slx”. Configure los parámetros de la simulación (menú “Simulation/Model Simulation Parameters”) tal y como se muestra en la Figura 10. Guarde el esquema de Simulink con un nombre reconocible, por ejemplo “PositionControl.slx”.

El modelo se ha guardado como simulationControl.slx

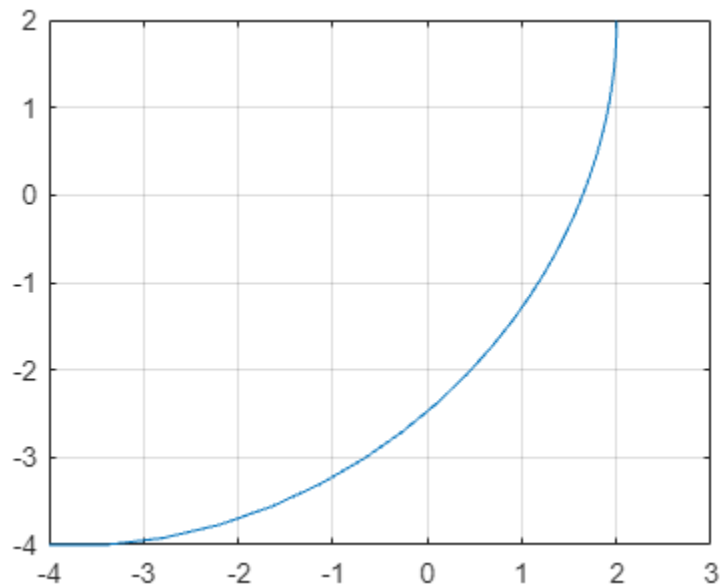
b) En el entorno de programación de Matlab, cree un script nuevo con el nombre “RunPositionControl.m”. Con el siguiente código se puede simular el diagrama PositionControl desde el entorno de comandos de Matlab, configurando el punto destino del robot mediante las variables refx y refy y el tiempo de muestreo Ts.

c) Ejecute el script RunPositionControl y compruebe que se generan las variables que contienen las salidas y entradas del controlador (variables E_d E_theta V y W) y las salidas del robot durante la simulación (salida_x, salida_y, salida_theta).

Workspace			
Name	Value	Size	Class
E_d	1x1 struct	1x1	struct
E_theta	1x1 struct	1x1	struct
Ts	0.1000	1x1	double
V	1x1 struct	1x1	struct
W	1x1 struct	1x1	struct
refx	2	1x1	double
refy	2	1x1	double
salida_theta	1x1 struct	1x1	struct
salida_x	1x1 struct	1x1	struct
salida_y	1x1 struct	1x1	struct
tout	69x1 double	69x1	double

Comprobamos que se generan las variables en el workspace.

d) Ejecute el siguiente código para mostrar la trayectoria del robot mediante el comando plot de Matlab



Vemos como su trayectoria acaba en el (2,2).

e) Realice $N=30$ simulaciones del controlador proporcionado mediante un bucle donde se varían los valores de ref_x y ref_y de manera aleatoria dentro del entorno de 10×10 metros. En cada simulación se deberá guardar el valor a lo largo del tiempo de las entradas (E_d y E_{θ}) y salidas (V y W) del bloque controlador. Genere la matriz “inputs” de tamaño $2 \times N$, donde se acumulen los valores de E_d y E_{θ} . Del mismo modo genere la matriz “outputs”, donde se acumulen los valores obtenidos de las variables V y W .

% Generar N posiciones aleatorias, simular y guardar en variables

```
N=30;
E_d_vec=[];
E_theta_vec=[];
V_vec=[];
W_vec=[];
for i=1:N
    refx=10*rand-5;
    refy=10*rand-5;
    sim('simulationControl')
    E_d_vec=[E_d_vec;E_d.signals.values];
    E_theta_vec=[E_theta_vec;E_theta.signals.values];
    V_vec=[V_vec; V.signals.values];
    W_vec=[W_vec; W.signals.values];
end
inputs=[E_d_vec'; E_theta_vec'];
outputs=[V_vec'; W_vec'];
```

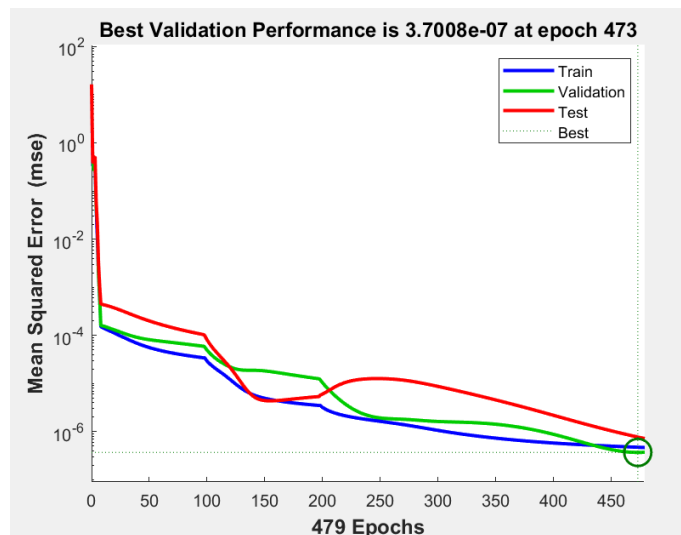
f) Diseñe una red neuronal con una capa oculta de tal manera que dicha red se comporte como el controlador proporcionado. El número de neuronas de la capa oculta se deberá encontrar mediante

experimentación. Justifique el valor elegido. El siguiente ejemplo muestra los comandos de Matlab para realizar dicho entrenamiento a partir de los vectores inputs, outputs.

```
% Generar N posiciones aleatorias, simular y guardar en variables
N=30;
E_d_vec=[];
E_theta_vec=[];
V_vec=[];
W_vec=[];
for i=1:N
    refx=10*rand-5;
    refy=10*rand-5;
    sim('simulationControl')
    E_d_vec=[E_d_vec;E_d.signals.values];
    E_theta_vec=[E_theta_vec;E_theta.signals.values];
    V_vec=[V_vec; V.signals.values];
    W_vec=[W_vec; W.signals.values];
end
inputs=[E_d_vec'; E_theta_vec'];
outputs=[V_vec'; W_vec'];
% Entrenar red neuronal con 9 neuronas en la capa oculta
net = feedforwardnet([9]);
net = configure(net,inputs,outputs);
net = train(net,inputs,outputs);
% Generar bloque de Simulink con el controlador neuronal
gensim(net,Ts)
```

Discuta los resultados del entrenamiento y observe el comportamiento de la red en los subconjuntos de entrenamiento, test y validación.

Experimentamos desde 1 hasta 15 neuronas y vemos que el valor más bajo de performance se consigue con **9 neuronas** en la época **473** con un valor de **3,7 e-7**.

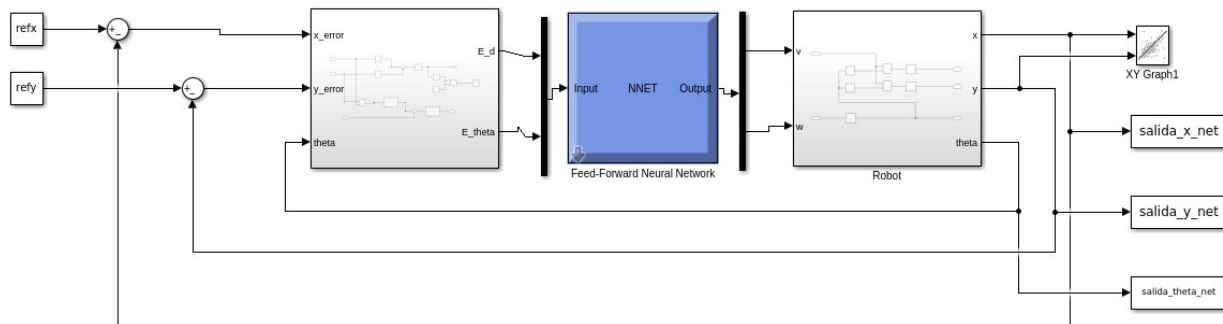


g) Genere, mediante la orden `gensim` de Matlab, un bloque con la red neuronal propuesta:

```
% Generar bloque de Simulink con el controlador neuronal  
gensim(net,Ts)
```

h) Cree un nuevo archivo de simulación “PositionControlNet.slx” con los mismos bloques utilizados en “PositionControl.slx” y donde se utilice la red neuronal en lugar del bloque controlador (Figura 12). Se utilizan bloques multiplexores y demultiplexores para adaptar las señales de entrada y salida como se muestra en la Figura.

En nuestro caso lo hemos llamado **simulationControlNet.slx**



i) Compare el comportamiento de “PositionControlNet.slx” con “PositionControl.slx” para diferentes valores de `refx` y `refy`. Calcule el error entre las trayectorias realizadas por ambos controladores. Se recomienda realizar un script de Matlab para automatizar dicha comparación y mostrar los resultados.

Hemos creado un script llamado **comparacion.m** el cual calcula el error entre las trayectorias con `N` valores diferentes y aleatorios de `refx` y `refy`. Aparecerán `N` gráficas donde se muestra la trayectoria original y la de la red neuronal. Además, se mostrará una gráfica extra con el error entre ambas trayectorias.

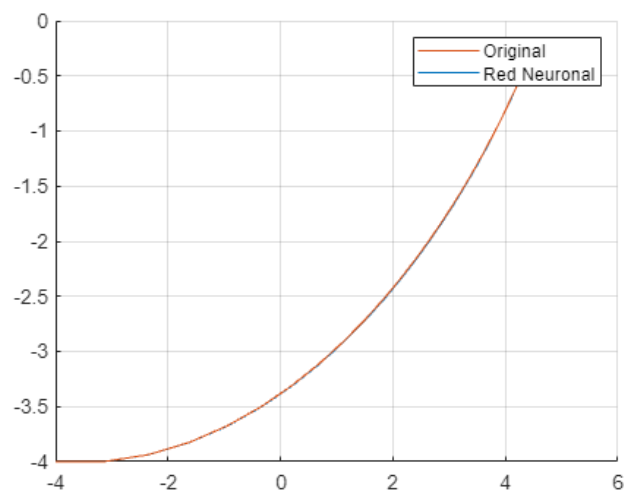
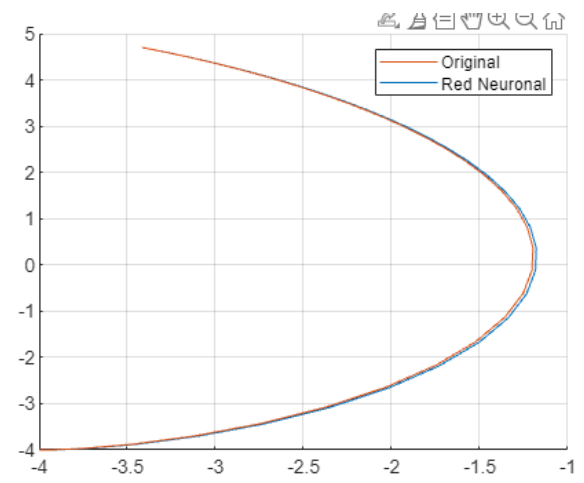
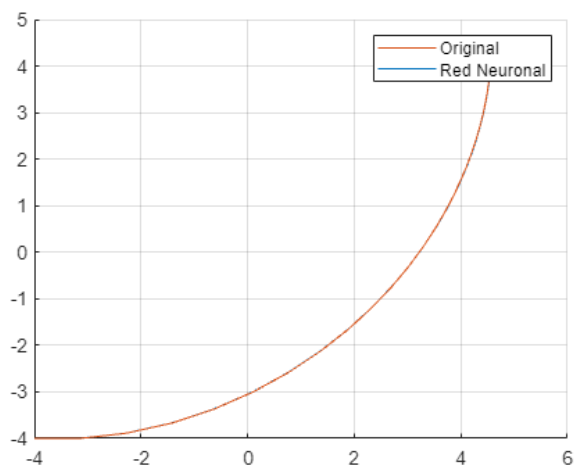
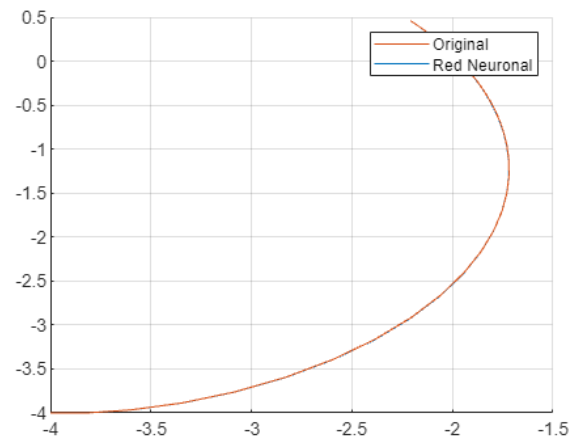
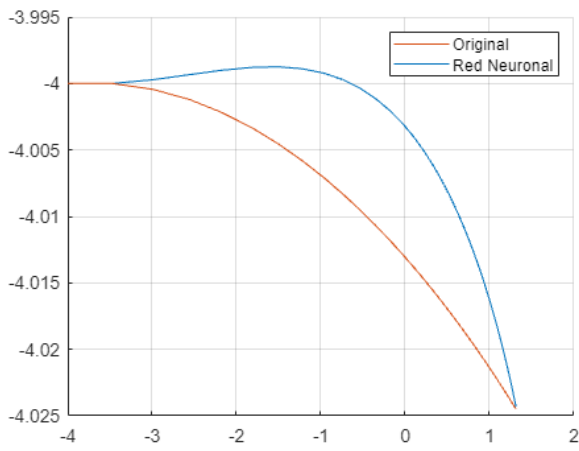
```
clc;  
clear all;  
close all;  
% Tiempo de muestreo  
Ts = 100e-3;  
% N valores diferentes de refx y refy  
N = 5;  
% Creamos los vectores de errores de x e y  
errors_x = [];  
errors_y = [];  
for i = 1:N  
    % Generamos los valores aleatorios de refx y refy  
    refx = 10 * rand - 5;  
    refy = 10 * rand - 5;  
    % Simulación con el sistema original  
    sim('simulationControlNet.slx');  
    % Guardamos resultados de x e y
```

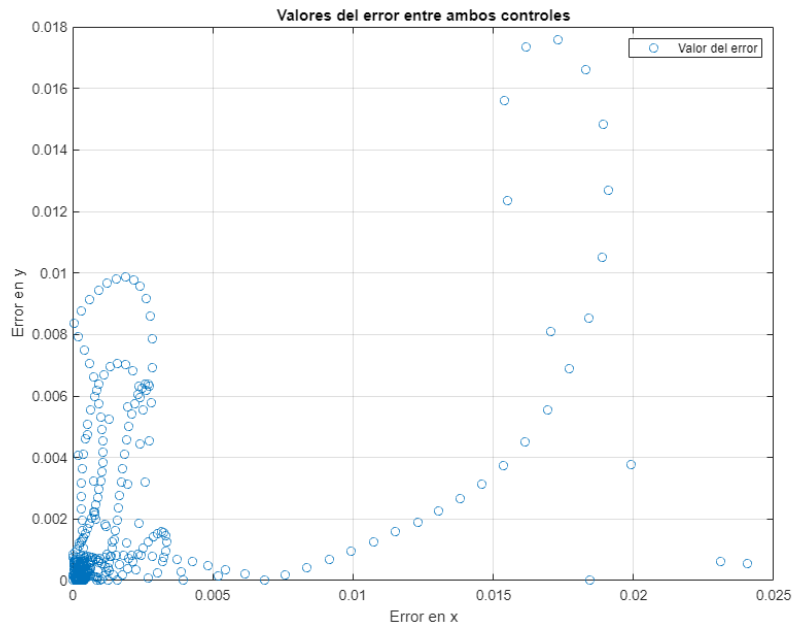
```

x = salida_x.signals.values;
y = salida_y.signals.values;
% Simulación con el sistema con red neuronal (feed forward neural
network)
sim('simulationControlNet.slx');
% Guardamos resultados de x e y
x_NN = salida_x_net.signals.values;
y_NN = salida_y_net.signals.values;
% Mostrar la trayectoria resultante comparando ambos controles
figure(i);
hold on;
grid on;
red = plot(x_NN, y_NN);
original = plot(x, y);
legend([original red], {'Original', 'Red Neuronal'});
hold off;
% Calcular el error entre ambos valores
partial_error_x = [];
partial_error_y = [];
for j=1:min(length(x),length(x_NN))
    partial_error_x = [partial_error_x ; abs(x(j) - x_NN(j))];
    partial_error_y = [partial_error_y ; abs(y(j) - y_NN(j))];
end
errors_x = [errors_x ; partial_error_x];
errors_y = [errors_y ; partial_error_y];
end
% Mostrar error entre las dos simulaciones
figure(N + 1);
plot_errors = plot(errors_x, errors_y, 'o');
grid on;
title('Valores del error entre ambos controles');
legend(plot_errors, {'Valor del error'});
xlabel('Error en x');
ylabel('Error en y');

```

Los resultados para N=5 son los siguientes:





Podemos comprobar como la gran mayoría de los errores son muy próximos a 0. Los puntos más alejados no llegan a una diferencia en valor absoluto de 0.025, por lo que el **error es bastante bajo** entre estos sistemas. Esto se puede corroborar si echamos un vistazo a las gráficas comparativas de las trayectorias de ambos sistemas.

Problemas encontrados y soluciones

Tuvimos problemas con las versiones de Matlab y Simulink, ya que, se realizó en las versiones R2022a, R2022b e incluso desde Matlab Online (R2023b). Para poder abrir y editar los modelos desde las distintas versiones se exportó a la versión anterior que se necesitaba en ese momento. Sin embargo, seguían saliendo mensajes de error al intentar abrirlos, por lo que, tuvimos que permitir cargar modelos creados con una versión más reciente de Simulink. Para ello, en la barra de herramientas de MATLAB, en la pestaña INICIO, pulsamos en Preferencias -> Simulink -> Abrir Preferencias de Simulink.

En el panel de Archivo de Modelo desmarcamos la opción "No cargar modelos creados con una versión más reciente de Simulink".

