DISEÑO DE UN SISTEMA DE CONTROL DEL NIVEL PARA TANQUES BASADO EN REDES NEURONALES

Trabajo Final de Máster Ingeniería Industrial

Alumno: Adrián Vieco Jiménez

Mail: adrivj96@gmail.com

PASAPORTE: PAI596058

Teléfono: +34 662227582

Fecha: 26/08/2020

1 ÍNDICE:

2 TABLA DE ILUSTRACIONES:

3 TABLA DE ECUACIONES:

4 TABLA DE GRÁFICAS:

5 ANTEPROYECTO:

5.1 INTRODUCCIÓN Y PRESENTACIÓN DEL TEMA:

Desde que Warren Mc Culloch y Walter Pitts presentaron la primera publicación con respecto al uso de redes neuronales en 1943 (José Danilo Rairán, 2012) las redes neuronales han ido ganando importancia, esto se puede apreciar en el mayor número de artículos y publicaciones sobre el tema que han sido elaborados (José Danilo Rairán, 2012).

La razón por la que las redes neuronales están siendo utilizadas cada vez más reside en que los sistemas actuales son cada vez más complejos, son no lineales y presentan más de un lazo de control, por lo que se busca nuevos métodos inteligentes que puedan realizar el control de estos (Faiber Ignacio Robayo, 2015).

Básicamente las redes neuronales intentan imitar el funcionamiento de sus homólogas biológicas. Hay diferentes tipos en función de su arquitectura y forma de aprendizaje. Por otra parte, las conexiones entre ellas se realizan aleatoriamente y se ajustan progresivamente a medida que se entrenan con los datos disponibles, de manera que la red es capaz de reconocer paulatinamente todos los casos del conjunto de datos utilizado para su entrenamiento, este finaliza cuando es capaz de reconocerlos todos sin fallar. La red es capaz de reconocer patrones con todo tipo de formas, no sólo lineales, como se ha comentado en párrafos anteriores (N. Sáenz Bajo, 2002).

Uno de estos sistemas a los que se refiere es el de sistemas multivariables. En (Faiber Ignacio Robayo B, 2015-2014) se puede ver como se realiza el control de un sistema mediante redes neuronales utilizando MATLAB, por lo que se propone el estudio y desarrollo de una red neuronal para control de nivel, para comprobar su utilización en un sistema como este, y estudiar su desempeño utilizando la herramienta OCTAVE. Con esto por una parte se comprobará y comparará el funcionamiento de una red neuronal frente a un controlador clásico PID. Por otra parte, se comprobará y justificará como realizar la red en la herramienta OCTAVE frente a la herramienta clásica utilizada MATLAB. El sistema que va a ser estudiado consiste en una planta compuesta por 3 tanques, uno de perturbación, el tanque principal y el de reserva, dos electroválvulas (una de perturbación (V1) y otra de control (V2)) y una bomba sumergible. Un sensor de nivel en el tanque principal (S1) y un sensor de caudal en la tubería (S2) después de la válvula de control completan la instrumentación del proceso.

5.2 ESTADO DEL ARTE DE LAS SOLUCIONES EXISTENTES:

Respecto a las soluciones existentes para realizar el control de un sistema se encuentra diferentes posibilidades. A lo largo de este punto se explicarán diferentes soluciones encontradas a una red neuronal y posteriormente casos en los que las redes neuronales se han utilizado para el control de un sistema.

Hablando primero de las soluciones diferentes al control de un sistema multivariable, se puede ver en (Jurado, 2012) como el control vía PID de un sistema multivariable utilizando el desacoplo inverso es muy efectivo. En esta solución se aborda las diferentes posibilidades de configuración que permite el control PID.

En (Montiel, 2009) se puede ver un caso que aborda la necesidad de mantener la seguridad en el acceso de centros de trabajo, para ello se utiliza redes neuronales en el reconocimiento de la voz.

En (N. Sáenz Bajo, 2002) se puede observar que las redes neuronales en el campo de la medicina son múltiples y variadas. Pues estas permiten extraer información útil y producir inferencias a partir de los datos disponibles gracias a su capacidad de aprendizaje. Por su capacidad de reconocimiento de patrones su utilidad es indudable en el campo del diagnóstico anatomopatológico, las pruebas de imagen o las pruebas electrofisiológicas.

En (Rubén Darío Cárdenas Espinosa, 2017) nos acercamos con una aplicación más relacionada con el campo del que trata esta publicación. Aquí se presenta la identificación de parámetros dinámicos para modelar un sistema hidráulico y el diseño de su controlador digital utilizando redes neuronales.

En (Yang & Gong, 2019) se explica las ventajas de utilizar un control PID con redes neuronales para dominar el sistema de bombeo de petróleo, realizando un análisis de su comportamiento y estructura de implementación.

En (Jiang Chang, 2009) se presenta un control PID con redes neuronales para el generador de hidroturbina Francis donde se especifica su relevancia para el control de sistemas no lineales.

En (Jiang-Jiang Wang, 2008) se aplica un control adaptativo PID para la temperatura en una microturbina de gas, se especifica su efectividad y eficiencia de la estrategia de control propuesta.

Como hemos visto en los anteriores ejemplos, existe multitud de análisis sobre el control de distintas variables mediante una red neuronal por lo que se justifica la realización de un control mediante red neuronal para el sistema propuesto.

5.3 DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA:

En base a todo lo planteado anteriormente se propone el control del sistema físico de tanques mediante redes neuronales. La recepción de los datos leídos desde el sistema planta por sensores y el envío de las señales a los actuadores se realizará mediante microcontroladores. Estos serán los que conectarán los sensores de nivel y caudal del sistema, recibirán los datos, los trasmitirán a la red neuronal, esta a su vez generará la respuesta que será enviada al microcontrolador para que modifique los actuadores de la bomba y la válvula en consecuencia. Por las condiciones actuales en las que se encuentra la sociedad debido a la pandemia provocada por el COVID-19 la simulación del sistema será únicamente virtual.

Puesto que se trata de un proyecto de diseño y validación no es útil incluir una valoración económica de la solución propuesta respecto a la realidad o a otras de las soluciones existentes, ya que lo que se estudiará es su desempeño comparándolo con otros controladores y del mismo tipo de controlador, pero en herramientas diferentes.

5.4 OBJETIVO GENERAL:

Adaptar y simular un sistema de control de tanques de nivel mediante la utilización de un controlador basado en redes neuronales.

5.5 OBJETIVOS ESPECÍFICOS:

1. Desarrollar una red neuronal en Octave que mejore el control del sistema frente a otros controladores clásicos.
2. Ajustar todos los parámetros del sistema mediante la realización de pruebas en las simulaciones.

3. Estudio detallado de los parámetros más influyentes en la buena realización y desempeño de una red neuronal.

5.6 METODOLOGÍA Y PLAN DE TRABAJO:

- Creación de la simulación del sistema físico virtualmente.
- Desarrollo de los datos para entrenar la red neuronal.
- Creación de la red neuronal.
- Desarrollo del sistema completo.
- Comparación del funcionamiento de la red frente a otros controladores.
- Escritura detallada del proyecto.
- Comprobación del correcto funcionamiento del sistema.

5.7 ENTREGABLES:

- Simulación del sistema real con el control aplicado.
- Programación del control de red neuronal PID.
- Comparativa del control propuesto respecto a otros controles.

5.8 CARTA GANTT DEL PROYECTO:

	SEMANAS																			
Actividades:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1. Creación de la simulación del sistema físico virtualmente.																				
2. Desarrollo de los datos para entrenar a la red neuronal.																				
3. Creación de la red neuronal.																				
5. Desarrollo del sistema completo																				
6. Comparación del funcionamiento de la red frente a otros controladores.																				
7. Escritura detallada del proyecto.																				
8. Comprobación del correcto funcionamiento del sistema.																				

6 SISTEMA:

A continuación, se realizará una explicación detallada de todos los componentes que intervienen en el sistema y su funcionamiento que justifique las decisiones tomadas para la construcción de la red neuronal en los puntos posteriores.

Como se ha comentado en un punto anterior, el sistema físico consiste en una planta compuesta por 3 tanques, uno de perturbación, el tanque principal y el de reserva, dos electroválvulas (una de perturbación (V1) y otra de control (V2)) y una bomba sumergible. Un sensor de nivel en el tanque principal (S1) y un sensor de caudal en la tubería (S2) después de la válvula de control completan la instrumentación del proceso. En la siguiente ilustración se puede observar el sistema:

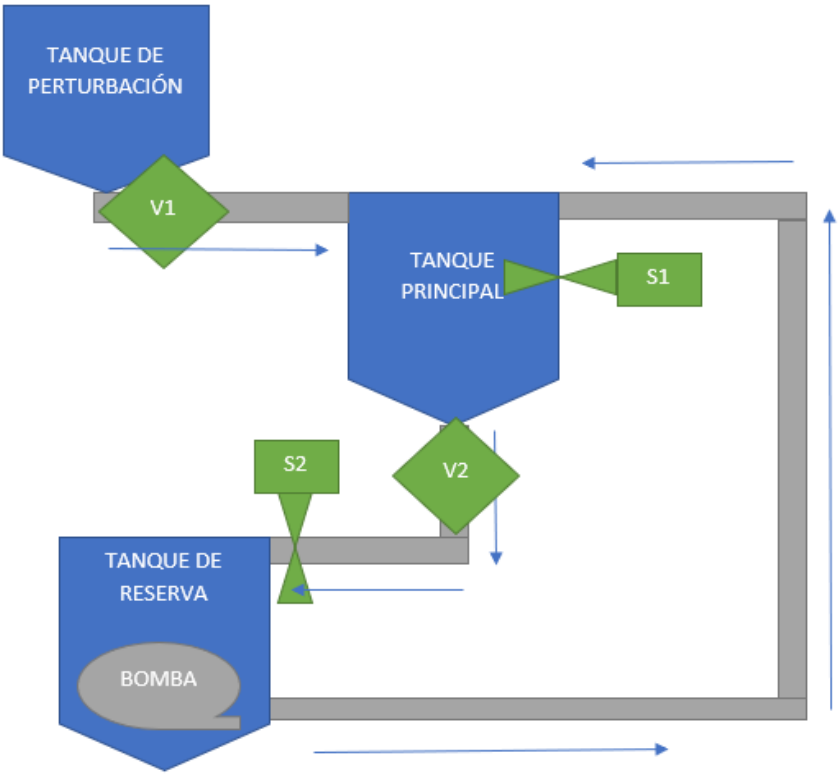


Ilustración 1. Sistema de tanques.

En (Faiber Ignacio Robayo B, 2015-2014) se presenta un diagrama de bloques que representa todos los componentes que intervienen, como se puede ver a continuación:

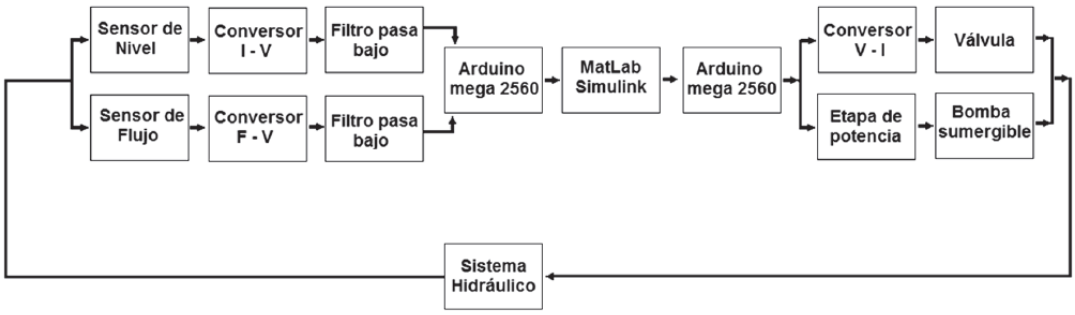


Ilustración 2. Diagrama de bloques del sistema.

Para introducir cuál es el comportamiento real de la instalación que se quiere controlar se ha utilizado las expresiones que modulan su comportamiento en (Faiber Ignacio Robayo B, 2015-2014). Se indica claramente que se puede interpretar su control como dos sistemas SISO desacoplados pues la independencia entre variables es total. Esto se ha comprobado por el método de ganancias relativas. Por lo que se establece que el Nivel debe ser controlado solo y únicamente por la Bomba y el Caudal debe ser controlado solo y únicamente por la Válvula. A partir de ahí se realiza la identificación del sistema físico, esto se realiza mediante la herramienta "Ident" de Matlab. Las soluciones son las siguientes:

Para la variable Nivel, se considera como entrada el voltaje de la bomba y como salida el voltaje de nivel, se aproxima el modelo del subsistema con una aproximación del 93.78% como:

$$G_{11}(s)=\frac{0.0002823}{s^2+0.04295s+0.0002129}$$

Ecuación 1. Función de transferencia de Nivel-Bomba

Para la variable Caudal, se considera como entrada el voltaje de la válvula y como salida el voltaje del caudal, se aproxima el modelo del subsistema con una aproximación del 62.68% como:

$$G_{22}(s)=\frac{0.1452}{s+0.2127}$$

Ecuación 2. Función de transferencia de Caudal-Válvula

7 PROGRAMACIÓN MATLAB:

Puesto que Matlab presenta una herramienta mucho más completa que OCTAVE para la simulación en línea de un sistema como es Simulink, se ha decidido hacer la simulación del sistema con los controladores PID en Matlab, para posteriormente con los datos necesarios para entrenar la red neuronal, transmitírselos al programa Octave.

Se explicará detalladamente la programación y al final del documento, en los anexos, se encuentra la explicación de cada variable junto con el código completo.

7.1 SIMULACIÓN SIMULINK:

En la construcción de toda la programación se ha tenido como máxima la simplicidad del código para que, por una parte, el lector del documento y cualquier persona que vea el código pueda entenderlo y por otra parte para simplificar los cambios que se tengan que realizar. Por ello, se ha intentado introducir el mínimo de variables si estas pueden ser extraídas de otras y también que todos los cambios en el archivo Simulink se realicen desde el archivo de simulación.m, de manera que con un simple cambio en el código se realicen las transformaciones pertinentes.

Para la creación del archivo de simulación se ha tomado como ejemplo el que se puede observar en (Faiber Ignacio Robayo B, 2015-2014). Primero se diseña en lazo abierto para ver cuál es el comportamiento del mismo ante cambios en la referencia, para posteriormente con ese análisis, calcular cuál es el PID óptimo del que se quiere utilizar para entrenar la red neuronal, pues para su entrenamiento se tomará como ejemplo el comportamiento de este PID.

7.1.1 SISTEMA EN LAZO ABIERTO:

Teniendo claro cuál es el comportamiento del sistema real se construye su estructura en el archivo introduciendo las dos funciones de transferencia. Este sistema consistirá en dos entradas, el voltaje de la bomba y el voltaje del caudal que mediante las funciones de transferencia realizarán su efecto en el sistema en forma de voltaje en el sensor de nivel y voltaje en el sensor del caudal. Como queremos saber que PID es más adecuado para cada función de transferencia vamos a construir un escalón de entrada y ver cómo responde cada salida. Con la gráfica de la respuesta y mediante un método de los múltiples disponibles para obtención de los parámetros del PID, se calcula las variables necesarias.

A continuación, se puede observar el sistema completo en lazo abierto:

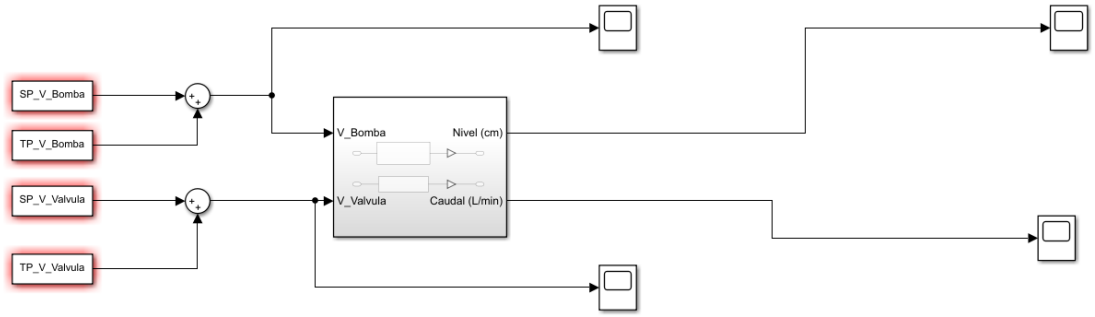


Ilustración 3. Modelo de la Planta en Lazo Abierto.

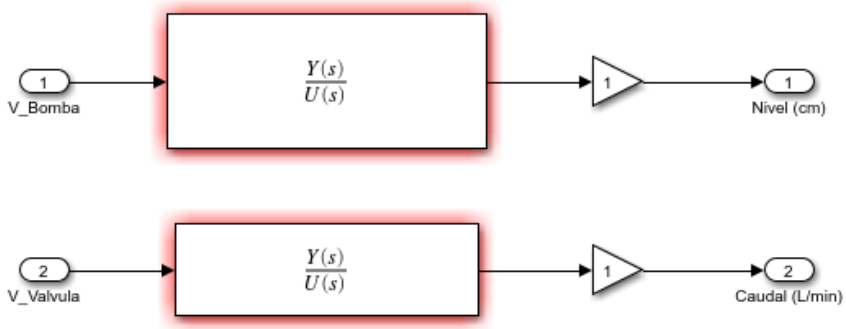


Ilustración 4. Planta Modelo Lazo Abierto.

Se pueden apreciar 4 entradas, dos por variable de entrada:

- SP_V_Bomba: Set Point en para la bomba en lazo abierto.
- TP_V_Bomba: Tren de pulsos para la bomba en lazo abierto.
- SP_V_Válvula: Set Point para la válvula en lazo abierto.
- TP_V_Válvula: Tren de pulsos para la válvula en lazo abierto.

Las entradas se suman, pues una compone el Set Point al que se le va a aplicar el tren de pulsos para así poder cambiarlo fácilmente. Esa entrada será la que se aplique al sistema para ver cuál es su respuesta. Esta respuesta es extraída para ser evaluada.

7.1.2 SISTEMA EN LAZO CERRADO:

A partir del modelo del sistema en lazo abierto se construyó el modelo en lazo cerrado. En este caso tenemos las mismas 4 entradas del apartado anterior junto con las 8 entradas que modulan los parámetros de los dos PIDs:

- TP_Nivel: Tren de Pulsos para el nivel en lazo cerrado.
- SP_Nivel: Set Point para el nivel en lazo cerrado.
- TP_Caudal: Tren de Pulsos para el caudal en lazo cerrado.
- SP_Caudal: Set Point para el caudal en lazo cerrado.

$$P + I\frac{1}{s} + D\frac{N}{1 + N\frac{1}{s}}$$

Ilustración 5. Estructura del PID.

- PB_PID: Valor proporcional del PID de la bomba.
- IB_PID: Valor integral del PID de la bomba.
- DB_PID: Valor derivativo del PID de la bomba.
- NB_PID Valor de N de la parte derivativa del PID de la bomba.
- PV_PID: Valor proporcional del PID de la válvula.
- IV_PID: Valor integral del PID de la válvula.
- DV_PID: Valor derivativo del PID de la válvula.
- NV_PID: Valor de N de la parte derivativa del PID de la válvula.

A continuación, se muestra la estructura del archivo Simulink de lazo cerrado:

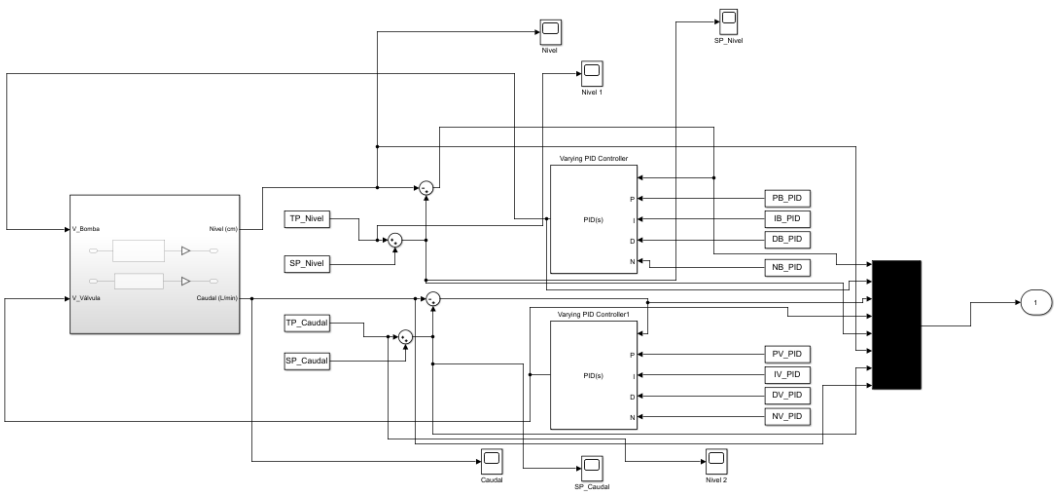


Ilustración 6. Archivo Simulink de Lazo Cerrado.

En él se puede observar a la izquierda el modelo de la planta que se presenta en la siguiente ilustración, como se puede apreciar es la misma que en lazo abierto ya que la planta en ambos casos es igual. Se compone de la entrada que en cada caso es el voltaje de la bomba o de la válvula, pasa a través de la función de transferencia y de un multiplicador que lo transforma a centímetros y L/min respectivamente.

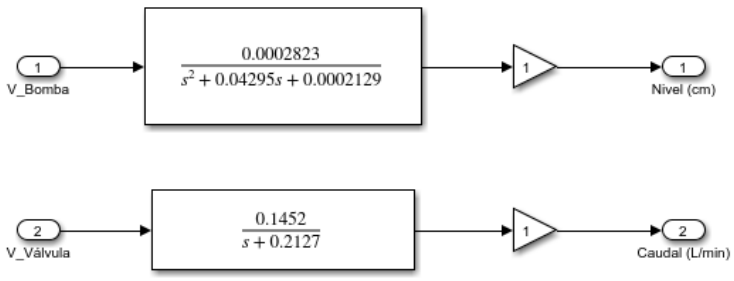


Ilustración 7. Planta Modelo Lazo Cerrado.

La salida de la planta es comparada con la entrada que tenemos como referencia para así poder restarlas y ver cuál es el error. La entrada de la referencia se ha construido de la misma manera que en Lazo Abierto, se tiene una entrada que es el Set Point al que se le suma el Tren de Pulsos. Con el error entre el estado actual y la referencia se introduce en el PID. Los PIDs con construidos como se ve a continuación, con el bloque, las entradas de los parámetros y la entrada con el error.

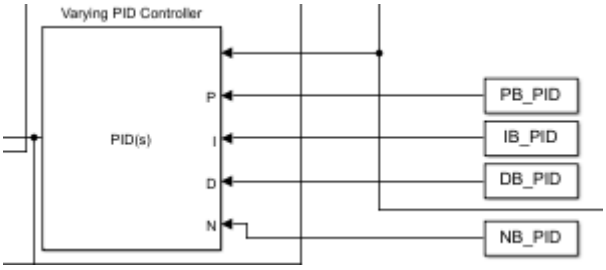


Ilustración 8. PID de la Válvula.

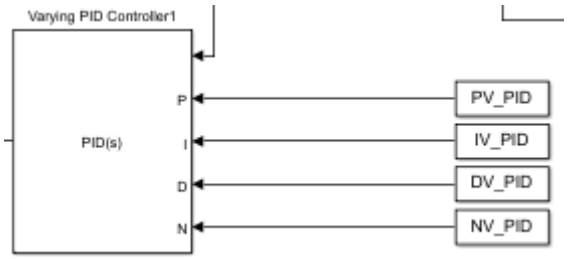


Ilustración 9. PID de la Bomba.

La salida de los PIDs es transferida al sistema como entrada y así continuar el bucle. Los datos que se han elegido salidas para poder obtenerlos y tratar con ellos son la entrada y salida del PID de nivel, la entrada y salida del PID de caudal, la referencia y salida de la planta de nivel y la referencia y salida de la planta de caudal respectivamente.

7.2 ARCHIVO DE SIMULACIÓN:

A continuación, se expondrá el código del archivo.m de Matlab que se encarga de cargar los parámetros para la simulación, crear los tiempos, trenes de pulsos y variables necesarias y que simula el archivo de Simulink explicando detalladamente cuál es la función de cada parte del código en todo el ensamblado.

7.2.1 INICIACIÓN:

```
%Archivo para definir los parámetros de la planta y simular el archivo de
%simulink .mdl
%tren=1; %Validate Data
tren=2; %Train Data
%tren=3; %Val Train Data
```

Ilustración 10. Tipo de Tren.

Lo primero es definir qué tipo de tren vamos a construir, esto se hará dependiendo del valor que introduzcamos en “tren”.

- 1: Datos de Validación.
- 2: Datos de entrenamiento.

- 3: Datos para validar cuando hayas entrenado.

7.2.2 PARÁMETROS:

Cuando ya hemos elegido que tipo de datos vamos a entrenar se establecen los parámetros y constantes que necesitamos para la simulación.

```
%% PARÁMETROS
%Funciones de transferencia
Num_V_Bomba = 0.0002823;
Den_V_Bomba = [1 0.04295 0.0002129];
Num_V_Valvula = 0.1452;
Den_V_Valvula = [1 0.2127];
%Temporales
ts=0.01;
```

Ilustración 11. Parámetros de Funciones De Transferencia.

Son el numerador y denominador de cada función de transferencia, de la de la bomba y la de la válvula respectivamente. También se establece el intervalo de tiempo para cada discretizar el vector de tiempos.

```
%Parámetros del PID de la Bomba/Nivel
PB=250;
IB=2;
DB=5800;
NB=1;
PB_PID=[0,PB;tend,PB];
IB_PID=[0,IB;tend,IB];
DB_PID=[0,DB;tend,DB];
NB_PID=[0,NB;tend,NB];
%Parámetros del PID de la Válvula/Caudal
PV=7;%PV=10;
IV=2;
DV=0;
NV=0;
PV_PID=[0,PV;tend,PV];
IV_PID=[0,IV;tend,IV];
DV_PID=[0,DV;tend,DV];
NV_PID=[0,NV;tend,NV];
Sim='cerrado';
```

Ilustración 12. Parámetros del PID para la Bomba/Nivel y Válvula/Caudal.

Puesto que se va a introducir los valores desde al archivo simulación se tiene que introducir como un archivo temporal, pero para hacerlo constante se establece que valga lo mismo desde el instante inicial al final.

7.2.3 SIMULACIÓN MODELO EN LAZO CERRADO:

Por último, se tiene la variable “Sim” que indica cuál de los dos archivos Simulink queremos simular, el de lazo abierto (Sim == ’abierto’) o lazo cerrado (Sim == ’cerrado’). Según el valor de la variable se ejecutará una de las partes que tenemos a continuación:

```
%% SIMULACIÓN MODELO LAZO CERRADO
if Sim=='cerrado'
    vt = 0:ts:tend;
    [t, estados, resultados_cer]=sim('modelo_planta',vt);
    %Exportar datos a excel, según estemos utilizando un patrón de tren de
    %pulsos u otro
    if tren==2
        xlswrite('Train_Data.xlsx',resultados_cer);
    end
    if tren==1
        xlswrite('Validate_Data.xlsx',resultados_cer);
    end
    if tren==3
        xlswrite('Val_train_Data.xlsx',resultados_cer);
    end
end
```

Ilustración 13. Simulación Modelo Cerrado.

En el código de simulación, empezando por el caso del modelo cerrado, primero se genera un vector vt. Este compone los intervalos de simulación, desde el instante inicial al final ("tend") con un intervalo ("ts"). Posteriormente se simula el modelo eligiendo el nombre del modelo que se quiere simular y el vector de tiempos de simulación. Según las variables que se pongan de salida se entregará unos datos y otros. La primera variable ("t") representa la salida del vector de tiempos, la segunda ("estados") los estados de todas las variables que intervienen en el proceso y la tercera ("resultados_cer") cuáles son las salidas que se ha introducido en el modelo. Cómo tenemos diferentes trenes de pulsos tenemos que guardar las salidas en excels diferentes, por lo tanto, dependiendo del valor de la variable "tren" los datos se exportarán a un Excel con un nombre diferente.

7.2.4 SIMULACIÓN MODELO EN LAZO ABIERTO:

```
%% SIMULACIÓN MODELO LAZO ABIERTO
if Sim=='abierto'
    vt = 0:ts:tend;
    [t, resultados_abi]=sim('modelo_planta_lazo_abierto',vt);
    xlswrite('Res_Abierto.xlsx',resultados_abi);
end
```

Ilustración 14. Simulación Modelo Abierto

En el caso del modelo abierto, igual que antes, se genera el vector de tiempos de simulación, se realiza la simulación y por último se exporta los datos en el Excel.

7.2.5 PARÁMETROS DEL PID:

Si imponemos una acción escalón al modelo de la planta en lazo abierto y aplicamos el método de Ziegler-Nichols obtenemos, para el lazo de Nivel las siguientes características del PID:

```
PB=250;
IB=2;
DB=5800;
NB=1;
```

Ilustración 15. Características PID Nivel.

Por otra parte, si analizamos el lazo de Caudal, obtenemos:

```
PV=7;
IV=2;
DV=0;
NV=0;
```

Ilustración 16. Caracterísitcas PID Caudal.

Con estas condiciones ya podemos realizar simulaciones con el controlador PID.

7.2.6 PRUEBAS CON DATOS DE VALIDACIÓN (TREN=1):

Primero, cuando se comprueba que se ha seleccionado los datos de validación y que la variable tren vale 1, se introduce los parámetros estáticos de la simulación empezando por la variable nivel: El Set Point sobre el que se quiere sumar el tren de pulsos, que tiempo necesita la bomba para alcanzar un cambio en la referencia, cuál es el tiempo de establecimiento final y el vector que con la constante del Set Point genera la variable que se puede introducir en Simulink.

```
%% Pruebas PID
if tren == 1
    SPNivel=20;
    Te_Bomba =6.5*2;
    tend=Te_Bomba*23;
    SP_Nivel=[0,SPNivel; tend,SPNivel];
```

Ilustración 17. Datos de Validación, Parámetros de Nivel.

7.2.6.1 TREN DE PULSOS VARIABLE PARA EL LAZO DE NIVEL:

El tren de pulsos se construye como un vector de dos columnas, la primera columna constituye el vector de tiempos y la segunda que valor vale la variable para ese instante. De manera que cada instante será un múltiplo de la variable “Te_Bomba”. Para que el vector se construya como cambios en un instante se tienen que estructurar de la siguiente manera, empezando en un valor en un instante, en el instante que quieras cambiar tienes que poner ese instante, el valor que tenía anteriormente y a continuación el mismo instante con el nuevo valor de la variable, así en un instante ínfimo de tiempo pasará de un valor a otro en forma de escalón. Para la validación se construirá un tren de pulsos con cambios muy leves que no generen grandes cambios, por lo que serán escalones de como máximo 1 unidad y en total se moverá 3 unidades por encima y 3 por debajo.

```
TP_Nivel=[0,0; Te_Bomba,0; Te_Bomba,1;...
    Te_Bomba*2,1;Te_Bomba*2,-1;...
    Te_Bomba*3,-1; Te_Bomba*3,0;...
    Te_Bomba*4,0;Te_Bomba*4,1;...
    Te_Bomba*5,1; Te_Bomba*5,1.5;...
    Te_Bomba*6,1.5;Te_Bomba*6,2;...
    Te_Bomba*7,2; Te_Bomba*7,3;...
    Te_Bomba*8,3;Te_Bomba*8,2.5;...
    Te_Bomba*9,2.5; Te_Bomba*9,2;...
    Te_Bomba*10,2; Te_Bomba*10,1.5;...
    Te_Bomba*11,1.5; Te_Bomba*11,1;...
    Te_Bomba*12,1; Te_Bomba*12,0;...
    Te_Bomba*13,0; Te_Bomba*13,-1;...
    Te_Bomba*14,-1; Te_Bomba*14,-1.5;...
    Te_Bomba*15,-1.5; Te_Bomba*15,-2;...
    Te_Bomba*16,-2; Te_Bomba*16,-2.5;...
    Te_Bomba*17,-2.5; Te_Bomba*17,-3;...
    Te_Bomba*18,-3; Te_Bomba*18,-2.5;...
    Te_Bomba*19,-2.5; Te_Bomba*19,-2;...
    Te_Bomba*20,-2; Te_Bomba*20,-1.5;...
    Te_Bomba*21,-1.5; Te_Bomba*21,-1;...
    Te_Bomba*22,-1; Te_Bomba*22,0;...
    tend,0];
```

Ilustración 18. Datos de Validación, Construcción de Tren de Pulsos de Nivel.

Si simulamos el código anterior y graficamos la variable TP_Nivel, obtendremos la siguiente gráfica con el tren de pulsos correspondiente. Se puede observar cómo son escalones pequeños con cambios leves para ver una respuesta leve del sistema.

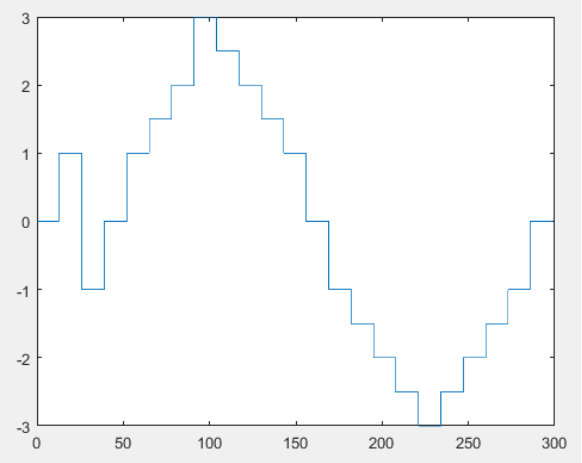


Ilustración 19. Datos de Validación, Tren de Pulsos Variable de Nivel.

Se realiza la simulación para ver cuál es el funcionamiento del PID frente a este tren de pulsos. El resultado es el siguiente:

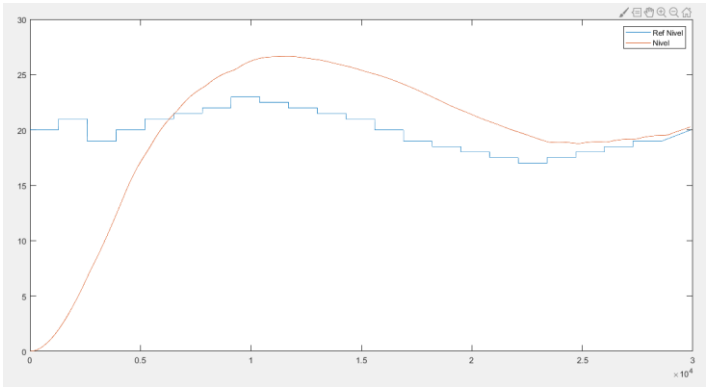


Ilustración 20. Datos de Validación, Comportamiento PID al Tren de Pulsos Variable.

Como podemos observar, el PID no es capaz de alcanzar correctamente el Tren de Pulsos. Esto es debido a que al limitar la salida del PID entre 0 y 100% para hacerlo de una manera más real, encuentra que sus acciones son insuficientes cuando el tren de pulsos cambia tan rápidamente.

Se puede observar en los datos que todas las acciones iniciales toman el máximo de sus posibilidades.

7.2.6.2 TREN DE PULSOS VARIABLE PARA EL LAZO DE CAUDAL:

Continuamos con la variable caudal, definimos el Set Point, el tiempo que tarda en reaccionar y calculamos el vector final de simulación para esta variable, como las dos se simulan a la vez, se calcula que variable va a tardar más en realizar su simulación y en base a ello simular las dos ese tiempo. Pues la que tarde menos, llegará un momento que se parará y se mantendrá en cero hasta el final de la simulación. Finalmente generamos el vector que vamos a traspasar al archivo de simulación.

```
SPCaudal=5;
Te_Valvula = 3.7*2;
if Te_Valvula*23>tend %Si el tiempo de simulación del sp_caudal
    %es más grande que el de nivel, lo ponemos como final de sim
    tend=Te_Valvula*9;
end
SP_Caudal=[0,SPCaudal;tend,SPCaudal];
```

Ilustración 21. Datos de Validación, Iniciación Parámetros de Caudal.

De la misma manera que se ha construido el tren de pulsos para la referencia de nivel, se genera el tren de pulsos para la referencia del caudal, con la misma estructura como se puede observar a continuación.

```
TP_Caudal=[0,0; Te_Valvula,0; Te_Valvula,1;...
   Te_Valvula*2,1;Te_Valvula*2,-1;...
    Te_Valvula*3,-1; Te_Valvula*3,0;...
    Te_Valvula*4,0; Te_Valvula*4,1;...
    Te_Valvula*5,1; Te_Valvula*5,1.5;...
    Te_Valvula*6,1.5; Te_Valvula*6,2;...
    Te_Valvula*7,2; Te_Valvula*7,3;...
    Te_Valvula*8,3; Te_Valvula*8,2.5;...
    Te_Valvula*9,2.5; Te_Valvula*9,2;...
    Te_Valvula*10,2; Te_Valvula*10,1.5;...
    Te_Valvula*11,1.5; Te_Valvula*11,1;...
    Te_Valvula*12,1; Te_Valvula*12,0;...
    Te_Valvula*13,0; Te_Valvula*13,-1;...
    Te_Valvula*14,-1; Te_Valvula*14,-1.5;...
    Te_Valvula*15,-1.5; Te_Valvula*15,-2;...
    Te_Valvula*16,-2; Te_Valvula*16,-2.5;...
    Te_Valvula*17,-2.5; Te_Valvula*17,-3;...
    Te_Valvula*18,-3; Te_Valvula*18,-2.5;...
    Te_Valvula*19,-2.5; Te_Valvula*19,-2;...
    Te_Valvula*20,-2; Te_Valvula*20,-1.5;...
    Te_Valvula*21,-1.5; Te_Valvula*21,-1;...
    Te_Valvula*22,-1; Te_Valvula*22,0;...
    tend,0];

end
```

Ilustración 22. Datos de Validación, Construcción de Tren de Pulsos de Caudal.

Si simulamos el código anterior y graficamos una de las variables de TP_Caudal sobre la otra, se obtiene el siguiente resultado:

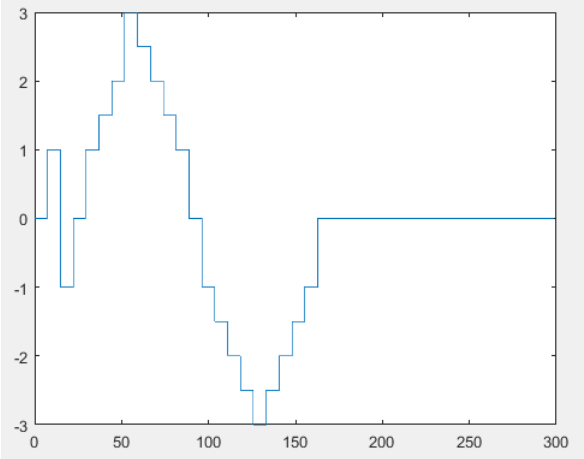


Ilustración 23. Datos de Validación, Tren de Pulsos Variable de Caudal.

Se puede observar los cambios leves en el tren de pulsos de la referencia del caudal que suben hasta 3 unidades del Set Point y bajan las mismas. También se puede ver como en efecto la variable Caudal por su tiempo de establecimiento menor hace la simulación en menos tiempo, pero como se simulan las dos variables a la vez tiene que esperar a que la primera acabe, manteniéndose en valor nulo.

Se realiza la simulación para ver cuál es el funcionamiento del PID frente a este tren de pulsos. El resultado es el siguiente:

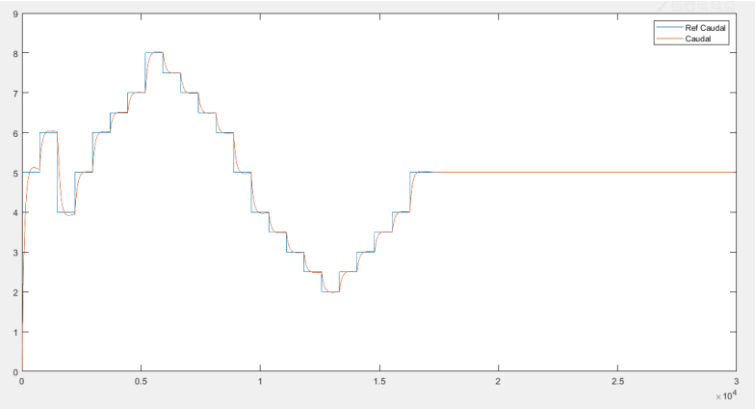


Ilustración 24. Comportamiento del PID frente al Tren de Pulsos Variable de Caudal.

Como se puede observar, en este caso el caudal si es capaz de alcanzar completamente la referencia sin problemas, aunque se haya limitado el PID. Este suceso es debido a que las acciones que el PID necesita realizar se encontraban ya dentro de los límites, por lo que, aunque los limitemos a 0 y 100% no hay ningún problema.

7.2.6.3 TREN DE PULSOS ASCENDENTE PARA EL LAZO DE NIVEL:

Como se encontró que el comportamiento de la red frente a todo el patrón de subidas y bajadas era extraño, se realizó un análisis más minucioso sobre cómo se comportaba esa respecto a las subidas y después respecto a las bajadas para entender mejor su comportamiento. Por lo que se realizó otro patrón en el que ascendiese escalones pequeños (incrementos de 0.1). A continuación, se expresa el código que crea el patrón para la variable Nivel.

```
x=0.1;
TP_Nivel=[0,0; Te_Bomba,0; Te_Bomba,x;...
    Te_Bomba*2,x;Te_Bomba*2,2*x;...
    Te_Bomba*3,2*x; Te_Bomba*3,3*x;...
    Te_Bomba*4,3*x;Te_Bomba*4,4*x;...
    Te_Bomba*5,4*x; Te_Bomba*5,5*x;...
    Te_Bomba*6,5*x;Te_Bomba*6,6*x;...
    Te_Bomba*7,6*x; Te_Bomba*7,7*x;...
    Te_Bomba*8,7*x;Te_Bomba*8,8*x;...
    Te_Bomba*9,8*x; Te_Bomba*9,9*x;...
    Te_Bomba*10,9*x; Te_Bomba*10,10*x;...
    Te_Bomba*11,10*x; Te_Bomba*11,11*x;...
    Te_Bomba*12,11*x; Te_Bomba*12,12*x;...
    Te_Bomba*13,12*x; Te_Bomba*13,13*x;...
    Te_Bomba*14,13*x; Te_Bomba*14,14*x;...
    Te_Bomba*15,14*x; Te_Bomba*15,15*x;...
    Te_Bomba*16,15*x; Te_Bomba*16,16*x;...
    Te_Bomba*17,16*x; Te_Bomba*17,17*x;...
    Te_Bomba*18,17*x; Te_Bomba*18,18*x;...
    Te_Bomba*19,18*x; Te_Bomba*19,19*x;...
    Te_Bomba*20,19*x; Te_Bomba*20,20*x;...
    Te_Bomba*21,20*x; Te_Bomba*21,21*x;...
    Te_Bomba*22,21*x; Te_Bomba*22,22*x;...
    tend,22*x];
```

Ilustración 25. Datos de validación. Construcción de Tren Ascendente de Nivel.

Ahora se muestra la representación del tren de pulsos, y como el PID controla la variable nivel hasta alcanzar y seguir el tren de pulsos.

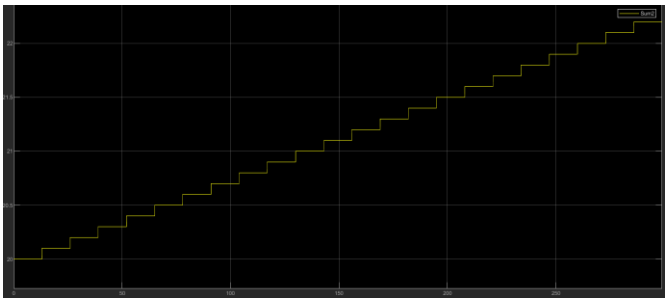


Ilustración 26. Datos de Validación. Tren de Pulsos Ascendente Nivel.

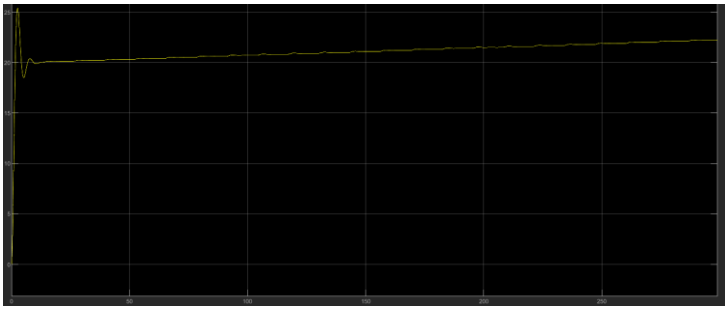


Ilustración 27. Datos de Validación. Comportamiento del Nivel frente Tren de Pulsos Ascendente.

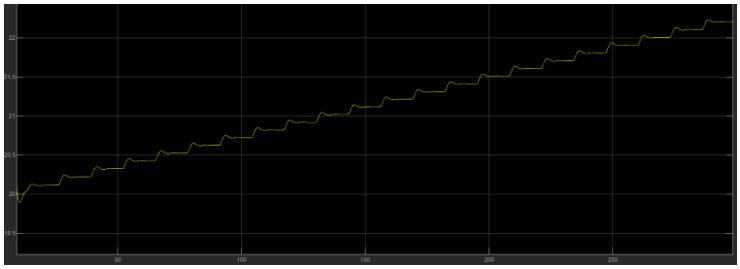


Ilustración 28. Datos de Validación. Comportamiento del Nivel frente Tren de Pulsos Ascendente Zoom.

7.2.6.4 TREN DE PULSOS ASCENDENTE PARA EL LAZO DE CAUDAL:

A continuación, el código para construir el tren de pulsos ascendente para la variable caudal.

```
y=0.1;
SP_Caudal=[0,SPCaudal;tend,SPCaudal];
TP_Caudal=[0,0; Te_Valvula,0; Te_Valvula,2*y;...
   Te_Valvula*2,2*y;Te_Valvula*2,3*y;...
    Te_Valvula*3,3*y; Te_Valvula*3,4*y;...
    Te_Valvula*4,4*y; Te_Valvula*4,5*y;...
    Te_Valvula*5,5*y; Te_Valvula*5,6*y;...
    Te_Valvula*6,6*y; Te_Valvula*6,7*y;...
    Te_Valvula*7,7*y; Te_Valvula*7,8*y;...
    Te_Valvula*8,8*y; Te_Valvula*8,9*y;...
    Te_Valvula*9,9*y; Te_Valvula*9,10*y;...
    Te_Valvula*10,10*y; Te_Valvula*10,11*y;...
    Te_Valvula*11,11*y; Te_Valvula*11,12*y;...
    Te_Valvula*12,12*y; Te_Valvula*12,13*y;...
    Te_Valvula*13,13*y; Te_Valvula*13,14*y;...
    Te_Valvula*14,14*y; Te_Valvula*14,15*y;...
    Te_Valvula*15,15*y; Te_Valvula*15,16*y;...
    Te_Valvula*16,16*y; Te_Valvula*16,17*y;...
    Te_Valvula*17,17*y; Te_Valvula*17,18*y;...
    Te_Valvula*18,18*y; Te_Valvula*18,19*y;...
    Te_Valvula*19,19*y; Te_Valvula*19,20*y;...
    Te_Valvula*20,20*y; Te_Valvula*20,21*y;...
    Te_Valvula*21,21*y; Te_Valvula*21,22*y;...
    Te_Valvula*22,22*y; Te_Valvula*22,23*y;...
    tend,23*y];
```

Ilustración 29. Datos de validación. Construcción de Tren Ascendente de Caudal.

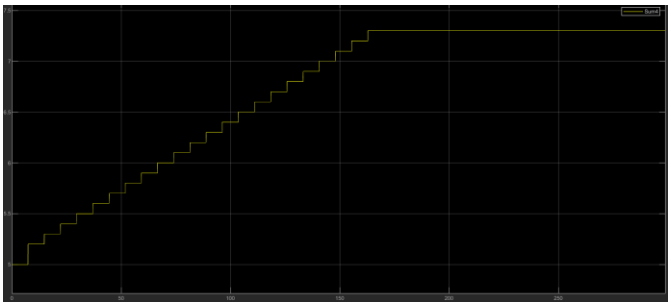


Ilustración 30. Datos de Validación. Tren de Pulsos Ascendente Caudal.

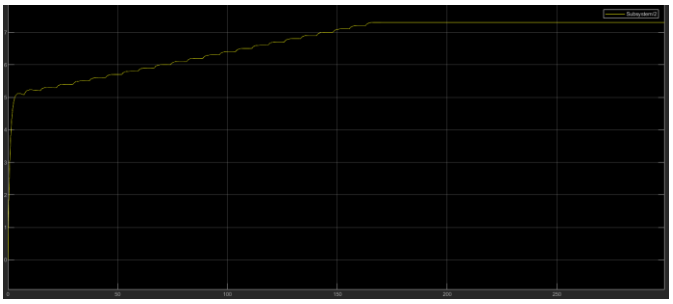


Ilustración 31. Datos de Validación. Comportamiento del Caudal frente Tren de Pulsos Ascendente.

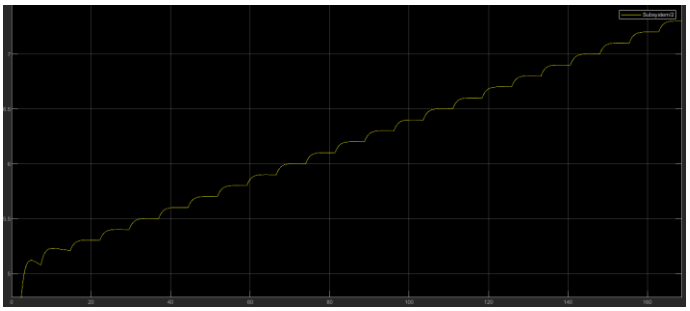


Ilustración 32. Datos de Validación. Comportamiento del Caudal frente Tren de Pulsos Ascendente Zoom.

7.2.6.5 TREN DE PULSOS DESCENDENTE PARA EL LAZO DE NIVEL:

De la misma manera que se ha construido el tren de pulsos ascendente, se realiza de manera descendente para las dos variables mencionadas. Primero se muestra el código de la variable nivel.

```
x=-0.1;
TP_Nivel=[0,0; Te_Bomba,0; Te_Bomba,x;...
    Te_Bomba*2,x;Te_Bomba*2,2*x;...
    Te_Bomba*3,2*x; Te_Bomba*3,3*x;...
    Te_Bomba*4,3*x;Te_Bomba*4,4*x;...
    Te_Bomba*5,4*x; Te_Bomba*5,5*x;...
    Te_Bomba*6,5*x;Te_Bomba*6,6*x;...
    Te_Bomba*7,6*x; Te_Bomba*7,7*x;...
    Te_Bomba*8,7*x;Te_Bomba*8,8*x;...
    Te_Bomba*9,8*x; Te_Bomba*9,9*x;...
    Te_Bomba*10,9*x; Te_Bomba*10,10*x;...
    Te_Bomba*11,10*x; Te_Bomba*11,11*x;...
    Te_Bomba*12,11*x; Te_Bomba*12,12*x;...
    Te_Bomba*13,12*x; Te_Bomba*13,13*x;...
    Te_Bomba*14,13*x; Te_Bomba*14,14*x;...
    Te_Bomba*15,14*x; Te_Bomba*15,15*x;...
    Te_Bomba*16,15*x; Te_Bomba*16,16*x;...
    Te_Bomba*17,16*x; Te_Bomba*17,17*x;...
    Te_Bomba*18,17*x; Te_Bomba*18,18*x;...
    Te_Bomba*19,18*x; Te_Bomba*19,19*x;...
    Te_Bomba*20,19*x; Te_Bomba*20,20*x;...
    Te_Bomba*21,20*x; Te_Bomba*21,21*x;...
    Te_Bomba*22,21*x; Te_Bomba*22,22*x;...
    tend,22*x];
```

Ilustración 33. Datos de validación. Construcción de Tren Descendente de Nivel.

Ahora la representación del tren de pulsos junto con el comportamiento del sistema controlado por el PID para alcanzar ese tren descendente en la variable Nivel.

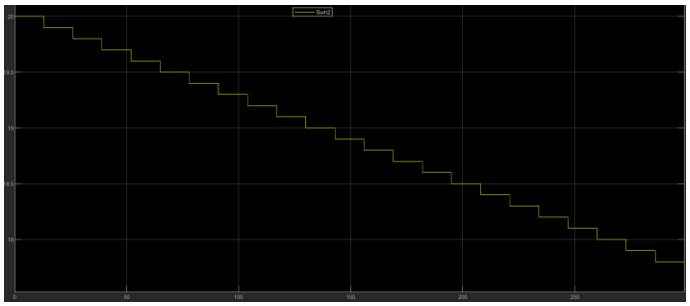


Ilustración 34. Ilustración 31. Datos de Validación. Tren de Pulsos Descendente Nivel.

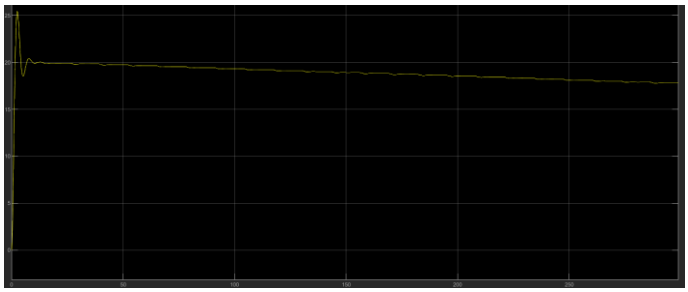


Ilustración 35. Datos de Validación. Comportamiento del Nivel frente Tren de Pulsos Descendente.

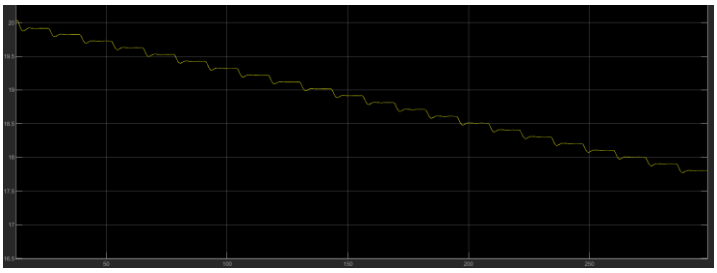


Ilustración 36. Datos de Validación. Comportamiento del Nivel frente Tren de Pulsos Descendente Zoom.

7.2.6.6 TREN DE PULSOS DESCENDENTE PARA EL LAZO DE CAUDAL:

De la misma manera, se procede para la variable caudal, primero con el código.

```
y=-0.1;
SP_Caudal=[0,SPCaudal;tend,SPCaudal];
TP_Caudal=[0,0; Te_Valvula,0; Te_Valvula,2*y;...
   Te_Valvula*2,2*y;Te_Valvula*2,3*y;...
    Te_Valvula*3,3*y; Te_Valvula*3,4*y;...
    Te_Valvula*4,4*y; Te_Valvula*4,5*y;...
    Te_Valvula*5,5*y; Te_Valvula*5,6*y;...
    Te_Valvula*6,6*y; Te_Valvula*6,7*y;...
    Te_Valvula*7,7*y; Te_Valvula*7,8*y;...
    Te_Valvula*8,8*y; Te_Valvula*8,9*y;...
    Te_Valvula*9,9*y; Te_Valvula*9,10*y;...
    Te_Valvula*10,10*y; Te_Valvula*10,11*y;...
    Te_Valvula*11,11*y; Te_Valvula*11,12*y;...
    Te_Valvula*12,12*y; Te_Valvula*12,13*y;...
    Te_Valvula*13,13*y; Te_Valvula*13,14*y;...
    Te_Valvula*14,14*y; Te_Valvula*14,15*y;...
    Te_Valvula*15,15*y; Te_Valvula*15,16*y;...
    Te_Valvula*16,16*y; Te_Valvula*16,17*y;...
    Te_Valvula*17,17*y; Te_Valvula*17,18*y;...
    Te_Valvula*18,18*y; Te_Valvula*18,19*y;...
    Te_Valvula*19,19*y; Te_Valvula*19,20*y;...
    Te_Valvula*20,20*y; Te_Valvula*20,21*y;...
    Te_Valvula*21,21*y; Te_Valvula*21,22*y;...
    Te_Valvula*22,22*y; Te_Valvula*22,23*y;...
    tend,23*y];
```

Ilustración 37. Datos de validación. Construcción de Tren Descendente de Caudal.

Ahora con la representación y el comportamiento de la variable Caudal.

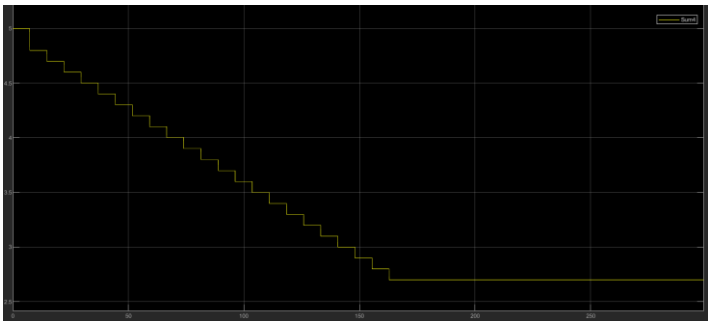


Ilustración 38. Datos de Validación. Tren de Pulsos Descendente Caudal.

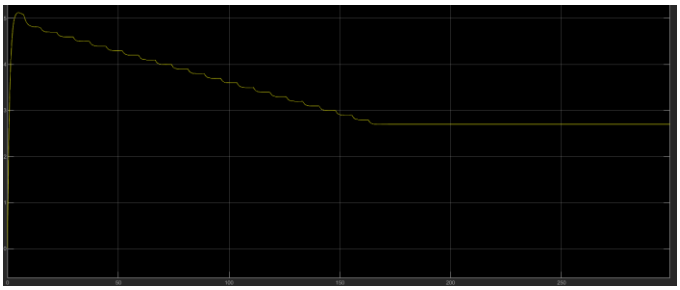


Ilustración 39. Datos de Validación. Comportamiento del Caudal frente Tren de Pulsos Descendente.

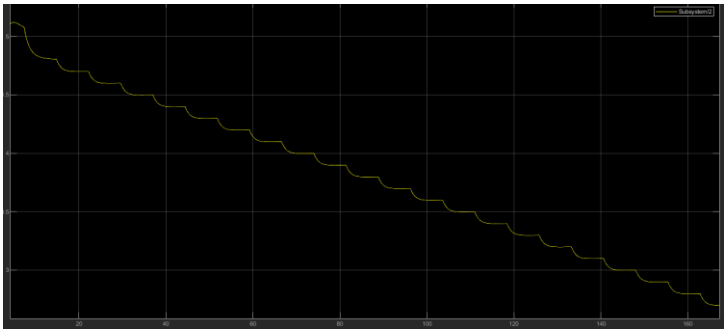


Ilustración 40. Datos de Validación. Comportamiento del Caudal frente Tren de Pulsos Descendente Zoom.

7.2.7 PRUEBAS CON DATOS DE ENTRENAMIENTO (TREN=2):

Cuando la variable de Tren es igual a dos, se va a simular el archivo Simulink con los controladores PIDs con el patrón de entrenamiento de la red neuronal, para conocer cómo se comporta el PID y entrenar la red en consecuencia intentando imitar su comportamiento.

El patrón de entrenamiento se compondrá de un tren de pulsos que se generará en un bucle "for", desde fuera del bucle se le indicará las condiciones que va a tener ese tren para que se cree automáticamente como se ve a continuación para el tren de pulsos de la bomba. Así, si se requiere cambiar alguna variable simplemente se cambia en esta parte y no en cada línea del bucle "for".

```
%% Trenes de pulsos
if tren == 2
    %Bomba
    %Si es un pulso de 2T, otro de T, otro de T/2 y otro de T/4,
    %cada pulso necesita 2 periodos mas otro de separacion
    Te_Bomba =6.5*2;
    N_Pulsos = 12;
    T_ini = Te_Bomba;
    %Set Point del Nivel
    SPNivel=20;
    Por_Su_Ba=0.02*SPNivel;
    TP_Nivel=zeros(23*N_Pulsos,2);
    Salto_Tren_Nivel=SPNivel*0.25;
    x=0;
```

Ilustración 41. Datos de Entrenamiento, Condiciones Tren de Pulsos Bomba.

Se define el tiempo de establecimiento de la bomba, para así crear escalones que se espacien lo suficiente para que se pueda alcanzar la referencia, cuantos pulsos se necesita, el tiempo que tarda en empezar la bomba, que como parte del estado de reposo y tiene que alcanzar el Set Point se ha dejado también el tiempo de establecimiento al inicio antes de que empiece el tren de pulsos. También se define el Set Point desde el que se quiere hacer el tren de pulsos ("SPNivel"), el salto del Set Point, cuando ya se haya realizado el tren de pulsos ("Por_Su_Ba"), que en este caso es de un 2%. Esto se ha realizado para tener un patrón de entrenamiento más completo, ya que no solo se realiza el tren de pulsos sobre el punto de funcionamiento normal sino otro un poco por encima y otro un poco por debajo con lo que se obtiene mucha más información de cómo debe trabajar. Después se inicializa el vector del tren de pulsos y se introduce de cuánto van a ser los escalones del tren de pulsos, en este caso será un 25% del Set Point.

A continuación, se muestra el bucle for que construye el tren a partir de los datos anteriores:

```
for i=1:N_Pulsos
    for j=1:4
        TP_Nivel(i+(23*(i-1))+(6*(j-1)),:)= [T_ini+x, 0];
        TP_Nivel(i+(23*(i-1))+(6*(j-1))+1,:)= [T_ini+x, Salto_Tren_Nivel];

        TP_Nivel(i+(23*(i-1))+(6*(j-1))+2,:)= [T_ini+x+(Te_Bomba/(2^(j-1))), Salto_Tren_Nivel];
        TP_Nivel(i+(23*(i-1))+(6*(j-1))+3,:)= [T_ini+x+(Te_Bomba/(2^(j-1))), -Salto_Tren_Nivel];

        TP_Nivel(i+(23*(i-1))+(6*(j-1))+4,:)= [T_ini+x+(Te_Bomba/(2^(j-1)))*2, -Salto_Tren_Nivel];
        TP_Nivel(i+(23*(i-1))+(6*(j-1))+5,:)= [T_ini+x+(Te_Bomba/(2^(j-1)))*2, 0];

        x=TP_Nivel(i+(23*(i-1))+(6*(j-1))+5,1);
    end
    x=x+Te_Bomba*2;
end
```

Ilustración 42. Datos de Entrenamiento, Tren de Pulsos Bomba.

El primer bucle indica el número de pulsos que se quiere realizar, el segundo bucle construye los 4 escalones que componen cada pulso, estos 4 escalones tienen el mismo salto, pero tienen una duración diferente, esta duración es el doble del período, el período, la mitad y un cuarto respectivamente. Para hacer esto se introduce en la duración la variable j.

```
tt=Te_Bomba*2;
tend=x;%igualamos el tiempo de simulación al máximo tiempo de SP de nivel
SP_Nivel=[0,SPNivel;TP_Nivel(96,1)+tt,SPNivel;TP_Nivel(96,1)+tt,SPNivel+Por_Su_Ba;...
    TP_Nivel(192,1)+tt,SPNivel+Por_Su_Ba;TP_Nivel(192,1)+tt,SPNivel-Por_Su_Ba;...
    tend,SPNivel-Por_Su_Ba];
```

Ilustración 43. Datos de Entrenamiento, Set Point del Nivel.

```
%Válvula
Te_Valvula = 3.7*2;
N_Pulsos = 12;
T_ini = Te_Valvula;
x=0;
%Set Point de la Válvula
SPCaudal=5;
Por_Su_Ba=0.02*SPCaudal;
TP_Caudal=zeros(23*N_Pulsos,2);
Salto_Tren_Caudal=0.25*SPCaudal;
```

Ilustración 44. Datos de Entrenamiento, Condiciones Tren de Pulsos Válvula.

```
for i=1:N_Pulsos
    for j=1:4
        TP_Caudal(i+(23*(i-1))+(6*(j-1)),:)= [T_ini+x, 0];
        TP_Caudal(i+(23*(i-1))+(6*(j-1))+1,:)= [T_ini+x, Salto_Tren_Caudal];

        TP_Caudal(i+(23*(i-1))+(6*(j-1))+2,:)= [T_ini+x+(Te_Valvula/(2^(j-1))), Salto_Tren_Caudal];
        TP_Caudal(i+(23*(i-1))+(6*(j-1))+3,:)= [T_ini+x+(Te_Valvula/(2^(j-1))), -Salto_Tren_Caudal];

        TP_Caudal(i+(23*(i-1))+(6*(j-1))+4,:)= [T_ini+x+(Te_Valvula/(2^(j-1)))*2, -Salto_Tren_Caudal];
        TP_Caudal(i+(23*(i-1))+(6*(j-1))+5,:)= [T_ini+x+(Te_Valvula/(2^(j-1)))*2, 0];

        x=TP_Caudal(i+(23*(i-1))+(6*(j-1))+5,1);
    end
    x=x+Te_Valvula*2;
end
```

Ilustración 45. Datos de Entrenamiento, Tren de Pulsos Válvula.

```
tt=Te_Valvula*2;
SP_Caudal=[0,SPCaudal;TP_Caudal(96,1)+tt,SPCaudal;TP_Caudal(96,1)+tt,SPCaudal+Por_Su_Ba;...
    TP_Caudal(192,1)+tt,SPCaudal+Por_Su_Ba;TP_Caudal(192,1)+tt,SPCaudal-Por_Su_Ba;...
    tend,SPCaudal-Por_Su_Ba];
if x>tend %Si el tiempo de simulación del sp_caudal es más grande que el de nivel, lo ponemos como final de sim
    tend=x;
end
```

Ilustración 46. Datos de Entrenamiento, Set Point Caudal.

A continuación, se expondrá cuáles son los resultados del control mediante PID para el sistema descrito para en puntos posteriores poder compararlo con el generado por la red neuronal. Primero partimos de la gráfica de la referencia respecto al caudal.

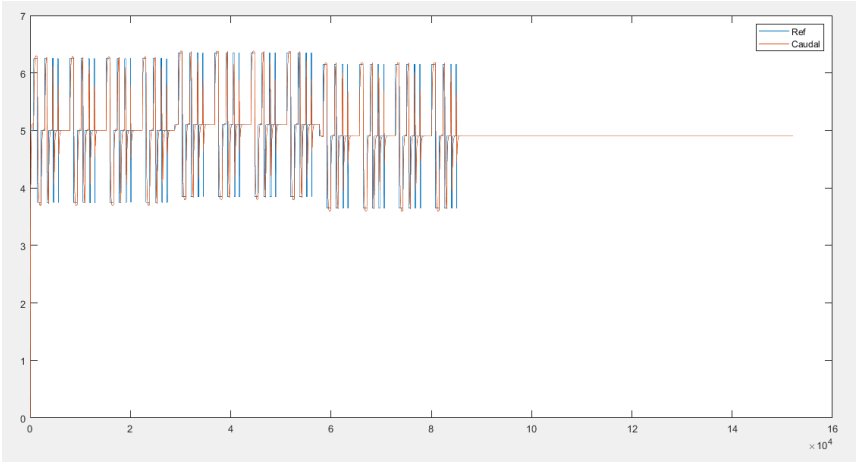


Ilustración 47. Referencia Frente al Caudal con Control PID.

Como se tienen demasiados datos y puesto que hay una gran parte de la simulación que no se utiliza pues está esperando a la variable Nivel a acabar, se van a seleccionar los 20000 primeros datos para poder apreciar mejor cuál es el comportamiento.

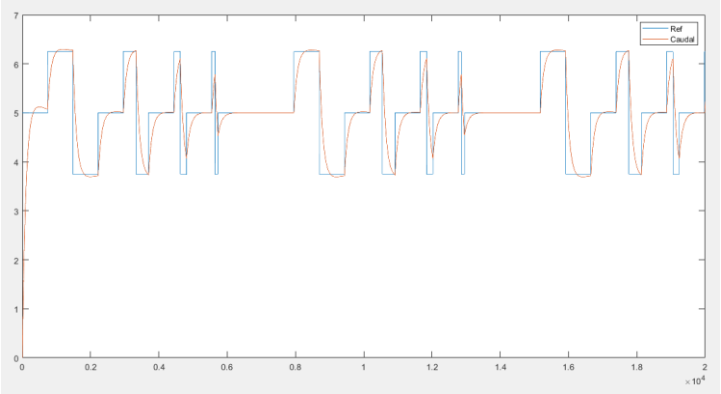


Ilustración 48. Referencia Frente al Caudal con Control PID Zoom.

En la anterior ilustración se puede apreciar en azul la referencia del tren de pulsos y en rojo como el PID es capaz de llevar el sistema y en este caso el caudal a alcanzar la referencia en pocos instantes realizando así un control bastante robusto.

Por otra parte, el control y seguimiento de la referencia en la variable nivel es el siguiente:

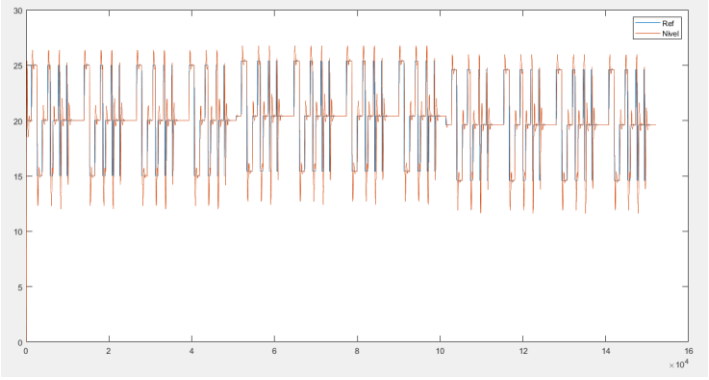


Ilustración 49. Referencia Frente al Nivel con Control PID.

Aunque en este caso se utilice toda la simulación para arrojar datos, hay tantos datos que no se puede apreciar correctamente cuál es el comportamiento del sistema. Si como en el anterior caso, graficamos únicamente los primeros 20000 datos, obtenemos lo siguiente:

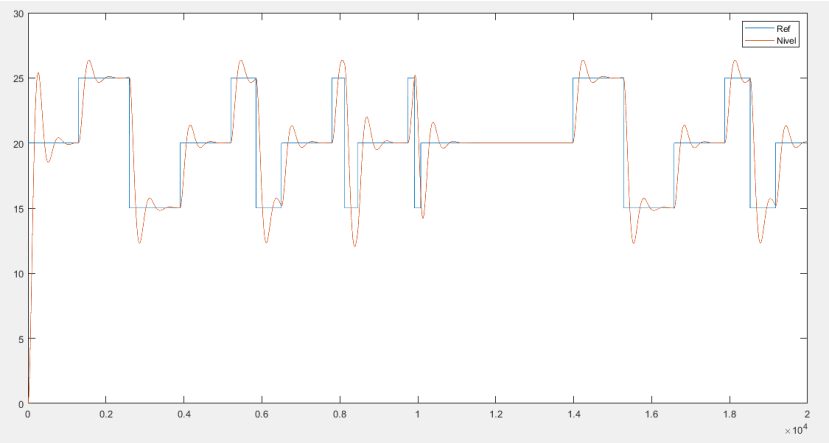


Ilustración 50. Referencia Frente al Nivel con Control PID Zoom.

En la anterior ilustración se puede observar de manera adecuada como el controlador PID es capaz de llevar la variable Nivel del sistema (Rojo) a alcanzar la referencia (Azul) aunque con una ligera sobreoscilación producida por el tipo de sistema que es y el controlador PID que lo domina.

Con estos datos, sabiendo como se debe comportar el PID, estamos en posición de crear la red neuronal y entrenarla para que intente imitar y si se pudiese mejorar el control realizado por el PID.

8 RED NEURONAL OCTAVE:

8.1 PREPARACIÓN DE LOS DATOS:

Con los datos obtenidos en la herramienta Matlab de cuál es el comportamiento que la red neuronal tiene que imitar se traslada a Octave para continuar con la investigación.

A continuación, se mostrará el código general que se ha implementado en para la creación y el entrenamiento de las redes neuronales. Se mostrará el código y paso a paso se explicará cada una de las partes que lo componen. Se ha intentado en todo momento que el código sirva para cualquier caso de red neuronal y por lo tanto sea lo más automatizado posible, por lo que habrá muchas variables que sean leídas desde dados de dentro del código.

```
% Neural network %
%%Cargamos paquetes
pkg load nnet
pkg load io

%% Definición de los parámetros de la red neuronal
%Estructura:
L=3;%Layers
TF_1= "tansig";
TF_2= "tansig";
TF_3= "tansig";
BTF = "trainlm" %Backpropagation network trainig
BLF = "not Used" %Only for matlab compatibility
PF = "mse" %Peformance function
```

Ilustración 51. Código General Octave 1.

Primero cargamos los paquetes necesarios para el correcto funcionamiento del programa. El paquete de las redes neuronales y el de interferencia de entradas y salidas. Después se establecen los parámetros generales que se van a utilizar y que no dependen de casos específicos ni de los tipos de redes neuronales que se van a emplear.

Se establece el número de capas (L=3), la función de transferencia para cada capa que en nuestro caso será tansig.

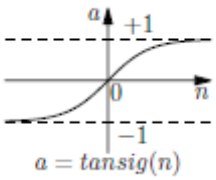


Ilustración 52. Función de transferencia tansig.

La función de entrenamiento de propagación hacia atrás (BTF=trainlm), y la función de actuación (PF=mse) ya que la variable BLF se utiliza solo para ser compatible con matlab.

```
%%Cargar los datos de entrada
In_D_NN_Col_M=xlsread('Train_Data.xlsx');
Red=4;
%1 Red neuronal para el control de nivel con 1 entrada
%2 Red neuronal para el control de nivel con 2 entradas
%3 Red neuronal para el control de caudal con 1 entrada
%4 Red neuronal para el control de caudal con 2 entradas
```

Ilustración 53. Código General Octave 2.

Se carga los datos que se utilizarán para entrenar la red y se almacenan en la variable "IN_D_NN_Col_M" mediante el comando "xlsread" que se utiliza para leer desde un archivo de Excel. Octave leerá todas las celdas hasta la que esté en la última fila y columna y lo cargará en forma de matriz en la variable indicada. Si existen blancos se cargarán como NaN.

Posteriormente se define la variable más importante de todo el código y de la que dependerán muchas ejecuciones de líneas. Como se explica ahí la variable "Red" indica de que red estamos hablando. Como se ha explicado en puntos anteriores, el sistema que se quiere controlar se ha definido como dos sistemas individuales SISO, con una entrada y una salida, pero para poder

hacer un estudio más complejo y exacto, se va a probar diferentes redes para cada lazo, para poder comprobar cuál es la más adecuada y en cuál el desempeño de la red neuronal es mejor respecto al PID calculado. Más concretamente se supondrán dos supuestos para cada lazo, teniendo su entrada como única entrada o en cambio, las dos entradas que representan el sistema.

- Red=1. Red neuronal para el control de nivel con 1 entrada.
- Red=2. Red neuronal para el control de nivel con 2 entradas.
- Red=3. Red neuronal para el control de caudal con 1 entrada.
- Red=4. Red neuronal para el control de caudal con 2 entradas.

Posteriormente se hace la separación de los datos cargados:

```
23  [m,n]=size(In_D_NN_Col_M);
24  switch Red
25      case 1
26      In_D_NN_Col=zeros(m,2);
27      In_D_NN_Col(:,1)=In_D_NN_Col_M(:,1);
28      In_D_NN_Col(:,2)=In_D_NN_Col_M(:,2);
29      N_L1=1;N_L2=13;N_L3=1;
30      case 2
31      In_D_NN_Col=zeros(m,2);
32      In_D_NN_Col(:,1)=In_D_NN_Col_M(:,3);
33      In_D_NN_Col(:,2)=In_D_NN_Col_M(:,4);
34      N_L1=1;N_L2=13;N_L3=1;
35      case 3
36      In_D_NN_Col=zeros(m,3);
37      In_D_NN_Col(:,1)=In_D_NN_Col_M(:,1);
38      In_D_NN_Col(:,2)=In_D_NN_Col_M(:,3);
39      In_D_NN_Col(:,3)=In_D_NN_Col_M(:,2);
40      N_L1=2;N_L2=13;N_L3=1;
41      case 4
42      In_D_NN_Col=zeros(m,3);
43      In_D_NN_Col(:,1)=In_D_NN_Col_M(:,3);
44      In_D_NN_Col(:,2)=In_D_NN_Col_M(:,1);
45      In_D_NN_Col(:,3)=In_D_NN_Col_M(:,4);
46      N_L1=2;N_L2=13;N_L3=1;
47  endswitch
```

Ilustración 54. Código General Octave 3.

Primero se calcula cuál es el número de filas y columnas de los datos cargados, después, dependiendo del valor de la variable “Red” se armará los datos de una forma u otra. Se coloca los datos de entrada en las primeras columnas y los de salida en las últimas dependiendo de si hay una entrada o más. Se inicializan los vectores para que nunca retengan valores de anteriores simulaciones, se colocan los datos y se introduce el número de neuronas de cada capa. Es necesario introducirlo aquí ya que, aunque la segunda y tercera capa son generales para los dos casos y no presentan ningún cambio, la capa inicial si cambia si se utiliza una entrada o dos.

Cuando ya tenemos los datos colocados como toca, necesitamos estandarizar los datos para que la red consiga converger mucho más rápido y de manera más eficiente, se necesita que los datos tengan media 0 y desviación estándar 1. Para ello el paquete de redes neuronales presenta 3 funciones:

- Prestd

$$[\mathrm{PN}, \mathrm{MEANP}, \mathrm{STDP}, \mathrm{TN}, \mathrm{MEANT}, \mathrm{STDT}] = \mathrm{prestd}(\mathrm{P}, \mathrm{T})$$

Ecuación 3. Prestd.

Donde:

- P: Valores a normalizar P.
- PN: Valores normalizados de P.
- MEANP: Media de los datos de P.
- STDP: Desviación estandar de los datos P.
- T: Valores a normalizar T.
- TN: Values normalized of T
- MEANT: Mean of the data T
- STDT: Standard deviation of the data T

- Poststd

$$[\mathrm{PP}, \mathrm{TT}] = \mathrm{poststd}(\mathrm{PN}, \mathrm{MEANP}, \mathrm{STDP}, \mathrm{TN}, \mathrm{MEANT}, \mathrm{STDT})$$

Ecuación 4. Poststd.

Esta ecuación se utiliza para con los datos ya utilizados que han sido procesados desprocesarlos para poder ser evaluados correctamente.

Donde:

- PP: Valores no normalizados de P.
- PN: Valores normalizados de P.
- MEANP: Media de los datos de P.
- STDP: Desviación estándar de P.
- TT: Valores no normalizados de T.
- TN: Valores normalizados de T.
- MEANT: Media de los datos de T.
- STDT: Desviación estándar de T.

- Trastd

Se utiliza para obtener los datos normalizados de P, con los datos no normalizados e imponiendo la media y la desviación estándar.

$$[\mathrm{PN}] = \mathrm{trastd}(\mathrm{P}, \mathrm{MEANP}, \mathrm{STDP})$$

Ecuación 5. Trastd.

Donde:

- PN: Datos normalizados de P.
- P: Datos a normalizar P.
- MEANP: Media de los datos normalizados.
- STDP: Desviación estándar de los datos normalizados.

Es importante saber que estas funciones trabajan por filas, por lo que todos los valores que se quieren normalizar tienen que estar en una misma fila, como los datos que hemos obtenido para entrenar la red vienen en columnas hay que invertidos. Después de invertirlos, ya podemos normalizar los datos.

```
%Reordenamos las columnas en filas
[m,n]=size(In_D_NN_Col);
In_D_NN=zeros(n,m);
for i=1:n
  In_D_NN(i,:)=In_D_NN_Col(:,i);
end
%%Normalizamos los datos de entrada
[In_D,Mean_In_D, STD_In_D]=prestd(In_D_NN);
```

Ilustración 55. Código General Octave 4.

Para invertir, simplemente hemos utilizado un bucle "for" que coge todos los valores de una columna y los coloca en una fila, podríamos haber utilizado el comando invertir (') de Octave pero se ha preferido realizar el bucle para que sea más visible lo que se está realizado. Normalizamos los datos y los guardamos en las variables In_D (Datos Normalizados), Mean_In_D (Media de los datos no normalizados), STD_IN_D (Desviación estándar de los datos no normalizados).

Según se puede ver en (Schmid, 2009), los datos obtenidos se tienen que dividir en tres partes, datos de entrenamiento, datos de prueba y datos de validación, con una proporción de 1/2, 1/3, 1/6 respectivamente. En nuestro caso solo lo dividiremos en datos de entrenamiento y datos de validación pues los casos de simulación, que aquí son llamados de test, los obtendremos desde otro patrón. Por lo que dividimos la cantidad total de datos en 6 partes y 5 serán de entrenamiento y 1 de validación.

Como se ha explicado en puntos anteriores se va a suponer 4 casos de redes, 2 por lazo. Para que los comandos comunes sean lo más parecidos posibles se utilizan las mismas variables pero que dependiendo del valor de Red, se entra en bucles "switch" que colocan los datos de la manera necesaria para que no haya errores.

```
%Divimos y asignamos los datos a las variables:
Fila_train=round(m/6);
%1/2 for train, 1/3 for test and 1/6 for validation data
```

Ilustración 56. Código General Octave 5.

Primero dividimos la cantidad de datos que tenemos (m) entre 6 partes, redondeamos con el comando "round" para tener valores enteros y ya podemos dividirlo aproximadamente. El caso 1 y 2 tienen una estructura muy parecida, pues los dos son redes con la misma estructura de entrada por lo que la división de datos es muy parecida. Ya que en las anteriores partes del código se ha distribuido los datos, aquí se llama a la misma fila de la misma variable, pero los datos en cada caso son diferentes.

Primero asignamos la entrada de datos de entrenamiento y los correspondientes objetivos de las primeras 5 partes de los datos que tenemos. Los datos restantes se asignan a la variable que datos de validación que como se explica muy bien en (Schmid, 2009), los datos tienen que asignarse en la variable VV, en la parte VV. P si son entradas o VV. T si son objetivos.

Para poder crear la red necesitamos el valor máximo y mínimo de los datos, en que rango se mueven los datos estandarizados. Para poder utilizar la función "min_max" que el paquete de redes neuronales propone necesitamos dos entradas mínimo por lo que para el caso de una entrada se ha creado esa parte del código. Simplemente asignamos a la variable min un número muy alto, y se evalúa cada valor de manera que, si encuentra un valor más pequeño, lo sustituye

por ese, y así sucesivamente. De la misma manera para el valor máximo, que en este caso empieza con un valor muy pequeño para que siempre encuentre uno mayor.

```
switch Red
   case 1
      In_D_Train_N=In_D(1,1:Fila_train*5);%Input data for training
      Tar_D_Train_N=In_D(2,1:Fila_train*5);%Target data for training
      VV.P=In_D(1,Fila_train*5+1:m);%Validation data for training
      VV.T=In_D(2,Fila_train*5+1:m);%Validation data for training
      %Estandarice the data
      %Val_Min_Max_DNN=min_max(In_D(1,1:Fila_train*5));
      min=1000000;max=-1000000;
      for i=1:Fila_train*5
         if In_D(i)>max
            max=In_D(i);
         end
         if In_D(i)<min
            min=In_D(i);
         end
      end
      Val_Min_Max_DNN=[min,max];
```

Ilustración 57. Distribución de Datos Case 1.

```
   case 2
      In_D_Train_N=In_D(1,1:Fila_train*5);%Input data for training
      Tar_D_Train_N=In_D(2,1:Fila_train*5);%Target data for training
      VV.P=In_D(1,Fila_train*5+1:m);%Validation data for training
      VV.T=In_D(2,Fila_train*5+1:m);%Validation data for training
      %Estandarice the data
      %Val_Min_Max_DNN=min_max(In_D(1,1:Fila_train*5));
      min=1000000;max=-1000000;
      for i=1:Fila_train*5
         if In_D(i)>max
            max=In_D(i);
         end
         if In_D(i)<min
            min=In_D(i);
         end
      end
      Val_Min_Max_DNN=[min,max];
```

Ilustración 58. Distribución de Datos Case 2.

Para el caso 3 y 4, la estructura es la misma, aunque en este caso, se asignan dos filas en las entradas, que son diferentes en cada caso. De la misma manera para los datos de validación. Finalmente, para calcular el valor máximo y mínimo, el rango en el que se mueven se va a utilizar la función min_max que proporciona el paquete.

$$[mMinMaxElements] = \min\ \_\max\ (RxN)$$

Ecuación 6. Min_max.

Donde:

- [mMinMaxElements]: Matriz de R filas y 2 columnas con los valores máximos y mínimos de cada fila de la matriz (RxN).
- R: Elementos de entrada.
- N: Columnas.

A continuación, podemos ver un ejemplo de cómo funciona esta función:

$$\begin{bmatrix} 1 & 11 \\ 0 & 21 \end{bmatrix} = min\_max \begin{bmatrix} 3 & 1 & 3 & 5 & 11 \\ 12 & 0 & 21 & 8 & 6 \end{bmatrix}$$

Ilustración 59. Ejemplo Función min_max.

```
94      case 3
95          In_D_Train_N=In_D(1:2,1:Fila_train*5);%Input data for training
96          Tar_D_Train_N=In_D(3,1:Fila_train*5);%Target data for training
97          VV.P=In_D(1:2,Fila_train*5+1:m);%Validation data for training
98          VV.T=In_D(2,Fila_train*5+1:m);%Validation data for training
99          %Estandarice the data
100         Val_Min_Max_DNN=min_max(In_D(1:2,1:Fila_train*5));
```

Ilustración 60. Distribución de Datos Case 3.

```
101     case 4
102         In_D_Train_N=In_D(1:2,1:Fila_train*5);%Input data for training
103         Tar_D_Train_N=In_D(3,1:Fila_train*5);%Target data for training
104         VV.P=In_D(1:2,Fila_train*5+1:m);%Validation data for training
105         VV.T=In_D(2,Fila_train*5+1:m);%Validation data for training
106         %Estandarice the data
107         Val_Min_Max_DNN=min_max(In_D(1:2,1:Fila_train*5));
108     endswitch
```

Ilustración 61. Distribución de Datos Case 4.

8.2 CREACIÓN DE LA RED:

Cuando ya tenemos todos los datos que necesitamos para crear y entrenar la red continuamos de forma común ya que el comando para crear es independiente en cada caso y así se genera un código mucho más optimizado.

```
109 %%RED NEURONAL
110 %Creación de la red neuronal
111 net_nt=newff(Val_Min_Max_DNN,[N_L1,N_L2,N_L3],{TF_1,TF_2,TF_3},BTF,BLF,PF);
```

Ilustración 62. Código General Octave 6.

Aquí utilizamos la función del paquete “newff” que a partir de los datos de entrada genera la estructura de una red neuronal.

$$net = newff\ (Rx2, [S1\ S2 \dots SN], \{TF1\ TF2 \dots TFN\}, BTF, BLF, PF)$$

Ecuación 7. Newff.

Donde:

- net: Nombre de la variable donde se quiere guardar la red neuronal.
- Rx2: Matriz de R entradas y 2 columnas con los valores máximos y mínimos de cada entrada.
- S1, S2.. SN: Tamaño de neuronas de cada capa desde la primera (S1) hasta la última (SN).
- TF1, TF2.. TFN: Función de transferencia de cada capa, desde la primera (TF1) hasta la última (TFN). Por defecto se tiene “tansig”.
- BTF: Función de entrenamiento de la red de backpropagation, por defecto se tiene “trainlm”.
- BLF: No utilizado, solo para poder compatibilizar con Matlab.
- PF: Función de actuación, por defecto se tiene “mse”.

Menos el valor de máximos y mínimos que se ha calculado recientemente, los demás parámetros se establecieron al principio del código ya que eran comunes para todos, menos el tamaño de cada capa que se define cuando se organizan los datos. Con la red ya creada, podemos pasar a entrenarla.

8.3 ENTRENAMIENTO DE LA RED:

Para ello utilizamos la función de proporciona el paquete "train":

$$net = train\ (MLPnet, P, T, [], [], VV)$$

Ecuación 8. Train.

Donde:

- net: Red neuronal entrenada.
- MLPnet: Red neuronal sin entrenar.
- P: Datos de entrada para entrenamiento.
- T: Datos objetivo para el entrenamiento.
- [], []: Solo para compatibilidad con Matlab.
- VV: Datos de validación.

Se vuelven a abrir los 4 casos para antes de entrenar la red cambiar los parámetros de entrenamiento, que se encuentran creados cuando se crea la red. Para modificarlos se tiene que acceder mediante la parte "net_nt.trainParam."junto con el nombre del parámetro que se desea modificar. Junto con el comando de entrenamiento de la red, se tiene en comentarios los mejores resultados que se ha conseguido con cada tipo de red.

Donde 'epoch' significa el número de iteraciones que ha necesitado para alcanzar el resultado, 'gradient' el salto que ha necesitado y por último 'MSE' es el mínimo error que ha conseguido. Cuando ya tenemos la red entrenada tenemos que guardarla para poder cargarla en futuros momentos, cuando queramos simularla. En el paquete de redes neuronales se propone el comando para guardar de "saveMLPStruct", aunque este comando solo te guarda la estructura de la red, y no sus valores internos, lo cual no es suficiente.

$$saveMLPStruct\ (net, "initNetwork.txt")$$

Ecuación 9. SaveMLPStruct.

Donde:

- Net: Red neuronal que se quiere guardar.
- InitNetwork.txt: Nombre del archivo de texto donde se quiere guardar la red neuronal.

Para guardar la red simplemente se debe utilizar el comando "save" de Octave, después para cargarla de la misma manera con el comando "load" se realiza de manera sencilla.

$$save\ Net\_x$$

Ecuación 10. Save.

A continuación, se muestran los 4 casos.

```
switch Red
  case 1
      net_nt.trainParam.epochs = 200;
      net_nt.trainParam.goal = 0.3;
      net_nt.trainParam.max_fail = 5;
      net_nt.trainParam.mem_reduc = 1;
      net_nt.trainParam.min_grad = 1.0000e-5;
      net_nt.trainParam.mu = 0.0010;
      net_nt.trainParam.mu_dec = 0.1;
      net_nt.trainParam.mu_inc = 10;
      net_nt.trainParam.mu_max = 1.0000e+010;
      net_nt.trainParam.show = 50;
      net_nt.trainParam.time = Inf;
      Net_1_1=train(net_nt,In_D_Train_N,Tar_D_Train_N,[],[],VV);
      %Save the neural network
      %TRAINLM, 1_3 Epoch 200/200, MSE 0.576451/0.4, Gradient 618.343/0.0001
      save Net_1_1;
```

Ilustración 63. Entrenamiento de la Red Caso 1.

```
  case 2
      net_nt.trainParam.epochs = 100;
      net_nt.trainParam.goal = 0;
      net_nt.trainParam.max_fail = 100;
      net_nt.trainParam.mem_reduc = 1;
      net_nt.trainParam.min_grad = 1.0000e-4;
      net_nt.trainParam.mu = 0.0010;
      net_nt.trainParam.mu_dec = 0.1;
      net_nt.trainParam.mu_inc = 10;
      net_nt.trainParam.mu_max = 1.0000e+010;
      net_nt.trainParam.show = 50;
      net_nt.trainParam.time = Inf;
      Net_2_3=train(net_nt,In_D_Train_N,Tar_D_Train_N,[],[],VV);
      %Save the neural network
      %TRAINLM, Epoch 2/100, MSE 0.605243/0.65, Gradient 13760.7/1e-10
      save Net_2_3;
```

Ilustración 64. Entrenamiento de la Red Caso 2.

```
case 3
      net_nt.trainParam.epochs = 100;
      net_nt.trainParam.goal = 0;
      net_nt.trainParam.max_fail = 5;
      net_nt.trainParam.mem_reduc = 1;
      net_nt.trainParam.min_grad = 1.0000e-4;
      net_nt.trainParam.mu = 0.0010;
      net_nt.trainParam.mu_dec = 0.1;
      net_nt.trainParam.mu_inc = 10;
      net_nt.trainParam.mu_max = 1.0000e+010;
      net_nt.trainParam.show = 50;
      net_nt.trainParam.time = Inf;
      Net_3_1=train(net_nt,In_D_Train_N,Tar_D_Train_N,[],[],VV);
      %Save the neural network
      %TRAINLM, Epoch 6/100, MSE 0.616448/0.65, Gradient 9983.82/1e-10
      save Net_3_1;
      %Net_3_1 0.61
```

Ilustración 65. Entrenamiento de la Red Caso 3.

```
case 4
      net_nt.trainParam.epochs = 100;
      net_nt.trainParam.goal = 0;
      net_nt.trainParam.max_fail = 100;
      net_nt.trainParam.mem_reduc = 1;
      net_nt.trainParam.min_grad = 1.0000e-4;
      net_nt.trainParam.mu = 0.0010;
      net_nt.trainParam.mu_dec = 0.1;
      net_nt.trainParam.mu_inc = 10;
      net_nt.trainParam.mu_max = 1.0000e+010;
      net_nt.trainParam.show = 50;
      net_nt.trainParam.time = Inf;
      Net_4_2=train(net_nt,In_D_Train_N,Tar_D_Train_N,[],[],VV);
      %Save the neural network
      %TRAINLM, Epoch 2/100, MSE 0.629569/0.65, Gradient 10419/1e-10
      %Net_4_2 0.596
      save Net_4_2;
endswitch
```

Ilustración 66. Entrenamiento de la Red Caso 4.

8.4 VALIDACIÓN:

Ahora que ya tenemos las redes creadas y entradas, es momento de comparar su ejecución respecto al funcionamiento del PID, para ello, se ha creado un script diferente que se encargará

de cargar la red que se desee, cargar los datos de validación que previamente se han creado en Matlab, ordenar esos datos y calcular cómo reacciona la red con esas entradas y la comparación respecto a cómo lo hace el PID.

8.4.1 INTRODUCCIÓN DE LOS DATOS:

```
%Este script cargará los datos de validación, obtendrá
%la salida de las redes neuronales previamente creadas
%y comparará los datos.
%%Cargamos paquetes
pkg load nnet
pkg load io
%%Cargamos las redes neuronales y se lo asignamos a los
%nombres que hemos utilizado
%%Cargamos los datos de validación
clear;
Val_Data_Col_MR=xlsread('Validate_Data');
%Le quitamos los primeros 5000 valores para no tener errores
%iniciales
[m,n]=size(Val_Data_Col_MR);
Val_quitados=10000;
Val_Data_Col_M=zeros(m-Val_quitados,n);
Val_Data_Col_M=Val_Data_Col_MR(Val_quitados:m,:);
  Red=2;
```

Ilustración 67. Código General Validación 1. Introducción de Datos.

Primero cargamos los paquetes necesarios que son los mismos que en los demás scripts. Es importante limpiar la memoria de variables de octave para que las redes se carguen correctamente ya que, si no, al asignar los nombres de las redes puede haber errores. Posteriormente cargamos los datos de validación y lo asignamos en la variable "Val_Data_Col_MR", que se corresponde a los datos de validación en columnas con mala cantidad de datos. Se ha comprobado que el desempeño de las redes es mejor si se retira los primeros datos, pues en estos, hay saltos muy grandes hasta que se alcanza el punto de funcionamiento lo que provoca una mala adaptación de la red por lo que en esta parte de código se quitan la cantidad de valores que haya en la variable "Val_quitados". En este caso, con quitar los primeros 10000 datos, la red es capaz de funcionar correctamente. Para acabar con la introducción de datos, se selecciona que tipo de red se va a simular, como se ha realizado en todos los scripts anteriores.

8.4.2 PREPARACIÓN DE LAS REDES:

Ahora que tenemos todos los datos, preparamos las matrices según que red se haya seleccionado. Se elige en que figura se va a graficar, se limpia esa gráfica para no tener datos remanentes. Dentro de cada tipo de red, se carga la red con su nombre.

```
  switch Red
    case 1
      figure(1)
      clf(1);
      load Net_1_3;
      net_1=Net_1_3;
      [m,n]=size(Val_Data_Col_M);
      In_D_NN_Col=zeros(m,2);
      In_D_NN_Col(:,1)=Val_Data_Col_M(:,1);
      In_D_NN_Col(:,2)=Val_Data_Col_M(:,2);
```

Ilustración 68. Código General Validación 2. Preparación de Red 1.

```
    case 2
    figure(2)
    clf(2);
    load Net_2_2;
    net_2=Net_2_2;
    [m,n]=size(Val_Data_Col_M);
    In_D_NN_Col=zeros(m,2);
    In_D_NN_Col(:,1)=Val_Data_Col_M(:,3);
    In_D_NN_Col(:,2)=Val_Data_Col_M(:,4);
```

Ilustración 69. Código General Validación 3. Preparación de Red 2.

```
    case 3
    figure(3)
    clf(3);
    net_3=Net_3_1;
    %net_3=load("Net_3_1");
    [m,n]=size(Val_Data_Col_M);
    In_D_NN_Col=zeros(m,3);
    In_D_NN_Col(:,1)=Val_Data_Col_M(:,1);
    In_D_NN_Col(:,2)=Val_Data_Col_M(:,3);
    In_D_NN_Col(:,3)=Val_Data_Col_M(:,2);
```

Ilustración 70. Código General Validación 4. Preparación de Red 3.

```
    case 4
    figure(4)
    clf(4)
    net_4=Net_4_2;
    %net_4=load("Net_4_1");
    [m,n]=size(Val_Data_Col_M);
    In_D_NN_Col=zeros(m,3);
    In_D_NN_Col(:,1)=Val_Data_Col_M(:,3);
    In_D_NN_Col(:,2)=Val_Data_Col_M(:,1);
    In_D_NN_Col(:,3)=Val_Data_Col_M(:,4);
endswitch
```

Ilustración 71. Código General Validación 5. Preparación de Red 4.

8.4.3 NORMALIZACIÓN DE LOS DATOS:

Cuando ya se tienen los datos preparados dependiendo de la red que se desee validar, de manera general se normalizan los datos. Manteniendo las partes generales en las mismas líneas de código hace que se sucedan la menor cantidad de errores posible.

```
%%Normalizamos los datos de entrada
  [In_D_Sim_N,Mean_In_D, STD_In_D]=prestd(In_D_NN);
%%Sacamos la salida de cada entrada y las
%%guardamos en variables diferentes para poder
%%compararlas
```

Ilustración 72. Código General Validación 6. Normalización de los Datos.

8.4.4 SIMULACIÓN:

Cuando ya se tienen todos los datos preparados se puede proceder a la simulación de la red con los datos de entrada y a la posterior evaluación de estos. A continuación, se expone el código de los cuatro casos posibles ya explicados anteriormente.

Primero se simula la red “net_1” que es diferente en cada caso con los datos de entrada “In_D_Sim_N (1,:)”, como los dato se habían prenormalizado se deben post normalizar para

obtener una escala adecuada de los mismos. Para poder comparar los datos se postnormalizan también los datos del PID.

```
switch Red
    case 1
        Out_D_N_SP=sim(net_1,In_D_Sim_N(1,:));
        Out_D_N_1=poststd(Out_D_N_SP,Mean_In_D(2),STD_In_D(2));
        In_D_Sim_N_2=poststd(In_D_Sim_N,Mean_In_D,STD_In_D);
        Val_N_1=Check_NN(In_D_Sim_N_2(2,:),Out_D_N_1)
        [m,n]=size(In_D_Sim_N_2(2,:));
        Data=zeros(2,n);
        Data(1,:)=In_D_Sim_N_2(2,:);
        Data(2,:)=Out_D_N_1;
        Ejex=[1:n];
        figure(1)
        plot(Ejex,Data);
        legend("PID","NN");
```

Ilustración 73. Código General Validación 7. Simulación Case 1.

A continuación, se comparan los datos evaluándose mediante la función creada "Check_NN" que se expone en la siguiente ilustración:

```
function Status=Check_NN(Out_PID,Out_NN)
    [m,n]=size(Out_PID);
    [o,p]=size(Out_NN);
    Status=0;
    x=(abs((Out_PID-Out_NN)./Out_PID))*100;
    Status=sum(x)/(m*n);
endfunction
```

Ilustración 74. Función Check_NN.

Esta función compara las dos variables de entrada que recibe, calcula el tamaño de ese vector, que deben de ser los mismos. Se inicializa la variable "Status" a 0 y posteriormente se evalúa la diferencia entre un valor del primer vector y su homónimo en el segundo. En lugar de hacerlo uno a uno se realiza con todos los valores del vector a la vez para hacerlo en un solo cálculo. Se calcula la diferencia y se divide por el valor para hacerlo en forma de porcentaje, calculando su valor absoluto y multiplicándolo por 100. Por último, se calcula la media de todos los valores sumándolos y dividiéndolos por la cantidad de valores que hay. Ese es el valor que te devuelve la función, por lo que puede ser empleada en los 4 cases.

Cuando ya se ha evaluado la red neuronal frente al PID se prepara los datos para ser graficados y obtener una visualización de cómo es uno frente al otro. Primero se calcula el tamaño de los vectores, en la matriz data se coloca en una columna los valores del PID y en la otra los de la red neuronal. Se gráfica y se visualiza. A continuación, se expone los casos que faltan, que como en casos anteriores, son parecidos, pero se ha creído conveniente exponerlos todos para que se entienda completamente cuáles son sus variaciones.

```
84  ···case·2
85  ·····Out_D_N_SP=sim(net_2,In_D_Sim_N(1,:));
86  ·····Out_D_N_1=poststd(Out_D_N_SP,Mean_In_D(2),·STD_In_D(2));
87  ·····In_D_Sim_N_2=poststd(In_D_Sim_N,Mean_In_D,·STD_In_D);
88  ·····Out_D_N_1=Out_D_N_1-0.05;
89  ·····Val_N_2=Check_NN(In_D_NN(2,:),Out_D_N_1)
90  ·····[m,n]=size(In_D_NN(2,:));
91  ·····Data=zeros(2,n);
92  ·····Data(1,:)=In_D_NN(2,:);
93  ·····Data(2,:)=Out_D_N_1;
94  ·····Ejex=1:n;
95  ·····figure(2);
96  ·····plot(Ejex,Data);
97  ·····legend("PID","NN");
```

Ilustración 75. Código General Validación 8. Simulación Case 2.

```
113 ···case·3
114 ·····[m,n]=size(In_D_NN_Col);
115 ·····In_D2_Sim_N=zeros(2,m);
116 ·····In_D2_Sim_N(1,:)=In_D_Sim_N(1,:);
117 ·····In_D2_Sim_N(2,:)=In_D_Sim_N(3,:);
118 ·····Out_D_N_3=sim(net_3,In_D2_Sim_N);
119 ·····Out_D_N_1=poststd(Out_D_N_3,Mean_In_D(2),·STD_In_D(2));
120 ·····In_D_Sim_N_2=poststd(In_D_Sim_N,Mean_In_D,·STD_In_D);
121 ·····Val_N_3=Check_NN(In_D_Sim_N_2(3,:),Out_D_N_1)
122 ·····[m,n]=size(In_D_Sim_N(3,:));
123 ·····Data=zeros(2,n);
124 ·····Data(1,:)=In_D_Sim_N(3,:);
125 ·····Data(2,:)=Out_D_N_1;
126 ·····Ejex=1:n;
127 ·····figure(3);
128 ·····plot(Ejex,Data);
129 ·····legend("PID","NN");
```

Ilustración 76. Código General Validación 9. Simulación Case 3.

```
131 ···case·4
132 ·····[m,n]=size(In_D_NN_Col);
133 ·····In_D2_Sim_N=zeros(2,m);
134 ·····In_D2_Sim_N(1,:)=In_D_Sim_N(1,:);
135 ·····In_D2_Sim_N(2,:)=In_D_Sim_N(3,:);
136 ·····Out_D_N_4=sim(net_4,In_D2_Sim_N);
137 ·····Out_D_N_1=poststd(Out_D_N_4,Mean_In_D(2),·STD_In_D(2));
138 ·····In_D_Sim_N_2=poststd(In_D_Sim_N,Mean_In_D,·STD_In_D);
139 ·····Val_N_4=Check_NN(In_D_Sim_N_2(3,:),Out_D_N_1)
140 ·····[m,n]=size(In_D_Sim_N(3,:));
141 ·····Data=zeros(2,n);
142 ·····Data(1,:)=In_D_Sim_N(3,:);
143 ·····Data(2,:)=Out_D_N_1;
144 ·····Ejex=1:n;
145 ·····figure(4);
146 ·····plot(Ejex,Data);
147 ·····legend("PID","NN");
```

Ilustración 77. Código General Validación 10. Simulación Case 4.

8.5 SIMULACIÓN EN CONTÍNUO:

Cuando ya tenemos la red evaluada se propone la creación del script que haga posible la simulación en continuo de la red. Como Octave no posee una herramienta de simulación como podría ser Simulink de Matlab, necesitamos que se simule todo mediante código.

El trabajo final es que las entradas a la red provengan de una fuente externa que son los microcontroladores Arduino, pero para empezar se va a realizar la simulación solo con las referencias de cada instante para ver si la red neuronal es capaz de con las referencias ir llevando la variable a su destino, cuando funcione se construirá la simulación recibiendo los datos de manera externa.

8.5.1 SIMULACIÓN EN BUCLE:

Para este caso se han construido dos scripts, uno para simular la variable caudal y otro para simular la variable nivel, aunque se pueden simular los dos a la vez y es como debería ser el

script final, pero para poder estudiar mejor cada caso, en principio se ha creado uno individualmente.

No se ha creído necesario poner el script de nivel y el de caudal pues la única diferencia es la red que estas cargando y la función de transferencia que evalúas en el step, por lo que para no introducir datos redundantes se ha omitido.

Lo primero es cargar los paquetes necesarios para la compilación del código. Los paquetes son los mismos que en anteriores casos, pero se añade el paquete de control, que es necesario cuando se van a utilizar comandos con funciones de transferencia, en este caso se utilizará la función "step" que se explicará a continuación.

8.5.1.1 CARGA DE PAQUETES Y DATOS NECESARIOS:

```
1  %Cargamos los paquetes necesarios
2  pkg load nnet
3  pkg load io
4  pkg load control
5  clear;
```

Ilustración 78. Código General Simulación en Bucle 1. Carga de Paquetes.

También se limpia la libreta de variables para que no haya confusiones y errores con simulaciones anteriores. Después, se carga la red neuronal que se quiere utilizar. Como en casos anteriores, para una simulación más sencilla, en este paso se sustituye la red en la variable Net_1, de manera que solo cambiando la red en este paso se sustituye en todas las partes del código y no hay que ir individualmente cambiándolas.

```
6  %Cargamos que red neuronal queremos simular
7  load Net_2_4;
8  Net_1=Net_2_4;
```

Ilustración 79. Código General Simulación en Bucle 2.

Como la red se ha entrenado con los valores normalizados, en el bucle que vamos a construir necesitamos normalizar los datos de entrada y los de salida. Las funciones que se han utilizado en pasos anteriores del paquete de redes neuronales (prestd,poststd..) necesitan un mínimo de dos datos para normalizarlos, pues los normalizan en función de su media y desviación estándar, como en la simulación que tenemos solo vamos a tener un dato de entrada y de salida, no vamos a poder utilizarlas, por lo que la normalización ha de ser realizada manualmente, para ello se carga la media y la desviación estándar de los valores con los que se entrenó la red. Después se define la variable 's' como variable de función de transferencia y se crea las dos funciones de transferencia que van a representar el funcionamiento de los dos lazos. G11 representará el lazo de nivel y G22 de caudal.

```
9  %Cargamos medias y desviaciones estandar
10 Mean1=0.0053164;Mean2=19.7704273;Mean3=0.0024142;Mean4=7.2803693;
11 STD1=2.08929;STD2=10203.65151;STD3=0.36044;STD4=2.65699;
12 %Definimos dos funciones de transferencia
13 s=tf('s');
14 G11=tf([0.0002823],[1 0.04295 0.0002129],0.001);
15 G22=tf([0.1452],[1 0.2127],0.001);
```

Ilustración 80. Código General Simulación en Bucle 3.

Ahora es el momento de cargar las referencias que se va a ir introduciendo en el sistema en cada momento para que la red neuronal intente seguir. Para realizar pruebas iniciales sin tener que cargar todos los datos en cada iteración se ha creado un vector ascendente y descendente con escalones pequeños que permita formalizar una idea previa de cómo va a trabajar la red.

```
%Cargamos el vector de referencias
Refsize_Data=xlsread('Validate_Data');
t=0;
x=0.5;
yl=0.5;
y2=-1;
A=zeros(350,1);
A(1:50,1)=t;
A(51:100,1)=yl;
A(101:150,1)=yl*2;
A(151:200,1)=yl*5;
A(201:250,1)=-yl*2;
A(251:300,1)=-yl*4;
A(301:350,1)=-yl*2;
%Ref_Data=A;
Ref_Data=Refsize_Data;
```

Ilustración 81. Código General Simulación en Bucle 4.

8.5.1.2 INICIALIZACIÓN DE VARIABLES:

Cuando ya se tiene las referencias que la red ha de seguir, se inicializa todas las variables que van a formar parte de la simulación. Para ello se calcula el tamaño del vector de referencias y se crean las variables en función de este ya que en cada instante de referencias habrá una casilla de cada vector que almacenará un valor. Se inicializa la entrada al sistema "Ent_Sist" como 1 en el primer instante de tiempo que permita que todo empiece a funcionar.

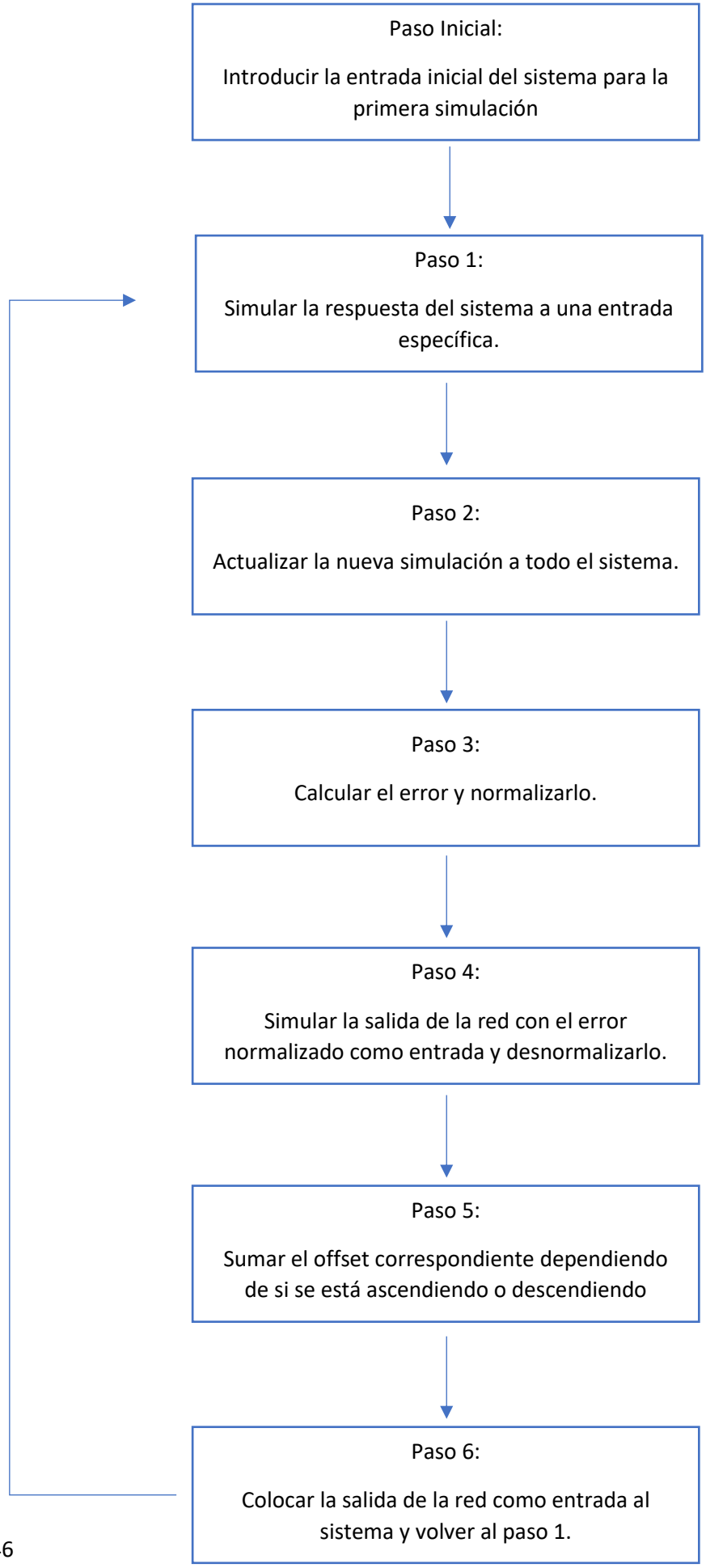
```
[m,n]=size(Ref_Data); %Tiempos de simulación
%Creamos vectores
Ent_Sist=zeros(m,1);
Ent_Sist(1)=0;%Inicializamos el vector de
%entradas ya que en el instante 1 no hay
Sal_Nivel=zeros(m,1);
Sal_Nivel_Act=zeros(m,1);
Error_1=zeros(m,1);
Error_1_NN=zeros(m,1);
Sal_NN_Nivel_NN=zeros(m,1);
Sal_NN_Nivel_NN_1=zeros(m,1);
Sal_NN_Nivel=zeros(m,1);
Data=zeros(m,2);
offset=0;
```

Ilustración 82. Código General Simulación en Bucle 5.

8.5.1.3 BUCLE DE SIMULACIÓN:

Cuando se tienen todos los datos cargados se empieza con el bucle que va a ir simulando el funcionamiento en tiempo real del sistema y como la red es capaz o no de ir llevando la variable a la referencia indicada. Se trata de un bucle "for" que tiene tantos ciclos como instantes de simulación de la referencia haya "m".

Para que se pueda comprender mejor el funcionamiento del código se ha realizado un flujo de actividades que se realizan continuamente en el bucle y en qué orden cada una.


Paso Inicial:
Introducir la entrada inicial del sistema para la primera simulación
Paso 1:
Simular la respuesta del sistema a una entrada específica.
Paso 2:
Actualizar la nueva simulación a todo el sistema.
Paso 3:
Calcular el error y normalizarlo.
Paso 4:
Simular la salida de la red con el error normalizado como entrada y desnormalizarlo.
Paso 5:
Sumar el offset correspondiente dependiendo de si se está ascendiendo o descendiendo
Paso 6:
Colocar la salida de la red como entrada al sistema y volver al paso 1.

Para simular cómo funciona el sistema se va a utilizar el comando “Step”. En Simulink se ha discretizado el funcionamiento continuo con un período de 0.01 segundos, por lo que para hacer la simulación en Octave tenemos que hacerlo de la misma manera. Para ello, consideraremos que cada espacio de 0.01 segundos, se suministrará a la instalación un escalón con la salida de la red neuronal. El comando step hará esta simulación. Se utilizará el último valor del vector que da como salida el comando step como valor final y será el que se utilice para calcular el error con este la entrada a la red. En el primer instante de simulación la entrada al sistema es de 0 para comenzar en algún valor.

```
46 - for i=1:m
47     %Tenemos que distinguir si es el momento 1
48     %de simulación o si es continuo
49     [a,b]=step(G22*Ent_Sist(i),0.01);
50     Sal_Nivel_Act(i)=a(end);
```

Ilustración 83. Código General Simulación en Bucle 6.

Este comando genera dos salidas, el vector de valores de simulación junto con el de tiempos. Como ejemplo, cuando se ejecuta con la función de transferencia G11 para una entrada de 1 presenta la siguiente estructura:

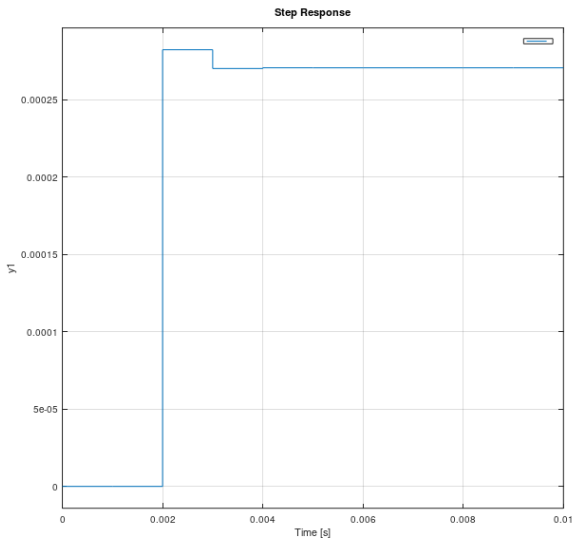


Ilustración 84. Simulación con Comando Step.

En la anterior ilustración se puede ver cómo evoluciona la instalación cuando hay una entrada constante de escalón durante 0.01 segundos. Se puede ver como el sistema se estabiliza antes en un valor determinado después de tener una ligera sobre oscilación.

Cuando ya se tiene la evolución del sistema se realiza un bucle para calcular el offset necesario de la red. Puesto que la red debe seguir la tendencia de la referencia, pero puede tener dificultad con la magnitud, para ello se añade un offset que se sumará o restará dependiendo de si se está ascendiendo o descendiendo, pues el funcionamiento es diferente y por tanto la magnitud también.

Para hacerlo y que el código sepa si se está ascendiendo o descendiendo se calcula el offset en cada iteración, si la referencia en este instante es mayor que la anterior se supone que se está ascendiendo y por tanto se proporciona un offset, si es a la inversa y es menor se proporciona

otro, y si no hay ningún cambio porque es la misma referencia que en el punto anterior se mantiene el mismo offset y no se realiza ningún cambio. Como en el instante inicial no hay referencia anterior se calcula viendo si es la referencia en el primer instante es superior o inferior a 0.

Por último, se suma la salida que ha proporcionado el step junto a la salida anterior, ya que se ha evaluado de un punto hasta otro, pero hay que añadir las condiciones del instante anterior por lo que se juntan. Como en el primer instante no hay estado anterior, y se parte desde 0 esta condición no se reproduce. De manera que así reproduciremos la simulación que se genera en Simulink, analizamos que condiciones tenemos, realizamos el step, y se lo añadimos a esas condiciones iniciales para crear las condiciones iniciales del siguiente punto de simulación.

```
    offset_asc=13.7;
    offset_desc=0;
    if i==1 %Si es el primer instante no
        if Ref_Data(1)>0
            offset=offset_asc;
        elseif Ref_Data(1)<0
            offset_desc=offset_desc;
        endif
    else %Si es otro instante, calculamos si estamos subiendo o bajando
    %para aplicar un offset
        if Ref_Data(i)>Ref_Data(i-1)
            offset=offset_asc;
        elseif Ref_Data(i)<Ref_Data(i-1)
            offset_desc=offset_desc;
        endif
        Sal_Nivel(i)=Sal_Nivel(i-1)+Sal_Nivel_Act(i); %Sumamos cual ha sido la
        %evolución al instante anterior, a como esta la señal en el instante anterior
    endif
```

Ilustración 85. Código General Simulación en Bucle 7.

Cuando ya se tiene el estado del sistema se calcula cuál es el error actual del sistema respecto a la referencia o punto al que se desea llegar. Con una simple resta se conseguirá obtener esa diferencia. Posteriormente se normalizará para que la red pueda conocer el valor en su verdadera magnitud.

Con el error normalizado y siguiendo el diagrama de flujo representado anteriormente se calcula cual es la respuesta de la red neuronal cuando en la entrada se introduce el error normalizado. La salida se normalizará para que se pueda introducir un valor válido a nivel de magnitud en el sistema.

```
    Error_1_NN(i)=Ref_Data(i)-Sal_Nivel(i);
    Error_1(i)=(Error_1_NN(i)-Mean2)/STD2;
    Error_1(i)=Error_1_NN(i);
    Sal_NN_Nivel_NN(i)=sim(Net_1,Error_1(i));
    Sal_NN_Nivel_NN_1(i)=(Sal_NN_Nivel_NN(i)*STD4)+Mean4;
    Sal_NN_Nivel_NN_1(i)=Sal_NN_Nivel_NN(i);
```

Ilustración 86. Código General Simulación en Bucle 8.

Como se ha comentado anteriormente, la red puede no tener un error de offset. Es decir, de nivelación, con lo que es necesario que a la salida de la red se le añada o se le reste un cierto valor que haga que cuadre totalmente con el valor esperado. Por lo que se le añade el offset que en puntos anteriores se haya calculado.

Cuando ya se tiene la salida de la red calculada, se sustituye su valor en el vector de entradas del sistema de la siguiente simulación, para que cuando se ejecute el siguiente ciclo del bucle "for", ya se tenga la entrada del sistema y se pueda ejecutar correctamente el código.

```
66      if Error_1_NN(i)==0
67          Sal_NN_Nivel(i)=0;
68      else
69          Sal_NN_Nivel(i)=(Sal_NN_Nivel_NN_1(i)+offset);
70      endif
71      if i==m
72      else
73          Ent_Sist(i+1)=Sal_NN_Nivel(i);
74      endif
75  endfor
```

Ilustración 87. Código General Simulación en Bucle 9.

Como se ha visto, la simulación se constituye principalmente en ir almacenando los datos en cada bucle en una posición determinada de los vectores, simulando así los cambios temporales de cada ciclo.

8.5.1.4 EVALUACIÓN DE LOS DATOS:

Cuando ya se ha finalizado el bucle, se tiene todos los datos necesarios para la simulación. Para agilizar el trabajo se ha introducido una porción de código que crea una matriz donde cada columna representa uno de los vectores utilizados. Esta matriz, se mostrará en cada simulación del archivo, para que se pueda observar los datos y obtener de manera rápida los fallos, a la vez que se obtiene una idea de magnitud de lo que está sucediendo.

```
76  %Creamos una matriz para comparar los datos
77  M_Com=zeros(m,7);
78  M_Com(:,1)=Sal_Nivel_Act;
79  M_Com(:,2)=Sal_Nivel;
80  M_Com(:,3)=Ref_Data(:,1);
81  M_Com(:,4)=Error_1_NN;
82  M_Com(:,5)=Error_1;
83  M_Com(:,6)=Sal_NN_Nivel_NN;
84  M_Com(:,7)=Sal_NN_Nivel_NN_1;
85  M_Com(:,8)=Sal_NN_Nivel;
86  M_Com(:,9)=Ent_Sist;
87  M_Com
```

Ilustración 88. Código General Simulación en Bucle 10.

Por último, se selecciona el vector de referencias y el vector de salidas del sistema y se grafican. Así, obtenemos un resultado visual de si la red ha sido capaz de llevar el sistema a las referencias.

```
88  Data(:,1)=Ref_Data(:,1);
89  Data(:,2)=(Sal_Nivel);
90  Ejex=1:m;
91  figure(1);
92  clf(1);
93  plot(Ejex,Data);
94  legend("Ref_Data","Sal_Nivel");
```

Ilustración 89. Código General Simulación en Bucle 11.

9 EVALUACIONES:

En este apartado, se va a profundizar en las diferentes variaciones que se pueden realizar sobre la construcción de los datos de entrenamiento de la Red Neuronal y sobre su propia construcción. Se analizará como afectan todas estas variaciones al funcionamiento final de la red, dependiendo de que red de las cuatro a estudiar y su desempeño en el control de las variables. También servirá de utilidad para encontrar cuál de las 2 redes por lazo funciona mejor y por tanto realizar con ella el control del sistema.

9.1 ELIMINANDO VALORES INICIALES:

Cuando se extrae los datos desde un controlador para entrenar la red, se extrae la mayor cantidad de datos posible, aunque hay ciertos intervalos de esos datos que pueden engañar a la red, con el funcionamiento que queremos obtener.

El comienzo de los datos de entrenamiento es relevante para esto, ya que supone el tránsito desde valores nulos hasta puntos de equilibrio, desde los cuales se va a mover buscando la referencia. En este tránsito, los valores suelen ser extremos, con lo que pueden alterar por su magnitud diferente, a todos los demás valores. Por lo que se ha realizado el estudio de cómo afecta al desempeño de la red, cuando se eliminan datos en el entrenamiento de esta. Se estudiará como afecta si se quita el 5%, 10% y 15% primeros datos.

Se va a diferenciar el estudio entre los tipos de red considerados en este documento.

9.1.1 RED 1 (CONTROL DE NIVEL CON 1 ENTRADA):

Se empezará por el tipo de red 1, es decir, el lazo de nivel con 1 entrada.

Para cambiar la cantidad de datos con los que se entrenan las redes, hemos introducido un bloque de comandos que elimina una cantidad de datos determinada por una variable. Después se ha creado y entrenado la red cambiando los datos a eliminar y por último se ha simulado esa red con una referencia de escalones.

El resultado del entrenamiento es el siguiente:

	Epoch	MSE	Gradient:
Sin eliminar:	120/120	0.814896/0.3	969.836/0.0001
Eliminando el 5%:	120/120	0.864175/0.3	701.566/0.0001
Eliminando el 10%:	62/120	0.861509/0.3	2666.67/0.0001
Eliminando el 15%:	14/120	0.823391/0.3	528.604/0.0001

Tabla 1. Resultados Entrenamiento Valores Iniciales. Red 1.

El resultado de todas las simulaciones es el siguiente:

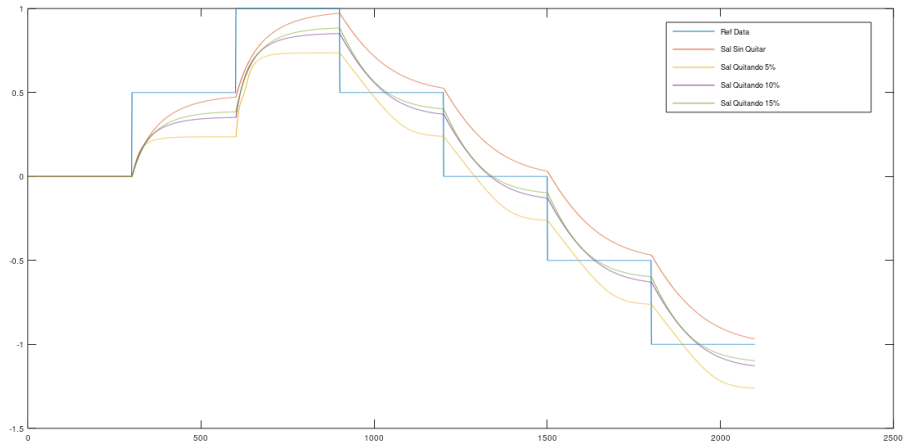


Ilustración 90. Red 1 Eliminando Valores.

En la anterior ilustración se representa en diferentes tonalidades las redes mencionadas. Se va a analizar primero como funciona cuando la referencia es ascendente, analizando el primer escalón.

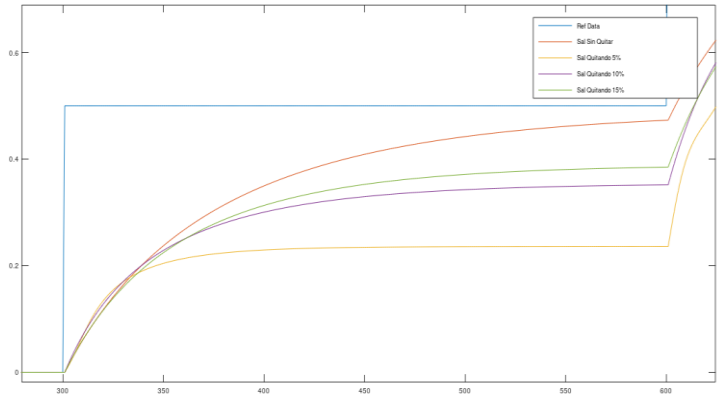


Ilustración 91. Red 1 Eliminando Valores 2. Referencia Ascendente.

Se puede apreciar la referencia en azul, y como todas las redes, siguen la forma de la referencia, aunque, en todos los casos, tienen un error de posición permanente. También, se puede observar que no hay sobreoscilación en ningún caso, aunque esto es debido a la forma en la que se controla ya que todos tienen esa misma forma. Al final de este apartado se encuentra la tabla resumen con resultados numéricos.

Por otra parte, si se enfoca en el primer salto descendente, se encuentra lo siguiente:

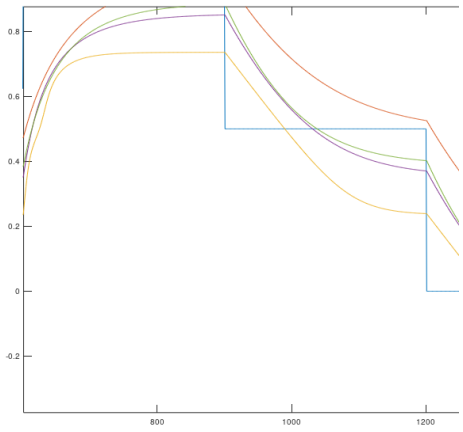


Ilustración 92. Red 1 Eliminando Valores 3. Referencia Descendente.

De la misma manera que para la referencia ascendente, las redes son capaces de seguir la referencia cuando esta desciende. Aunque en este caso los errores en régimen permanente han cambiado respecto al caso anterior. Sin embargo, el orden en el que las redes se acercan más a la referencia se mantiene. En la siguiente tabla se muestra un resumen del error en régimen permanente que tiene cada red.

	Referencia Ascendente			Referencia Descendente		
Redes:	V.Ini (Ud)	V.Fin (Ud)	Error Ref Asc (%)	V.Ini (Ud)	V.Fin (Ud)	Error Ref Desc (%)
Referencia	0	0,5		1	0,5	
Red Sin Eliminar Datos	0	0,473	5,40%	0,972	0,525	5,00%
Red Eliminando 5% de los Datos	0	0,236	52,80%	0,884	0,239	52,20%
Red Eliminando 10% de los Datos	0	0,352	29,60%	0,85	0,37	26,00%
Red Eliminando 15% de los Datos	0	0,382	23,60%	0,736	0,402	19,60%

Tabla 2. Desempeño Red 1 Eliminando Valores.

Se puede apreciar en la tabla anterior como para la red neuronal en el lazo de nivel con 1 entrada, el mejor comportamiento de todos se consigue cuando no se elimina ningún porcentaje de datos. Por otra parte, si se eliminan pocos datos (5%), el comportamiento difiere mucho del ideal, con un error del alrededor del 50% en escalones ascendentes y descendentes.

Cuando se eliminan más datos (10%), el error desciende en gran medida, con lo que se empieza a apreciar el efecto de evitar el tránsito a las condiciones de funcionamiento normales. Sin embargo, aún se mantiene, ya que cuando se eliminan más datos (15%), la red sigue mejorando su desempeño. Aunque lejos de acercarse al funcionamiento cuando no se han eliminado ninguna cantidad de datos. Se puede concluir que, en la red del lazo de nivel con una entrada, es mejor cuantos más datos se le proporcione a la red, aunque incluya algún tránsito.

9.1.2 RED 3 (CONTROL DE NIVEL CON 2 ENTRADAS):

El resultado del entrenamiento de las redes para el lazo de nivel con 2 entradas se presenta a continuación:

	Epoch	MSE	Gradient:
Sin eliminar:	15/100	0.629583/0	4572.5/0.0001
Eliminando el 5%:	6/100	0.889765/0	6415.87/0.0001
Eliminando el 10%:	8/100	0.744029/0	1559.1/0.0001
Eliminando el 15%:	10/100	0.647049/0	225.396/0.0001

Tabla 3. Resultados Entrenamiento Valores Iniciales. Red 3.

Se puede ver como a diferencia de la tipología de red 1, esta ha necesitado muchas menos iteraciones para converger, en algunos casos incluso con errores más pequeños con lo que en principio debería tener un mejor desempeño en la simulación. Se presenta a continuación la simulación junto con los datos resultado:

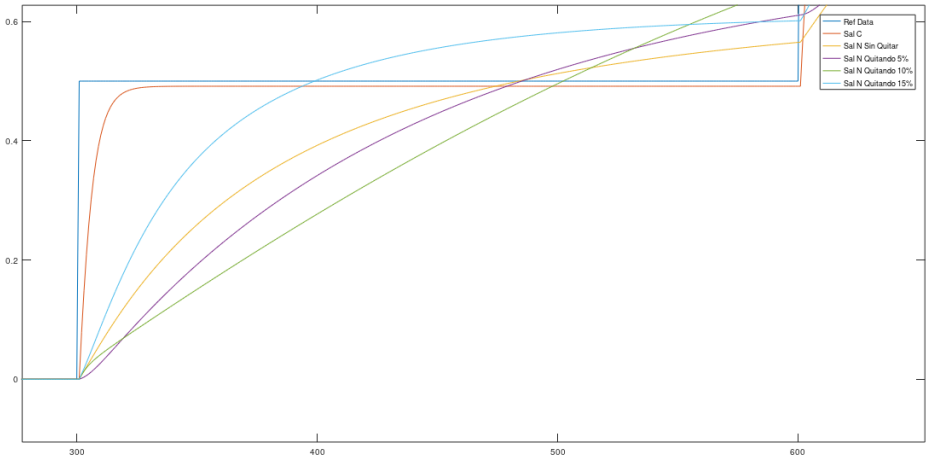


Ilustración 93. Red 3 Eliminando Valores.

	Referencia Ascendente			Referencia Descendente		
Redes:	V.Ini (Ud)	V.Fin (Ud)	Error Ref Asc (%)	V.Ini (Ud)	V.Fin (Ud)	Error Ref Desc (%)
Referencia	20	20,5		21	20,5	
Red Sin Eliminar Datos	20	22,92	484,00%	23,42	22,71	442,00%
Red Eliminando 5%	20	20,581	16,20%	21,081	20,579	15,80%
Red Eliminando 10%	20	30,51	2002,00%	42,02	52,046	6309,20%
Red Eliminando 15%	20	20,485	3,00%	20,985	20,488	2,40%

Tabla 4. Desempeño Red 3 Eliminando Valores.

Se puede apreciar que las redes tienen un funcionamiento similar, aunque con diferencias. Hay dos redes que no han sido capaces de tener buenos resultados, por una parte, cuando no se eliminan datos y cuando solo se eliminan el 10%. Es un comportamiento diferente al que encontramos con una sola entrada. Aquí se puede ver que los transitorios que se encuentran en los datos tienen un efecto más grande.

9.1.3 RED 2 (CONTROL DE CAUDAL CON 1 ENTRADA):

El resultado del entrenamiento es el siguiente:

	Epoch	MSE	Gradient:
Sin eliminar:	100/100	0.629583/0	4572.5/0.0001
Eliminando el 5%:	100/100	0.889765/0	6415.87/0.0001
Eliminando el 10%:	100/100	0.744029/0	1559.1/0.0001
Eliminando el 15%:	100/100	0.647049/0	225.396/0.0001

Tabla 5. Resultados Entrenamiento Valores Iniciales. Red 2.

Para el lazo de caudal, se hace el mismo proceso, evaluando primero la red 2, control de caudal con 2 entradas.

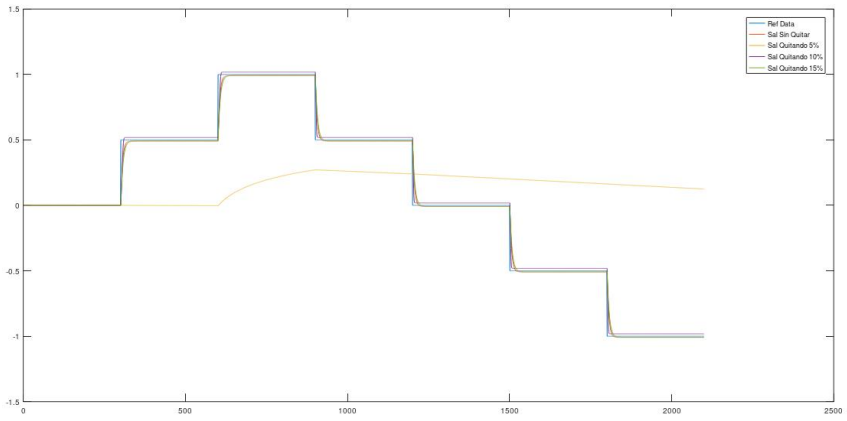


Ilustración 94.Red 2 Eliminando Valores.

En este caso, menos la red que se ha eliminado el 5% de los datos, las demás funcionan de una manera muy parecida, alcanzando la referencia sin problemas, aunque con alguna variación. El funcionamiento de la red con el 5% de los datos eliminados, puede deberse a una mala convergencia de la red en el entrenamiento. Se podría asociar a la cantidad de datos eliminada, aunque viendo cómo se comportan las demás redes, es más posible pensar que se debe a una mala convergencia en el entrenamiento por razones de aleatoriedad en encontrar un mínimo local del error en lugar de un mínimo global.

Evaluando los siguientes escalones podemos obtener los resultados que se presentan a continuación.

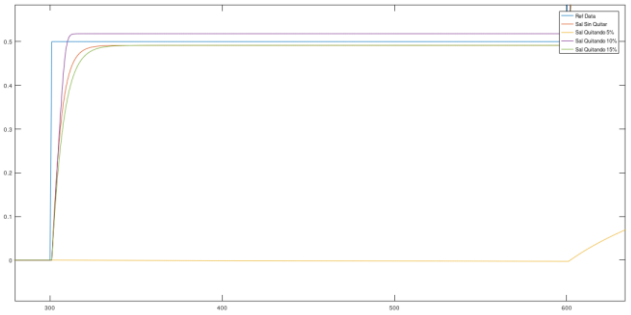


Ilustración 95. Red 2 Eliminando Valores. Escalón Ascendente.

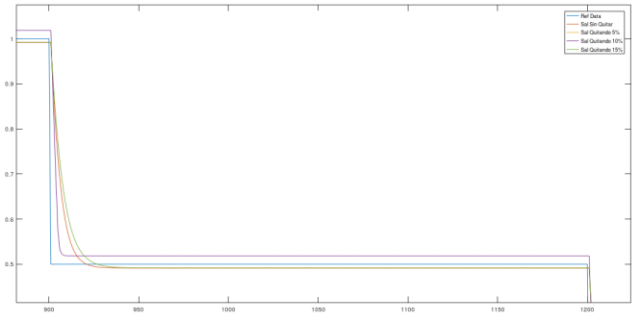


Ilustración 96. Red 2 Eliminando Valores. Escalón Descendente.

	Referencia Ascendente			Referencia Descendente		
Redes:	V.Ini (Ud)	V.Fin (Ud)	Error Ref Asc (%)	V.Ini (Ud)	V.Fin (Ud)	Error Ref Desc (%)
Referencia	5	5,5		6	5,5	
Red Sin Eliminar Datos	5	5,4915	1,70%	5,9914	5,4917	3,60%
Red Eliminando 5%	5	5,5185	3,70%	5,27	5,24	52,00%
Red Eliminando 10%	5	5	100,00%	6,017	5,518	3,60%
Red Eliminando 15%	5	5,4917	1,66%	5,9918	5,4918	1,64%

Tabla 6. Desempeño Red 2 Eliminando Valores.

A partir de los datos anteriores se puede extraer que, de las 4 redes, hay 3 que funcionan mejor para una referencia ascendente cuando se eliminan datos y otras 3 para escalón descendente cuando se eliminan otros y no son las mismas. Para el escalón ascendente sin eliminar, eliminando el 10% y eliminando el 15% funcionan con un error pequeño siendo la red de eliminando el 15% la que mejor funciona. Por otra parte, para escalón descendente sin eliminar, eliminando el 5% y eliminando el 15% funcionan con un error pequeño siendo la del 15% la que mejor funciona. Por lo tanto, la red que mejor funciona para todos los escalones es eliminar el 15% de los primeros datos al entrenar la red.

9.1.4 RED 4 (CONTROL DE CAUDAL CON 2 ENTRADAS):

El resultado del entrenamiento de las redes es el siguiente:

	Epoch	MSE	Gradient:
Sin eliminar:	43/100	0.627962/0	6207.56/0.0001
Eliminando el 5%:	100/100	0.626249/0	2120.23/0.0001
Eliminando el 10%:	100/100	0.637456/0	2179.07/0.0001
Eliminando el 15%:	100/100	0.646338/0	812.043/0.0001

Tabla 7. Resultados Entrenamiento Valores Iniciales. Red 4.

De los resultados del entrenamiento no se puede extraer ninguna conclusión ya que todas las redes han convergido sobre los mismos errores (alrededor de 0.62) por lo en principio deben funcionar de manera parecida.

El resultado de la simulación se presenta a continuación:

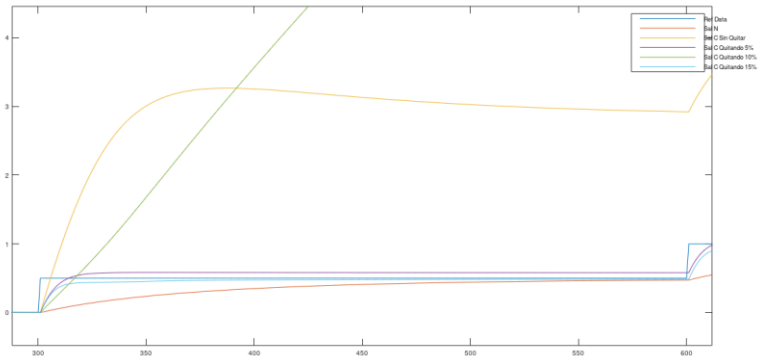


Ilustración 97. Red 2 Eliminando Valores. Escalón Descendente.

Si calculamos el error de cada red respecto al escalón y la referencia, se presentan los siguientes resultados:

	Referencia Ascendente			Referencia Descendente		
Redes:	V.Ini (Ud)	V.Fin (Ud)	Error Ref Asc (%)	V.Ini (Ud)	V.Fin (Ud)	Error Ref Desc (%)
Referencia	5	5,5		5,5	5	
Red Sin Eliminar Datos	5	5,565	13,00%	8,42	8,42	684,00%
Red Eliminando 5%	5	5,611	22,20%	5,58	5,08	16,00%
Red Eliminando 10%	5	5,667	33,40%	20,85	31,32	5264,00%
Red Eliminando 15%	5	5,6105	22,10%	5,498	4,882	23,60%

Tabla 8. Desempeño Red 4 Eliminando Valores.

A diferencia de cómo se presentaba el entrenamiento de la red, la simulación funciona de manera diferente y como se puede ver en la tabla anterior los errores en régimen permanente difieren mucho unas redes de otras.

Para escalones ascendentes cuando se elimina un 5 y 15% el error es similar, aunque con un 10% funciona peor y cuando no se elimina nada mejor aún. Lo que indica que el eliminar valores ha hecho que cuando se quiera un escalón ascendente la red no sea capaz de aprender lo suficiente. Para escalones descendentes funciona de manera muy diferente ya que hay dos redes que se han descontrolado y tienen errores muy altos para el escalón que tenemos. En cambio, cuando eliminamos un 5 o 15% la red ha funcionado mejor, con una mejoría si solo eliminamos el 5%.

9.1.5 CONCLUSIÓN:

El resumen de los resultados anteriores es:

RED	SOLUCIÓN:
1_NIVEL 1 ENTRADA	RED SIN ELIMINAR
2_CAUDAL 1 ENTRADA	RED ELIMINANDO EL 15%
3_NIVEL 2 ENTRADAS	RED ELIMINANDO EL 15%
4_CAUDAL 2 ENTRADAS	RED ELIMINANDO EL 15%

Tabla 9. Conclusión Eliminar Valores.

Se puede concluir que la red neuronal funciona mejor cuando se elimina el 15% de los primeros datos, ya que permite eliminar los transitorios que se generan en los datos de entrenamiento que pueden hacer pensar a la red neuronal que se encuentra en otro punto de funcionamiento diferente al que debería encontrarse.

9.2 NORMALIZACIÓN DE LOS VALORES:

Cuando se va a entrenar una red neuronal es importante que los datos estén acotados en un intervalo pequeño para que la red pueda converger de manera más fácil. Como ya se ha explicado en los apartados del código, existen funciones previamente diseñadas para este hecho. En este apartado se va a estudiar cómo afecta al entrenamiento de la red y a su posterior desempeño cuando se normalizan los datos para el entrenamiento. También, hay que tener en cuenta que, si la red ha sido normalizada, cuando se quiera simular, los datos de entrada de la simulación también deben ser normalizados ya que, si no, se encontrarán en magnitudes diferentes.

Para poder comprobar todos los casos posibles y obtener mejores conclusiones se abordarán los 4 casos posibles por red, es decir, primero entrenamos dos redes por tipo, una normalizando los valores y otra no y después de cada una de ellas se simula comparando si se normaliza la entrada y desnormaliza la salida o no.

Así, se puede comprobar si realmente la normalización es necesaria o no, y en qué casos. Cómo los resultados han sido muy similares en todas las redes, se expondrán primero los datos y finalmente la conclusión de estos.

9.2.1 RED 1 (CONTROL DE NIVEL CON 1 ENTRADA):

El resultado del entrenamiento es el siguiente:

	Epoch	MSE	Gradient:
Sin normalizar:	1/120	1862.99/0.3	0/0.0001
Normalizando:	120/120	0.814832/0.3	1186.01/0.0001

Tabla 10. Resultados Entrenamiento Normalización. Red 1.

Se puede ver claramente que la red al no normalizar los datos de entrada presenta un error muy grande respecto a cuando si lo hace, lo que prevé un mal funcionamiento en simulación. Si se simula las dos redes anteriores, se obtiene:

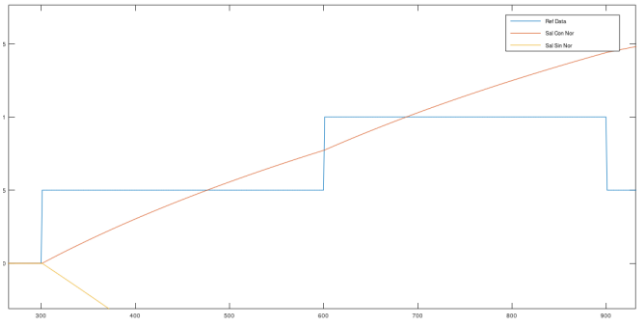


Ilustración 98. Simulación Red 1 Normalizando. Red Normalizada y No Normalizada.

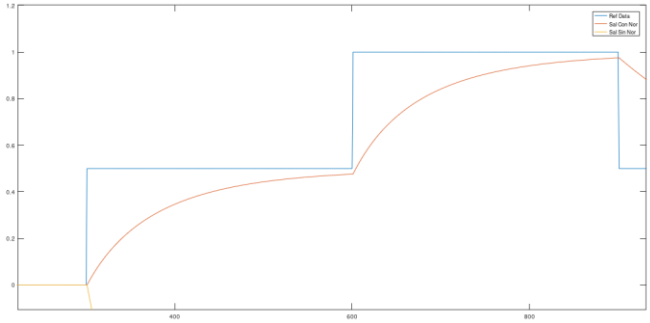


Ilustración 99. Simulación Red 1 No Normalizando. Red Normalizada y No Normalizada.

De las gráficas anteriores podemos extraer los datos necesarios para completar la tabla resultado de esta red.

	Referencia Ascendente			Referencia Descendente		
Redes:	V.Ini (Ud)	V.Fin (Ud)	Error Ref Asc (%)	V.Ini (Ud)	V.Fin (Ud)	Error Ref Desc (%)
Referencia	20	20,5		21	20,5	
Simulación 1 Normalizando						
Entrenamiento Normalizado	20	20,773	54,60%	21,452	21,795	259,00%
Entrenamiento No Normalizado	20	18,605	379,00%	16,99	15,315	1037,00%
Simulación 1 No Normalizando						
Entrenamiento Normalizado	20	20,476	4,80%	20,976	20,535	7,00%
Entrenamiento No Normalizado	20	18,606	378,80%	13,6	-0,5	4200,00%

Tabla 11. Desempeño Red 1. Normalizando.

9.2.2 RED 3 (CONTROL DE NIVEL CON 2 ENTRADAS):

Si se construyen las redes se obtienen los siguientes resultados:

	Epoch	MSE	Gradient:
Sin normalizar:	1/100	1797.6/0	0/0.0001
Normalizando:	8/100	0.811027/0	4027.48/0.0001

Tabla 12. Resultados Entrenamiento Normalización. Red 2.

Simulando las redes obtenemos:

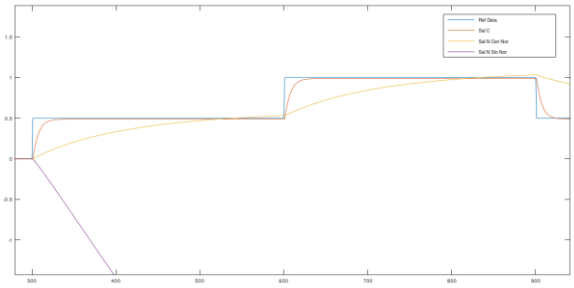


Ilustración 100. Simulación Red 3 Normalizando. Red Normalizada y No Normalizada.

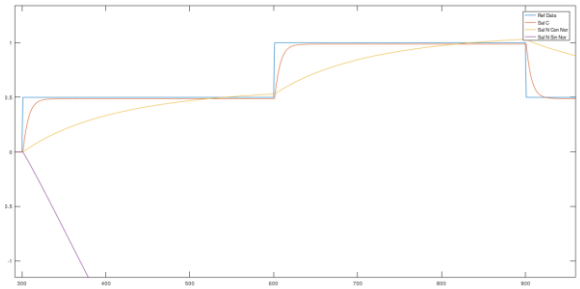


Ilustración 101. Simulación Red 3 No Normalizando. Red Normalizada y No Normalizada.

	Referencia Ascendente			Referencia Descendente		
Redes:	V.Ini (Ud)	V.Fin (Ud)	Error Ref Asc (%)	V.Ini (Ud)	V.Fin (Ud)	Error Ref Desc (%)
Referencia	20	20,5		21	20,5	
Simulación 3 Normalizando						
Entrenamiento Normalizado	20	20,531	6,20%	21,033	20,641	28,20%
Entrenamiento No Normalizado	20	15,75	950,00%	12,31	9,48	2204,00%
Simulación 3 No Normalizando						
Entrenamiento Normalizado	20	20,531	6,20%	21,033	20,641	28,20%
Entrenamiento No Normalizado	20	15,75	950,00%	12,31	9,48	2204,00%

Tabla 13. Desempeño Red 3. Normalizando.

9.2.3 RED 2 (CONTROL DE CAUDAL CON 1 ENTRADA):

	Epoch	MSE	Gradient:
Sin normalizar:	1/100	46.7742/0	0/0.0001
Normalizando:	100/100	0.623294/0	2526.37/0.0001

Tabla 14. Resultados Entrenamiento Normalización. Red 3.

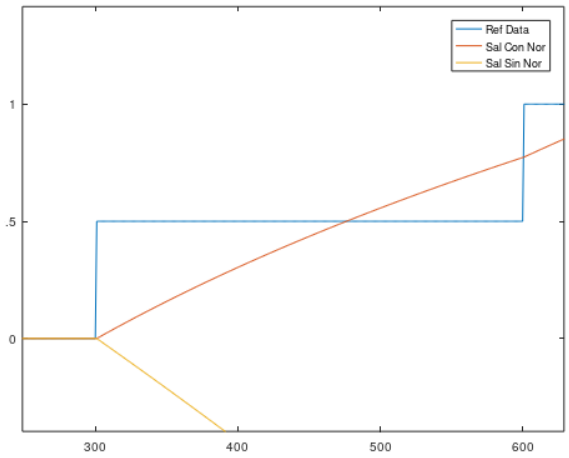


Ilustración 102. Simulación Red 2 Normalizando. Red Normalizada y No Normalizada.

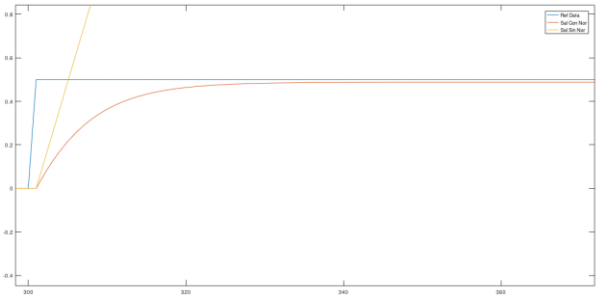


Ilustración 103. Simulación Red 2 No Normalizando. Red Normalizada y No Normalizada.

	Referencia Ascendente			Referencia Descendente		
Redes:	V.Ini (Ud)	V.Fin (Ud)	Error Ref Asc (%)	V.Ini (Ud)	V.Fin (Ud)	Error Ref Desc (%)
Referencia	5	5,5		6	5,5	
Simulación 2 Normalizando						
Entrenamiento Normalizado	5	5,773	54,60%	6,44	6,79	258,00%
Entrenamiento No Normalizado	5	3,57	386,00%	2,98	1,33	834,00%
Simulación 2 No Normalizando						
Entrenamiento Normalizado	5	5,38	24,00%	5,98	5,488	2,40%
Entrenamiento No Normalizado	5	36,5	6200,00%	73,2	107,6	20420,00%

Tabla 15. Desempeño Red 2. Normalizando.

9.2.4 RED 4 (CONTROL DE CAUDAL CON 2 ENTRADAS):

	Epoch	MSE	Gradient:
Sin normalizar:	3/100	54.3091/0	0.0045862/0.0001
Normalizando:	100/100	0.626412/0	2098.12/0.0001

Tabla 16. Resultados Entrenamiento Normalización. Red 4.

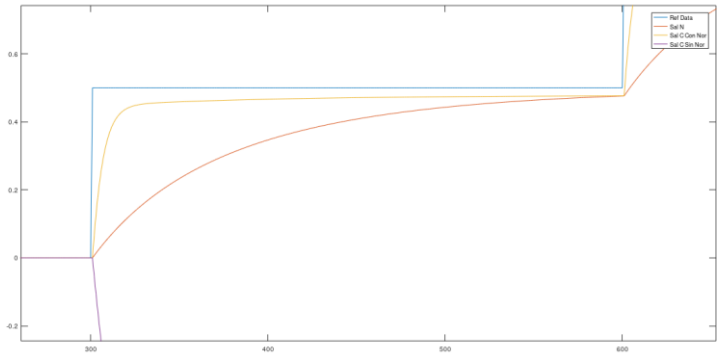


Ilustración 104. Simulación Red 3 No Normalizando. Red Normalizada y No Normalizada.

	Referencia Ascendente			Referencia Descendente		
Redes:	V.Ini (Ud)	V.Fin (Ud)	Error Ref Asc (%)	V.Ini (Ud)	V.Fin (Ud)	Error Ref Desc (%)
Referencia	5	5,5		5,5	5	
Simulación 4 Normalizando						
Entrenamiento Normalizado	5	5,476	4,80%	5,476	4,981	3,80%
Entrenamiento No Normalizado	5	2,79	542,00%	-1,7	-2,342	1468,40%
Simulación 4 No Normalizando						
Entrenamiento Normalizado	5	5,476	4,80%	-1,724	-2,361	1472,20%
Entrenamiento No Normalizado	5	2,79	542,00%	-8,9	-2,342	1468,40%

Tabla 17. Desempeño Red 4. Normalizando.

9.2.5 CONCLUSIÓN:

Se puede observar claramente que cuando en la simulación se normaliza los valores que entran a la red, esta no es capaz de realizar el control correctamente. Pero, por otra parte, es muy importante normalizar los mismos cuando se está entrenando la red ya que si no se realiza, la red no será capaz de seguir la referencia. Se puede apreciar en que los errores de la simulación normalizada son muy grandes, incluso más cuando no se normaliza. Por ello, la mejor solución es normalizar en el entrenamiento, pero no hacerlo en la simulación.

RED	SOLUCIÓN	
	ENTRENAMIENTO	SIMULACIÓN
1_NIVEL 1 ENTRADA	CON NORMALIZACIÓN	SIN NORMALIZACIÓN
2_CAUDAL 1 ENTRADA	CON NORMALIZACIÓN	SIN NORMALIZACIÓN
3_NIVEL 2 ENTRADAS	CON NORMALIZACIÓN	SIN NORMALIZACIÓN
4_CAUDAL 2 ENTRADAS	CON NORMALIZACIÓN	SIN NORMALIZACIÓN

Tabla 18. Conclusión Normalización.

9.3 LIMITANDO LOS PIDs:

Para la construcción de los datos que se necesitan para entrenar la red se construyó el archivo de simulación de Simulink ya explicado en apartados anteriores. Después de varias simulaciones con las que el PID funcionaba correctamente, se encontró que, para que la simulación fuera lo más real posible el PID tenía que estar limitado a las acciones que los actuadores pueden realizar. Puesto que los actuadores en esta instalación son la bomba para el lazo de nivel y la válvula para el lazo del caudal, no son infinitos, sino que tienen un rango de acción determinado. Este rango de acción se tiene que incluir en la programación.

En el montaje real, la salida de la red neuronal es conducida hasta el actuador, siendo la bomba o la válvula. De manera que generan una transformación de la señal en una acción que se traduce a voltaje. Como no son infinitos los actuadores, se puede limitar la salida de la red a el límite en voltaje de esos actuadores y conducir directamente la señal de la red al sistema, o limitar en porcentaje la salida de la red y lo que se limite sea la acción entre un 0 y 100%. Para el montaje real, esta última forma de limitar tendrá introducidos los actuadores que harán que ese 100% sea su máxima acción y el 0% la mínima.

Analizaremos el comportamiento del PID y de la Red Neuronal en cada lazo por separado para las diferentes combinaciones de limitaciones, es decir, si se limita entre 100% y 0%, si se limita con los rangos de los actuadores o si simplemente no se limita.

9.3.1 LAZO CONTROL DE NIVEL:

El primer lazo para analizar será el de Nivel. Primero se realizará la simulación del PID sin cambios y después se introducirá los límites y se volverá a simular.

9.3.1.1 PID:

Si se grafica las dos respuestas del controlador PID respecto a la referencia se obtiene lo siguiente:

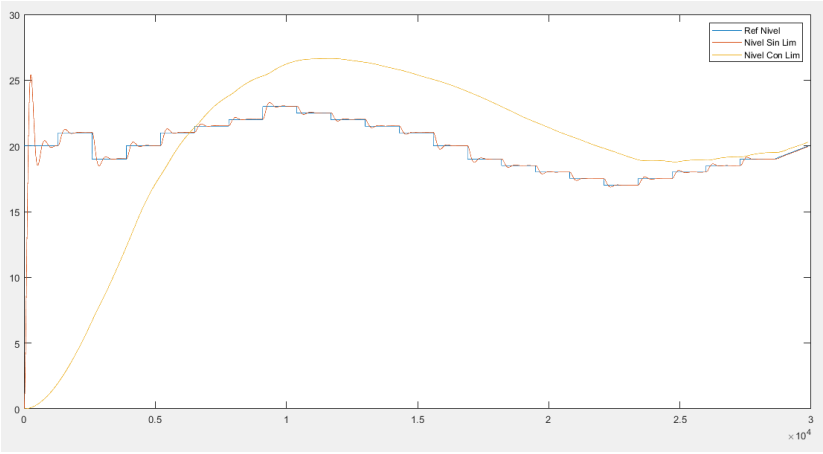


Ilustración 105. Respuesta del PID Con y Sin Límites en el Lazo de Nivel.

Por una parte, se puede observar como la respuesta sin limitaciones del PID es capaz de llevar la variable a la referencia con un poco de sobreoscilación pero con error en régimen permanente nulo. En cada salto consigue acercarse a la referencia sin problemas. En cambio, cuando se limita el PID se puede observar como la acción de control que ha de tomar es mucho más grande que el límite y por tanto es incapaz de alcanzar la referencia. Se puede ver como el problema principalmente viene con alcanzar el punto de equilibrio desde condiciones nulas, aunque aun así no consigue seguir los cambios de variable.

9.3.1.2 RED NEURONAL:

Con los datos de simulación del PID, podemos entrenar la red neuronal y ver su funcionamiento, primero calcularemos la red sin haber limitado y después con los límites introducidos, así podremos observar si la variabilidad en los datos de entrenamiento se refleja en el funcionamiento de la red.

Los resultados sobre el entrenamiento de las diferentes redes necesitadas son los siguientes:

- Sin Límites:

	Epoch	MSE	Gradient:
1 Entrada:	10/120	0.969978/0	11043.8/0.0001
2 Entradas:	8/100	0.818565/0	9107.39/0.0001

Tabla 19. Resultados Entrenamiento Límites Lazo Nivel. Sin Límites.

- Límite entre 0 y 100%:

	Epoch	MSE	Gradient:
1 Entrada:	11/120	0.815317/0	1259.68/0.0001
2 Entradas:	8/100	0.814687/0	13280.3/0.0001

Tabla 20. Resultados Entrenamiento Límites Lazo Nivel. Límite entre 0 y 100%.

- Límite entre rango de actuadores:

	Epoch	MSE	Gradient:
1 Entrada:	12/120	0.634441/0	1496.61/0.0001
2 Entradas:	9/100	0.549085/0	2016.41/0.0001

Tabla 21. Resultados Entrenamiento Límites Lazo Nivel. Límite entre Rango de Actuadores.

A primera vista se puede extraer que la red converge con gran facilidad ya que solo necesita unas pocas simulaciones, alrededor de 10. También se puede ver que el error varía bastante dependiendo de los límites con los que se haya entrenado siendo la red con menor error cuando se limita entre actuadores y la que peor cuando no se imponen límites. Para poder sacar una mejor conclusión se va a simular todas las redes con las mismas entradas.

Los resultados de la red 1 del lazo de nivel son los siguientes:

	Referencia Ascendente			Referencia Descendente		
Redes:	V.Ini (Ud)	V.Fin (Ud)	Error Ref Asc (%)	V.Ini (Ud)	V.Fin (Ud)	Error Ref Desc (%)
Referencia	20	20,5		21	20,5	
Red Sin Limitar	20	20,574	14,80%	21,074	20,477	4,60%
Red Lim 0-100%	20	20,589	17,80%	21,09	20,488	2,40%
Red Lim Actuadores	20	20,488	2,40%	20,985	20,43	14,00%

Tabla 22. Resultados Simulación Límites Lazo Nivel. Red 1.

Por otra parte, los resultados de la red 3 del lazo de nivel son los siguientes:

	Referencia Ascendente			Referencia Descendente		
Redes:	V.Ini (Ud)	V.Fin (Ud)	Error Ref Asc (%)	V.Ini (Ud)	V.Fin (Ud)	Error Ref Desc (%)
Referencia	5	5,5		6	5,5	
Red Sin Limitar	4,99	5,49	2,00%	5,9899	5,482	3,60%
Red Lim 0-100%	5,001	5,501	0,20%	6,001	5,4975	0,50%
Red Lim Actuadores	4,609	5,11	78,00%	5,67	5,67	34,00%

Tabla 23. Resultados Simulación Límites Lazo Nivel. Red 3.

Como se puede observar por los datos anteriores, el lazo de nivel en general funciona mejor cuando se construye y controla en base a dos entradas que en 1. Ya que los errores son menores en las redes sin limitar y con límites de 0 y 100%. En cambio, para la red limitada con el rango de los actuadores la red se entrena y simula mejor cuando solo tiene una entrada. Esto puede deberse a que, al ser una red con un funcionamiento complejo, cuantos más datos tenga la red para controlarlo mejor.

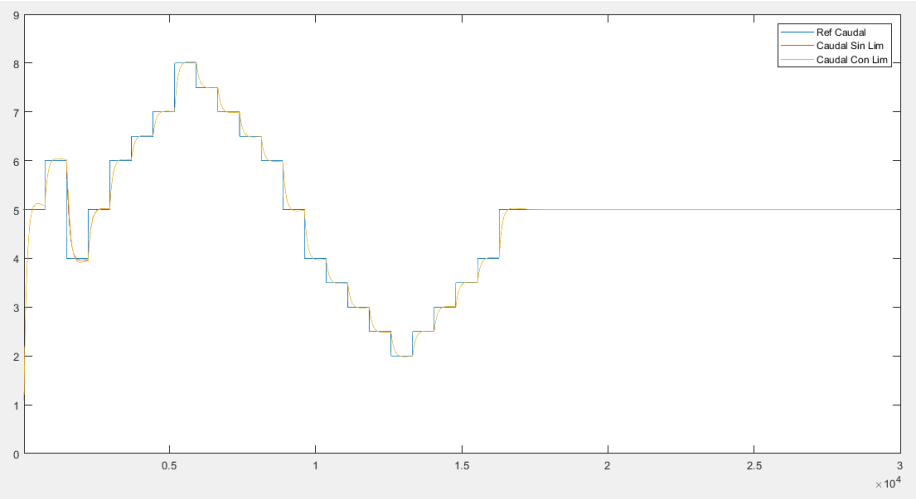
Se puede concluir que las redes funcionan mejor cuando son limitas además de que así se consigue una simulación mejor de cómo será la red en el montaje real. También se puede ver que cuando hay una entrada los errores son más pequeños en referencias descendentes, pero en cambio, cuando son 2 entradas los errores más pequeños se encuentran en referencias ascendentes, quitando de la red con los límites de los actuadores que su funcionamiento parece ser diferente al de las demás. Esto puede ser causado por una mala convergencia en su entrenamiento que no se debe cumplir de manera generalizada.

Específicamente para la Red 1, la que funciona mejor es cuando tiene los límites entre actuadores, lo que concuerda con los datos obtenidos en el entrenamiento. Para la Red 2 sin duda la que funciona mejor es cuando los límites se encuentran entre 0 y 100%. Finalmente, la que funciona mejor para controlar el lazo de nivel es con dos entradas y límites entre 0 y 100%, la Red 3.

9.3.2 LAZO CONTROL DE CAUDAL:

9.3.2.1 PID:

Si se grafica las dos respuestas del controlador PID respecto a la referencia se obtiene lo siguiente:



En este caso, la respuesta con o sin límites es exactamente la misma y por tanto se encuentran solapadas. Por ello solo se puede observar una línea. La razón es que, aunque se haya limitado el rango de acción del PID, como las acciones sin limitaciones, ya estaban dentro de ese límite, por tanto, no ha habido alteraciones.

9.3.2.2 RED NEURONAL:

Los resultados sobre el entrenamiento del lazo de caudal variando los límites son los siguientes:

- Sin Límites:

	Epoch	MSE	Gradient:
1 Entrada:	115/200	0.626889/0	1144.61/0.0001
2 Entradas:	103/200	0.577885/0	8355.08/0.0001

Tabla 24. Resultados Entrenamiento Límites Lazo Caudal. Sin Límites.

- Límite entre 0 y 100%:

	Epoch	MSE	Gradient:
1 Entrada:	200/200	0.626542/0	1123.16/0.0001
2 Entradas:	100/200	0.57573/0	16888/0.0001

Tabla 25. Resultados Entrenamiento Límites Lazo Caudal. Límite entre 0 y 100%.

- Límite entre rango de actuadores:

	Epoch	MSE	Gradient:
1 Entrada:	159/200	0.549085/0	852.85/0.0001
2 Entradas:	100/200	0.626652/0	2186.49/0.0001

Tabla 26. Resultados Entrenamiento Límites Lazo Caudal. Límite entre Rango de Actuadores.

Se puede apreciar como a diferencia del lazo de nivel, la red necesita más iteraciones para converger, aunque el funcionamiento es mejor ya que los errores de todas las redes se encuentran entre 0.549 y 6268. Para poder sacar una mejor conclusión se va a simular todas las redes con las mismas entradas.

Los resultados de la red 2 del lazo de caudal son los siguientes:

	Referencia Ascendente			Referencia Descendente		
Redes:	V.Ini (Ud)	V.Fin (Ud)	Error Ref Asc (%)	V.Ini (Ud)	V.Fin (Ud)	Error Ref Desc (%)
Referencia	5	5,5		6	5,5	
Red Sin Limitar	4,99	5,49	2,00%	5,9899	5,482	3,60%
Red Lim 0-100%	5,001	5,501	0,20%	6,001	5,4975	0,50%
Red Lim Actuadores	4,609	5,11	78,00%	5,67	5,67	34,00%

Tabla 27. Resultados Simulación Límites Lazo Nivel. Red 4.

Los resultados de la red 4 del lazo de caudal son los siguientes:

	Referencia Ascendente			Referencia Descendente		
Redes:	V.Ini (Ud)	V.Fin (Ud)	Error Ref Asc (%)	V.Ini (Ud)	V.Fin (Ud)	Error Ref Desc (%)
Referencia	5	5,5		6	5,5	
Red Sin Limitar	5,602	6,015	103,00%	6,43	5,79	58,00%
Red Lim 0-100%	4,84	5,389	22,20%	4,9375	5,453	9,40%
Red Lim Actuadores	5,603	6,015	103,00%	6,426	5,787	57,40%

Tabla 28. Resultados Simulación Límites Lazo Nivel. Red 4.

De los resultados anteriores se pueden extraer varias conclusiones. Para la Red 2, cuando se limita entre 0 y 100% el funcionamiento es muy preciso, con unos errores entorno al 0% con lo que se obtiene un muy buen control. Los errores con la red sin limitar son muy buenos también, aunque algo peores, esto se debe a que casi todos los valores de actuación en este lazo son pequeños con lo que, aunque se limiten, como ya se encontraban dentro de este límite el funcionamiento es parecido.

Por otra parte, al controlar el caudal con una red con dos entradas nos encontramos que los errores son mayores, aunque la mejor solución sigue siendo la de limitar el caudal entre 0 y 100%.

9.3.3 CONCLUSIÓN:

El resumen de los resultados obtenidos es el siguiente:

RED:	LIMITACIÓN:
1_NIVEL 1 ENTRADA	ENTRE ACTUADORES:
2_CAUDAL 1 ENTRADA	ENTRE 0 Y 100%
3_NIVEL 2 ENTRADAS	ENTRE 0 Y 100%
4_CAUDAL 2 ENTRADAS	ENTRE 0 Y 100%

Tabla 29. Conclusión Limitación.

Como se puede observar la mayoría de las redes funciona mejor cuando se limita el funcionamiento de las salidas ya que también hace que estas estén en un rango mas pequeño. Dentro de la limitación se puede ver como solo una red funciona mejor con otro límite al de

limitar la red entre el 0 y 100% de la acción de control. Aunque esta diferencia para la Red 1 que es la diferente, es muy pequeña, por lo que se puede concluir que la mejor forma de limitar las Red Neuronal para todos los lazos es entre el0 y 100% de la acción de control.

9.4 CORRIGIENDO LA SALIDA DE LA RED NEURONAL:

Cuando la red ya ha sido entrenada y se estudia su simulación, es posible que se encuentre una red que está desfasada, es decir, que es capaz de seguir la forma de la referencia, pero a una escala diferente y lo que necesita es que manualmente se corrija su funcionamiento aumentando o disminuyendo esa escala (aumentando normalmente). La razón del desfase puede venir del error que existe cuando se entrena la red, que en la instalación que se estudia en este documento suele rondar el 0.5 hasta 0.6. Para solucionarlo se puede corregir cada neurona por individual, actuando sobre el "bias" o simplemente actuar sobre la salida final de la red que es como se solucionará en este caso. Se va a estudiar cómo afecta al desempeño de la red en la simulación si se corrige o no.

Como solo se va a cambiar si se corrige o no la salida, solo hace falta entrenar una red por tipo.

9.4.1 LAZO CONTROL DE NIVEL:

Los resultados sobre el entrenamiento del lazo de nivel son los siguientes:

	Epoch	MSE	Gradient:
1 Entrada:	120/200	0.633843/0	606.432/0.0001
2 Entradas:	9/200	0.695016/0	30988.1/0.0001

Tabla 30. Resultados Entrenamiento Corrigiendo Lazo Nivel.

Si se simula el tipo de Red 1, con los cambios mencionados se obtiene lo siguiente:

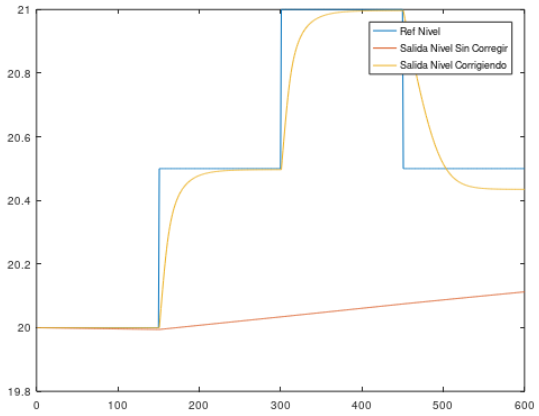


Ilustración 106. Resultados Gráficos Simulación Corrigiendo Lazo Nivel. Red 1.

De los datos del entrenamiento no podemos sacar ninguna conclusión, en cambio, viendo la simulación se observa claramente que la red necesita ser corregida sumando un valor para que sea capaz de alcanzar la referencia sin problemas.

9.4.2 LAZO CONTROL DE CAUDAL:

Los resultados sobre el entrenamiento del lazo de caudal son los siguientes:

	Epoch	MSE	Gradient:
1 Entrada:	117/200	0.634187/0	2635.34/0.0001
2 Entradas:	100/200	0.577299/0	1867.95/0.0001

Tabla 31. Resultados Entrenamiento Corrigiendo Lazo Nivel.

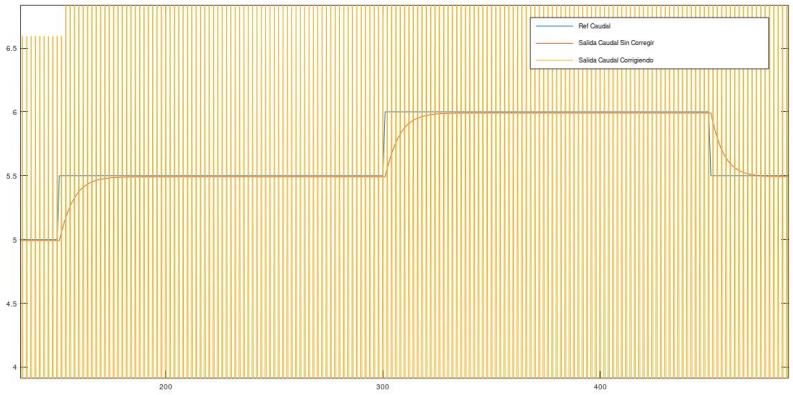


Tabla 32. Resultados Gráficos Simulación Corrigiendo Lazo Caudal. Red 2.

Observando la simulación se puede ver claramente como si la salida no es corregida, esta va a oscilar haciendo que no se alcance la referencia.

9.4.3 CONCLUSIÓN:

Como se ha visto en los anteriores puntos, cuando no se corrige la red después de entrenarla, esta tiene un desfase que provoca que no pueda seguir las referencias al completo por lo que para cada red:

Redes:	CON/SIN CORRECCIÓN
Lazo Nivel	
Red 1: 1 Entrada	CON CORRECIÓN
Red 1: 2 Entradas	CON CORRECIÓN
Lazo Caudal	
Red 1: 1 Entrada	CON CORRECIÓN
Red 1: 2 Entradas	CON CORRECIÓN

Tabla 33. Conclusión Correción Offset.

9.5 VARIANDO LAS REFERENCIAS:

Para evaluar cómo se comportan las diferentes redes cuando se cambian las referencias se van a construir y entrenar una de cada tipo para después ser simuladas con diferentes referencias y ver su comportamiento.

El resultado del entrenamiento de las redes es el siguiente:

	Epoch	MSE	Gradient:
Nivel 1 Entrada	10/120	0.634565/0	1471.45/0.0001
Caudal 1 Entrada	100/100	0.631224/0	1736.96/0.0001
Nivel 2 Entradas	8/100	0.633684/0	3017.84/0.0001
Caudal 2 Entradas	100/100	0.612327/0	2131.36/0.0001

Respecto a las redes vemos que las de caudal siguen necesitando muchas iteraciones para converger, a diferencia de las del lazo de nivel.

Evaluaremos cómo se comportan las dos redes de cada lazo con las diferentes referencias. Primero con el de nivel, con un escalón ascendente y con uno descendente:

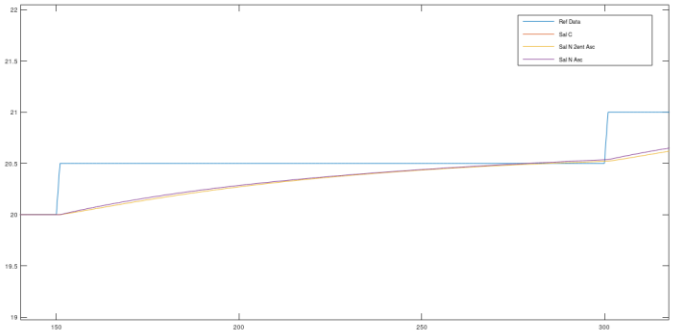


Ilustración 107. Evaluación Referencia Ascendente. Lazo de Nivel.

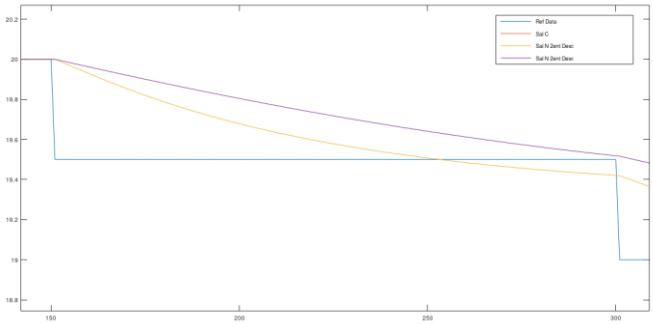


Ilustración 108. Evaluación Referencia Descendente. Lazo de Nivel.

A continuación, se muestra el lazo de caudal con un escalón ascendente y otro descendente:

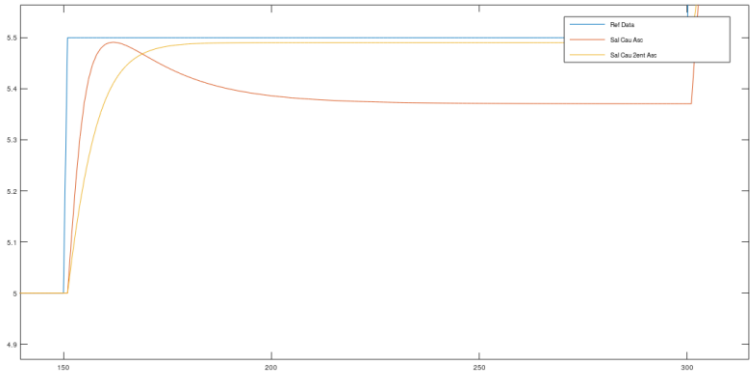


Ilustración 109. Evaluación Referencia Ascendente. Lazo de Caudal.

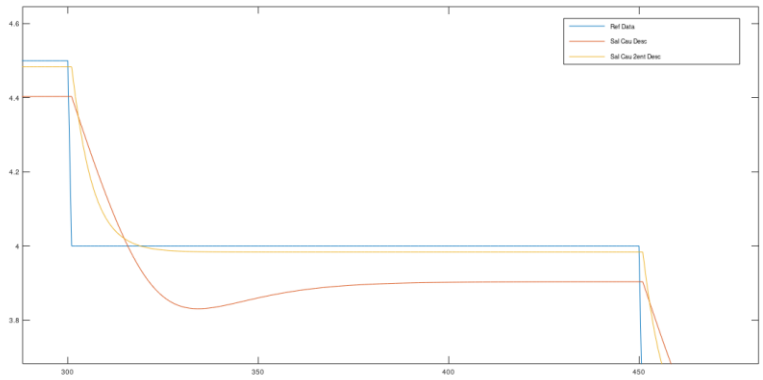


Ilustración 110. Evaluación Referencia Descendente. Lazo de Caudal.

Con los datos anteriores ya se puede realizar los cálculos para poder comparar unas redes con otras:

	Referencia Ascendente			Referencia Descendente		
Redes:	V.Ini (Ud)	V.Fin (Ud)	Error Ref Asc (%)	V.Ini (Ud)	V.Fin (Ud)	Error Ref Desc (%)
Referencia	20	20,5		20	19,5	
Lazo Nivel						
Red 1: 1 Entrada	20	20,535	7,00%	20	19,52	4,00%
Red 1: 2 Entradas	20	20,52	4,00%	20	19,42	16,00%
Referencia	5	5,5		4,5	4	
Lazo Caudal						
Red 1: 1 Entrada	5	5,371	25,80%	4,406	3,982	3,60%
Red 1: 2 Entradas	5	5,49	2,00%	4,482	3,905	19,00%

Tabla 34. Resultados Simulación Diferentes Referencias.

9.5.1 CONCLUSIÓN:

Según los datos anteriores se puede apreciar como si la referencia es ascendente, las redes con dos entradas trabajan mejor. Se puede ver en el lazo de nivel con una pequeña diferencia en el error pero que se aprecia en mayor magnitud para el lazo de caudal. Por otra parte, si la referencia es ascendente nos encontramos que las redes con una entrada trabajan mucho mejor que las de dos, con una diferencia de error grande.

Como en el funcionamiento real nos vamos a encontrar con variabilidades de escalones ascendentes y descendentes, sacamos el error medio para poder decidir con que lazo nos quedamos:

Redes:	Error Medio:
Lazo Nivel	
Red 1: 1 Entrada	5,50%
Red 1: 2 Entradas	10,00%
Lazo Caudal	
Red 1: 1 Entrada	14,70%
Red 1: 2 Entradas	10,50%

Tabla 35. Conclusión Referencias.

10 CONCLUSIONES:

Después del trabajo realizado es importante comprobar si se han realizado todos los objetivos que se habían planteado en un principio y que eran objetivos de esta investigación. Lo primero era construir y aplicar el control de un sistema de tanques mediante la aplicación de una Red Neuronal. Este objetivo ha sido ampliamente cubierto ya que se ha conseguido el control de este de diversas formas.

También, se ha comparado este controlador junto con el controlador al que quería imitar consiguiendo así una comparación de su funcionamiento. Por último, también se requería realizar un estudio sobre el funcionamiento de la Red en este sistema, obteniendo que parámetros son los más importantes para la realización y el buen funcionamiento de la red neuronal.

En base al último objetivo descrito se pueden extraer diferentes conclusiones. En la construcción y el entrenamiento de una red neuronal se necesita primero, una buena fuente de datos con los que entrenar la red, ya que estos serán muy importantes en su funcionamiento.

Como hemos visto en el apartado anterior, si estos datos incluyen transitorios, pueden hacer que la red neuronal entienda que trabaja en puntos diferentes de los que se requiere, haciendo que el control del sistema sea erróneo. Por lo tanto, es importante eliminar una cantidad suficiente de datos iniciales del controlador a imitar por la red, para que esta no encuentre puntos de equilibrio o funcionamientos diferentes y pueda aprender correctamente.

Por otra parte, se ha comprobado como de importante es que estos datos de entrenamiento se normalicen para que la red pueda converger más deprisa y de manera más precisa, ya que en todas las redes que se han simulado, los resultados así lo demostraban. Es importante tener en cuenta que los datos solo hace falta que estén normalizados para el entrenamiento, ya que después la red neuronal se puede corregir. Por lo que se necesita que la red aprenda la dinámica del sistema a controlar y ya después ajustando con el offset correspondiente se puede conseguir un buen control.

De esta corrección también se ha comprobado su finalidad, ya que como las redes al entrenarse siempre conllevan un error, es importante saber que solo con un ajuste sobre la salida de la red se consigue un mejor resultado. Aunque esto sería inviable si la red no hubiera aprendido la dinámica del sistema.

Para que la red lo consiga, también ha de ajustarse las condiciones en las que trabaja a la realidad, por ello, es importante que los datos de entrenamiento se ajusten a ello. En este caso, se ha encontrado que como el sistema real funciona con microcontroladores Arduino para hacer de intermediarios entre la red, los sensores y los actuadores, ahí hay unos cambios de voltaje que en la realidad se van a aplicar, por lo que es mejor que se lo introduzcamos a la red en el entrenamiento y también en la simulación. Había dos formas de realizarlo, una era limitando la acción de la red entre porcentajes que después serían transformados a valores de voltaje reales o directamente saltarte ese paso y aplicar los valores de voltaje reales. Es importante realizar la programación lo más adaptativa posible a los cambios y realizando la programación con acciones de porcentaje hacía que si se cambian los sensores o actuadores, simplemente se cambie esa transformación. Además, los datos mostraban que cuando los límites se realizaban entre esos valores, el control era más preciso.

Por último, una de las finalidades de esta investigación era comprobar también, para los dos lazos del sistema que teníamos, cuál de las dos posibilidades de control era mejor. Si una red

neuronal con una entrada y que se compusiese como un sistema SISO o una red neuronal donde se tuviese en cuenta la otra variable del sistema. Por lo tanto, se han calculado todas las variables para los dos tipos en los dos lazos, nunca descartando ninguna posibilidad. Los datos han mostrado que es diferente según el lazo del que se hable, lo cual es lógico y valida este punto de la investigación. Para calcularlo se han realizado diferentes escalones y comprobando el error también se ha visto que las redes no funcionan igual si el escalón es ascendente que descendente. Justo la en cada lazo, la red que era peor para un escalón, era mejor para el otro, por lo que después de calcular el error medio, se ha llegado a la conclusión que el lazo de Nivel debe ser controlado por una Red Neuronal con 1 entrada, ya que tiene una dinámica más compleja en la que la otra variable no afecta lo suficiente. Por otra parte, en el caso del Caudal, la variable de la bomba si afecta lo suficiente por lo que el control es mejor cuando se tiene en cuenta las dos.

11 FUTURO DE LA INVESTIGACIÓN Y PROYECTO:

Después de la realización de la investigación se proponen aspectos que están fuera del ámbito de este documento pero que serían muy interesantes a desarrollar en un futuro.

Lo más importante sería aplicar el control diseñado al sistema real, para poder probar y comprobar los aspectos estudiados anteriormente.

Es interesante esta continuación del proyecto para estudiar si todos los aspectos que en la simulación se han tenido en cuenta se aplican de la misma forma en el sistema real.

Por otra parte, también sería interesante aplicar los conocimientos aprendidos sobre redes con el código diseñado para el control de otros sistemas y comprobar si las conclusiones obtenidas en este proyecto se pueden aplicar a los demás y generalizar su funcionamiento.

12 BIBLIOGRAFÍA:

Faiber Ignacio Robayo B, A. M. (2015-2014). Desarrollo de un controlador basado en redes neuronales para un sistema multivariable de nivel y caudal. *Revista de Ingeniería y Región*, 43-54.

Faiber Ignacio Robayo, A. M. (2015). Desarrollo de un controlador basado en redes neuronales para un sistema multivariable de nivel y caudal. *Revista Ingeniería y Región*, 43-54.

Jiang Chang, Y. P. (2009). Neural Network PID Control System for the FTGS. *IEEE Xplore*.

Jiang-Jiang Wang, C.-F. Z.-Y. (2008). Self-adaptive RBF neural network PID control in exhaust temperature of micro gas turbine. *IEEE Xplore*.

José Danilo Rairán, F. J. (2012). Control de procesos por medio de redes neuronales. *SlideShare*, 33.

Jurado, J. G. (2012). *Diseño de sistemas de control multivariable por desacoplo con controladores PID. .* Madrid: Escuela Técnica Superior de Ingeniería Informática, Universidad Nacional de Educación a Distancia.

Montiel, S. E. (2009). *Reconocimiento de voz para un control de acceso mediante red neuronal de retropropagación.* México: INSTITUTO POLITÉCNICO NACIONAL ESCUELA SUPERIOR INGENIERÍA MECÁNICA ELÉCTRICA UNIDAD CALHUACAN.

N. Sáenz Bajo, M. Á. (2002). Redes neuronales: concepto, aplicaciones y utilidad en medicina. *ELSEVIER*, 119-120.

Rubén Darío Cárdenas Espinosa, R. G. (2017). Identificación y control digital con redes neuronales para un sistema hidráulico. *Vector*, 32-39.

Schmid, M. D. (2009). *A neural network package for Octave User's Guide Version: 0.1.9.1.*

Yang, S., & Gong, C. (2019). Application of Fuzzy Neural Network PID Algorithm in Oil Pump Control. *IEEE Xplore*.

13 ANEXOS:

13.1 CÓDIGO DE PROGRAMACIÓN:

13.1.1 MATLAB:

13.1.1.1 SIMULACIÓN:

```
%Archivo para definir los parámetros de la planta y simular el archivo
de
%simulink .mdl
%tren=1; %Validate Data
tren=2; %Train Data
%tren=3; %Val_Train Data
%% Pruebas PID
if tren == 1
    SPNivel=20;
    Te_Bomba =6.5*2;
    tend=Te_Bomba*23;
    SP_Nivel=[0,SPNivel; tend,SPNivel];
%     x=0.1;
%     x=-0.1;
%     TP_Nivel=[0,0; Te_Bomba,0; Te_Bomba,x;...
%         Te_Bomba*2,x;Te_Bomba*2,2*x;...
%         Te_Bomba*3,2*x; Te_Bomba*3,3*x;...
%         Te_Bomba*4,3*x;Te_Bomba*4,4*x;...
%         Te_Bomba*5,4*x; Te_Bomba*5,5*x;...
%         Te_Bomba*6,5*x;Te_Bomba*6,6*x;...
%         Te_Bomba*7,6*x; Te_Bomba*7,7*x;...
%         Te_Bomba*8,7*x;Te_Bomba*8,8*x;...
%         Te_Bomba*9,8*x; Te_Bomba*9,9*x;...
%         Te_Bomba*10,9*x; Te_Bomba*10,10*x;...
%         Te_Bomba*11,10*x; Te_Bomba*11,11*x;...
%         Te_Bomba*12,11*x; Te_Bomba*12,12*x;...
%         Te_Bomba*13,12*x; Te_Bomba*13,13*x;...
%         Te_Bomba*14,13*x; Te_Bomba*14,14*x;...
%         Te_Bomba*15,14*x; Te_Bomba*15,15*x;...
%         Te_Bomba*16,15*x; Te_Bomba*16,16*x;...
%         Te_Bomba*17,16*x; Te_Bomba*17,17*x;...
%         Te_Bomba*18,17*x; Te_Bomba*18,18*x;...
%         Te_Bomba*19,18*x; Te_Bomba*19,19*x;...
%         Te_Bomba*20,19*x; Te_Bomba*20,20*x;...
%         Te_Bomba*21,20*x; Te_Bomba*21,21*x;...
%         Te_Bomba*22,21*x; Te_Bomba*22,22*x;...
%         tend,22*x];
   TP_Nivel=[0,0; Te_Bomba,0; Te_Bomba,1;...
        Te_Bomba*2,1;Te_Bomba*2,-1;...
        Te_Bomba*3,-1; Te_Bomba*3,0;...
        Te_Bomba*4,0;Te_Bomba*4,1;...
        Te_Bomba*5,1; Te_Bomba*5,1.5;...
        Te_Bomba*6,1.5;Te_Bomba*6,2;...
        Te_Bomba*7,2; Te_Bomba*7,3;...
        Te_Bomba*8,3;Te_Bomba*8,2.5;...
        Te_Bomba*9,2.5; Te_Bomba*9,2;...
        Te_Bomba*10,2; Te_Bomba*10,1.5;...
        Te_Bomba*11,1.5; Te_Bomba*11,1;...
        Te_Bomba*12,1; Te_Bomba*12,0;...
        Te_Bomba*13,0; Te_Bomba*13,-1;...
        Te_Bomba*14,-1; Te_Bomba*14,-1.5;...
        Te_Bomba*15,-1.5; Te_Bomba*15,-2;...
        Te_Bomba*16,-2;Te_Bomba*16,-2.5;...
        Te_Bomba*17,-2.5; Te_Bomba*17,-3;...
        Te_Bomba*18,-3; Te_Bomba*18,-2.5;...
```

```
        Te_Bomba*19,-2.5;Te_Bomba*19,-2;...
        Te_Bomba*20,-2; Te_Bomba*20,-1.5;...
        Te_Bomba*21,-1.5; Te_Bomba*21,-1;...
        Te_Bomba*22,-1; Te_Bomba*22,-1;...
        tend,0];

    SPCaudal=5;
    Te_Valvula = 3.7*2;
    if Te_Valvula*23>tend %Si el tiempo de simulación del sp_caudal
        %es más grande que el de nivel, lo ponemos como final de sim
        tend=Te_Valvula*9;
    end
%     y=0.1;
%     y=-0.1;
      SP_Caudal=[0,SPCaudal;tend,SPCaudal];
%     TP_Caudal=[0,0; Te_Valvula,0; Te_Valvula,2*y;...
%        Te_Valvula*2,2*y;Te_Valvula*2,3*y;...
%         Te_Valvula*3,3*y; Te_Valvula*3,4*y;...
%         Te_Valvula*4,4*y; Te_Valvula*4,5*y;...
%         Te_Valvula*5,5*y; Te_Valvula*5,6*y;...
%         Te_Valvula*6,6*y; Te_Valvula*6,7*y;...
%         Te_Valvula*7,7*y; Te_Valvula*7,8*y;...
%         Te_Valvula*8,8*y; Te_Valvula*8,9*y;...
%         Te_Valvula*9,9*y; Te_Valvula*9,10*y;...
%         Te_Valvula*10,10*y; Te_Valvula*10,11*y;...
%         Te_Valvula*11,11*y; Te_Valvula*11,12*y;...
%         Te_Valvula*12,12*y; Te_Valvula*12,13*y;...
%         Te_Valvula*13,13*y; Te_Valvula*13,14*y;...
%         Te_Valvula*14,14*y; Te_Valvula*14,15*y;...
%         Te_Valvula*15,15*y; Te_Valvula*15,16*y;...
%         Te_Valvula*16,16*y; Te_Valvula*16,17*y;...
%         Te_Valvula*17,17*y; Te_Valvula*17,18*y;...
%         Te_Valvula*18,18*y; Te_Valvula*18,19*y;...
%         Te_Valvula*19,19*y; Te_Valvula*19,20*y;...
%         Te_Valvula*20,20*y; Te_Valvula*20,21*y;...
%         Te_Valvula*21,21*y; Te_Valvula*21,22*y;...
%         Te_Valvula*22,22*y; Te_Valvula*22,23*y;...
%         tend,23*y];
    TP_Caudal=[0,0; Te_Valvula,0; Te_Valvula,1;...
       Te_Valvula*2,1;Te_Valvula*2,-1;...
        Te_Valvula*3,-1; Te_Valvula*3,0;...
        Te_Valvula*4,0; Te_Valvula*4,1;...
        Te_Valvula*5,1; Te_Valvula*5,1.5;...
        Te_Valvula*6,1.5; Te_Valvula*6,2;...
        Te_Valvula*7,2; Te_Valvula*7,3;...
        Te_Valvula*8,3; Te_Valvula*8,2.5;...
        Te_Valvula*9,2.5; Te_Valvula*9,2;...
        Te_Valvula*10,2; Te_Valvula*10,1.5;...
        Te_Valvula*11,1.5; Te_Valvula*11,1;...
        Te_Valvula*12,1; Te_Valvula*12,0;...
        Te_Valvula*13,0; Te_Valvula*13,-1;...
        Te_Valvula*14,-1; Te_Valvula*14,-1.5;...
        Te_Valvula*15,-1.5; Te_Valvula*15,-2;...
        Te_Valvula*16,-2; Te_Valvula*16,-2.5;...
        Te_Valvula*17,-2.5; Te_Valvula*17,-3;...
        Te_Valvula*18,-3; Te_Valvula*18,-2.5;...
        Te_Valvula*19,-2.5; Te_Valvula*19,-2;...
        Te_Valvula*20,-2; Te_Valvula*20,-1.5;...
        Te_Valvula*21,-1.5; Te_Valvula*21,-1;...
        Te_Valvula*22,-1; Te_Valvula*22,0;...
        tend,0];
```

```
end
%% Trenes de pulsos
if tren == 2
    %Bomba
    %Si es un pulso de 2T, otro de T, otro de T/2 y otro de T/4,
    %cada pulso necesita 2 periodos mas otro de separacion
    %Te_Bomba =165*2;
    Te_Bomba =30*6.5*2;
    N_Pulsos = 2;
    T_ini = Te_Bomba;
    %Set Point del Nivel
    SPNivel=20;
    Por_Su_Ba=0.02*SPNivel;
    TP_Nivel=zeros(23*N_Pulsos,2);
    Salto_Tren_Nivel=SPNivel*0.25;
    x=0;

    for i=1:N_Pulsos
        for j=1:4
            TP_Nivel(i+(23*(i-1))+(6*(j-1)),:)= [T_ini+x, 0];
            TP_Nivel(i+(23*(i-1))+(6*(j-1))+1,:)= [T_ini+x,
Salto_Tren_Nivel];

            TP_Nivel(i+(23*(i-1))+(6*(j-1))+2,:)=
[T_ini+x+(Te_Bomba/(2^(j-1))), Salto_Tren_Nivel];
            TP_Nivel(i+(23*(i-1))+(6*(j-1))+3,:)=
[T_ini+x+(Te_Bomba/(2^(j-1))), -Salto_Tren_Nivel];

            TP_Nivel(i+(23*(i-1))+(6*(j-1))+4,:)=
[T_ini+x+(Te_Bomba/(2^(j-1)))*2, -Salto_Tren_Nivel];
            TP_Nivel(i+(23*(i-1))+(6*(j-1))+5,:)=
[T_ini+x+(Te_Bomba/(2^(j-1)))*2, 0];

            x=TP_Nivel(i+(23*(i-1))+(6*(j-1))+5,1);
        end
        x=x+Te_Bomba*2;
    end
    tt=Te_Bomba*2;
    tend=x;%igualamos el tiempo de simulación al máximo tiempo de SP
de nivel
    SP_Nivel=[0,SPNivel;tend,SPNivel];

%SP_Nivel=[0,SPNivel;TP_Nivel(96,1)+tt,SPNivel;TP_Nivel(96,1)+tt,SPNiv
el+Por_Su_Ba;...

%TP_Nivel(192,1)+tt,SPNivel+Por_Su_Ba;TP_Nivel(192,1)+tt,SPNivel-
Por_Su_Ba;...
        %tend,SPNivel-Por_Su_Ba];
    %Válvula
    Te_Valvula = 3.7*2;
    N_Pulsos = 2;
    T_ini = Te_Valvula;
    x=0;
    %Set Point de la Válvula
    SPCaudal=2;
    Por_Su_Ba=0.02*SPCaudal;
    TP_Caudal=zeros(23*N_Pulsos,2);
    Salto_Tren_Caudal=0.25*SPCaudal;
```

```
    for i=1:N_Pulsos
        for j=1:4
            TP_Caudal(i+(23*(i-1))+(6*(j-1)),:)= [T_ini+x, 0];
            TP_Caudal(i+(23*(i-1))+(6*(j-1))+1,:)= [T_ini+x,
Salto_Tren_Caudal];

            TP_Caudal(i+(23*(i-1))+(6*(j-1))+2,:)=
[T_ini+x+(Te_Valvula/(2^(j-1))), Salto_Tren_Caudal];
            TP_Caudal(i+(23*(i-1))+(6*(j-1))+3,:)=
[T_ini+x+(Te_Valvula/(2^(j-1))), -Salto_Tren_Caudal];

            TP_Caudal(i+(23*(i-1))+(6*(j-1))+4,:)=
[T_ini+x+(Te_Valvula/(2^(j-1)))*2, -Salto_Tren_Caudal];
            TP_Caudal(i+(23*(i-1))+(6*(j-1))+5,:)=
[T_ini+x+(Te_Valvula/(2^(j-1)))*2, 0];

            x=TP_Caudal(i+(23*(i-1))+(6*(j-1))+5,1);
        end
        x=x+Te_Valvula*2;
    end
    tt=Te_Valvula*2;
    SP_Caudal=[0,SPCaudal;tend,SPCaudal];
%
SP_Caudal=[0,SPCaudal;TP_Caudal(96,1)+tt,SPCaudal;TP_Caudal(96,1)+tt,S
PCaudal+Por_Su_Ba;...
%
TP_Caudal(192,1)+tt,SPCaudal+Por_Su_Ba;TP_Caudal(192,1)+tt,SPCaudal-
Por_Su_Ba;...
%         tend,SPCaudal-Por_Su_Ba];
    if x>tend %Si el tiempo de simulación del sp_caudal es más grande
que el de nivel, lo ponemos como final de sim
        tend=x;
    end
end
%% Data for validate when training
if tren==3
    %Bomba
    %Si es un pulso de 2T, otro de T, otro de T/2 y otro de T/4, cada
pulso necesita 2 periodos mas otro de separacion
    Te_Bomba = 6.5*2;
    N_Pulsos = 12;
    T_ini = Te_Bomba;
    %Set Point del Nivel
    SPNivel=20;
    Por_Su_Ba=0.02*SPNivel;
    TP_Nivel=zeros(23*N_Pulsos,2);
    Salto_Tren_Nivel=SPNivel*0.25;
    x=0;
    ini=500;
    for i=1:N_Pulsos
        for j=1:4
            TP_Nivel(i+(23*(i-1))+(6*(j-1)),:)= [T_ini+x+ini, 0];
            TP_Nivel(i+(23*(i-1))+(6*(j-1))+1,:)= [T_ini+x+ini,
Salto_Tren_Nivel];

            TP_Nivel(i+(23*(i-1))+(6*(j-1))+2,:)=
[T_ini+x+ini+(Te_Bomba/(2^(j-1))), Salto_Tren_Nivel];
            TP_Nivel(i+(23*(i-1))+(6*(j-1))+3,:)=
[T_ini+x+ini+(Te_Bomba/(2^(j-1))), -Salto_Tren_Nivel];
```

```
            TP_Nivel(i+(23*(i-1))+(6*(j-1))+4,:)=
[T_ini+x+ini+(Te_Bomba/(2^(j-1)))*2, -Salto_Tren_Nivel];
            TP_Nivel(i+(23*(i-1))+(6*(j-1))+5,:)=
[T_ini+x+ini+(Te_Bomba/(2^(j-1)))*2, 0];

            x=TP_Nivel(i+(23*(i-1))+(6*(j-1))+5,1);
            ini=0;
        end
        x=x+Te_Bomba*2;
    end
    tt=Te_Bomba*2;
    tend=x;%igualamos el tiempo de simulación al máximo tiempo de SP
de nivel

SP_Nivel=[0,SPNivel;TP_Nivel(96,1)+tt,SPNivel;TP_Nivel(96,1)+tt,SPNive
l+Por_Su_Ba;...

TP_Nivel(192,1)+tt,SPNivel+Por_Su_Ba;TP_Nivel(192,1)+tt,SPNivel-
Por_Su_Ba;...
        tend,SPNivel-Por_Su_Ba];
    %Válvula
    Te_Valvula = 3.7*2;
    N_Pulsos = 12;
    T_ini = Te_Valvula;
    x=0;
    %Set Point de la Válvula
    SPCaudal=5;
    Por_Su_Ba=0.02*SPCaudal;
    TP_Caudal=zeros(23*N_Pulsos,2);
    Salto_Tren_Caudal=0.25*SPCaudal;
    ini=300;
    for i=1:N_Pulsos
        for j=1:4
            TP_Caudal(i+(23*(i-1))+(6*(j-1)),:)= [T_ini+x+ini, 0];
            TP_Caudal(i+(23*(i-1))+(6*(j-1))+1,:)= [T_ini+x+ini,
Salto_Tren_Caudal];

            TP_Caudal(i+(23*(i-1))+(6*(j-1))+2,:)=
[T_ini+x+ini+(Te_Valvula/(2^(j-1))), Salto_Tren_Caudal];
            TP_Caudal(i+(23*(i-1))+(6*(j-1))+3,:)=
[T_ini+x+ini+(Te_Valvula/(2^(j-1))), -Salto_Tren_Caudal];

            TP_Caudal(i+(23*(i-1))+(6*(j-1))+4,:)=
[T_ini+x+ini+(Te_Valvula/(2^(j-1)))*2, -Salto_Tren_Caudal];
            TP_Caudal(i+(23*(i-1))+(6*(j-1))+5,:)=
[T_ini+x+ini+(Te_Valvula/(2^(j-1)))*2, 0];

            x=TP_Caudal(i+(23*(i-1))+(6*(j-1))+5,1);
            ini=0;
        end
        x=x+Te_Valvula*2;
    end
    tt=Te_Valvula*2;

SP_Caudal=[0,SPCaudal;TP_Caudal(96,1)+tt,SPCaudal;TP_Caudal(96,1)+tt,S
PCaudal+Por_Su_Ba;...

TP_Caudal(192,1)+tt,SPCaudal+Por_Su_Ba;TP_Caudal(192,1)+tt,SPCaudal-
Por_Su_Ba;...
        tend,SPCaudal-Por_Su_Ba];
```

```
    if x>tend %Si el tiempo de simulación del sp_caudal es más grande
que el de nivel, lo ponemos como final de sim
        tend=x;
    end
end
%% PARÁMETROS
%Funciones de transferencia
Num_V_Bomba = 0.0002823;
Den_V_Bomba = [1 0.04295 0.0002129];
Num_V_Valvula = 0.1452;
Den_V_Valvula = [1 0.2127];
%Temporales
ts=0.01;
%Parámetros del PID de la Bomba/Nivel
PB=250;
IB=2;
DB=5800;
NB=1;
PB_PID=[0,PB;tend,PB];
IB_PID=[0,IB;tend,IB];
DB_PID=[0,DB;tend,DB];
NB_PID=[0,NB;tend,NB];
Num_Cons_PID_Nivel_En=1;
Cons_PID_Nivel_En=[0,Num_Cons_PID_Nivel_En;tend,Num_Cons_PID_Nivel_En]
;
Num_Cons_PID_Nivel_Sa=1;
Cons_PID_Nivel_Sa=[0,Num_Cons_PID_Nivel_Sa;tend,Num_Cons_PID_Nivel_Sa]
;
%Parámetros del PID de la Válvula/Caudal
Num_Cons_PID_Caudal=1;
Cons_PID_Caudal=[0,Num_Cons_PID_Caudal;tend,Num_Cons_PID_Caudal];
PV=7;%PV=10;
IV=2;
DV=0;
NV=0;
PV_PID=[0,PV;tend,PV];
IV_PID=[0,IV;tend,IV];
DV_PID=[0,DV;tend,DV];
NV_PID=[0,NV;tend,NV];
Sim='cerrado';
%% SIMULACIÓN MODELO LAZO CERRADO
if Sim=='cerrado'
    vt = 0:ts:tend;
    [t, estados, resultados_cer]=sim('modelo_planta',vt);
    %Exportar datos a excel, según estemos utilizando un patrón de
tren de
    %pulsos u otro
    if tren==2
        xlswrite('Train_Data_Lim.xlsx',resultados_cer);
    end
    if tren==1
        resultados_cer(:,9)=t;
        xlswrite('Validate_Data_Lim_35_5.xlsx',resultados_cer);
        %xlswrite('Validate_Data_Asc.xlsx',resultados_cer);
        %xlswrite('Validate_Data_Desc.xlsx',resultados_cer);
    end
    if tren==3
        xlswrite('Val_train_Data.xlsx',resultados_cer);
    end
end
%% SIMULACIÓN MODELO LAZO ABIERTO
```

```
if Sim=='abierto'
    vt = 0:ts:tend;
    [t, resultados_abi]=sim('modelo_planta_lazo_abierto',vt);
    xlswrite('Res_Abierto.xlsx',resultados_abi);
end
```

13.1.2 OCTAVE:

13.1.2.1 NN:

```
% Neural network %
%%Cargamos paquetes
%pkg load nnet
%pkg load io
clear;
%% Definición de los parámetros de la red neuronal
%Estructura:
L=1;%Layers
TF_1= "tansig";
TF_2= "tansig";
TF_3= "tansig";
BTF = "trainlm";%Backpropagation network trainig
BLF = "not Used";%Only for matlab compatibility
PF = "mse";%Peformance function

%%Cargar los datos de entrada
In_D_NN_Col_M1=xlsread('Train_Data_Lim_SM.xlsx');
%Quitamos valores iniciales
[m,n]=size(In_D_NN_Col_M1);
Val_quitados_por=5;%En porcentaje
Val_quitados=round(Val_quitados_por*(1/100)*m);
In_D_NN_Col_M=zeros(m-Val_quitados,n);
In_D_NN_Col_M=In_D_NN_Col_M1(Val_quitados:m,:);

%Calculamos el máximo y el mínimo para poder interpolar
 Red=1;
 %1 Red neuronal para el control de nivel con 1 entrada
 %2 Red neuronal para el control de caudal con 1 entrada
 %3 Red neuronal para el control de nivel con 2 entradas
 %4 Red neuronal para el control de caudal con 2 entradas
 [m,n]=size(In_D_NN_Col_M);
 switch Red
   case 1
    In_D_NN_Col=zeros(m,2);
    In_D_NN_Col(:,1)=In_D_NN_Col_M(:,1);
    In_D_NN_Col(:,2)=In_D_NN_Col_M(:,2);
    N_L1=1;N_L2=13;N_L3=1;
   case 2
    In_D_NN_Col=zeros(m,2);
    In_D_NN_Col(:,1)=In_D_NN_Col_M(:,3);
    In_D_NN_Col(:,2)=In_D_NN_Col_M(:,4);
    N_L1=1;N_L2=13;N_L3=1;
   case 3
    In_D_NN_Col=zeros(m,3);
    In_D_NN_Col(:,1)=In_D_NN_Col_M(:,1);
    In_D_NN_Col(:,2)=In_D_NN_Col_M(:,3);
    In_D_NN_Col(:,3)=In_D_NN_Col_M(:,2);
    N_L1=2;N_L2=13;N_L3=1;
   case 4
    In_D_NN_Col=zeros(m,3);
    In_D_NN_Col(:,1)=In_D_NN_Col_M(:,3);
    In_D_NN_Col(:,2)=In_D_NN_Col_M(:,1);
    In_D_NN_Col(:,3)=In_D_NN_Col_M(:,4);
    N_L1=2;N_L2=13;N_L3=1;
 endswitch
%Reordenamos las columnas en filas
[m,n]=size(In_D_NN_Col);
In_D_NN=zeros(n,m);
```

```
for i=1:n
  In_D_NN(i,:)=In_D_NN_Col(:,i);
end
%%Normalizamos los datos de entrada
[In_D,Mean_In_D,STD_In_D]=prestd(In_D_NN);
%Divimos y asignamos los datos a las variables:
Fila_train=round(m/6);
%1/2 for train, 1/3 for test and 1/6 for validation data
switch Red
   case 1
      In_D_Train_N=In_D(1,1:Fila_train*5);%Input data for training
      Tar_D_Train_N=In_D(2,1:Fila_train*5);%Target data for training
      VV.P=In_D(1,Fila_train*5+1:m);%Validation data for training
      VV.T=In_D(2,Fila_train*5+1:m);%Validation data for training
      %Estandarice the data
      %Val_Min_Max_DNN=min_max(In_D(1,1:Fila_train*5));
      min=1000000;max=-1000000;
      for i=1:Fila_train*5
        if In_D(i)>max
          max=In_D(i);
        end
        if In_D(i)<min
          min=In_D(i);
        end
      end
      Val_Min_Max_DNN=[min,max];
   case 2
      In_D_Train_N=In_D(1,1:Fila_train*5);%Input data for training
      Tar_D_Train_N=In_D(2,1:Fila_train*5);%Target data for training
      VV.P=In_D(1,Fila_train*5+1:m);%Validation data for training
      VV.T=In_D(2,Fila_train*5+1:m);%Validation data for training
      %Estandarice the data
      %Val_Min_Max_DNN=min_max(In_D(1,1:Fila_train*5));
      min=1000000;max=-1000000;
      for i=1:Fila_train*5
        if In_D(i)>max
          max=In_D(i);
        end
        if In_D(i)<min
          min=In_D(i);
        end
      end
      Val_Min_Max_DNN=[min,max];
   case 3
      In_D_Train_N=In_D(1:2,1:Fila_train*5);%Input data for training
      Tar_D_Train_N=In_D(3,1:Fila_train*5);%Target data for training
      VV.P=In_D(1:2,Fila_train*5+1:m);%Validation data for training
      VV.T=In_D(2,Fila_train*5+1:m);%Validation data for training
      %Estandarice the data
      Val_Min_Max_DNN=min_max(In_D(1:2,1:Fila_train*5));
   case 4
      In_D_Train_N=In_D(1:2,1:Fila_train*5);%Input data for training
      Tar_D_Train_N=In_D(3,1:Fila_train*5);%Target data for training
      VV.P=In_D(1:2,Fila_train*5+1:m);%Validation data for training
      VV.T=In_D(2,Fila_train*5+1:m);%Validation data for training
      %Estandarice the data
      Val_Min_Max_DNN=min_max(In_D(1:2,1:Fila_train*5));
 endswitch
%%RED NEURONAL
%Creación de la red neuronal
```

```
net_nt=newff(Val_Min_Max_DNN,[N_L1,N_L2,N_L3],{TF_1,TF_2,TF_3},BTF,BLF
,PF);
%Train the neural network
      %Parámetros iniciales
      %net_nt.trainParam.epochs = 100;
      %net_nt.trainParam.goal = 0;
      %net_nt.trainParam.max_fail = 5;
      %net_nt.trainParam.mem_reduc = 1;
      %net_nt.trainParam.min_grad = 1.0000e-4;
      %net_nt.trainParam.mu = 0.0010;
      %net_nt.trainParam.mu_dec = 0.1;
      %net_nt.trainParam.mu_inc = 10;
      %net_nt.trainParam.mu_max = 1.0000e+010;
      %net_nt.trainParam.show = 50;
      %net_nt.trainParam.time = Inf;
switch Red
  case 1
      net_nt.trainParam.epochs = 120;
      net_nt.trainParam.goal = 0.3;
      net_nt.trainParam.max_fail = 5;
      net_nt.trainParam.mem_reduc = 1;
      net_nt.trainParam.min_grad = 1.0000e-5;
      net_nt.trainParam.mu = 0.0010;
      net_nt.trainParam.mu_dec = 0.1;
      net_nt.trainParam.mu_inc = 10;
      net_nt.trainParam.mu_max = 1.0000e+010;
      net_nt.trainParam.show = 50;
      net_nt.trainParam.time = Inf;

Net_1_Quitando_5=train(net_nt,In_D_Train_N,Tar_D_Train_N,[],[],VV);
      %Save the neural network
      %TRAINLM,1_3 Epoch 200/200, MSE 0.576451/0.4, Gradient
618.343/0.0001
      save Net_1_Quitando_5;
  case 2
      net_nt.trainParam.epochs = 100;
      net_nt.trainParam.goal = 0;
      net_nt.trainParam.max_fail = 100;
      net_nt.trainParam.mem_reduc = 1;
      net_nt.trainParam.min_grad = 1.0000e-4;
      net_nt.trainParam.mu = 0.0010;
      net_nt.trainParam.mu_dec = 0.1;
      net_nt.trainParam.mu_inc = 10;
      net_nt.trainParam.mu_max = 1.0000e+010;
      net_nt.trainParam.show = 50;
      net_nt.trainParam.time = Inf;

Net_2_Quitando_15=train(net_nt,In_D_Train_N,Tar_D_Train_N,[],[],VV);
      %Save the neural network
      %TRAINLM, Net_2_3 Epoch 100/100, MSE 0.54139/0, Gradient
173.754/0.0001
      save Net_2_Quitando_15;
      case 3
      net_nt.trainParam.epochs = 100;
      net_nt.trainParam.goal = 0;
      net_nt.trainParam.max_fail = 5;
      net_nt.trainParam.mem_reduc = 1;
      net_nt.trainParam.min_grad = 1.0000e-4;
      net_nt.trainParam.mu = 0.0010;
      net_nt.trainParam.mu_dec = 0.1;
      net_nt.trainParam.mu_inc = 10;
```

```
      net_nt.trainParam.mu_max = 1.0000e+010;
      net_nt.trainParam.show = 50;
      net_nt.trainParam.time = Inf;

Net_3_Quitando_15=train(net_nt,In_D_Train_N,Tar_D_Train_N,[],[],VV);
      %Save the neural network
      %TRAINLM, Epoch 6/100, MSE 0.616448/0.65, Gradient 9983.82/1e-10
      save Net_3_Quitando_15;
  case 4
      net_nt.trainParam.epochs = 100;
      net_nt.trainParam.goal = 0;
      net_nt.trainParam.max_fail = 100;
      net_nt.trainParam.mem_reduc = 1;
      net_nt.trainParam.min_grad = 1.0000e-4;
      net_nt.trainParam.mu = 0.0010;
      net_nt.trainParam.mu_dec = 0.1;
      net_nt.trainParam.mu_inc = 10;
      net_nt.trainParam.mu_max = 1.0000e+010;
      net_nt.trainParam.show = 50;
      net_nt.trainParam.time = Inf;

Net_4_Quitando_15=train(net_nt,In_D_Train_N,Tar_D_Train_N,[],[],VV);
      %Save the neural network
      %Net_4_2 0.596
      save Net_4_Quitando_15;
endswitch
```

13.1.2.2 OUTPUT_NN

```
%Este script cargará los datos de validación, obtendrá
%la salida de las redes neuronales previamente creadas
%y comparará los datos.
%%Cargamos paquetes
%pkg load nnet
%pkg load io
%%Cargamos las redes neuronales y se lo asignamos a los nombres que
hemos utilizado
%de redes
%%Cargamos los datos de validación
Val_Data_Col_MR=xlsread('Validate_Data');
%Le quitamos los primeros 5000 valores para no tener errores
%iniciales
[m,n]=size(Val_Data_Col_MR);
Val_Data_Col_M=zeros(m-5000,n);
Val_Data_Col_M=Val_Data_Col_MR(5000:m,:);
for i=1:4
  Red=i;
  switch Red
   case 1
   figure(1)
    load Net_1_1;
    net_1=Net_1_1;
    [m,n]=size(Val_Data_Col_M);
    In_D_NN_Col=zeros(m,2);
    In_D_NN_Col(:,1)=Val_Data_Col_M(:,1);
    In_D_NN_Col(:,2)=Val_Data_Col_M(:,2);
   case 2
    figure(2)
    load Net_2_1;
    net_2=Net_2_1;
    [m,n]=size(Val_Data_Col_M);
    In_D_NN_Col=zeros(m,2);
    In_D_NN_Col(:,1)=Val_Data_Col_M(:,3);
    In_D_NN_Col(:,2)=Val_Data_Col_M(:,4);
   case 3
    figure(3)
    load Net_3_1;
    net_3=Net_3_1;
    [m,n]=size(Val_Data_Col_M);
    In_D_NN_Col=zeros(m,3);
    In_D_NN_Col(:,1)=Val_Data_Col_M(:,1);
    In_D_NN_Col(:,2)=Val_Data_Col_M(:,3);
    In_D_NN_Col(:,3)=Val_Data_Col_M(:,2);
   case 4
    figure(4)
    load Net_4_1;
    net_4=Net_4_1;
    [m,n]=size(Val_Data_Col_M);
    In_D_NN_Col=zeros(m,3);
    In_D_NN_Col(:,1)=Val_Data_Col_M(:,3);
    In_D_NN_Col(:,2)=Val_Data_Col_M(:,1);
    In_D_NN_Col(:,3)=Val_Data_Col_M(:,4);
  endswitch
    %Reordenamos las columnas en filas
  [m,n]=size(In_D_NN_Col);
  In_D_NN=zeros(n,m);
  for i=1:n
    In_D_NN(i,:)=In_D_NN_Col(:,i);
  end
```

```
%%Normalizamos los datos de entrada
  [In_D_Sim_N,Mean_In_D, STD_In_D]=prestd(In_D_NN);
%%Sacamos la salida de cada entrada y las
%%guardamos en variables diferentes para poder
%%compararlas
  switch Red
   case 1
       Out_D_N_1=sim(net_1,In_D_Sim_N(1,:));
       Val_N_1=Check_NN(In_D_Sim_N(2,:),Out_D_N_1)
       [m,n]=size(In_D_Sim_N(2,:));
       Data=zeros(2,n);
       Data(1,:)=In_D_Sim_N(2,:);
       Data(2,:)=Out_D_N_1;
       Ejex=1:n;
       figure(1);
       plot(Ejex,Data);
   case 2
       Out_D_N_2=sim(net_2,In_D_Sim_N(1,:));
       Val_N_2=Check_NN(In_D_Sim_N(2,:),Out_D_N_1)
       [m,n]=size(In_D_Sim_N(2,:));
       Data=zeros(2,n);
       Data(1,:)=In_D_Sim_N(2,:);
       Data(2,:)=Out_D_N_1;
       Ejex=1:n;
       figure(2);
       plot(Ejex,Data);
   case 3
       [m,n]=size(In_D_NN_Col);
       In_D2_Sim_N=zeros(2,m);
       In_D2_Sim_N(1,:)=In_D_Sim_N(1,:);
       In_D2_Sim_N(2,:)=In_D_Sim_N(3,:);
       Out_D_N_3=sim(net_3,In_D2_Sim_N);
       Val_N_3=Check_NN(In_D_Sim_N(3,:),Out_D_N_1)
       [m,n]=size(In_D_Sim_N(3,:));
       Data=zeros(2,n);
       Data(1,:)=In_D_Sim_N(3,:);
       Data(2,:)=Out_D_N_1;
       Ejex=1:n;
       figure(3);
       plot(Ejex,Data);
   case 4
       [m,n]=size(In_D_NN_Col);
       In_D2_Sim_N=zeros(2,m);
       In_D2_Sim_N(1,:)=In_D_Sim_N(1,:);
       In_D2_Sim_N(2,:)=In_D_Sim_N(3,:);
       Out_D_N_4=sim(net_4,In_D2_Sim_N);
       Val_N_4=Check_NN(In_D_Sim_N(3,:),Out_D_N_1)
       [m,n]=size(In_D_Sim_N(3,:));
       Data=zeros(2,n);
       Data(1,:)=In_D_Sim_N(3,:);
       Data(2,:)=Out_D_N_1;
       Ejex=1:n;
       figure(4);
       plot(Ejex,Data);
  endswitch
endfor
```

13.1.2.3 RED 1 (NIVEL CON 1 ENTRADA):

```
%Cargamos los paquetes necesarios
%pkg load nnet
%pkg load io
%pkg load control
cargar_redes=1;
if cargar_redes==1
clear
      load Net_1_Con_Nor;
        Red_1=Net_1_Con_Nor;
      load Net_1_Sin_Nor
        Red_2=Net_1_Sin_Nor;
endif
for j=1:2
  clear Net_1;
  %Cargamos que red neuronal queremos simular
  switch j
    case 1
    Net_1=Red_1;
    case 2
    Net_1=Red_2;
  endswitch
  %Le asignamos la red a la variable Net_1 que es
  %la que se utiliza en todo el código.
  %Cargamos medias y desviaciones estandar
  Mean1=0.0053164;Mean2=19.7704273;Mean3=0.0024142;Mean4=7.2803693;
  STD1=2.08929;STD2=10203.65151;STD3=0.36044;STD4=2.65699;
  %Definimos dos funciones de transferencia
  s=tf('s');
  G11=tf([0.0002823],[1 0.04295 0.0002129],0.001);
  G22=tf([0.1452],[1 0.2127],0.001);
  %Cargamos el vector de referencias
  t=0;
  x=0.25;
  y1=0.5;
  y2=-1;
  temp=300;
  A=zeros(7*temp,1);
  A(1:temp,1)=t;
  A(temp+1:2*temp,1)=y1;
  A(2*temp+1:3*temp,1)=y1*2;
  A(3*temp+1:4*temp,1)=y1;
  A(4*temp+1:5*temp,1)=0;
  A(5*temp+1:6*temp,1)=-y1;
  A(6*temp+1:7*temp,1)=-y1*2;
  %Refsize_Data_1=xlsread('Validate_Data');
  %Refsize_Data_2=xlsread('Validate_Data_Lim_SM');
  %Refsize_Data_1=Refsize_Data_2(:,5);
  Refsize_Data_1=A;
  [m,n]=size(Refsize_Data_1);
  Val_quitados=0001;
  Refsize_Data=zeros(m-Val_quitados,n);
  Refsize_Data=Refsize_Data_1(Val_quitados:m,:);
  %Creamos el vector de tren de pulsos.
  Ref_Data=Refsize_Data;
  [m,n]=size(Ref_Data); %Tiempos de simulación
  %Creamos vectores
  Ent_Sist=zeros(m,1);
  Ent_Sist(1)=0;%Inicializamos el vector de
  %entradas ya que en el instante 1 no hay
  Sal_Nivel=zeros(m,1);
```

```
  Sal_Nivel_Act=zeros(m,1);
  Error_1=zeros(m,1);
  Error_1_NN=zeros(m,1);
  Sal_NN_Nivel_NN=zeros(m,1);
  Sal_NN_Nivel_NN_1=zeros(m,1);
  Sal_NN_Nivel=zeros(m,1);
  %Data=zeros(m,5);
  offset=0;
   for i=1:m
      %Tenemos que distinguir si es el momento 1
      %de simulación o si es continuo
      [a,b]=step(G11*Ent_Sist(i),0.01); %Calculamos el step
      Sal_Nivel_Act(i)=a(end); %De todos los valores del step
      %Cogemos el valor del último instante, el de 0.01 segundos
      offset_asc=13.7;
      offset_desc=0;
      if i==1 %Si es el primer instante no
          if Ref_Data(1)>0
            offset=offset_asc;
          elseif Ref_Data(1)<0
            offset=offset_desc;
          endif
      else %Si es otro instante, calculamos si estamos subiendo o
bajando
      %para aplicar un offset
         if Ref_Data(i)>Ref_Data(i-1)
            offset=offset_asc;
         elseif Ref_Data(i)<Ref_Data(i-1)
            offset_desc=offset_desc;
         endif
         Sal_Nivel(i)=Sal_Nivel(i-1)+Sal_Nivel_Act(i); %Sumamos cual
ha sido la
          %evolución al instante anterior, a como esta la señal en el
instante anterior
      endif
      Error_1_NN(i)=Ref_Data(i)-Sal_Nivel(i); %Calculamos el error
      Error_1(i)=(Error_1_NN(i)-Mean1)/STD1; %Lo normalizamos
      %Error_1(i)=Error_1_NN(i); %Hemos bypaseado la normalización por
que
        %funciona mejor sin ella
      Sal_NN_Nivel_NN(i)=sim(Net_1,Error_1(i)); %Calculamos la
simulación de la
      %entrada del error

Sal_NN_Nivel_NN_1(i)=(Sal_NN_Nivel_NN(i)*STD3)+Mean3;%Normalizamos
      %la salida
      %Sal_NN_Nivel_NN_1(i)=Sal_NN_Nivel_NN(i); %Bypaseamos la
normalización
      %de la salida por que funciona mejor sin.
      if Error_1_NN(i)==0 %Si el error es 0 la señal de la red tiene
que ser 0.
        Sal_NN_Nivel(i)=0;
      else %En caso contrario elevamos la salida de la red y le
añadimos un offset
        Sal_NN_Nivel(i)=(Sal_NN_Nivel_NN_1(i)*100+offset);
      endif
      if i==m
      else %traspasamos el valor de la señal de la red a la variable
entrada del
      %sistema para empezar la siguiente iteración
          Ent_Sist(i+1)=Sal_NN_Nivel(i);
```

```
      endif
   endfor
   %Creamos una matriz para comparar los datos
   M_Com=zeros(m,7);
   M_Com(:,1)=Sal_Nivel_Act;
   M_Com(:,2)=Sal_Nivel;
   M_Com(:,3)=Ref_Data(:,1);
   M_Com(:,4)=Error_1_NN;
   M_Com(:,5)=Error_1;
   M_Com(:,6)=Sal_NN_Nivel_NN;
   M_Com(:,7)=Sal_NN_Nivel_NN_1;
   M_Com(:,8)=Sal_NN_Nivel;
   M_Com(:,9)=Ent_Sist;
   M_Com
   Data(:,j+1)=Sal_Nivel(:,1)
   switch j
      case 1
        Sal_Nivel_Con_Nor_1=Sal_Nivel;
        %Save Sal_Nivel_Sin_Quitar_2;
      case 2
        Sal_Nivel_Sin_Nor_1=Sal_Nivel;
        %Save Sal_Nivel_Quitando_5_2
   endswitch
endfor
Data(:,1)=Ref_Data(:,1);
Ejex=1:m;
figure(1);
clf(1);
plot(Ejex,Data);
legend("Ref Data","Sal Con Nor","Sal Sin Nor");
```

13.1.2.4 RED 2 (CAUDAL CON 1 ENTRADA):

```
%Cargamos los paquetes necesarios
%pkg load nnet
%pkg load io
%pkg load control
cargar_redes=1;
if cargar_redes==1
clear
      load Net_2_Var_Ref;
        Net_1=Net_2_Var_Ref
endif
  %Cargamos que red neuronal queremos simular
  %Le asignamos la red a la variable Net_1 que es
  %la que se utiliza en todo el código.
  %Cargamos medias y desviaciones estandar
  Mean1=0.0053164;Mean2=19.7704273;Mean3=0.0024142;Mean4=7.2803693;
  STD1=2.08929;STD2=10203.65151;STD3=0.36044;STD4=2.65699;
  %Definimos dos funciones de transferencia
  s=tf('s');
  G11=tf([0.0002823],[1 0.04295 0.0002129],0.001);
  G22=tf([0.1452],[1 0.2127],0.001);
  %Cargamos el vector de referencias
  t=0;
  x=0.25;
  y1=0.5;
  y2=-1;
  temp=150;
  A=zeros(4*temp,1);
  A(1:temp,1)=t;
  A(temp+1:2*temp,1)=y1;
  A(2*temp+1:3*temp,1)=y1*2;
```

```
  A(3*temp+1:4*temp,1)=y1;
  %A(4*temp+1:5*temp,1)=0;
  %A(5*temp+1:6*temp,1)=-y1;
  %A(6*temp+1:7*temp,1)=-y1*2;
  %Refsize_Data_1=xlsread('Validate_Data');
  %Refsize_Data_2=xlsread('Validate_Data_Lim_SM');
  %Refsize_Data_1=Refsize_Data_2(:,5);
  Refsize_Data_1=A;
  [m,n]=size(Refsize_Data_1);
  Val_quitados=0001;
  Refsize_Data=zeros(m-Val_quitados,n);
  Refsize_Data=Refsize_Data_1(Val_quitados:m,:);
  %Creamos el vector de tren de pulsos.
  Ref_Data=Refsize_Data;
  [m,n]=size(Ref_Data); %Tiempos de simulación
  %Creamos vectores
  Ent_Sist=zeros(m,1);
  Ent_Sist(1)=0;%Inicializamos el vector de
  %entradas ya que en el instante 1 no hay
  Sal_Nivel=zeros(m,1);
  Sal_Nivel_Act=zeros(m,1);
  Error_1=zeros(m,1);
  Error_1_NN=zeros(m,1);
  Sal_NN_Nivel_NN=zeros(m,1);
  Sal_NN_Nivel_NN_1=zeros(m,1);
  Sal_NN_Nivel=zeros(m,1);
  %Data=zeros(m,5);
  offset=0;
   for i=1:m
      %Tenemos que distinguir si es el momento 1
      %de simulación o si es continuo
      [a,b]=step(G22*Ent_Sist(i),0.01); %Calculamos el step
      Sal_Nivel_Act(i)=a(end); %De todos los valores del step
      %Cogemos el valor del último instante, el de 0.01 segundos
      offset_asc=0.02;
      offset_desc=0.01;
      if i==1 %Si es el primer instante no
          if Ref_Data(1)>0
            offset=offset_asc;
          elseif Ref_Data(1)<0
            offset=offset_desc;
          endif
      else %Si es otro instante, calculamos si estamos subiendo o
bajando
      %para aplicar un offset
         if Ref_Data(i)>Ref_Data(i-1)
            offset=offset_asc;
         elseif Ref_Data(i)<Ref_Data(i-1)
            offset=offset_desc;
         endif
         Sal_Nivel(i)=Sal_Nivel(i-1)+Sal_Nivel_Act(i); %Sumamos cual
ha sido la
          %evolución al instante anterior, a como esta la señal en el
instante anterior
      endif
      Error_1_NN(i)=Ref_Data(i)-Sal_Nivel(i); %Calculamos el error
      Error_1(i)=(Error_1_NN(i)-Mean2)/STD2; %Lo normalizamos
      Error_1(i)=Error_1_NN(i); %Hemos bypaseado la normalización por
que
        %funciona mejor sin ella
```

```
      Sal_NN_Nivel_NN(i)=sim(Net_1,Error_1(i)); %Calculamos la
simulación de la
      %entrada del error

Sal_NN_Nivel_NN_1(i)=(Sal_NN_Nivel_NN(i)*STD4)+Mean4;%Normalizamos
      %la salida
      Sal_NN_Nivel_NN_1(i)=Sal_NN_Nivel_NN(i); %Bypaseamos la
normalización
      %de la salida por que funciona mejor sin.
      if Error_1_NN(i)==0 %Si el error es 0 la señal de la red tiene
que ser 0.
        Sal_NN_Nivel(i)=0;
      else %En caso contrario elevamos la salida de la red y le
añadimos un offset
        Sal_NN_Nivel(i)=(Sal_NN_Nivel_NN_1(i)+offset);
      endif
      if i==m
      else %traspasamos el valor de la señal de la red a la variable
entrada del
      %sistema para empezar la siguiente iteración
          Ent_Sist(i+1)=Sal_NN_Nivel(i);
      endif
   endfor
   %Creamos una matriz para comparar los datos
   M_Com=zeros(m,7);
   M_Com(:,1)=Sal_Nivel_Act;
   M_Com(:,2)=Sal_Nivel;
   M_Com(:,3)=Ref_Data(:,1);
   M_Com(:,4)=Error_1_NN;
   M_Com(:,5)=Error_1;
   M_Com(:,6)=Sal_NN_Nivel_NN;
   M_Com(:,7)=Sal_NN_Nivel_NN_1;
   M_Com(:,8)=Sal_NN_Nivel;
   M_Com(:,9)=Ent_Sist;
   M_Com
   Data(:,j+1)=Sal_Nivel(:,1)
endfor
Data(:,1)=Ref_Data(:,1);
Ejex=1:m;
figure(2);
clf(2);
plot(Ejex,Data);
legend("Ref Data","Sal Cau Asc");
```

13.1.2.5 RED 3 (NIVEL CON 2 ENTRADAS):

```
%Quitando_Valores_Ambas
%Cargamos los paquetes necesarios
%pkg load nnet
%pkg load io
%pkg load control
cargar_redes=1;
if cargar_redes==1
clear
      load Net_3_Con_Nor;
        Red_1=Net_3_Con_Nor;
      load Net_3_Sin_Nor
        Red_2=Net_3_Sin_Nor;
      load Net_2_Con_Nor
        Net_5=Net_2_Con_Nor;
endif
for j=1:2
  clear Net_1;
  %Cargamos que red neuronal queremos simular
  switch j
    case 1
    Net_1=Red_1;
    case 2
    Net_1=Red_2;
  endswitch
  %Le asignamos la red a la variable Net_1 que es
  %la que se utiliza en todo el código.
  %Cargamos medias y desviaciones estandar
  Mean1=0.0053164;Mean2=19.7704273;Mean3=0.0024142;Mean4=7.2803693;
  STD1=2.08929;STD2=10203.65151;STD3=0.36044;STD4=2.65699;
  %Definimos dos funciones de transferencia
  s=tf('s');
  G11=tf([0.0002823],[1 0.04295 0.0002129],0.001);
  G22=tf([0.1452],[1 0.2127],0.001);
  %Cargamos el vector de referencias
  t=0;
  x=0.25;
  y1=0.5;
  y2=-1;
  temp=300;
  A=zeros(7*temp,1);
  A(1:temp,1)=t;
  A(temp+1:2*temp,1)=y1;
  A(2*temp+1:3*temp,1)=y1*2;
  A(3*temp+1:4*temp,1)=y1;
  A(4*temp+1:5*temp,1)=0;
  A(5*temp+1:6*temp,1)=-y1;
  A(6*temp+1:7*temp,1)=-y1*2;
  %Refsize_Data_1=xlsread('Validate_Data');
  %Refsize_Data_2=xlsread('Validate_Data_Lim_SM');
  %Refsize_Data_1=Refsize_Data_2(:,5);
  Refsize_Data_1_N=A;
  [m,n]=size(Refsize_Data_1_N);
  Val_quitados=0001;
  Refsize_Data_N=zeros(m-Val_quitados,n);
  Refsize_Data_N=Refsize_Data_1_N(Val_quitados:m,:);
   Refsize_Data_1_C=A;
  [m,n]=size(Refsize_Data_1_C);
  Val_quitados=0001;
  Refsize_Data_C=zeros(m-Val_quitados,n);
  Refsize_Data_C=Refsize_Data_1_C(Val_quitados:m,:);
```

```
  %Creamos el vector de tren de pulsos.
  Ref_Data_N=Refsize_Data_N;Ref_Data_C=Refsize_Data_C;
  [m_N,n_N]=size(Ref_Data_N);[m_C,n_C]=size(Ref_Data_C); %Tiempos de
simulación
  %Creamos vectores
  Ent_Sist_N=zeros(m_N,1);Ent_Sist_C=zeros(m_C,1);
  Ent_Sist_N(1)=0;Ent_Sist_C(1)=0;
  %Inicializamos el vector de
  %entradas ya que en el instante 1 no hay
  Sal_N=zeros(m_N,1);Sal_C=zeros(m_C,1);
  Sal_N_Act=zeros(m_N,1);Sal_C_Act=zeros(m_C,1);
  Error_1_N=zeros(m_N,1);Error_1_C=zeros(m_C,1);
  Error_1_NN_N=zeros(m_N,1);Error_1_NN_C=zeros(m_C,1);
  Sal_NN_N_NN=zeros(m_N,1);Sal_NN_C_NN=zeros(m_C,1);
  Sal_NN_N_NN_1=zeros(m_N,1);Sal_NN_C_NN=zeros(m_C,1);
  Sal_NN_N=zeros(m_N,1);Sal_NN_C=zeros(m_C,1);
  %Data=zeros(m,5);
  offset_n=0;
  offset_c=0;
   for i=1:m_C
      %Tenemos que distinguir si es el momento 1
      %de simulación o si es continuo
      [a_n,b_n]=step(G11*Ent_Sist_N(i),0.01); %Calculamos el step
      Sal_N_Act(i)=a_n(end);
      [a_c,b_c]=step(G22*Ent_Sist_C(i),0.01); %Calculamos el step
      Sal_C_Act(i)=a_c(end); %De todos los valores del step
      %Cogemos el valor del último instante, el de 0.01 segundos
      offset_asc_n=13.7;
      offset_desc_n=0;
      offset_asc_c=0.02;
      offset_desc_c=0.01;
      if i==1 %Si es el primer instante no
          if Ref_Data_N(1)>0
            offset_n=offset_asc_n;
          elseif Ref_Data_N(1)<0
            offset_n=offset_desc_n;
          endif
          if Ref_Data_C(1)>0
            offset_c=offset_asc_c;
          elseif Ref_Data_C(1)<0
            offset_c=offset_desc_c;
          endif
      else %Si es otro instante, calculamos si estamos subiendo o
bajando
      %para aplicar un offset
         if Ref_Data_N(i)>Ref_Data_N(i-1)
            offset_n=offset_asc_n;
         elseif Ref_Data_N(i)<Ref_Data_N(i-1)
            offset_n=offset_desc_n;
         endif
         Sal_N(i)=Sal_N(i-1)+Sal_N_Act(i); %Sumamos cual ha sido la
          %evolución al instante anterior, a como esta la señal en el
instante anterior
      if Ref_Data_C(i)>Ref_Data_C(i-1)
            offset_c=offset_asc_c;
         elseif Ref_Data_C(i)<Ref_Data_C(i-1)
            offset_c=offset_desc_c;
         endif
         Sal_C(i)=Sal_C(i-1)+Sal_C_Act(i);
      endif
      Error_1_NN_C(i)=Ref_Data_C(i)-Sal_C(i); %Calculamos el error
```

```
      Error_1_C(i,1)=(Error_1_NN_C(i)-Mean2)/STD2; %Lo normalizamos
      Error_1_C(i,1)=Error_1_NN_C(i);
      Error_1_NN_N(i)=Ref_Data_N(i)-Sal_N(i); %Calculamos el error
      Error_1_N(i)=(Error_1_NN_N(i)-Mean1)/STD1; %Lo normalizamos
      Error_1_N(i)=Error_1_NN_N(i);
      Error_N(1,1)=Error_1_NN_N(i);
      Error_N(2,1)=Error_1_NN_C(i);
      %Hemos bypaseado la normalización por que
        %funciona mejor sin ella
      Sal_NN_N_NN(i)=sim(Net_1,Error_N); %Calculamos la simulación de
la
      Sal_NN_C_NN(i)=sim(Net_5,Error_1_C(i));
      %entrada del error
      Sal_NN_N_NN_1(i)=(Sal_NN_N_NN(i)*STD3)+Mean3;%Normalizamos
      Sal_NN_C_NN_1(i)=(Sal_NN_C_NN(i)*STD4)+Mean4;
      %la salida
      Sal_NN_N_NN_1(i)=Sal_NN_N_NN(i); %Bypaseamos la normalización
      Sal_NN_C_NN_1(i)=Sal_NN_C_NN(i);
      %de la salida por que funciona mejor sin.
      if Error_1_NN_N(i)==0 %Si el error es 0 la señal de la red tiene
que ser 0.
        Sal_NN_N(i)=0;
      else %En caso contrario elevamos la salida de la red y le
añadimos un offset
        Sal_NN_N(i)=(Sal_NN_N_NN_1(i)*100+offset_n);
      endif
      if Error_1_NN_C(i)==0 %Si el error es 0 la señal de la red tiene
que ser 0.
        Sal_NN_C(i)=0;
      else %En caso contrario elevamos la salida de la red y le
añadimos un offset
        Sal_NN_C(i)=(Sal_NN_C_NN_1(i)+offset_c);
      endif
      if i==m
      else %traspasamos el valor de la señal de la red a la variable
entrada del
      %sistema para empezar la siguiente iteración
          Ent_Sist_N(i+1)=Sal_NN_N(i);
          Ent_Sist_C(i+1)=Sal_NN_C(i);
      endif
   endfor
   %Creamos una matriz para comparar los datos
   M_Com=zeros(m_C,7);
   M_Com(:,1)=Sal_C_Act;
   M_Com(:,2)=Sal_C;
   M_Com(:,3)=Ref_Data_C(:,1);
   M_Com(:,4)=Error_1_NN_C;
   M_Com(:,5)=Error_1_C(:,1);
   M_Com(:,6)=Sal_NN_C_NN;
   M_Com(:,7)=Sal_NN_C_NN_1;
   M_Com(:,8)=Sal_NN_C;
   M_Com(:,9)=Ent_Sist_C;
   M_Com;
   Data(:,j+2)=Sal_N(:,1);
endfor
Data(:,2)=Sal_C(:,1);
Data(:,1)=Ref_Data_C(:,1);
Ejex=1:m_C;
figure(3);
clf(3);
plot(Ejex,Data);
```

```
legend("Ref Data","Sal C","Sal N Con Nor","Sal N Sin Nor");
```

13.1.2.6 RED 4 (CAUDAL CON 2 ENTRADAS):

```
%Quitando_Valores_Ambas
%Cargamos los paquetes necesarios
%pkg load nnet
%pkg load io
%pkg load control
cargar_redes=1;
if cargar_redes==1
clear
      load Net_4_Con_Nor;
        Red_1=Net_4_Con_Nor;
      load Net_4_Sin_Nor;
        Red_2=Net_4_Sin_Nor;
      load Net_1_Con_Nor;
        Net_5=Net_1_Con_Nor;
endif
for j=1:2
  clear Net_1;
  %Cargamos que red neuronal queremos simular
  switch j
    case 1
    Net_1=Red_1;
    case 2
    Net_1=Red_2;
  endswitch
  %Le asignamos la red a la variable Net_1 que es
  %la que se utiliza en todo el código.
  %Cargamos medias y desviaciones estandar
  Mean1=0.0053164;Mean2=19.7704273;Mean3=0.0024142;Mean4=7.2803693;
  STD1=2.08929;STD2=10203.65151;STD3=0.36044;STD4=2.65699;
  %Definimos dos funciones de transferencia
  s=tf('s');
  G11=tf([0.0002823],[1 0.04295 0.0002129],0.001);
  G22=tf([0.1452],[1 0.2127],0.001);
  %Cargamos el vector de referencias
  t=0;
  x=0.25;
  y1=0.5;
  y2=-1;
  temp=300;
  A=zeros(7*temp,1);
  A(1:temp,1)=t;
  A(temp+1:2*temp,1)=y1;
  A(2*temp+1:3*temp,1)=y1*2;
  A(3*temp+1:4*temp,1)=y1;
  A(4*temp+1:5*temp,1)=0;
  A(5*temp+1:6*temp,1)=-y1;
  A(6*temp+1:7*temp,1)=-y1*2;
  %Refsize_Data_1=xlsread('Validate_Data');
  %Refsize_Data_2=xlsread('Validate_Data_Lim_SM');
  %Refsize_Data_1=Refsize_Data_2(:,5);
  Refsize_Data_1_N=A;
  [m,n]=size(Refsize_Data_1_N);
  Val_quitados=0001;
  Refsize_Data_N=zeros(m-Val_quitados,n);
  Refsize_Data_N=Refsize_Data_1_N(Val_quitados:m,:);
   Refsize_Data_1_C=A;
  [m,n]=size(Refsize_Data_1_C);
  Val_quitados=0001;
  Refsize_Data_C=zeros(m-Val_quitados,n);
  Refsize_Data_C=Refsize_Data_1_C(Val_quitados:m,:);
```

```
  %Creamos el vector de tren de pulsos.
  Ref_Data_N=Refsize_Data_N;Ref_Data_C=Refsize_Data_C;
  [m_N,n_N]=size(Ref_Data_N);[m_C,n_C]=size(Ref_Data_C); %Tiempos de
simulación
  %Creamos vectores
  Ent_Sist_N=zeros(m_N,1);Ent_Sist_C=zeros(m_C,1);
  Ent_Sist_N(1)=0;Ent_Sist_C(1)=0;
  %Inicializamos el vector de
  %entradas ya que en el instante 1 no hay
  Sal_N=zeros(m_N,1);Sal_C=zeros(m_C,1);
  Sal_N_Act=zeros(m_N,1);Sal_C_Act=zeros(m_C,1);
  Error_1_N=zeros(m_N,1);Error_1_C=zeros(m_C,2);
  Error_1_NN_N=zeros(m_N,1);Error_1_NN_C=zeros(m_C,1);
  Sal_NN_N_NN=zeros(m_N,1);Sal_NN_C_NN=zeros(m_C,1);
  Sal_NN_N_NN_1=zeros(m_N,1);Sal_NN_C_NN=zeros(m_C,1);
  Sal_NN_N=zeros(m_N,1);Sal_NN_C=zeros(m_C,1);
  %Data=zeros(m,5);
  offset_n=0;
  offset_c=0;
   for i=1:m_C
      %Tenemos que distinguir si es el momento 1
      %de simulación o si es continuo
      [a_n,b_n]=step(G11*Ent_Sist_N(i),0.01); %Calculamos el step
      Sal_N_Act(i)=a_n(end);
      [a_c,b_c]=step(G22*Ent_Sist_C(i),0.01); %Calculamos el step
      Sal_C_Act(i)=a_c(end); %De todos los valores del step
      %Cogemos el valor del último instante, el de 0.01 segundos
      offset_asc_n=13.7;
      offset_desc_n=0;
      offset_asc_c=0.02;
      offset_desc_c=0.01;
      if i==1 %Si es el primer instante no
          if Ref_Data_N(1)>0
            offset_n=offset_asc_n;
          elseif Ref_Data_N(1)<0
            offset_n=offset_desc_n;
          endif
          if Ref_Data_C(1)>0
            offset_c=offset_asc_c;
          elseif Ref_Data_C(1)<0
            offset_c=offset_desc_c;
          endif
      else %Si es otro instante, calculamos si estamos subiendo o
bajando
      %para aplicar un offset
         if Ref_Data_N(i)>Ref_Data_N(i-1)
            offset_n=offset_asc_n;
         elseif Ref_Data_N(i)<Ref_Data_N(i-1)
            offset_desc_n=offset_desc_n;
         endif
         Sal_N(i)=Sal_N(i-1)+Sal_N_Act(i); %Sumamos cual ha sido la
          %evolución al instante anterior, a como esta la señal en el
instante anterior
      if Ref_Data_C(i)>Ref_Data_C(i-1)
            offset_c=offset_asc_c;
         elseif Ref_Data_C(i)<Ref_Data_C(i-1)
            offset_desc_c=offset_desc_c;
         endif
         Sal_C(i)=Sal_C(i-1)+Sal_C_Act(i);
      endif
      Error_1_NN_N(i)=Ref_Data_N(i)-Sal_N(i); %Calculamos el error
```

```
      Error_1_N(i)=(Error_1_NN_N(i)-Mean1)/STD1; %Lo normalizamos
      Error_1_N(i)=Error_1_NN_N(i);
      Error_1_NN_C(i)=Ref_Data_C(i)-Sal_C(i); %Calculamos el error
      Error_1_C(i,1)=(Error_1_NN_C(i)-Mean2)/STD2; %Lo normalizamos
      Error_1_C(i,1)=Error_1_NN_C(i);
      Error_1_C(i,2)=Error_1_NN_N(i);
      Error_C(1,1)=Error_1_NN_C(i);
      Error_C(2,1)=Error_1_NN_N(i);
      %Hemos bypaseado la normalización por que
        %funciona mejor sin ella
      Sal_NN_N_NN(i)=sim(Net_5,Error_1_N(i)); %Calculamos la
simulación de la
      Sal_NN_C_NN(i)=sim(Net_1,Error_C);
      %entrada del error
      Sal_NN_N_NN_1(i)=(Sal_NN_N_NN(i)*STD3)+Mean3;%Normalizamos
      Sal_NN_C_NN_1(i)=(Sal_NN_C_NN(i)*STD4)+Mean4;
      %la salida
      Sal_NN_N_NN_1(i)=Sal_NN_N_NN(i); %Bypaseamos la normalización
      Sal_NN_C_NN_1(i)=Sal_NN_C_NN(i);
      %de la salida por que funciona mejor sin.
      if Error_1_NN_N(i)==0 %Si el error es 0 la señal de la red tiene
que ser 0.
        Sal_NN_N(i)=0;
      else %En caso contrario elevamos la salida de la red y le
añadimos un offset
        Sal_NN_N(i)=(Sal_NN_N_NN_1(i)*100+offset_n);
      endif
      if Error_1_NN_C(i)==0 %Si el error es 0 la señal de la red tiene
que ser 0.
        Sal_NN_C(i)=0;
      else %En caso contrario elevamos la salida de la red y le
añadimos un offset
        Sal_NN_C(i)=(Sal_NN_C_NN_1(i)+offset_c);
      endif
      if i==m
      else %traspasamos el valor de la señal de la red a la variable
entrada del
      %sistema para empezar la siguiente iteración
          Ent_Sist_N(i+1)=Sal_NN_N(i);
          Ent_Sist_C(i+1)=Sal_NN_C(i);
      endif
   endfor
   %Creamos una matriz para comparar los datos
   M_Com=zeros(m_C,7);
   M_Com(:,1)=Sal_C_Act;
   M_Com(:,2)=Sal_C;
   M_Com(:,3)=Ref_Data_C(:,1);
   M_Com(:,4)=Error_1_NN_C;
   M_Com(:,5)=Error_1_C(:,1);
   M_Com(:,6)=Sal_NN_C_NN;
   M_Com(:,7)=Sal_NN_C_NN_1;
   M_Com(:,8)=Sal_NN_C;
   M_Com(:,9)=Ent_Sist_C;
   M_Com;
   Data(:,j+2)=Sal_C(:,1);
endfor
Data(:,2)=Sal_N(:,1);
Data(:,1)=Ref_Data_C(:,1);
Ejex=1:m_C;
figure(4);
clf(4);
```

```
plot(Ejex,Data);
legend("Ref Data","Sal N","Sal C Con Nor","Sal C Sin Nor");
```

13.2 CATÁLOGO DE VARIABLES:

13.2.1 MATLAB

13.2.1.1 ARCHIVO SIMULACION:

Tren: Variable escalar que indica que datos se quieren generar. Puede tomar valores entre 1 y 4 (1=Validate_Data, 2=Train_Data, 3=Val_Train_Data, 4=Val_short_Data).

SPNivel: Punto de equilibrio de la variable Nivel sobre el que se quiere generar el tren de pulsos.

Te_Bomba: Tiempo de establecimiento de la bomba.

tend: Tiempo final de simulación, para saber cuantos segundos va a necesitar esta variable ser simulada, y así construir los vectores de simulación de las variables.

SP_Nivel: Vector continuo que establece el valor de equilibrio de la variable Nivel. Como se trata de una simulación, no solo basta con una constante, sino que hay que indicar en que tiempo vale que magnitud esa constante. Vendrá marcada por la variable "SPNivel".

x: Incremento en cada tiempo de establecimiento del tren de pulsos de la variable Nivel.

TP_Nivel: Tren de pulsos de la variable Nivel. Serán cambios que se sumarán al punto de equilibrio marcado por "SP_Nivel".

SPCaudal: Punto de equilibrio de la variable Caudal sobre el que se quiere generar el tren de pulsos.

Te_Valvula: Tiempo de establecimiento de la válvula.

SP_Caudal: Vector continuo que establece el valor de equilibrio de la variable Nivel. Como se trata de una simulación, no solo basta con una constante, sino que hay que indicar en que tiempo vale que magnitud esa constante. Vendrá marcada por la variable "SPCaudal".

y: Incremento en cada tiempo de establecimiento del tren de pulsos de la variable caudal.

TP_Caudal: Tren de pulsos de la variable Caudal. Serán cambios que se sumarán al punto de equilibrio marcado por "SP_Caudal".

T_ini: En que tiempo se quiere empezar cada pulso. Se utiliza para que los transitorios de otros pulsos o señales modifiquen o afecten los datos.

N_Pulsos: Número de pulsos que se quiere crear en el tren de pulsos.

Por_Su_Ba: Incremento y decremento del punto de equilibrio de la variable Nivel/Caudal.

Salto_Tren_Nivel: Que incrementos queremos que el tren de pulsos de la variable Nivel tenga.

tt: Tiempo que se va a añadir en el tren de pulsos, para que el PID, lleve el sistema a los cambios de referencia en su punto de equilibrio.

Salto_Tren_Caudal: Que incrementos queremos que el tren de pulsos de la variable Caudal tenga.

ini: Cantidad a sumar en los tiempos del primer ciclo del bucle for. Se utiliza para que el sistema sea capaz de llevar la variable a la referencia.

Num_V_Bomba: Numerador de la función de transferencia de la variable Bomba.

Den_V_Bomba: Denominador de la función de transferencia de la variable Bomba.

Num_V_Valvula: Numerador de la función de transferencia de la variable Valvula.

Den_V_Valvula: Denominador de la función de transferencia de la variable Valvula.

ts: Intervalo de tiempo para cada ciclo. Dividirá el vector de tiempos en escalones de magnitud ts.

PB: Valor de proporcionalidad del PID de la bomba.

IB: Valor de integrabilidad del PID de la bomba.

DB: Valor de derivatividad del PID de la bomba.

NB: Valor de derivabilidad del PID de la bomba.

PB_PID: Vector que representa el valor de la constante PB, como valor de proporcionalidad del PID de la bomba. Marcado por la variable PB.

IB_PID: Vector que representa el valor de la constante IB, como valor de derivabilidad del PID de la bomba. Marcado por la variable IB.

DB_PID: Vector que representa el valor de la constante DB, como valor de integracionalidad del PID de la bomba. Marcado por la variable DB.

NB_PID: Vector que representa el valor de la constante NB, como valor de derivabilidad del PID de la bomba. Marcado por la variable NB.

Num_Cons_PID_Nivel_En: Constante por la que hay que multiplicar la entrada del controlador de nivel, para suplir el efecto del microcontrolador Arduino que transforma las señales en voltaje.

Cons_PID_Nivel_En: Vector de la constante por la que hay que multiplicar la entrada del controlador de nivel, para suplir el efecto del microcontrolador Arduino que transforma las señales en voltaje.

Num_Cons_PID_Nivel_Sa: Constante por la que hay que multiplicar la entrada del controlador de nivel, para suplir el efecto del microcontrolador Arduino que transforma las señales en voltaje.

Cons_PID_Nivel_Sa: Vector de la constante por la que hay que multiplicar la entrada del controlador de nivel, para suplir el efecto del microcontrolador Arduino que transforma las señales en voltaje.

Num_Cons_PID_Caudal: Constante por la que hay que multiplicar la entrada del controlador de caudal, para suplir el efecto del microcontrolador Arduino que transforma las señales en voltaje.

Cons_PID_Caudal: Vector de la constante por la que hay que multiplicar la entrada del controlador de caudal, para suplir el efecto del microcontrolador Arduino que transforma las señales en voltaje.

PV: Valor de proporcionalidad del PID de la válvula.

IV: Valor de integralidad del PID de la válvula.

DV: Valor de derivabilidad del PID de la válvula.

NV: Valor de derivabilidad del PID de la válvula.

PV_PID: Vector que representa el valor de la constante PV, como valor de proporcionalidad del PID de la válvula. Marcado por la variable PV.

IV_PID: Vector que representa el valor de la constante IV, como valor de derivabilidad del PID de la válvula. Marcado por la variable IV.

DV_PID: Vector que representa el valor de la constante DV, como valor de integralidad del PID de la válvula. Marcado por la variable DV.

NV_PID: Vector que representa el valor de la constante NV, como valor de derivabilidad del PID de la válvula. Marcado por la variable NV.

Sim: Variable que indica que archivo de Simulink se va a utilizar, si el sistema de control por bucle cerrado (cerrado=control del ciclo cerrado por realimentación) o si por bucle abierto (abierto=control del ciclo abierto).

vt: Vector de tiempos de la simulación.

t: Vector que contiene los tiempos con los que se ha simulado.

estados: Estado de las variables relevantes durante la simulación.

resultados_cer: Variable que contiene las salidas que en el archivo Simulink de lazo cerrado se hayan delimitado como outputs.

Train_Data: Nombre del archivo de exportación Excel en el que se escribirán los datos del entrenamiento.

Validate_Data: Nombre del archivo de exportación Excel en el que se escribirán los datos de la validación.

Val_train_Data: Nombre del archivo de exportación Excel en el que se escribirán los datos de validación del entrenamiento.

resultados_abi: Variable que contiene las salidas que en el archivo Simulink de lazo abierto se hayan delimitado como outputs.

res_Abierto: Nombre del archivo de exportación Excel en el que se escribirán los datos de comportamiento del lazo cerrado

13.2.2 OCTAVE

13.2.2.1 ARCHIVO NN.

L: Número de capas (layers).

TF_N: Tipo de función de transferencia de cada capa (TF_1, función de transferencia de la capa 1). Si no se especifica, se utiliza “tansig”.

BTF: Modo de entrenamiento “backpropagation”. Si no se especifica, se utiliza “trainlm”.

BLF: Se utiliza solo para compatibilidad con Matlab, no es útil aquí.

PF: Función de actuación. Si no se especifica, se utiliza “mse”.

In_D_NN_Col_M1: Los datos de entrada que se necesitan, no normalizados y ordenados en columnas sin quitar ningún valor y sin escoger los datos necesarios por red.

M: Se usará en todo el código para leer el número de filas de una variable.

N: Se usará en todo el código para leer el número de columnas de una variable.

In_D_NN_Col_M: Los datos de entrada que se necesitan, no normalizados y ordenados en columnas sin seleccionar los datos necesarios por red y habiendo quitado los valores seleccionados.

Red: Valor entre 1 y 4 que selecciona que tipo de red se quiere simular (1=Control de nivel con 1 entrada, 2=Control de caudal con 1 entrada, 3=Control de nivel con dos entradas, 4=Control de caudal con 2 entradas).

In_D_NN_Col: Los datos de entrada que se necesitan, no normalizados y ordenados en columnas seleccionando los datos necesarios por red y habiendo quitado los valores seleccionados.

N_Ln: Número de neuronas por capa n (N_L1= Número de neuronas de la capa 1, N_L2=Número de neuronas de la capa 2...).

In_D_NN: Los datos de entrada que se necesitan, no normalizados y ordenados en filas seleccionando los datos necesarios por red y habiendo quitado los valores seleccionados.

In_D: Los datos de entrada que se necesitan, normalizados y ordenados en filas seleccionando los datos necesarios por red y habiendo quitado los valores seleccionados.

Mean_In_D: Media de los datos que se han normalizado (In_D_NN).

STD_In_D: Desviación estándar de los datos que se han normalizado (In_D_NN).

Fila_train: Valor redondeado de dividir la cantidad total de datos (m, filas de la matriz In_D_NN_Col) que se utiliza para dividir la cantidad de datos en datos de entrenamiento, target y validación.

In_D_Train_N: Valores normalizados de entrenamiento seleccionados para las entradas.

Tar_D_Train_N: Valores normalizados de entrenamiento seleccionados para target.

VV.P: Valores normalizados de validación, seleccionados para las entradas.

VV.T: Valores normalizados de validación seleccionados para los targets.

min: Vector de valores mínimos de los datos normalizados, que se inicializa como un valor muy grande para que cuando encuentre uno más pequeño se sustituya por él.

max: Vector de valores máximos de los datos normalizados, que se inicializa como un valor muy grande para que cuando encuentre uno más pequeño se sustituya por él.

Val_Min_Max_DNN: Vector con el valor mínimo y máximo de las entradas para normalizar.

net_nt: Variable que almacena la red creada.

net_nt.trainParam.epochs: Parámetro epoch de la red creada.

net_nt.trainParam.goal: Parámetro goal de la red creada.

net_nt.trainParam.max_fail: Parámetro max_fail de la red creada.

net_nt.trainParam.mem_reduc: Parámetro mem_reduc de la red creada.

net_nt.trainParam.min_grad: Parámetro min_grad de la red creada.

net_nt.trainParam.mu: Parámetro mu de la red creada.

net_nt.trainParam.mu_dec: Parámetro mu_dec de la red creada.

net_nt.trainParam.mu_inc: Parámetro mu_inc de la red creada.

net_nt.trainParam.mu_max: Parámetro mu_max de la red creada.

net_nt.trainParam.show: Parámetro show de la red creada.

net_nt.trainParam.time: Parámetro time de la red creada.

Net_X_Y: Nombre de la red creada, donde X es el tipo de red (1,2,3 o 4) y la Y es a que red de ese tipo nos referimos.

13.2.2.2 ARCHIVO OUT1_NN.

Val_Data_Col_MR: Los datos de entrada que se necesitan, no normalizados y ordenados en columnas sin quitar ningún valor y sin escoger los datos necesarios por red.

Val_quitados: Número de valores que se quiere quitar desde el principio sobre los datos de entrada, para eliminar problemas de transición.

Val_Data_Col_M: Los datos de entrada que se necesitan, no normalizados y ordenados en columnas quitando los valores seleccionados y sin escoger los datos necesarios por red.

Red: Valor entre 1 y 4 que selecciona que tipo de red se quiere simular (1=Control de nivel con 1 entrada, 2=Control de caudal con 1 entrada, 3=Control de nivel con dos entradas, 4=Control de caudal con 2 entradas).

Net_X_Y: Nombre de la red creada, donde X es el tipo de red (1,2,3 o 4) y la Y es a que red de ese tipo nos referimos.

Net_1: Variable en la que se almacena la red introducida, para así cuando se necesite cambiar la red sobre la que se va a actuar, simplemente se cambia la asociación a esta variable.

In_D_NN_Col: Los datos de entrada que se necesitan, no normalizados y ordenados en columnas quitando los valores seleccionados y escogiendo los datos necesarios por red.

In_D_NN: Los datos de entrada que se necesitan, normalizados y ordenados en columnas quitando los valores seleccionados y escogiendo los datos necesarios por red.

In_D_Sim_N: Los datos de entrada que se necesitan, normalizados y ordenados en filas seleccionando los datos necesarios por red y habiendo quitado los valores seleccionados.

Mean_In_D: Media de los datos que se han normalizado (In_D_NN).

STD_In_D: Desviación estándar de los datos que se han normalizado (In_D_NN).

Out_D_N_SP: Salidas de la simulación de la red sin desnormalizar, cuando se introduce un vector de entradas (In_D_SIM_N), y se quiere saber cuál es la respuesta de la red a cada una de ellas.

Out_D_N_1: Salidas de la simulación de la red desnormalizadas, cuando se introduce un vector de entradas (In_D_SIM_N), y se quiere saber cuál es la respuesta de la red a cada una de ellas.

In_D_Sim_N_2: Salidas del PID desnormalizadas, cuando se introduce el mismo vector de entradas que para la red neuronal.

Val_N_1: Evaluación en porcentaje de la diferencia entre los valores de respuesta de la red neuronal frente a los del PID para un mismo vector de entradas.

M: Se usará en todo el código para leer el número de filas de una variable.

N: Se usará en todo el código para leer el número de columnas de una variable.

Data: Matriz que almacena los datos que se quieren graficar. En este caso se trata de la respuesta del PID en la primera fila y la respuesta de la Red Neuronal en la segunda.

Ejex: Secuencia de números ascendente que representarán el eje x al realizar la gráfica para que se pueda graficar los datos de la matriz “Data” respecto a un vector. Representan cada instante en el que se va a comparar una respuesta frente a otra.

13.2.2.3 SIMULACIÓN CAUDAL

Net_X_Y: Nombre de la red creada, donde X es el tipo de red (1,2,3 o 4) y la Y es a que red de ese tipo nos referimos.

Net_1: Variable en la que se almacena la red introducida, para así cuando se necesite cambiar la red sobre la que se va a actuar, simplemente se cambia la asociación a esta variable.

Med: Variable que indica cuál de las siguientes medias y desviaciones estándar con las que se normalizó los datos al entrenar la red se va a utilizar.

MeanX: Media de los datos con los que se normalizó los datos de entrenamiento. Puede variar entre 1 y 4 refiriéndose a las 4 columnas, 2 entradas y 2 salidas de datos del PID. Se utilizará para normalizar los nuevos datos con los que operará la red.

STDX: Desviación estándar de los datos con los que se normalizó los datos de entrenamiento. Puede variar entre 1 y 4 refiriéndose a las 4 columnas, 2 entradas y 2

salidas de datos del PID. Se utilizará para normalizar los nuevos datos con los que operará la red.

s: Variable que se utilizará en la función de transferencia.

G11: Función de transferencia que relaciona la bomba con el nivel:

G22: Función de transferencia que relaciona la válvula con el caudal.

Refsize_Data_2: Los datos de entrada que se necesitan, no normalizados y ordenados en columnas sin quitar ningún valor y sin escoger la columna necesaria por red.

Refsize_Data_1: Los datos de entrada que se necesitan, no normalizados y ordenados en columnas sin quitar ningún valor, escogiendo la columna necesaria por red.

A: Vector de referencias creado manualmente para realizar pruebas rápidas con las que conocer el funcionamiento y desempeño de la red. Consiste en diferentes escalones ascendentes y descendentes.

Val_Ini: Posición inicial del vector "Refsize_Data_1" desde la que se quiere coger valores.

Val_final: Posición final del vector "Refsize_Data_1" desde la que se quiere coger valores.

Refsize_Data: Los datos de entrada que se necesitan, no normalizados y ordenados en columnas, escogiendo la columna necesaria por red y los valores seleccionados.

Ref_Data: Los datos de entrada que se necesitan, no normalizados y ordenados en columnas, escogiendo la columna necesaria por red y los valores seleccionados.

Ent_Sist: Vector que se rellena con las entradas del sistema.

Sal_Caudal: Vector que se rellena con las salidas del sistema cuando se genera una simulación, que permite obtener la salida del instante siguiente, con la salida de caudal actual y la salida en el instante anterior sumándolas.

Sal_Caudal_Act: Salida actual del sistema cuando se ha realizado una simulación.

Error_1: Error no normalizado, generado por la diferencia entre la salida del sistema actual y la referencia deseada.

Error_1_NN: Error normalizado, generado por la diferencia entre la salida del sistema actual y la referencia deseada.

Sal_NN_Caudal_NN: Salida de la Red Neuronal no normalizada sin aplicar offset.

Sal_NN_Caudal_NN_1: Salida de la Red Neuronal normalizada sin aplicar offset.

Sal_NN_Caudal: Salida de la Red Neuronal normalizada aplicando offset.

Data: Matriz que almacena los datos que se quieren graficar. En este caso se trata de la respuesta del PID en la primera fila y la respuesta de la Red Neuronal en la segunda.

Ejex: Secuencia de números ascendente que representarán el eje x al realizar la gráfica para que se pueda graficar los datos de la matriz "Data" respecto a un vector. Representan cada instante en el que se va a comparar una respuesta frente a otra.

b: Respuesta de tiempos ante el comando offset.

a: Respuesta del comando offset a una entrada del sistema.

offset: Cantidad a sumar a la salida de la red neuronal para ajustarla a la magnitud adecuada. En cada ciclo puede deberse al "offset_asc" o al "offset_desc", dependiendo de si estamos ascendiendo a una referencia mayor o menor.

offset_asc: Cantidad a sumar a la salida de la red neuronal cuando se está ascendiendo, para ajustarla a la magnitud adecuada. Puede componer o no la variable offset, dependiendo de si estamos ascendiendo a una referencia mayor o menor.

offset_desc: Cantidad a sumar a la salida de la red neuronal cuando se está descendiendo, para ajustarla a la magnitud adecuada. Puede componer o no la variable offset, dependiendo de si estamos ascendiendo a una referencia mayor o menor.

M_Com: Matriz que combina todos los datos para poder exponerlos por pantalla en una matriz. Con esto se obtiene una visión rápida del desempeño de la red.

13.2.2.4 SIMULACIÓN NIVEL

Net_X_Y: Nombre de la red creada, donde X es el tipo de red (1,2,3 o 4) y la Y es a que red de ese tipo nos referimos.

Net_1: Variable en la que se almacena la red introducida, para así cuando se necesite cambiar la red sobre la que se va a actuar, simplemente se cambia la asociación a esta variable.

Med: Variable que indica cuál de las siguientes medias y desviaciones estándar con las que se normalizó los datos al entrenar la red se va a utilizar.

MeanX: Media de los datos con los que se normalizó los datos de entrenamiento. Puede variar entre 1 y 4 refiriéndose a las 4 columnas, 2 entradas y 2 salidas de datos del PID. Se utilizará para normalizar los nuevos datos con los que operará la red.

STDX: Desviación estándar de los datos con los que se normalizó los datos de entrenamiento. Puede variar entre 1 y 4 refiriéndose a las 4 columnas, 2 entradas y 2 salidas de datos del PID. Se utilizará para normalizar los nuevos datos con los que operará la red.

s: Variable que se utilizará en la función de transferencia.

G11: Función de transferencia que relaciona la bomba con el nivel:

G22: Función de transferencia que relaciona la válvula con el caudal.

Refsize_Data_2: Los datos de entrada que se necesitan, no normalizados y ordenados en columnas sin quitar ningún valor y sin escoger la columna necesaria por red.

Refsize_Data_1: Los datos de entrada que se necesitan, no normalizados y ordenados en columnas sin quitar ningún valor, escogiendo la columna necesaria por red.

A: Vector de referencias creado manualmente para realizar pruebas rápidas con las que conocer el funcionamiento y desempeño de la red. Consiste en diferentes escalones ascendentes y descendentes.

Val_Ini: Posición inicial del vector “Refsize_Data_1” desde la que se quiere coger valores.

Val_final: Posición final del vector “Refsize_Data_1” desde la que se quiere coger valores.

Refsize_Data: Los datos de entrada que se necesitan, no normalizados y ordenados en columnas, escogiendo la columna necesaria por red y los valores seleccionados.

Ref_Data: Los datos de entrada que se necesitan, no normalizados y ordenados en columnas, escogiendo la columna necesaria por red y los valores seleccionados.

Ent_Sist: Vector que se rellena con las entradas del sistema.

Sal_Nivel: Vector que se rellena con las salidas del sistema cuando se genera una simulación, que permite obtener la salida del instante siguiente, con la salida de caudal actual y la salida en el instante anterior sumándolas.

Sal_Nivel_Act: Salida actual del sistema cuando se ha realizado una simulación.

Error_1: Error no normalizado, generado por la diferencia entre la salida del sistema actual y la referencia deseada.

Error_1_NN: Error normalizado, generado por la diferencia entre la salida del sistema actual y la referencia deseada.

Sal_NN_Nivel_NN: Salida de la Red Neuronal no normalizada sin aplicar offset.

Sal_NN_Nivel_NN_1: Salida de la Red Neuronal normalizada sin aplicar offset.

Sal_NN_Nivel: Salida de la Red Neuronal normalizada aplicando offset.

Data: Matriz que almacena los datos que se quieren graficar. En este caso se trata de la respuesta del PID en la primera fila y la respuesta de la Red Neuronal en la segunda.

Ejex: Secuencia de números ascendente que representarán el eje x al realizar la gráfica para que se pueda graficar los datos de la matriz “Data” respecto a un vector. Representan cada instante en el que se va a comparar una respuesta frente a otra.

b: Respuesta de tiempos ante el comando offset.

a: Respuesta del comando offset a una entrada del sistema.

offset: Cantidad a sumar a la salida de la red neuronal para ajustarla a la magnitud adecuada. En cada ciclo puede deberse al “offset_asc” o al “offset_desc”, dependiendo de si estamos ascendiendo a una referencia mayor o menor.

offset_asc: Cantidad a sumar a la salida de la red neuronal cuando se está ascendiendo, para ajustarla a la magnitud adecuada. Puede componer o no la variable offset, dependiendo de si estamos ascendiendo a una referencia mayor o menor.

offset_desc: Cantidad a sumar a la salida de la red neuronal cuando se está descendiendo, para ajustarla a la magnitud adecuada. Puede componer o no la variable offset, dependiendo de si estamos ascendiendo a una referencia mayor o menor.

M_Com: Matriz que combina todos los datos para poder exponerlos por pantalla en una matriz. Con esto se obtiene una visión rápida del desempeño de la red.