ANEXO A

PROGRAMAS PARA LA ESTIMACIÓN DE PARÁMETROS Y VALIDACIÓN DEL MODELO NO LINEAL MULTIVARIABLE OBTENIDO

A.1. p02AlatiqiInititalConditionsBL.m

```
%Determinación del estado estacionario inicial (condiciones %iniciales) del sistema.
%Este programa sirve también para calcular los valores reales de %kA y kB mediante prueba y error. Los valores reales de estos %parámetros serán aquellos que hacen que la salida del modelo %sea igual a la salida de la planta real (datos experimentales).
clc; close all; clear all;

%%Datos
%Corriente de alimentación
Ff=315.45; %g/s (5 gpm)
Pf=(900+14.7)/14.50373; %bar
Tf=25.0;  %°C
Cf=30000; %ppm
pHf=6.45;
Cdf=C2Cd(Cf); %uS/cm
dPb=0; %bar
dPp=0; %bar
%permeado
Pp=1.01325; %bar
%membrana
kA=2.4500*10^-3; %m/s
kB=1.3605*10^-2; %g/(s*m2*bar)
A=152; %m2
k=kB/kA; %bar-1
%soluto
Mmolar=58.44; %g/gmol
%Condiciones iniciales para solución del sistema de EDO's
mb0=43/50*1000; %g
mp0=35/50*1000; %g
Tb0=0; %°C
Tp0=0; %°C
Cb0=0; %ppm
Cp0=0; %ppm
%%Simulación del modelo no lineal
open('AlatiqiInitialConditionsBL');
simOut=sim('AlatiqiInitialConditionsBL');
```

A.2 p03AlatiqiBLComparacionPFC.m

```
%Comparación de los resultados del modelo no lineal con
los %valores experimentales de Alatiqi et al. (1989).
%Efecto de la presión de la alimentación sobre el flujo y
la %conductividad del permeado.
clc; close all; clear all; format short g;

%Se ejecuta el programa AlatiqiInititalConditionsBL.m para
%determinar el valor de todas las variables en el estado
%estacionario inicial y se asignan estos valores a sus
%variables %respectivas
p02AlatiqiInititalConditionsBL
%Condiciones iniciales de la corriente de alimentación
Ff0=Ff; %g/s
Pf0=Pf; %bar
Tf0=Tf; %°C
Cf0=Cf; %ppm
pHf0=pHf;
%Condiciones iniciales sistema
mb0=mb(end); %g
mp0=mp(end); %g
Fb0=Fb(end); %g/s
Fp0=Fp(end); %g/s
Cb0=Cb(end); %ppm
Cp0=Cp(end); %ppm
Tb0=Tb(end); %°C
Tp0=Tp(end); %°C
Pb0=Pb(end); %bar
Pp0=Pp(end); %bar

%Se simula el sistema en lazo abierto frente a los mismos
%cambios realizados en la referencia. Luego se presentan
los %resultados en forma gráfica.
open('AlatiqiBLROSimulationRVarP');
simOut=sim('AlatiqiBLROSimulationRVarP');
tsim=retentate.time;
Fretentate=retentate.signals(1,1).values;
Cretentate=retentate.signals(1,2).values;
Tretentate=retentate.signals(1,3).values;
Pretentate=retentate.signals(1,4).values;
% clear 'retentate'
Fpermeate=permeate.signals(1,1).values;
Cpermeate=permeate.signals(1,2).values;
Tpermeate=permeate.signals(1,3).values;
Ppermeate=permeate.signals(1,4).values;
% clear 'permeate'

%Se definen las variables para comparar gráficamente
%resultados.
Pf=Pfdata.signals.values; %bar
```

```
Fp=Fpermeate; %g/s
Condp=Kohlrausch(Cpermeate).*(1+0.0212*(Tpermeate-
25));%uS/cm,
%Se calculan las variables de desviación los resultados
de %simulación
Pfn=Pf-Pf(1);
Fpn=Fp-Fp(1);
Condpn=Condp-Condp(1);

%%Valores experimentales de la referencia.
tA=30*(0:45)';%s
PfA=[900 900 900 900 900 900 900 900 1000 1000 1000 1000
1000 900 900 900 900 900 900 900 900 900 900 900 900 900
900 900 900 900 800 800 700 700 900 900 900 900 900 900
900 900 900 900 900 900]';%psi
FpA=[1.05 1.05 1.05 1.05 1.05 1.05 1.05 1.05 1.25 1.25
1.25 1.25 1.25 1.25 1.05 1.05 1.05 1.05 1.05 1.05 1.05
1.05 1.05 1.05 1.05 1.05 1.05 1.05 1.05 1.05 0.85 0.85
0.65 0.65 1.05 1.05 1.05 1.05 1.05 1.05 1.05 1.05 1.05
1.05 1.05 1.05]';%gpm
CondpA=[435 438 437 437 437 436 436 437 401 403 401 402
404 436 435 436 437 435 436 437 435 438 436 437 436 435
437 436 435 436 486 484 558 562 435 436 435 436 437 436
437 435 436 436 436 437]';%uS/cm
disp('      [tiempo(s) presión(psi) flujo(gpm)
conductividad(uS/cm)]')
disp([tA PfA FpA CondpA])
%se escriben los datos experimentales en las unidades del
%presente trabajo
PfA=(PfA+14.7)/14.50373;
FpA=FpA/0.01585033;
%Se calculan las variables de desviación de los datos
%experimentales.
PfAn=PfA-PfA(1);
FpAn=FpA-FpA(1);
CondpAn=CondpA-CondpA(1);

%Se toman los valores de simulación del modelo
%correspondientes a los mismos tiempos que los datos
%disponibles en la referencia.
puntos=tA+1;
tcomp=tsim(puntos);
Fpcomp=Fp(puntos);
Condpcomp=Condp(puntos);
%Se calcula el error cuadrático medio (MSE) y la bondad
de %ajuste (Fit)
errorFp=Fpcomp-FpA;
MSEFp=sum(errorFp.^2)/(length(errorFp)-1)
fitFp=goodnessOfFit(Fpcomp,FpA,'NMSE');
fitFptext=strcat(num2str(round(fitFp*10000)/100),' %')
errorCondp=Condpcomp-CondpA;
```

```
MSECondp=sum(errorCondp.^2)/(length(errorCondp)-1)
fitCondp=goodnessOfFit(Condpcomp,CondpA,'NMSE');
fitCondp=0.7457;%este valor es obtenido con el programa
%AlatiqiBLValidacionPFC.m
fitCondptext=strcat(num2str(round(fitCondp*10000)/100),'
%')

%%Comparación de resultados
%gráficas en unidades reales
figure(1);
plot(tA,FpA,'o','LineWidth',2);
hold all;
plot(tsim,Fp,'LineWidth',2);
xlabel('tiempo, s');
ylabel('Flujo, ml/s');
% title('Flujo de permeado ante cambios escalón en la
presión %de la alimentación');
legend('Experimental','Modelo');
text(1050,72.5,strcat('Fit = ',fitFptext));
grid on;
figure(2);
plot(tA,CondpA,'LineWidth',2);
hold all;
plot(tsim,Condp,'LineWidth',2);
xlabel('tiempo, s');
ylabel('Conductividad, uS/cm');
% title('Conductividad del permeado ante cambios escalón
en la %presión de la alimentación');
legend('Experimental','Modelo');
text(1050,590,strcat('Fit = ',fitCondptext));
grid on;
```

A.3 p04AlatiqiBLComparacionpHC

```
%Comparación de los resultados del modelo no lineal con
los %valores experimentales de Alatiqi et al. (1989).
Efecto del %pH %de la alimentación sobre la conductividad
del permeado.
clc; close all; clear all; format short g;

%Se ejecuta el programa AlatiqiInititalConditionsBL.m
p02AlatiqiInititalConditionsBL
%Condiciones iniciales de la corriente de alimentación
Ff0=Ff; %g/s
Pf0=Pf; %bar
Tf0=Tf; %°C
Cf0=Cf; %ppm
pHf0=pHf;
%Condiciones iniciales sistema
mb0=mb(end); %g
mp0=mp(end); %g
Fb0=Fb(end); %g/s
Fp0=Fp(end); %g/s
Cb0=Cb(end); %ppm
Cp0=Cp(end); %ppm
Tb0=Tb(end); %°C
Tp0=Tp(end); %°C
Pb0=Pb(end); %bar
Pp0=Pp(end); %bar

%Se simula el sistema en lazo abierto frente a los mismos
%cambios realizados en la referencia.
open('AlatiqiBLROSimulationRVarpH');
simOut=sim('AlatiqiBLROSimulationRVarpH');
tsim=retentate.time;
Fretentate=retentate.signals(1,1).values;
Cretentate=retentate.signals(1,2).values;
Tretentate=retentate.signals(1,3).values;
Pretentate=retentate.signals(1,4).values;
% clear 'retentate'
Fpermeate=permeate.signals(1,1).values;
Cpermeate=permeate.signals(1,2).values;
Tpermeate=permeate.signals(1,3).values;
Ppermeate=permeate.signals(1,4).values;
% clear 'permeate'

%Se definen las variables para comparar gráficamente
%resultados.
pHf=pHfdata.signals.values;
Condp=Kohlrausch(Cpermeate).*(1+0.0212*(Tpermeate-25));
%Se calculan las variables de desviación los resultados
de %simulación
pHfn=pHf-pHf(1);
```

```
Condpn=Condp-Condp(20*60);
%%Valores experimentales de la referencia.
tA=60*(0:39)';%s
pHfA=[6.45 6.45 6.45 6.45 6.45 6.45 6.45 6.45 6.45 6.66
6.66 6.66 6.66 6.66 6.77 6.77 6.77 6.77 6.77 6.77 6.77
6.77 6.77 6.77 6.77 7.08 7.08 7.08 7.08 7.08 7.08 7.08
7.08 7.08 7.08 7.08 7.08 7.08 7.08 7.08]';%-log([H+])
FpA=[1.05 1.05 1.05 1.05 1.05 1.05 1.05 1.05 1.05 1.05
1.05 1.05 1.05 1.05 1.05 1.05 1.05 1.05 1.05 1.05 1.05
1.05 1.05 1.05 1.05 1.05 1.05 1.05 1.05 1.05 1.05 1.05
1.05 1.05 1.05 1.05 1.05 1.05 1.05 1.05]';%gpm
CondpA=[442 442 443 440 441 441 441 440 441 441 440 425
426 426 426 425 426 420 419 419 418 419 418 418 419 419
418 404 405 404 405 404 405 404 404 403 404 404 403
404]';%uS/cm
disp('        [tiempo(s)    pH      flujo(gpm)
conductividad(uS/cm)]')
disp([tA pHfA FpA CondpA])
%se escriben los datos experimentales en las unidades del
%presente trabajo
FpA=FpA/0.01585033;
%Se calculan las variables de desviación con los valores
de la %referencia.
pHfAn=pHfA-pHfA(1);
CondpAn=CondpA-CondpA(20);

%Se toman los valores de simulación del modelo en los
mismos %tiempos que los datos disponibles en la
referencia.
puntos=tA+1;
tcomp=tsim(puntos);
pHfcomp=pHf(puntos);
%Se calcula el error cuadrático medio (MSE) y la bondad
de %ajuste (Fit)
errorpHf=pHfcomp-pHfA;
MSEpHf=sum(errorpHf.^2)/(length(errorpHf)-1)
fitpHf=goodnessOfFit(pHfcomp,pHfA,'NRMSE');
fitpHptext=strcat(num2str(round(fitpHf*10000)/100),' %')
%%Comparación de resultados
%gráficas en unidades reales
figure(1);
plot(tA,CondpA,'o','LineWidth',2);
hold all;
plot(tsim,Condp,'LineWidth',2);
xlabel('tiempo, s');
ylabel('Conductividad, uS/cm');
% title('Conductividad del permeado ante cambios escalón
en el pH de la alimentación');
legend('Experimental','Modelo');
text(1810,437,strcat('Fit = ',fitpHptext));
grid on;
```

ANEXO B

PROGRAMAS PARA LA IDENTIFICACIÓN EN EL DOMINIO DEL TIEMPO Y LA VALIDACIÓN DEL MODELO LINEAL OBTENIDO

B.1 p01ROGeneracionDatosPFC.m

```
%Simulación en lazo abierto y obtención de datos para la
%identificación del sistema en el dominio del tiempo.
Efecto de %la presión de la alimentación sobre el flujo y
la conductividad %del permeado
clc; close all; clear all;

%%Determinación del Estado Estacionario Inicial
disp('---Determinación del Estado Estacionario Inicial---
');
%Se ejecuta el programa AlatiqiInititalConditionsBL.m para
determinar las variables en el estado estacionario inicial
AlatiqiInititalConditionsBL
%%Generación de datos a partir del Estado Estacionario
Inicial
disp('-Efecto de la presión sobre el flujo y
conductividad-');
%Condiciones iniciales de la corriente de alimentación
Ff0=Ff; %g/s
Pf0=Pf; %bar
Tf0=Tf; %°C
Cf0=Cf; %ppm
pHf0=pHf;
%Condiciones iniciales sistema
kA0=kA(end); %g/(s*m2)
mb0=mb(end); %g
mp0=mp(end); %g
Fb0=Fb(end); %g/s
Fp0=Fp(end); %g/s
Cb0=Cb(end); %ppm
Cp0=Cp(end); %ppm
Tb0=Tb(end); %°C
Tp0=Tp(end); %°C
Pb0=Pb(end); %bar
Pp0=Pp(end); %bar
Condp0=Kohlrausch(Cp0);%uS/cm
%simulación en lazo abierto
open('ROIdentificationRVarP');
simOut=sim('ROIdentificationRVarP');
tsim=permeate.time;
Fpermeate=permeate.signals(1,1).values;
Cpermeate=permeate.signals(1,2).values;
Tpermeate=permeate.signals(1,3).values;
Ppermeate=permeate.signals(1,4).values;
%cálculos
Pfeed=Pretentate;
Condretentate=C2Cd(Cretentate);
Condpermeate=Kohlrausch(Cpermeate);
save datosPFC tsim Pfeed Fpermeate Condpermeate
```

B.2 p02ROGeneracionDatospHC.m

```
%Simulación en lazo abierto y obtención de datos para la
%identificación del sistema en el dominio del tiempo.
Efecto %del pH de la alimentación sobre la conductividad
del %permeado.
clc; close all; clear all;

%%Determinación del Estado Estacionario Inicial
disp('--Determinación del Estado Estacionario Inicial--');
%Se ejecuta el programa AlatiqiInititalConditionsBL.m para
%determinar el valor de todas las variables en el estado
%estacionario inicial.
AlatiqiInititalConditionsBL

%%Generación de datos a partir del Estado Estacionario
Inicial
disp('-----Efecto del pH sobre la conductividad-----');
%Condiciones iniciales de la corriente de alimentación
Ff0=Ff; %g/s
Pf0=Pf; %bar
Tf0=Tf; %°C
Cf0=Cf; %ppm
pHf0=pHf;
%Condiciones iniciales sistema
kA0=kA(end); %g/(s*m2)
mb0=mb(end); %g
mp0=mp(end); %g
Fb0=Fb(end); %g/s
Fp0=Fp(end); %g/s
Cb0=Cb(end); %ppm
Cp0=Cp(end); %ppm
Tb0=Tb(end); %°C
Tp0=Tp(end); %°C
Pb0=Pb(end); %bar
Pp0=Pp(end); %bar
Condp0=Kohlrausch(Cp0);%uS/cm
%simulación en lazo abierto
open('ROIdentificationRVarpH');
simOut=sim('ROIdentificationRVarpH');
tsim=permeate.time;
Fpermeate=permeate.signals(1,1).values;
Cpermeate=permeate.signals(1,2).values;
Tpermeate=permeate.signals(1,3).values;
Ppermeate=permeate.signals(1,4).values;
clear 'permeate'
%cálculos
pHfeed=pHfdata.signals(1,1).values;
Condretentate=C2Cd(Cretentate);
Condpermeate=Kohlrausch(Cpermeate);
save datospHC tsim pHfeed Condpermeate
```

B.3 p03PreparacionDatosIdenPF.m

```
%Se calculan los valores de las variables de desviación a
%utilizar en la identificación no paramétrica en el dominio
%del tiempo. La identificación se realiza utilizando el
System %Identification Toolbox de Matlab.
close all; clear all; clc;
%Se cargan los datos de la simulación del modelo de la
planta %almacenados en el archivo datosPFC.mat
load('datosPFC.mat');
time=tsim;
input=Pfeed;
output=Fpermeate;
%Estos datos de entrada y salida corresponden a la
respuesta del sistema a una entrada escalón del 10% en la
presión de la corriente de alimentación.
figure(1)
subplot(2,1,1)
plot(time,input,'LineWidth',2)
ylabel('Presión de la alimentación, bar');
grid on;
subplot(2,1,2)
plot(time,output,'LineWidth',2)
xlabel('Tiempo, s');
ylabel('Flujo de permeado, g/s');
grid on;

%Se determinan los valores de las variables de desviación
dtime=time;
dtime=dtime-dtime(30);
dtime(1:29)=[];
dinput=input;
dinput=dinput-dinput(30);
dinput(1:29)=[];
doutput=output;
doutput=doutput-doutput(30);
doutput(1:29)=[];
%Se utilizan las variables de desviación para la
identificación del sistema en el dominio del tiempo con el
System Identification Toolbox de MATLAB.
ident
%En el se importan los datos de entrada (dinput) y salida
(doutput) con un starting time = 0 y sampling inteval = 1.
Luego se ajustan a un modelo de segundo orden, sin ceros,
ni tiempo muerto, que se exportan al Workspace de Matlab
con el nombre MPF2orden, y finalmente, se guarda todo en
forma %manual el archivo PFident.mat usando la línea de
código:
save PFident
```

B.4 p04PreparacionDatosIdenPC.m

```
%Se calculan los valores de las variables de desviación a
%utilizar en la identificación no paramétrica en el
dominio %del tiempo. La identificación se realiza
utilizando el System %Identification Toolbox de Matlab.
close all; clear all; clc;
%Se cargan los datos de la simulación del modelo de la
planta %almacenados en el archivo datosPFC.mat
load('datosPFC.mat');
time=tsim;
input=Pfeed;
output=Condpermeate;
%Estos datos de entrada y salida corresponden a la
respuesta %del sistema a una entrada escalón del 10% en
la presión de la %corriente de alimentación
figure(1)
subplot(2,1,1)
plot(time,input,'LineWidth',2)
ylabel('Presión de la alimentación, bar');
grid on;
subplot(2,1,2)
plot(time,output,'LineWidth',2)
xlabel('Tiempo, s');
ylabel('Conductividad del permeado, uS/cm');
grid on;

%Se determinan los valores de las variables de desviación
dtime=time;
dtime=dtime-dtime(30);
dtime(1:29)=[];
dinput=input;
dinput=dinput-dinput(30);
dinput(1:29)=[];
doutput=output;
doutput=doutput-doutput(30);
doutput(1:29)=[];
%Se utilizan las variables de desviación para la
%identificación del sistema en el dominio del tiempo con
el %System Identification Toolbox de MATLAB.
ident
%En el se importan los datos de entrada (dinput) y salida
%(doutput) con un starting time = 0 y sampling inteval =
1. %Luego se ajustan a un modelo de segundo orden, sin
ceros, ni %tiempo muerto, que se exportan al Workspace de
Matlab con el %nombre MPC2orden, y finalmente, se guarda
todo en forma
%manual el archivo PCident.mat usando la línea de código:
save PCident
```

B.5 p05PreparacionDatosIdenpHC.m

```
%Se calculan los valores de las variables de desviación a
%utilizar en la identificación no paramétrica en el
dominio %del tiempo. La identificación se realiza
utilizando el System %Identification Toolbox de Matlab.
close all; clear all; clc;
%Se cargan los datos de la simulación del modelo de la
planta %almacenados en el archivo datosPFC.mat
load('datospHC.mat');
time=tsim;
input=pHfeed;
output=Condpermeate;
%Estos datos de entrada y salida corresponden a la
respuesta %del sistema a una entrada escalón del 10% en
la presión de la %corriente de alimentación
figure(1)
subplot(2,1,1)
plot(time,input,'LineWidth',2)
ylabel('pH de la alimentación, bar');
grid on;
subplot(2,1,2)
plot(time,output,'LineWidth',2)
xlabel('Tiempo, s');
ylabel('Conductivad del permeado, uS/cm');
grid on;

%Se determinan los valores de las variables de desviación
dtime=time;
dtime=dtime-dtime(1);
% dtime(1:1)=[];
dinput=input;
dinput=dinput-dinput(1);
% dinput(1:1)=[];
doutput=output;
doutput=doutput-doutput(1);
% doutput(1:1)=[];

%Se utilizan las variables de desviación para la
%identificación del sistema en el dominio del tiempo con
el %System Identification Toolbox de MATLAB.
ident
%En el se importan los datos de entrada (dinput) y salida
%(doutput) con un starting time = 0 y sampling inteval =
1. %Luego se ajustan a un modelo de segundo orden, sin
ceros, ni %tiempo muerto, que se exportan al Workspace de
Matlab con el %nombre MpHC2orden, y finalmente, se guarda
todo en forma %manual el archivo PFident.mat usando la
línea de código:
save pHCident
```

B.6 p06PFidentSim.m

```
%Se discretiza y lleva al espacio de estados el modelo
obtenido y también se realiza la simulación en lazo
abierto.
close all; clear all; clc;

load('PFident.mat');
%modelo de segundo orden con dos polos, sin ceros ni
tiempo muerto
Kp = MPF2orden.Kp; Tp1 = MPF2orden.Tp1; Tp2 =
MPF2orden.Tp2; Td = 0; Tz = 0; %Td = MPF2orden.Td; Tz =
MPF2orden.Tz;

%Modelo lineal continuo del sistema en el dominio de s
PFc1=tf([Kp*Tz Kp],[Tp1*Tp2 Tp1+Tp2 1],'iodelay',Td)
[num den]=tfdata(PFc1,'v')
PFb1=num(2)/den(1);%para que la ecuación característica
sea un polinómio mónico
PFb0=num(3)/den(1);
PFa1=den(2)/den(1);
PFa0=den(3)/den(1);
save PFGs PFb0 PFa0 PFa1
%Modelo lineal continuo en el espacio de estados (primera
forma, desarrollo propio)
%X=[x1 x2]'; x1p=x2
Ac1=[0 1; -PFa0 -PFa1]
Bc1=[0; PFb0]
Cc1=[1 0]
Dc1=[0]
PFc2=ss(Ac1,Bc1,Cc1,Dc1)
save PFsss2 PFc2

%Se discretiza el modelo
Ts=1;%1/100000*round(min(10000*[Tp1 Tp2]));%Ts= 1/10 de
la menor constante de tiempo
%Modelo lineal discreto del sistema en el dominio de z
PFz=c2d(PFc1,Ts,'zoh')
[numd dend]=tfdata(PFz,'v')
PFdb1=numd(2)/dend(1);%para que la ecuación
característica sea un polinómio mónico
PFdb0=numd(3)/dend(1);
PFda1=dend(2)/dend(1);
PFda0=dend(3)/dend(1);
%Modelo lineal discreto en el espacio de estados (primera
forma, a partir de la f.t. z)
Ad1=[-PFda1 1; -PFda0 0]
Bd1=[PFdb1; PFdb0]
Cd1=[1 0]
Dd1=[0]
PFd1=ss(Ad1,Bd1,Cd1,Dd1,Ts)
```

```
%respuesta al escalón del sistema continuo
U=dinput;
yGs=step(U(end)*PFc1,dtime);
figure(1);
subplot(2,1,1);
plot(dtime,doutput+Fpermeate(1),'o','LineWidth',2);
hold all;
plot(dtime,yGs+Fpermeate(1),'LineWidth',2);
xlabel('Tiempo, s');
ylabel('Flujo del permeado, ml/s');
legend('Modelo no lineal','Modelo lineal');
grid on;
subplot(2,1,2);
plot(dtime,dinput+Pfeed(1),'LineWidth',2);
xlabel('Tiempo,s');
ylabel('Presión de entrada, bar');
% axis([0 600 60 90]);
grid on;

%Comparación de las respuestas de los modelos discretos
%se presenta el modelo continuo y no lineal para
comparación
figure(2);
plot(dtime,doutput+Fpermeate(1),'o','LineWidth',2);
hold all;
plot(dtime,yGs+Fpermeate(1),'LineWidth',2);
xlabel('Tiempo,s');
ylabel('Flujo del permeado, g/s');
grid on;
%respuesta al escalón del modelo discreto en z
t=dtime(1):Ts:dtime(end);
U=dinput;
yGz=step(U(end)*PFz,t);
plot(t,yGz+Fpermeate(1),'d','LineWidth',2);
%respuesta al escalón de los modelos discretos en el
espacio de estados
%primera forma
yGz=step(U(end)*PFd1,t);
plot(t,yGz+Fpermeate(1),'--','LineWidth',2);
%segunda forma
x = [ 0;  0];
k = 1;
u = [0 linspace(U(end),U(end),length(t)-1)];
nn=length(t);
for n = 1:nn
    x1(k,1) = x(1,1);
    x2(k,1) = x(2,1);
    x = (Ad2)*x + Bd2*u(k);
    k = k + 1;
end
plot(t,x1+Fpermeate(1),'s','LineWidth',2);
```

B.7 p07PCidentSim.m

```
%Se discretiza y lleva al espacio de estados el modelo
obtenido y también se realiza la simulación en lazo
abierto.
close all; clear all; clc;

load('PCident.mat');
%modelo de segundo orden con dos polos, sin ceros ni
tiempo muerto
Kp = MPC2orden.Kp; Tp1 = MPC2orden.Tp1; Tp2 =
MPC2orden.Tp2; Td = 0; Tz = 0; %Td = MPC2orden.Td; Tz =
MPC2orden.Tz;

%Modelo lineal continuo del sistema en el dominio de s
PCc1=tf([Kp*Tz Kp],[Tp1*Tp2 Tp1+Tp2 1],'iodelay',Td)
[num den]=tfdata(PCc1,'v')
PCb1=num(2)/den(1);%para que la ecuación característica
sea un polinómio mónico
PCb0=num(3)/den(1);
PCa1=den(2)/den(1);
PCa0=den(3)/den(1);
save PCGs PCb0 PCa0 PCa1
%Modelo lineal continuo en el espacio de estados (primera
forma, desarrollo propio)
%X=[x1 x2]'; x1p=x2
Ac1=[0 1; -PCa0 -PCa1]
Bc1=[0; PCb0]
Cc1=[1 0]
Dc1=[0]
PCc2=ss(Ac1,Bc1,Cc1,Dc1)
save PCsss2 PCc2

%Se discretiza el modelo
Ts=1;%1/10000*round(min(1000*[Tp1 Tp2]));%Ts= 1/10 de la
menor constante de tiempo
%Modelo lineal discreto del sistema en el dominio de z
PCz=c2d(PCc1,Ts,'zoh')
[numd dend]=tfdata(PCz,'v')
PCdb1=numd(2)/dend(1);%para que la ecuación
característica sea un polinómio mónico
PCdb0=numd(3)/dend(1);
PCda1=dend(2)/dend(1);
PCda0=dend(3)/dend(1);
%Modelo lineal discreto en el espacio de estados (primera
forma, a partir de la f.t. z)
Ad1=[-PCda1 1; -PCda0 0]
Bd1=[PCdb1; PCdb0]
Cd1=[1 0]
Dd1=[0]
PCd1=ss(Ad1,Bd1,Cd1,Dd1,Ts)
```

```
%respuesta al escalón del sistema continuo
U=dinput;
yGs=step(U(end)*PCc1,dtime);
figure(1);
subplot(2,1,1);
plot(dtime,doutput+Condpermeate(1),'o','LineWidth',2);
hold all;
plot(dtime,yGs+Condpermeate(1),'LineWidth',2);
xlabel('Tiempo, s');
ylabel('Conductividad del permeado, uS/cm');
legend('Modelo no lineal','Modelo lineal');
grid on;
subplot(2,1,2);
plot(dtime,dinput+Pfeed(1),'LineWidth',2);
xlabel('Tiempo,s');
ylabel('Presión de entrada, bar');
grid on;

%Comparación de las respuestas de los modelos discretos
%se presenta el modelo continuo y no lineal para
comparación
figure(2);
plot(dtime,doutput+Condpermeate(1),'o','LineWidth',2);
hold all;
plot(dtime,yGs+Condpermeate(1),'LineWidth',2);
xlabel('Tiempo,s');
ylabel('Conductividad del permeado, uS/cm');
grid on;
%respuesta al escalón del modelo discreto en el dominio
de z
t=dtime(1):Ts:dtime(end);
U=dinput;
yGz=step(U(end)*PCz,t);
plot(t,yGz+Condpermeate(1),'d','LineWidth',2);
%respuesta al escalón de los modelos discretos en el
espacio de estados
%primera forma
yGz=step(U(end)*PCd1,t);
plot(t,yGz+Condpermeate(1),'--','LineWidth',2);
%segunda forma
x = [ 0;  0];
k = 1;
u = [0 linspace(U(end),U(end),length(t)-1)];
nn=length(t);
for n = 1:nn
    x1(k,1) = x(1,1);
    x2(k,1) = x(2,1);
    x = (Ad2)*x + Bd2*u(k);
    k = k + 1;
end
plot(t,x1+Condpermeate(1),'s','LineWidth',2);
```

B.8 p08pHCidentSim.m

```
%Se discretiza y lleva al espacio de estados el modelo
obtenido y también se realiza la simulación en lazo
abierto.
close all; clear all; clc;

load('pHCident.mat');
%modelo de segundo orden con dos polos, sin ceros ni
tiempo muerto
Kp = MpHC2orden.Kp; Tp1 = MpHC2orden.Tp1; Tp2 =
MpHC2orden.Tp2; Td = 0; Tz = 0; %Td = MpHC2orden.Td; Tz =
MpHC2orden.Tz;

%Modelo lineal continuo del sistema en el dominio de s
pHCc1=tf([Kp*Tz Kp],[Tp1*Tp2 Tp1+Tp2 1],'iodelay',Td)
[num den]=tfdata(pHCc1,'v')
pHCb1=num(2)/den(1);%para que la ecuación característica
sea un polinómio mónico
pHCb0=num(3)/den(1);
pHCa1=den(2)/den(1);
pHCa0=den(3)/den(1);
save pHCGs pHCb0 pHCa0 pHCa1
%Modelo lineal continuo en el espacio de estados (primera
forma, desarrollo propio)
%X=[x1 x2]'; x1p=x2
Ac1=[0 1; -pHCa0 -pHCa1]
Bc1=[0; pHCb0]
Cc1=[1 0]
Dc1=[0]
pHCc2=ss(Ac1,Bc1,Cc1,Dc1)
save pHCsss2 pHCc2

%Se discretiza el modelo
Ts=1;%1/10000*round(min(1000*[Tp1 Tp2]));%Ts= 1/10 de la
menor constante de tiempo
%Modelo lineal discreto del sistema en el dominio de z
pHCz=c2d(pHCc1,Ts,'zoh')
[numd dend]=tfdata(pHCz,'v')
pHCdb1=numd(2)/dend(1);%para que la ecuación
característica sea un polinómio mónico
pHCdb0=numd(3)/dend(1);
pHCda1=dend(2)/dend(1);
pHCda0=dend(3)/dend(1);
%Modelo lineal discreto en el espacio de estados (primera
forma, a partir de la f.t. z)
Ad1=[-pHCda1 1; -pHCda0 0]
Bd1=[pHCdb1; pHCdb0]
Cd1=[1 0]
Dd1=[0]
pHCd1=ss(Ad1,Bd1,Cd1,Dd1,Ts)
```

```
%respuesta al escalón del sistema continuo
U=dinput;
yGs=step(U(end)*pHCc1,dtime);
figure(1);
subplot(2,1,1);
plot(dtime,doutput+Condpermeate(1),'o','LineWidth',2);
hold all;
plot(dtime,yGs+Condpermeate(1),'LineWidth',2);
xlabel('Tiempo, s');
ylabel('Conductividad del permeado, uS/cm');
legend('Modelo no lineal','Modelo lineal');
grid on;
subplot(2,1,2);
plot(dtime,dinput+pHfeed(1),'LineWidth',2);
xlabel('Tiempo,s');
ylabel('pH de la alimentación');
% axis([0 600 60 90]);
grid on;

%Comparación de las respuestas de los modelos discretos
%se presenta el modelo continuo y no lineal para
comparación
figure(2);
plot(dtime,doutput+Condpermeate(1),'o','LineWidth',2);
hold all;
plot(dtime,yGs+Condpermeate(1),'LineWidth',2);
xlabel('Tiempo,s');
ylabel('Conductividad del permeado, uS/cm');
grid on;
%respuesta al escalón del modelo discreto en z
t=dtime(1):Ts:dtime(end);
U=dinput;
yGz=step(U(end)*pHCz,t);
plot(t,yGz+Condpermeate(1),'d','LineWidth',2);
%respuesta al escalón de los modelos discretos en el
espacio de estados
%primera forma
yGz=step(U(end)*pHCd1,t);
plot(t,yGz+Condpermeate(1),'--','LineWidth',2);
%segunda forma
x = [ 0;  0];
k = 1;
u = [0 linspace(U(end),U(end),length(t)-1)];
nn=length(t);
for n = 1:nn
    x1(k,1) = x(1,1);
    x2(k,1) = x(2,1);
    x = (Ad2)*x + Bd2*u(k);
    k = k + 1;
end
plot(t,x1+Condpermeate(1),'s','LineWidth',2);
```

ANEXO C

PROGRAMAS PARA EL DISEÑO DEL NEUROCONTROLADOR Y SU IMPLEMENTACIÓN EN SIMULINK

C.1 p05MultivariableNeuralController.m

```
%=========================================================
%Entrenamiento de un Neurocontrolador Dinámico para el
%control del Flujo y la Conductividad del Permeado en una
%Planta de Desalinización por O.I.
%Memoria de Tesis: "Modelado y Control Basado en Redes
%Neuronales Artificiales de una Planta Piloto de
%Desalinización de Agua de Mar por Ósmosis Inversa"
%=========================================================

%Se entrena el controlador neuronal para varias
condiciones %iniciales utilizando el algoritmo de
entrenamiento: Dynamic Back Propagation (DBP)
%=========================================================
clear all; close all; clc;
disp('ENTRENAMIENTO DEL CONTROLADOR NEURONAL
MULTIVARIABLE, ALGORITMO DBP')
disp(' ')

% PARA EL SISTEMA DISCRETO EN EL ESPACIO DE ESTADOS:
% x_(k+1) = Ak*x_k + Bk*u_k
% y_k = Ck*x_k
% los estados, las entradas y las salidas son:
% estados
% x1 = [FPP-FPPss]
% x2 = [FPPp],  x2ss = 0
% x3 = [CPP-CPPss]
% x4 = [CPPp],  x4ss = 0
% x5 = [CPpH-CPpHss]
% x6 = [CPpHp], x6ss = 0
% entradas
% u1 = [PA-PAss]
% u2 = [pHA-pHAss]
% salidas
% y1 = [FP-FPss]
% y2 = [CP-CPss]
% con:
% FP = Flujo de la corriente de permeado, ml/s
% CP = Conductividad de la corriente de permeado, uS/cm
% PA = Presión de la corriente de alimentación, bar
% pHA = pH de la corriente de alimentación, adimensional
% FPP: Flujo de permeado debido a la presión de la
alimentación
% FPPp: Tasa de cambio del Flujo de permeado debido a la
presión
% CPP: Conductividad del permeado debido a la presión de
la alimentación
% CPPp: Tasa de cambio de la Conductividad del permeado
debido a la presión
```

```
% CPpH: Conductividad del permeado debido al pH de la
alimentación
% CPpHp: Tasa de cambio Conductividad del permeado debido
al pH
%-----------------------------------------------------------
% Además, las matrices correspondientes son:
load('SS2system.mat');
Ts=1;
[Ak Bk]=c2d(A,B,Ts)
Ck = C
Dk = D

%Para entrenar el controlador neuronal multivariable se
utilizan las mismas condiciones de operación (inicial y
final) usadas para evaluar el desempeño de los
controladores PID. Es decir, se partirá de la condición:
%entradas
PAss  = 63.06654;%bar
pHAss = 6.45;
%salidas
FPss = 66.264;%ml/s
CPss = 439.06;%uS/cm

%Por comodidad, para el proceso: las variables de entrada
se llamarán control,las de salida se llamarán respuesta,
y las variables con las que se definen los estados se
llamarán variables_e. En estado estacionario, todas estas
%variables tienen los valores:
control1ss = PAss;
control2ss = pHAss;
controlss  = [control1ss; control2ss]
respuesta1ss = FPss;
respuesta2ss = CPss;
respuestass  = [respuesta1ss; respuesta2ss]
variable_e1ss = respuesta1ss;
variable_e2ss = 0;
variable_e3ss = respuesta2ss;%32 es la contribución del
pH a la conductividad
variable_e4ss = 0;           %desde pH 7 hasta 6.35
variable_e5ss = 0;
variable_e6ss = 0;
variables_ess = [variable_e1ss; variable_e2ss;
variable_e3ss; variable_e4ss; variable_e5ss;
variable_e6ss]
%observe que, en el modelo no lineal obtenido, en
correspondencia con la referencia (Alatiqi et al., 1989)
se considera que la contribución a la conductividad del
pH, a pH=6.45, es cero.
%Observe además que, independientemente de la elección
del estado estacionario inicial, todas las variables de
```

```
desviación en ese punto son cero, es decir, las entradas,
los estados y las salidas del modelo son cero.
u1ss = 0;
u2ss = 0;
uss  =[u1ss; u2ss];
y1ss = 0;
y2ss = 0;
yss  =[y1ss; y2ss];
x1ss = 0;
x2ss = 0;
x3ss = 0;
x4ss = 0;
x5ss = 0;
x6ss = 0;
xss  = [x1ss; x2ss; x3ss; x4ss; x5ss; x6ss];
%Se llegará a la condición en la que, el flujo de
permeado sea el 105% de su valor inicial manteniendose la
conductividad en el mismo valor inicial.

%Lo anterior significa que el estado final deseado
resulta de resolver el sistema de ecuaciones:
%y_k = Ck*x_k, con  y = [y1; y2], x = [x1; x2;...;x6]
%x_k = Ak*x_k + Bk*u_k, con u = [u1; u2]
%La primera ecuación matricial permite obtener:
%0.05*respuesta1 = x1d;
%               0 = x3d + x5d;
%y la segunda se puede escribir de forma conveniente
como:
%x_k = inv(eye(6)-Ak)*Bk*u_k
%al definir AA = inv(eye(6)-Ak)*Bk, se encuentra:
%x1d = AA(1,1)*u1d + AA(1,2)*u2d
%x2d = AA(2,1)*u1d + AA(2,2)*u2d
%x3d = AA(3,1)*u1d + AA(3,2)*u2d
%x4d = AA(4,1)*u1d + AA(4,2)*u2d
%x5d = AA(5,1)*u1d + AA(5,2)*u2d
%x6d = AA(6,1)*u1d + AA(6,2)*u2d
%sistema de ecuaciones que se reduce aún más:
%x1d = AA(1,1)*u1d
%x2d = AA(2,1)*u1d
%x3d = AA(3,1)*u1d
%x4d = AA(4,1)*u1d
%x5d = AA(5,2)*u2d
%x6d = AA(6,2)*u2d
%con la información disponible, se resuelve y obtiene:
AA = inv(eye(6)-Ak)*Bk
y1d = 0.05*respuesta1ss
x1d = y1d
u1d = 1/AA(1,1)*x1d
x2d = AA(2,1)*u1d
x3d = AA(3,1)*u1d
x4d = AA(4,1)*u1d
```

```
y2d = 0
x5d = -x3d
u2d = 1/AA(5,2)*x5d
x6d = AA(6,2)*u2d
%A partir de los valores anteriores, los estados deseados
son:
xd  = [x1d; x2d; x3d; x4d; x5d; x6d]
%y los valores deseados para las señales de control y
salidas "y":
ud  = [u1d; u2d]
yd  = [y1d; y2d]

%El conjunto de condiciones iniciales utilizadas para el
entrenamiento es:
cond_ini = [xss];

% Se realiza el escalamiento de las señales de entrada y
salida a la red neuronal al rango de [- 1, 1] para
facilitar el entrenamiento.
% Para las variables de entrada a la red neuronal se
considera lo siguiente:
% x1 - x1d, Flujo de permeado - Flujo deseado de permeado
= +- 10 ml/s alrededor del punto de operación
% x2 - x2d, Tasa de cambio del delta de flujo de permeado
+- 0.5 ml/s2 alrededor del punto de operación
% x3 - x3d, Conductividad del permeado debida a la
presión - Conductividad deseada del permeado debida a la
presión = +- 50 uS/cm alrededor del punto de operación
% x4 - x4d, Tasa de cambio del delta de Conductividad del
permeado debida a la presión +- 5 uS/cm/s alrededor del
punto de operación
% x5 - x5d, Conductividad del permeado debida al pH -
Conductividad deseada del permeado debida al pH = +- 50
uS/cm alrededor del punto de operación
% x6 - x6d, Tasa de cambio del delta de Conductividad del
permeado debida al pH +- 5 uS/cm/s alrededor del punto de
operación
% Para la variable de salida de la red neuronal se
considera lo siguiente:
% u1 - u1d, Presión de la alimentación - Presión deseada
de la alimentación = +- 5 bar alrededor del punto de
operación
% u - ud, pH de la alimentación - pH deseado de la
alimentación = +- 0.5ME alrededor del punto de operación
% Haciendo las correspondencias correctas, se tendrán los
factores de escalamiento m siguientes:
% Para x1:
me1=(1 - (-1))/(10 - (-10));
% Para x2:
me2=(1 - (-1))/(0.5 - (-0.5));
% Para x3:
```

```
me3=(1 - (-1))/(50 - (-50));
% Para x4:
me4=(1 - (-1))/(5 - (-5));
% Para x5:
me5=(1 - (-1))/(50 - (-50));
% Para x6:
me6=(1 - (-1))/(5 - (-5));
% Para u1:
ms1=(5 - (-5))/(1 - (-1));
% Para u2:
ms2=(0.5 - (-0.5))/(1 - (-1));
me=[me1;me2;me3;me4;me5;me6]
ms=[ms1;ms2]
%y las matrices de transformación
ME=diag(me)
MS=diag(ms)
save M ME MS

%definidos estos factores de escalamiento, se calculan
los valores escalados de todas las variables y se asignan
a las mismas variables (esto con el fin de no usar mas
variables en el programa)

Ak=ME*Ak*inv(ME)
Bk=ME*Bk*MS
Ck=Ck*inv(ME)

xss=ME*xss
uss=inv(MS)*uss
yss=Ck*xss

xd=ME*xd
ud=inv(MS)*ud
yd=Ck*xd

cond_ini=ME*cond_ini

ne = 6; %número de neuronas en la capa de entrada
nm = 3; %número de neuronas en la capa intermedia
ns = 2; %número de neuronas en la capa de salida

%Posibilidad de cargar los pesos guardados en
entrenamientos previos o usar valores iniciales
aleatorios
resp = menu('Desea utilizar los pesos guardados en
entrenamientos previos','SI','NO');
if resp == 1
load pesosMVnm3.mat
[ne_old,nm_old] = size(v);
if nm_old ~= nm
```

```
disp('Usted no puede utilizar los pesos guardados porque
estos corresponden a un número diferente de neuronas en
la capa intermedia. El número de neuronas con el que se
entrenó previamente la red es: ');
nm_old
break
end
else
v = 0.025*randn(ne,nm); %valores iniciales aleatorios
para los pesos v
w = 0.025*randn(nm,ns); %valores iniciales aleatorios
para los pesos w
c = zeros(nm,1); %centros de las funciones de activación
a = ones(nm,1);  %inclinaciones de las funciones de
activación
end

eta  = 0.0005;%ratio de aprendizaje de los pesos v y w
etac = 0.0000;%ratio de aprendizaje de los centros c
(ceros)
etaa = 0.0001;%ratio de aprendizaje de la inclinación a

ndata = 50; %número de veces que se integra la ecuación
de estado con cada combinación de pesos
n_max_iter_tot = 5000;%número máximo de iteraciones
totales a realizar, número de actualizaciones de pesos
%esto significa que n_max_iter_tot/ndata será el número
de veces que actualizan los pesos
error_tol = 0.01;%tolerancia para el error, una vez que
el error sea menor que error_tol, se aceptan los pesos y
se detiene el entrenamiento

r = [0 0 0 0 0 0]';
x_xd_buscado = ones(ndata,ne)*diag(r);%diferencia buscada
entre x y xd
dt = Ts; t=0;
niter = 1;%inicialización del contador del número de
iteraciones realizadas en el entrenamiento
n_ini = 1;%número de condiciones iniciales usadas para el
entrenamiento
error_max_per = 10000000;%se inicializa el error máximo
posible en un valor suficientemente grande
error = error_max_per;
while( (error > error_tol) & (niter < n_max_iter_tot) )
    for c_ini=1:n_ini
    ersum2 = zeros(ne,1);
    dx1dw_t = zeros(nm,ns);
    dx2dw_t = zeros(nm,ns);
    dx3dw_t = zeros(nm,ns);
    dx4dw_t = zeros(nm,ns);
    dx5dw_t = zeros(nm,ns);
```

```
        dx6dw_t = zeros(nm,ns);
        dx1dv_t = zeros(ne,nm);
        dx2dv_t = zeros(ne,nm);
        dx3dv_t = zeros(ne,nm);
        dx4dv_t = zeros(ne,nm);
        dx5dv_t = zeros(ne,nm);
        dx6dv_t = zeros(ne,nm);
        dx1dc_t = zeros(nm,1);
        dx2dc_t = zeros(nm,1);
        dx3dc_t = zeros(nm,1);
        dx4dc_t = zeros(nm,1);
        dx5dc_t = zeros(nm,1);
        dx6dc_t = zeros(nm,1);
        dx1da_t = zeros(nm,1);
        dx2da_t = zeros(nm,1);
        dx3da_t = zeros(nm,1);
        dx4da_t = zeros(nm,1);
        dx5da_t = zeros(nm,1);
        dx6da_t = zeros(nm,1);
        dJdw_t = zeros(nm,ns);
        dJdv_t = zeros(ne,nm);
        dJdc_t = zeros(nm,1);
        dJda_t = zeros(nm,1);
        %Condiciones inciales
        x = cond_ini(:,c_ini);
        t = 0;
            for k = 1:ndata-1
            %Entrada a la red neuronal, valores escalados
            in_red = x-xd;
            %Cálculos internos de la red neuronal
            m = v'*in_red;
            n = 2.0./(1 + exp(-(m-c)./a)) - 1;%función de
            activación sigmoidea 2
            out_red = w'*n;
            %Señal de control, desescalamiento y saturación
            u = out_red + ud;
            ureal = MS*u;
            if ureal(1) >  5; ureal(1) =  5; end
            if ureal(1) < -5; ureal(1) = -5; end
            if ureal(2) >  0.5; ureal(2) =  0.5; end
            if ureal(2) < -0.5; ureal(2) = -0.5; end
            u = inv(MS)*(ureal);
            control(k,:,c_ini) = u';%en variables de
desviación
            escaladas
            estado(k,:,c_ini) = x';%en variables de
desviación
            escaladas
            y = Ck*(x);%en variables de desviación reales
            salida(k,:,c_ini) = y';
            tsim(k,:,c_ini)=t;
```

```
        t=t+dt;
        %Cálculo de derivadas para la actualización de
los
        pesos de la red neuronal
        % Jacobiano = dX(k+1)/dX(k)
        jacob = Ak;
        % dxdu = dX(k+1)/dU(k)
        dxdu  = Bk;
        x = Ak*(x) + Bk*(u);
        % dU(k)/dW
        dndm = diag((1 - n.*n)./(2*a));
        dudx = w'*dndm*v';
        dudw_s = n;
        dudv_s1 = in_red*w(:,1)'*dndm;% s1 = salida 1
        dudv_s2 = in_red*w(:,2)'*dndm;
        dndc = ((n.*n-1)./(2.0.*a));
        dudc_s = [dndc.* w(:,1) dndc.* w(:,2)];
        dnda = ((n.*n-1).*(m-c)./(2*a.*a));
        duda_s = [dnda.* w(:,1) dnda.* w(:,2)];
        jacob_t = dxdu*dudx + jacob;
        %Actualizacion de derivadas en la Red Neuronal
        dx1dw_t = [dxdu(1,1).*dudw_s dxdu(1,2).*dudw_s] +
jacob_t(1,1).*dx1dw_t + jacob_t(1,2).*dx2dw_t +
jacob_t(1,3).*dx3dw_t + jacob_t(1,4).*dx4dw_t +
jacob_t(1,5).*dx5dw_t + jacob_t(1,6).*dx6dw_t;
        dx2dw_t = [dxdu(2,1).*dudw_s dxdu(2,2).*dudw_s] +
jacob_t(2,1).*dx1dw_t + jacob_t(2,2).*dx2dw_t +
jacob_t(2,3).*dx3dw_t + jacob_t(2,4).*dx4dw_t +
jacob_t(2,5).*dx5dw_t + jacob_t(2,6).*dx6dw_t;
        dx3dw_t = [dxdu(3,1).*dudw_s dxdu(3,2).*dudw_s] +
jacob_t(3,1).*dx1dw_t + jacob_t(3,2).*dx2dw_t +
jacob_t(3,3).*dx3dw_t + jacob_t(3,4).*dx4dw_t +
jacob_t(3,5).*dx5dw_t + jacob_t(3,6).*dx6dw_t;
        dx4dw_t = [dxdu(4,1).*dudw_s dxdu(4,2).*dudw_s] +
jacob_t(4,1).*dx1dw_t + jacob_t(4,2).*dx2dw_t +
jacob_t(4,3).*dx3dw_t + jacob_t(4,4).*dx4dw_t +
jacob_t(4,5).*dx5dw_t + jacob_t(4,6).*dx6dw_t;
        dx5dw_t = [dxdu(5,1).*dudw_s dxdu(5,2).*dudw_s] +
jacob_t(5,1).*dx1dw_t + jacob_t(5,2).*dx2dw_t +
jacob_t(5,3).*dx3dw_t + jacob_t(5,4).*dx4dw_t +
jacob_t(5,5).*dx5dw_t + jacob_t(5,6).*dx6dw_t;
        dx6dw_t = [dxdu(6,1).*dudw_s dxdu(6,2).*dudw_s] +
jacob_t(6,1).*dx1dw_t + jacob_t(6,2).*dx2dw_t +
jacob_t(6,3).*dx3dw_t + jacob_t(6,4).*dx4dw_t +
jacob_t(6,5).*dx5dw_t + jacob_t(6,6).*dx6dw_t;

        dx1dv_t = (dxdu(1,1).*dudv_s1 +
dxdu(1,2).*dudv_s2) + jacob_t(1,1).*dx1dv_t +
jacob_t(1,2).*dx2dv_t + jacob_t(1,3).*dx3dv_t +
jacob_t(1,4).*dx4dv_t + jacob_t(1,5).*dx5dv_t +
jacob_t(1,6).*dx6dv_t;
```

```
        dx2dv_t = (dxdu(2,1).*dudv_s1 +
dxdu(2,2).*dudv_s2) + jacob_t(2,1).*dx1dv_t +
jacob_t(2,2).*dx2dv_t + jacob_t(2,3).*dx3dv_t +
jacob_t(2,4).*dx4dv_t + jacob_t(2,5).*dx5dv_t +
jacob_t(2,6).*dx6dv_t;
        dx3dv_t = (dxdu(3,1).*dudv_s1 +
dxdu(3,2).*dudv_s2) + jacob_t(3,1).*dx1dv_t +
jacob_t(3,2).*dx2dv_t + jacob_t(3,3).*dx3dv_t +
jacob_t(3,4).*dx4dv_t + jacob_t(3,5).*dx5dv_t +
jacob_t(3,6).*dx6dv_t;
        dx4dv_t = (dxdu(4,1).*dudv_s1 +
dxdu(4,2).*dudv_s2) + jacob_t(4,1).*dx1dv_t +
jacob_t(4,2).*dx2dv_t + jacob_t(4,3).*dx3dv_t +
jacob_t(4,4).*dx4dv_t + jacob_t(4,5).*dx5dv_t +
jacob_t(4,6).*dx6dv_t;
        dx5dv_t = (dxdu(5,1).*dudv_s1 +
dxdu(5,2).*dudv_s2) + jacob_t(5,1).*dx1dv_t +
jacob_t(5,2).*dx2dv_t + jacob_t(5,3).*dx3dv_t +
jacob_t(5,4).*dx4dv_t + jacob_t(5,5).*dx5dv_t +
jacob_t(5,6).*dx6dv_t;
        dx6dv_t = (dxdu(6,1).*dudv_s1 +
dxdu(6,2).*dudv_s2) + jacob_t(6,1).*dx1dv_t +
jacob_t(6,2).*dx2dv_t + jacob_t(6,3).*dx3dv_t +
jacob_t(6,4).*dx4dv_t + jacob_t(6,5).*dx5dv_t +
jacob_t(6,6).*dx6dv_t;

        dx1dc_t = (dxdu(1,1).*dudc_s(:,1) +
dxdu(1,2).*dudc_s(:,2)) + jacob_t(1,1).*dx1dc_t +
jacob_t(1,2).*dx2dc_t + jacob_t(1,3).*dx3dc_t +
jacob_t(1,4).*dx4dc_t + jacob_t(1,5).*dx5dc_t +
jacob_t(1,6).*dx6dc_t;
        dx2dc_t = (dxdu(2,1).*dudc_s(:,1) +
dxdu(2,2).*dudc_s(:,2)) + jacob_t(2,1).*dx1dc_t +
jacob_t(2,2).*dx2dc_t + jacob_t(2,3).*dx3dc_t +
jacob_t(2,4).*dx4dc_t + jacob_t(2,5).*dx5dc_t +
jacob_t(2,6).*dx6dc_t;
        dx3dc_t = (dxdu(3,1).*dudc_s(:,1) +
dxdu(3,2).*dudc_s(:,2)) + jacob_t(3,1).*dx1dc_t +
jacob_t(3,2).*dx2dc_t + jacob_t(3,3).*dx3dc_t +
jacob_t(3,4).*dx4dc_t + jacob_t(3,5).*dx5dc_t +
jacob_t(3,6).*dx6dc_t;
        dx4dc_t = (dxdu(4,1).*dudc_s(:,1) +
dxdu(4,2).*dudc_s(:,2)) + jacob_t(4,1).*dx1dc_t +
jacob_t(4,2).*dx2dc_t + jacob_t(4,3).*dx3dc_t +
jacob_t(4,4).*dx4dc_t + jacob_t(4,5).*dx5dc_t +
jacob_t(4,6).*dx6dc_t;
        dx5dc_t = (dxdu(5,1).*dudc_s(:,1) +
dxdu(5,2).*dudc_s(:,2)) + jacob_t(5,1).*dx1dc_t +
jacob_t(5,2).*dx2dc_t + jacob_t(5,3).*dx3dc_t +
jacob_t(5,4).*dx4dc_t + jacob_t(5,5).*dx5dc_t +
jacob_t(5,6).*dx6dc_t;
```

```
        dx6dc_t = (dxdu(6,1).*dudc_s(:,1) +
dxdu(6,2).*dudc_s(:,2)) + jacob_t(6,1).*dx1dc_t +
jacob_t(6,2).*dx2dc_t + jacob_t(6,3).*dx3dc_t +
jacob_t(6,4).*dx4dc_t + jacob_t(6,5).*dx5dc_t +
jacob_t(6,6).*dx6dc_t;

        dx1da_t = (dxdu(1,1).*duda_s(:,1) +
dxdu(1,2).*duda_s(:,2)) + jacob_t(1,1).*dx1da_t +
jacob_t(1,2).*dx2da_t + jacob_t(1,3).*dx3da_t +
jacob_t(1,4).*dx4da_t + jacob_t(1,5).*dx5da_t +
jacob_t(1,6).*dx6da_t;
        dx2da_t = (dxdu(2,1).*duda_s(:,1) +
dxdu(2,2).*duda_s(:,2)) + jacob_t(2,1).*dx1da_t +
jacob_t(2,2).*dx2da_t + jacob_t(2,3).*dx3da_t +
jacob_t(2,4).*dx4da_t + jacob_t(2,5).*dx5da_t +
jacob_t(2,6).*dx6da_t;
        dx3da_t = (dxdu(3,1).*duda_s(:,1) +
dxdu(3,2).*duda_s(:,2)) + jacob_t(3,1).*dx1da_t +
jacob_t(3,2).*dx2da_t + jacob_t(3,3).*dx3da_t +
jacob_t(3,4).*dx4da_t + jacob_t(3,5).*dx5da_t +
jacob_t(3,6).*dx6da_t;
        dx4da_t = (dxdu(4,1).*duda_s(:,1) +
dxdu(4,2).*duda_s(:,2)) + jacob_t(4,1).*dx1da_t +
jacob_t(4,2).*dx2da_t + jacob_t(4,3).*dx3da_t +
jacob_t(4,4).*dx4da_t + jacob_t(4,5).*dx5da_t +
jacob_t(4,6).*dx6da_t;
        dx5da_t = (dxdu(5,1).*duda_s(:,1) +
dxdu(5,2).*duda_s(:,2)) + jacob_t(5,1).*dx1da_t +
jacob_t(5,2).*dx2da_t + jacob_t(5,3).*dx3da_t +
jacob_t(5,4).*dx4da_t + jacob_t(5,5).*dx5da_t +
jacob_t(5,6).*dx6da_t;
        dx6da_t = (dxdu(6,1).*duda_s(:,1) +
dxdu(6,2).*duda_s(:,2)) + jacob_t(6,1).*dx1da_t +
jacob_t(6,2).*dx2da_t + jacob_t(6,3).*dx3da_t +
jacob_t(6,4).*dx4da_t + jacob_t(6,5).*dx5da_t +
jacob_t(6,6).*dx6da_t;
        %Cálculo del error total acumulado
        out_des = x_xd_buscado(k+1,:);%diferencia entre x
y xd
        en cada instante
        out_des = out_des';
        er = (in_red - out_des);
        erJ = (in_red - out_des).^1;
        dJdw_t = dJdw_t + erJ(1,1).*dx1dw_t +
erJ(2,1).*dx2dw_t + erJ(3,1).*dx3dw_t + erJ(4,1).*dx4dw_t
+ erJ(5,1).*dx5dw_t + erJ(6,1).*dx6dw_t;
        dJdv_t = dJdv_t + erJ(1,1).*dx1dv_t +
erJ(2,1).*dx2dv_t + erJ(3,1).*dx3dv_t + erJ(4,1).*dx4dv_t
+ erJ(5,1).*dx5dv_t + erJ(6,1).*dx6dv_t;
```

```
        dJdc_t = dJdc_t + erJ(1,1).*dx1dc_t +
erJ(2,1).*dx2dc_t + erJ(3,1).*dx3dc_t + erJ(4,1).*dx4dc_t
+ erJ(5,1).*dx5dc_t + erJ(6,1).*dx6dc_t;
        dJda_t = dJda_t + erJ(1,1).*dx1da_t +
erJ(2,1).*dx2da_t + erJ(3,1).*dx3da_t + erJ(4,1).*dx4da_t
+ erJ(5,1).*dx5da_t + erJ(6,1).*dx6da_t;
        ersum2 = ersum2 + er.^2;
        end
    %Se calcula el promedio de todas las derivadas
    dJdw_t = dJdw_t/k;
    dJdv_t = dJdv_t/k;
    dJdc_t = dJdc_t/k;
    dJda_t = dJda_t/k;
    %Se almacenan las derivadas de cada condición inicial
en
    las variables
    %DJw, DJv, DJa, DJb
    DJw1(:,c_ini) = dJdw_t(:,1);
    DJw2(:,c_ini) = dJdw_t(:,2);
    DJv1(c_ini,:) = dJdv_t(1,:);
    DJv2(c_ini,:) = dJdv_t(2,:);
    DJv3(c_ini,:) = dJdv_t(3,:);
    DJv4(c_ini,:) = dJdv_t(4,:);
    DJv5(c_ini,:) = dJdv_t(5,:);
    DJv6(c_ini,:) = dJdv_t(6,:);
    DJa(:,c_ini) = dJda_t(:,1);
    DJc(:,c_ini) = dJdc_t(:,1);
    end
    %Se promedian las derivadas obtenidas para cada
condición
    inicial
    for aux=1:nm
    DJW1(aux,1) = sum(DJw1(aux,:))/n_ini;
    DJW2(aux,1) = sum(DJw2(aux,:))/n_ini;
    DJV1(1,aux) = sum(DJv1(:,aux))/n_ini;
    DJV2(1,aux) = sum(DJv2(:,aux))/n_ini;
    DJV3(1,aux) = sum(DJv3(:,aux))/n_ini;
    DJV4(1,aux) = sum(DJv4(:,aux))/n_ini;
    DJV5(1,aux) = sum(DJv5(:,aux))/n_ini;
    DJV6(1,aux) = sum(DJv6(:,aux))/n_ini;
    DJA(aux,1) = sum(DJa(aux,:))/n_ini;
    DJC(aux,1) = sum(DJc(aux,:))/n_ini;
    end
%Se actualizan los pesos
dw = [DJW1 DJW2];
dv = [DJV1; DJV2; DJV3; DJV4; DJV5; DJV6];
da = DJA;
dc = DJC;
w = w - eta*dw;
v = v - eta*dv;
c = c - etac*dc;
```

```
a = a - etaa*da;
error = sum(ersum2)
niter = niter + 1;
error_max_per = error;
end
%valores de los estados (variables de desviación) reales
estado=estado*inv(ME);
x1=estado(:,1);
x2=estado(:,2);
x3=estado(:,3);
x4=estado(:,4);
x5=estado(:,5);
x6=estado(:,6);
%señales de control (variables de desviación) reales
control=control*MS;
u1=control(:,1);
u2=control(:,2);
%señales de salida (variables de desviación) reales
y1=salida(:,1);
y2=salida(:,2);

Fpermeate = y1 + respuesta1ss;
SPFpermeate = [respuesta1ss
linspace(y1d+respuesta1ss,y1d+respuesta1ss,length(y1)-
1)];
Pfeed = u1 + control1ss;

Condp = y2 + respuesta2ss;
SPCondp = [respuesta2ss
linspace(y2d+respuesta2ss,y2d+respuesta2ss,length(y2)-
1)];
pHff = u2 + control2ss;

%variable controlada y variable manipulada
figure(1);
plot(tsim,Fpermeate,'LineWidth',2);
hold all;
plot(tsim,SPFpermeate,'LineWidth',2);
xlabel('Tiempo, s');
ylabel('Flujo de la corriente de permeado, ml/s');
legend('Variable controlada','Referencia');
axis([0 t 65 70])
grid on;

figure(2);
plot(tsim,Pfeed,'LineWidth',2);
xlabel('Tiempo, s');
ylabel('Presión de la corriente de alimentación, bar');
legend('Variable manipulada');
axis([0 t 62 66])
grid on;
```

```
figure(3);
plot(tsim,Condp,'LineWidth',2);
hold all;
plot(tsim,SPCondp,'--','LineWidth',2);
xlabel('Tiempo, s');
ylabel('Conductividad de la corriente de permeado,
uS/cm');
legend('Variable controlada','Referencia');
axis([0 t 400 500])
grid on;

figure(4);
plot(tsim,pHff,'LineWidth',2);
xlabel('Tiempo, s');
ylabel('pH de la corriente de alimentación');
legend('Variable manipulada');
axis([0 t 6 7])
grid on;
```

C.2 p06MultivariableNeuralControllerValidation.m

```
%=========================================================
% Validación del entrenamiento de un Neurocontrolador
Dinámico para el Flujo y la Conductividad del Permeado en
una Planta de Desalinización por O.I.
% Memoria de Tesis: "Modelado y Control Basado en Redes
Neuronales Artificiales de una Planta Piloto de
Desalinización de Agua de Mar por Ósmosis Inversa"
%=========================================================
clear all; close all; clc;
disp('VALIDACIÓN DEL ENTRENAMIENTO DEL CONTROLADOR
NEURONAL MULTIVARIABLE')
disp(' ')

% PARA EL SISTEMA DISCRETO EN EL ESPACIO DE ESTADOS:
% x_(k+1) = Ak*x_k + Bk*u_k
% y_k = Ck*x_k
% los estados, las entradas y las salidas son:
% estados
% x1 = [FPP-FPPss]
% x2 = [FPPp],  x2ss = 0
% x3 = [CPP-CPPss]
% x4 = [CPPp],  x4ss = 0
% x5 = [CPpH-CPpHss]
% x6 = [CPpHp], x6ss = 0
% entradas
% u1 = [PA-PAss]
% u2 = [pHA-pHAss]
% salidas
% y1 = [FP-FPss]
% y2 = [CP-CPss]
% con:
% FP = Flujo de la corriente de permeado, ml/s
% CP = Conductividad de la corriente de permeado, uS/cm
% PA = Presión de la corriente de alimentación, bar
% pHA = pH de la corriente de alimentación, adimensional
% FPP: Flujo de permeado debido a la presión de la
alimentación
% FPPp: Tasa de cambio del Flujo de permeado debido a la
presión
% CPP: Conductividad del permeado debido a la presión de
la alimentación
% CPPp: Tasa de cambio de la Conductividad del permeado
debido a la presión
% CPpH: Conductividad del permeado debido al pH de la
alimentación
% CPpHp: Tasa de cambio Conductividad del permeado debido
al pH
%---------------------------------------------------------
```

```
% Además, las matrices correspondientes son:
load('SS2system.mat');
Ts=1;
[Ak Bk]=c2d(A,B,Ts)
Ck = C
Dk = D

%Se partirá de la condición:
%entradas
PAss  = 63.06654;%bar
pHAss = 6.45;
%salidas
FPss = 66.264;%ml/s
CPss = 439.06;%uS/cm

%En estado estacionario se tienen los valores:
control1ss = PAss;
control2ss = pHAss;
controlss  = [control1ss; control2ss]
respuesta1ss = FPss;
respuesta2ss = CPss;
respuestass  = [respuesta1ss; respuesta2ss]
variable_e1ss = respuesta1ss;
variable_e2ss = 0;
variable_e3ss = respuesta2ss;
variable_e4ss = 0;
variable_e5ss = 0;
variable_e6ss = 0;
variables_ess = [variable_e1ss; variable_e2ss;
variable_e3ss; variable_e4ss; variable_e5ss;
variable_e6ss]

controlss = pHAss;
respuestass = CPss;
variable_e1ss = respuestass;
variable_e2ss = 0;
variables_ess = [variable_e1ss; variable_e2ss]
%Como el estado inicial es estacionario, todas las
variables %de desviación en ese punto son cero, es decir,
las entradas, %los estados y las salidas del modelo son
cero.
u1ss = 0;
u2ss = 0;
uss  =[u1ss; u2ss];
y1ss = 0;
y2ss = 0;
yss  =[y1ss; y2ss];
x1ss = 0;
x2ss = 0;
x3ss = 0;
x4ss = 0;
```

```
x5ss = 0;
x6ss = 0;
xss  = [x1ss; x2ss; x3ss; x4ss; x5ss; x6ss];
%Se llegará a la condición en la que, la conductividad
del permeado sea el 105% de su valor inicial.

%Lo anterior significa que el estado final deseado
resulta de %resolver el sistema de ecuaciones:
%y_k = Ck*x_k, con  y = [y1; y2], x = [x1; x2;...;x6]
%x_k = Ak*x_k + Bk*u_k, con u = [u1; u2]
%La primera ecuación matricial permite obtener:
%0.05*respuesta1 = x1d;
%               0 = x3d + x5d;
%y la segunda se puede escribir de forma conveniente
como:
%x_k = inv(eye(6)-Ak)*Bk*u_k
%al definir AA = inv(eye(6)-Ak)*Bk, se encuentra:
%x1d = AA(1,1)*u1d + AA(1,2)*u2d
%x2d = AA(2,1)*u1d + AA(2,2)*u2d
%x3d = AA(3,1)*u1d + AA(3,2)*u2d
%x4d = AA(4,1)*u1d + AA(4,2)*u2d
%x5d = AA(5,1)*u1d + AA(5,2)*u2d
%x6d = AA(6,1)*u1d + AA(6,2)*u2d
%sistema de ecuaciones que se reduce aún más:
%x1d = AA(1,1)*u1d
%x2d = AA(2,1)*u1d
%x3d = AA(3,1)*u1d
%x4d = AA(4,1)*u1d
%x5d = AA(5,2)*u2d
%x6d = AA(6,2)*u2d
%con la información disponible, se resuelve y obtiene:
AA = inv(eye(6)-Ak)*Bk
y1d = 0.10*respuesta1ss
x1d = y1d
u1d = 1/AA(1,1)*x1d
x2d = AA(2,1)*u1d
x3d = AA(3,1)*u1d
x4d = AA(4,1)*u1d
y2d = 0
x5d = -x3d
u2d = 1/AA(5,2)*x5d
x6d = AA(6,2)*u2d
%A partir de los valores anteriores, los estados deseados
son:
xd  = [x1d; x2d; x3d; x4d; x5d; x6d]
%y los valores deseados para las señales de control y
salidas "y":
ud  = [u1d; u2d]
yd  = [y1d; y2d]
```

```
%El conjunto de condiciones iniciales utilizadas para la
validación es:
cond_ini = [xss];
load('M.mat')

Ak=ME*Ak*inv(ME)
Bk=ME*Bk*MS
Ck=Ck*inv(ME)

xss=ME*xss
uss=inv(MS)*uss
yss=Ck*xss

xd=ME*xd
ud=inv(MS)*ud
yd=Ck*xd

cond_ini=ME*cond_ini

load pesosMVnm3.mat
c_ini=1;
k=1;
dt = Ts; t=0;
x = cond_ini(:,c_ini);
u = uss;
y = yss;
while( ( k < 100 ) )
estado(k,:,c_ini) = x';%variables de desviación escaladas
control(k,:,c_ini)= u';%variables de desviación escaladas
salida(k,:,c_ini) = y';%en variables de desviación reales
tsim(k,:,c_ini)=t;
%Entrada a la red neuronal, valores escalados
in_red = x-xd;
%Cálculos internos de la red neuronal
m = v'*in_red;
n = 2.0./(1 + exp(-(m-c)./a)) - 1;%función de activación
out_red = w'*n;
%Señal de control, desescalamiento y saturación
u = out_red + ud;
ureal = MS*u;
    if ureal(1) >  5; ureal(1) =  5; end
    if ureal(1) < -5; ureal(1) = -5; end
    if ureal(2) >  0.5; ureal(2) =  0.5; end
    if ureal(2) < -0.5; ureal(2) = -0.5; end
u = inv(MS)*(ureal);
%Se calcula el nuevo estado
x = Ak*(x) + Bk*(u);
y = Ck*(x);
t=t+dt;
k=k+1;
end
```

```
%señales de control (variables de desviación) reales
control=control*MS;
u1=control(:,1);
u2=control(:,2);
%señales de salida (variables de desviación) reales
y1=salida(:,1);
y2=salida(:,2);

Fpermeate = y1 + respuesta1ss;
SPFpermeate = [respuesta1ss
linspace(y1d+respuesta1ss,y1d+respuesta1ss,length(y1)-
1)];
Pfeed = u1 + control1ss;

Condp = y2 + respuesta2ss;
SPCondp = [respuesta2ss
linspace(y2d+respuesta2ss,y2d+respuesta2ss,length(y2)-
1)];
pHff = u2 + control2ss;
t=t/1.25
%variable controlada y variable manipulada
figure(1);
plot(tsim,Fpermeate,'LineWidth',2);hold all;
plot(tsim,SPFpermeate,'LineWidth',2);
xlabel('Tiempo, s');
ylabel('Flujo de la corriente de permeado, ml/s');
legend('Variable controlada','Referencia');
axis([0 t 65 74]);grid on;

figure(2);
plot(tsim,Pfeed,'LineWidth',2);
xlabel('Tiempo, s');
ylabel('Presión de la corriente de alimentación, bar');
legend('Variable manipulada');
axis([0 t 62 68]);grid on;

figure(3);
plot(tsim,Condp,'LineWidth',2);hold all;
plot(tsim,SPCondp,'--','LineWidth',2);
xlabel('Tiempo, s');
ylabel('Conductividad de la corriente de permeado,
uS/cm');
legend('Variable controlada','Referencia');
axis([0 t 400 500]);grid on;

figure(4);
plot(tsim,pHff,'LineWidth',2);
xlabel('Tiempo, s');
ylabel('pH de la corriente de alimentación');
legend('Variable manipulada');
axis([0 t 6 6.5]);grid on;
```

C.3 ControladorNeuronal.m

```
function u = fcn(x_xd)
ESCALAMIENTO=load('M.mat');
ME=ESCALAMIENTO.ME;
MS=ESCALAMIENTO.MS;
PESOS=load('pesosMVnm3.mat');
v=PESOS.v;
w=PESOS.w;
c=PESOS.c;
a=PESOS.a;
CONTROL_DESEADO=load('valores_deseados_no_linealp.mat');
ud=CONTROL_DESEADO.ud;
ud=[1 0;0 0.99]*ud;
% u2d=CONTROL_DESEADO.u2d;
in_red = ME*x_xd;
%Cálculos internos de la red neuronal
m = v'*in_red;
n = 2.0./(1 + exp(-(m-c)./a)) - 1;%función de activación
sigmoidea 2
out_red = w'*n;
%Señal de control, desescalamiento y saturación
u = out_red + MS\ud;
ureal = MS*u;
    if ureal(1) >  5; ureal(1) =  5; end
    if ureal(1) < -5; ureal(1) = -5; end
    if ureal(2) >  0.5; ureal(2) =  0.5; end
    if ureal(2) < -0.5; ureal(2) = -0.5; end
u = ureal;
u = u;
```

ANEXO D

PROGRAMAS PARA EL CONTROL NEURONAL DEL MODELO NO LINEAL DE LA UNIDAD DE ÓSMOSIS INVERSA DE UNA PLANTA PILOTO DE DESALINIZACIÓN DE AGUA DE MAR

D.1 p01controlNeuronalnoLineal.m

```
%Control neuronal del flujo y la conductividad del
permeado en %el modelo no lineal de una planta de
desalinización por O.I.
close all; clear all; clc;

%Determinación del Estado Estacionario Inicial
disp('-Determinación del Estado Estacionario Inicial-');
%Se ejecuta el programa InititalConditionsBL.m para
determinar %el valor de todas las variables en el estado
estacionario %inicial y se asignan estos valores a sus
variables %respectivas
InititalConditionsBL
%Condiciones iniciales de la corriente de alimentación
Ff0=Ff; %ml/s
Pf0=Pf; %bar
Tf0=Tf; %°C
Cf0=Cf; %ppm
pHf0=pHf;
%Condiciones iniciales sistema
kA0=kA(end); %ml/(s*m2)
mb0=mb(end); %ml
mp0=mp(end); %ml
Fb0=Fb(end); %ml/s
Fp0=Fp(end); %ml/s
Cb0=Cb(end); %ppm
Cp0=Cp(end); %ppm
Tb0=Tb(end); %°C
Tp0=Tp(end); %°C
Pb0=Pb(end); %bar
Pp0=Pp(end); %bar
Condp0=Kohlrausch(Cp0);%uS/cm

control10 = Pf0;
salida10 = Fp0;
control20 = pHf0;
salida20 = Condp0;

load('valores_deseados_no_linealp.mat');
x1d = xd(1);
x2d = xd(2);
x3d = xd(3);
x4d = xd(4);
x5d = xd(5);
y1d = yd(1);
y2d = yd(2);

Ts=1;%tiempo de integración
open('ROcontrolNeuronalnoLineal');
simOut=sim('ROcontrolNeuronalnoLineal');
```

```
%Se obtienen las variables de control para graficarlas
%variable de entrada
tsim=pHfdata.time;
PfeedNN=Pfdata.signals.values;
pHfeedNN=pHfdata.signals.values;

%variable de salida
FpermeateNN=Fpdata.signals.values(:,2);
CondpermeateNN=Condpdata.signals.values(:,1);
SPFpermeate = [salida10
linspace(y1d+salida10,y1d+salida10,length(FpermeateNN)-
1)];
SPCondpermeate = [salida20
linspace(y2d+salida20,y2d+salida20,length(CondpermeateNN)
-1)];
t=tsim(end)/2;

%gráficas en unidades reales
%Flujo de permeado
figure(1);
plot(tsim,SPFpermeate,'LineWidth',2);
hold all;
plot(tsim,FpermeateNN,'LineWidth',2);
xlabel('Tiempo, s');
ylabel('Flujo de la corriente de permeado, ml/s');
axis([0 t 66 70]);grid on;

figure(2);
plot(tsim,PfeedNN,'LineWidth',2);
hold all;
xlabel('Tiempo, s');
ylabel('Presión de la corriente de alimentación, bar');
legend('Variable manipulada');
axis([0 t 62 66]);grid on;
%Conductividad del permeado
figure(3);
plot(tsim,CondpermeateNN,'LineWidth',2);
hold all;
plot(tsim,SPCondpermeate,'LineWidth',2);
xlabel('Tiempo, s');
ylabel('Conductividad de la corriente de permeado,
uS/cm');
axis([0 t 430 450]);grid on;

figure(4);
plot(tsim,pHfeedNN,'LineWidth',2);
hold all;
xlabel('Tiempo, s');
ylabel('pH de la corriente de alimentación');
legend('Variable manipulada');
axis([0 t 6.1 6.4]);grid on;
```

ANEXO E

PROGRAMAS PARA LA SIMULACIÓN DEL SISTEMA DE CONTROL PID Y PARA LA COMPARACIÓN DE LOS CONTROLADORES PID vs. NEURONAL

E.1 p01ROFlowControl.m

```
%Control PID del Flujo de Permeado en una Planta de
Desaliniziación por O.I.
clc; close all; clear all;

%%Determinación del Estado Estacionario Inicial
disp('-Determinación del Estado Estacionario Inicial-');
%Se ejecuta el programa InititalConditionsBL.m para
determinar %el valor de todas las variables en el estado
estacionario %inicial y se asignan estos valores a sus
variables %respectivas
InititalConditionsBL

%%Control del Flujo de Permeado Manipulando la presión de
%Alimentación
disp('-----Control del Flujo de Permeado-----');
%Condiciones iniciales de la corriente de alimentación
Ff0=Ff; %g/s
Pf0=Pf; %bar
Tf0=Tf; %°C
Cf0=Cf; %ppm
pHf0=pHf;
%Condiciones iniciales sistema
kA0=kA(end); %g/(s*m2)
mb0=mb(end); %g
mp0=mp(end); %g
Fb0=Fb(end); %g/s
Fp0=Fp(end); %g/s
Cb0=Cb(end); %ppm
Cp0=Cp(end); %ppm
Tb0=Tb(end); %°C
Tp0=Tp(end); %°C
Pb0=Pb(end); %bar
Pp0=Pp(end); %bar
Condp0=Kohlrausch(Cp0);%uS/cm
%Simulación del sistema de control
open('ROPermeateFlowControl');
simOut=sim('ROPermeateFlowControl');
tsim=retentate.time;
Fretentate=retentate.signals(1,1).values;
Cretentate=retentate.signals(1,2).values;
Tretentate=retentate.signals(1,3).values;
Pretentate=retentate.signals(1,4).values;
clear 'retentate'
Fpermeate=permeate.signals(1,1).values;
Cpermeate=permeate.signals(1,2).values;
Tpermeate=permeate.signals(1,3).values;
Ppermeate=permeate.signals(1,4).values;
clear 'permeate'
SPFp=SPFpdata.signals.values;
```

```
%cálculos
Pfeed=Pretentate;
Condretentate=C2Cd(Cretentate).*(1+0.0212*(Tretentate-
25));
Condpermeate=Kohlrausch(Cpermeate).*(1+0.0212*(Tpermeate-
25));%incluye corrección por temperatura

%Variable controlada y variable manipulada
figure(11);
plot(tsim,Fpermeate,'LineWidth',2);
hold all;
plot(tsim,SPFp,'LineWidth',2);
xlabel('Tiempo, s');
ylabel('Flujo de la corriente de permeado, ml/s');
legend('Variable controlada','Referencia');
grid on;
figure(12);
plot(tsim,Pretentate,'LineWidth',2);
xlabel('Tiempo, s');
ylabel('Presión de la corriente de alimentación, bar');
legend('Señal de control');
grid on;

save FPIdata tsim Fpermeate SPFp Pretentate
```

E.2 p02ROConductivityControl.m

```
%Control PID de la Conductividad del Permeado en una
Planta de %Desaliniziación por O.I.
clc; close all; clear all;

%%Determinación del Estado Estacionario Inicial
disp('-Determinación del Estado Estacionario Inicial-');
%Se ejecuta el programa InititalConditionsBL.m
InititalConditionsBL

%%Control de la Conductividad del Permeado Manipulando el
pH %de la Alimentación
disp('--Control de la Conductividad del Permeado--');
%Condiciones iniciales de la corriente de alimentación
Ff0=Ff; %g/s
Pf0=Pf; %bar
Tf0=Tf; %°C
Cf0=Cf; %ppm
pHf0=pHf;
%Condiciones iniciales sistema
kA0=kA(end); %g/(s*m2)
mb0=mb(end); %g
mp0=mp(end); %g
Fb0=Fb(end); %g/s
Fp0=Fp(end); %g/s
Cb0=Cb(end); %ppm
Cp0=Cp(end); %ppm
Tb0=Tb(end); %°C
Tp0=Tp(end); %°C
Pb0=Pb(end); %bar
Pp0=Pp(end); %bar
Condp0=Kohlrausch(Cp0);%uS/cm
%simulación del sistema de control
open('ROPermeateConductivityControl');
simOut=sim('ROPermeateConductivityControl');
tsim=retentate.time;
Fretentate=retentate.signals(1,1).values;
Cretentate=retentate.signals(1,2).values;
Tretentate=retentate.signals(1,3).values;
Pretentate=retentate.signals(1,4).values;
clear 'retentate'
Fpermeate=permeate.signals(1,1).values;
Cpermeate=permeate.signals(1,2).values;
Tpermeate=permeate.signals(1,3).values;
Ppermeate=permeate.signals(1,4).values;
clear 'permeate'
SPCondp=SPCondpdata.signals.values;
Condp=Condpdata.signals.values;
pHff=pHfdata.signals.values;
```

```
%cálculos
Pfeed=Pretentate;
Condretentate=C2Cd(Cretentate).*(1+0.0212*(Tretentate-
25));
Condpermeate=Kohlrausch(Cpermeate).*(1+0.0212*(Tpermeate-
25));%incluye corrección por temperatura

%variable controlada y variable manipulada
figure(11);
plot(tsim,Condp,'LineWidth',2);
hold all;
plot(tsim,SPCondp,'LineWidth',2);
xlabel('Tiempo, s');
ylabel('Conductividad de la corriente de permeado,
uS/cm');
legend('Variable controlada','Referencia');
grid on;
figure(12);
plot(tsim,pHff,'LineWidth',2);
xlabel('Tiempo, s');
ylabel('pH de la corriente de Alimentación');
legend('Señal de control');
grid on;

save CondPIdata tsim Condp SPCondp pHff
```

E.3 p01controlNeuronalvsPInoLineal.m

```
%Comparación de los controladores Neuronal y PI para el
flujo %y la conductividad del permeado en el modelo no
lineal de una %planta de desalinización por O.I.
close all; clear all; clc;

%Determinación del Estado Estacionario Inicial
disp('-Determinación del Estado Estacionario Inicial-');
%Se ejecuta el programa InititalConditionsBL.m
InititalConditionsBL
%Condiciones iniciales de la corriente de alimentación
Ff0=Ff; %ml/s
Pf0=Pf; %bar
Tf0=Tf; %°C
Cf0=Cf; %ppm
pHf0=pHf;
%Condiciones iniciales sistema
kA0=kA(end); %ml/(s*m2)
mb0=mb(end); %ml
mp0=mp(end); %ml
Fb0=Fb(end); %ml/s
Fp0=Fp(end); %ml/s
Cb0=Cb(end); %ppm
Cp0=Cp(end); %ppm
Tb0=Tb(end); %°C
Tp0=Tp(end); %°C
Pb0=Pb(end); %bar
Pp0=Pp(end); %bar
Condp0=Kohlrausch(Cp0);%uS/cm

control10 = Pf0;
salida10 = Fp0;
control20 = pHf0;
salida20 = Condp0;

load('valores_deseados_no_lineal.mat');
x1d = xd(1);x2d = xd(2);x3d = xd(3);x4d = xd(4);x5d =
xd(5);
y1d = yd(1);y2d = yd(2);

Ts=1;%tiempo de integración
open('ROcontrolNeuronalvsPInoLineal');
simOut=sim('ROcontrolNeuronalvsPInoLineal');
%Se obtienen las vaiables de control para graficarlas
%variable de entrada
tsim=Pfdata3.time;
PfeedPI=Pfdata3.signals.values(:,1);
PfeedNN=Pfdata3.signals.values(:,2);
pHfeedPI=pHfdata3.signals.values(:,1);
pHfeedNN=pHfdata3.signals.values(:,2);
```

```
%variable de salida
FpermeatePI=Fpdata3.signals.values(:,1);
FpermeateNN=Fpdata3.signals.values(:,2);
CondpermeatePI=Condpdata3.signals.values(:,1);
CondpermeateNN=Condpdata3.signals.values(:,2);
SPFpermeate = [salida10 linspace(y1d+salida10,
y1d+salida10,length(FpermeateNN)-1)];
SPCondpermeate = [salida20 linspace(y2d+salida20,
y2d+salida20,length(CondpermeateNN)-1)];

%gráficas en unidades reales
%Flujo de permeado
figure(1);
plot(tsim,FpermeatePI,'LineWidth',2);hold all;
plot(tsim,SPFpermeate,'LineWidth',2);
plot(tsim,FpermeateNN,'LineWidth',2);
xlabel('Tiempo, s');
ylabel('Flujo de la corriente de permeado, ml/s');
legend('Controlador PI','Controlador
Neuronal','Referencia');
axis([0 tsim(end) 65 70]);grid on;

figure(2);
plot(tsim,PfeedPI,'LineWidth',2);hold all;
plot(tsim,PfeedNN,'LineWidth',2);
xlabel('Tiempo, s');
ylabel('Presión de la corriente de alimentación, bar');
legend('Controlador PI','Controlador Neuronal');
axis([0 tsim(end) 62 66]);grid on;

%Conductividad del permeado
figure(3);
plot(tsim,CondpermeatePI,'LineWidth',2);hold all;
plot(tsim,CondpermeateNN,'LineWidth',2);
plot(tsim,SPCondpermeate,'LineWidth',2);
xlabel('Tiempo, s');
ylabel('Conductividad de la corriente de permeado,
uS/cm');
legend('Controlador PI','Controlador
Neuronal','Referencia');
axis([0 tsim(end) 420 450]);grid on;

figure(4);
plot(tsim,pHfeedPI,'LineWidth',2);hold all;
plot(tsim,pHfeedNN,'LineWidth',2);
xlabel('Tiempo, s');
ylabel('pH de la corriente de alimentación');
legend('Controlador PI','Controlador Neuronal');
axis([0 tsim(end) 6.0 6.5]);
grid on;
```