Anexo 1.- Programa en Matlab para el Entrenamiento de la Red Neuronal Dinámica

```
% Entrenamiento de Red Neuronal Dinamica
% Entrenamiento tipo Batch
% Data extraida del DCS para el sistema de estabilizacion de
condensado

clear;
clc;
close all;

% Obtencion de la data completa a Matlab
datadcs = xlsread('datadcs.xls');
nf = length(datadcs);
% Asignacion de las variables para el entrenamiento del sistema neural
y(1:nf,1) = datadcs(:,1);
x1(1:nf,1) = datadcs(:,2);
x2(1:nf,1) = datadcs(:,3);
x3(1:nf,1) = datadcs(:,4);
x4(1:nf,1) = datadcs(:,5);
x5(1:nf,1) = datadcs(:,6);
x6(1:nf,1) = datadcs(:,7);
x7(1:nf,1) = datadcs(:,8);
x8(1:nf,1) = datadcs(:,9);
x9(1:nf,1) = datadcs(:,10);
x10(1:nf,1) = datadcs(:,11);
x11(1:nf,1) = datadcs(:,12);
x12(1:nf,1) = datadcs(:,13);
x13(1:nf,1) = datadcs(:,14);
x14(1:nf,1) = datadcs(:,15);

% Matrices de datos para el entrenamiento
xb = [x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14];
yb = y;
nv = length(xb);

ne = 15; %se considera que se realimenta la salida como una entrada
mas
nm = 15;
ns = 1;

% Escalamiento de datos para el entrenamiento de la red neuronal
factx = max(abs(xb));
facty = max(abs(yb));

xesc(:,1) = xb(:,1)./factx(1,1);
xesc(:,2) = xb(:,2)./factx(1,2);
xesc(:,3) = xb(:,3)./factx(1,3);
xesc(:,4) = xb(:,4)./factx(1,4);
xesc(:,5) = xb(:,5)./factx(1,5);
xesc(:,6) = xb(:,6)./factx(1,6);
xesc(:,7) = xb(:,7)./factx(1,7);
xesc(:,8) = xb(:,8)./factx(1,8);
xesc(:,9) = xb(:,9)./factx(1,9);
xesc(:,10) = xb(:,10)./factx(1,10);
xesc(:,11) = xb(:,11)./factx(1,11);
xesc(:,12) = xb(:,12)./factx(1,12);
xesc(:,13) = xb(:,13)./factx(1,13);
```

```
xesc(:,14) = xb(:,14)./factx(1,14);
yesc(:,1) = yb(:,1)./facty(1,1);

size(xesc)

bias = input('Bias:  SI = 1 : ');
if(bias == 1)
      ne = ne +1;
      xesc = [ xesc ones(nx,1) ];
end

ndata = nv;
dataoutesc = yesc;

v = 0.2*randn(ne,nm); % valores iniciales de los pesos v,w
w = 0.2*randn(nm,ns);
c = zeros(nm,1); % centro de la sigmoidea
a = ones(nm,1); % inclinación de la sigmoidea

%load dataentrenamiento

eta  = input('Introducir ratio de aprendizaje : ');
beta = input('Introducir momento : ');
etac = input('Introducir ratio de aprendizaje de centro c      : ');
etaa = input('Introducir ratio de aprendizaje de inclinacion a : ');

errormax = input('Introducir el valor maximo del error (%) : ');
errormax = errormax/100;
contmax = input('Introducir el maximo numero de etapas de
aprendizaje:');

outsum2 = sum(dataoutesc.^2);
outsum2 = outsum2';
outsum2total = sum(outsum2);
cont = 1;
erreltotal = 1;
   dw_old = 0;
   dv_old = 0;
   da_old = 0;
   dc_old = 0;

while( (erreltotal > errormax) & (cont < contmax) )
   ersum2 = zeros(ns,1);
   dJdw = 0;
   dJdv = 0;
   dJda = 0;
   dJdc = 0;
   dydw_t = zeros(nm,ns);  % y es la salida de la red neuronal
   dydv_t = zeros(ne,nm);
   dydc_t = zeros(nm,1);
   dyda_t = zeros(nm,1);
   dJdw_t  = zeros(nm,ns);
   dJdv_t  = zeros(ne,nm);
   dJdc_t  = zeros(nm,1);
   dJda_t  = zeros(nm,1);

   yesc_p = dataoutesc(1,1);
   yesc_p = yesc_p';
   for k = 1:ndata-1 %bucle interno son las etapas (nu etapas)
```

```
        in_red = [yesc_p
                  xesc(k,:)']; %  yesc_p
        m = v'*in_red;
        n = 2.0./(1 + exp(-(m-c)./a)) - 1; %Sigmoidea tipo2
        out_red = w'*n;
        outputesc(k,:) = (out_red*facty)';
        dndm = diag((1 - n.*n)./(2*a));
        dydw_s = n;
        dydv_s = in_red*w(:,1)'*dndm;
        dydc_s = w(:,1) .* ((n.*n-1)./(2.0.*a));
        dyda_s = w(:,1) .* ((n.*n-1).*(m-c)./(2*a.*a));
        jacob = w'*dndm*(v(1:ne-1,:))'; % se calcula el Jacobian
        dydw_t = dydw_s + jacob(1,1).*dydw_t;
        dydv_t = dydv_s + jacob(1,1).*dydv_t;
        dydc_t = dydc_s + jacob(1,1).*dydc_t;
        dyda_t = dyda_s + jacob(1,1).*dyda_t;

        out_des = dataoutesc(k+1,1);
        out_des = out_des';
        er = (out_red - out_des);
        erJ = (out_red - out_des).^1;
        dJdw_t = dJdw_t + erJ(1,1).*dydw_t;
        dJdv_t = dJdv_t + erJ(1,1).*dydv_t;
        dJdc_t = dJdc_t + erJ(1,1).*dydc_t;
        dJda_t = dJda_t + erJ(1,1).*dyda_t;
        ersum2 = ersum2 + er.^2;
        yesc_p = out_red;
    end
        dJdw_t = dJdw_t/ndata;
        dJdv_t = dJdv_t/ndata;
        dJdc_t = dJdc_t/ndata;
        dJda_t = dJda_t/ndata;
        dw = dJdw_t + beta*dw_old;
        dv = dJdv_t + beta*dv_old;
        dc = dJdc_t;
        da = dJda_t;
        w = w - eta*dw;
        v = v - eta*dv;
        c = c - etac*dc;
        a = a - etaa*da;
        dw_old = dw;
        dv_old = dv;
    ersum2total = sum(ersum2);
    cont = cont + 1;
    if ( rem(cont,1) == 0 )
        errorrel(cont/1,:) = sqrt(ersum2'./outsum2');
        errorreltotal(cont/1,1) = sqrt(ersum2total/outsum2total);
        erreltotal = errorreltotal(cont/1,1);
        cont
        erreltotal
    end
end
figure(1);
plot(errorreltotal*100); % ver como baja el error
figure(2);
plot(errorrel*100);
figure(3);
plot(y(:,1),'--b');%Compara salida real con salida de red neuronal
hold on;
plot(outputesc(:,1),'r');
save dataentrenamiento factx facty v w a c ne nm ns;
```

Anexo 2.- Programa en Matlab para la Validación de la Red Neuronal Dinámica

```
% Validadcion de Red Neuronal Dinamica
% Data extraida del DCS para el sistema de estabilizacion de
condensado

clear;
clc;
close all;

% Obtencion de la data completa a Matlab
datadcs = xlsread('datadcsvalida.xls');
nf = length(datadcs);
% Asignacion de las variables para la validacion del sistema neural
y(1:nf,1) = datadcs(:,1);
x1(1:nf,1) = datadcs(:,2);
x2(1:nf,1) = datadcs(:,3);
x3(1:nf,1) = datadcs(:,4);
x4(1:nf,1) = datadcs(:,5);
x5(1:nf,1) = datadcs(:,6);
x6(1:nf,1) = datadcs(:,7);
x7(1:nf,1) = datadcs(:,8);
x8(1:nf,1) = datadcs(:,9);
x9(1:nf,1) = datadcs(:,10);
x10(1:nf,1) = datadcs(:,11);
x11(1:nf,1) = datadcs(:,12);
x12(1:nf,1) = datadcs(:,13);
x13(1:nf,1) = datadcs(:,14);
x14(1:nf,1) = datadcs(:,15);

% Matrices de datos para la validacion
xb = [x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14];
yb = y;
nv = length(xb);

ne = 15; %se considera que se realimenta la salida como una entrada
mas
nm = 15;
ns = 1;

% Escalamiento de datos para el entrenamiento de la red neuronal
factx = max(abs(xb));
facty = max(abs(yb));

xesc(:,1) = xb(:,1)./factx(1,1);
xesc(:,2) = xb(:,2)./factx(1,2);
xesc(:,3) = xb(:,3)./factx(1,3);
xesc(:,4) = xb(:,4)./factx(1,4);
xesc(:,5) = xb(:,5)./factx(1,5);
xesc(:,6) = xb(:,6)./factx(1,6);
xesc(:,7) = xb(:,7)./factx(1,7);
xesc(:,8) = xb(:,8)./factx(1,8);
xesc(:,9) = xb(:,9)./factx(1,9);
xesc(:,10) = xb(:,10)./factx(1,10);
xesc(:,11) = xb(:,11)./factx(1,11);
xesc(:,12) = xb(:,12)./factx(1,12);
xesc(:,13) = xb(:,13)./factx(1,13);
xesc(:,14) = xb(:,14)./factx(1,14);
```

```
yesc(:,1) = yb(:,1)./facty(1,1);

ndata = nv;
dataoutesc = yesc;

v = 0.2*randn(ne,nm); % valores iniciales de los pesos v,w
w = 0.2*randn(nm,ns);
c = zeros(nm,1); % centro de la sigmoidea
a = ones(nm,1); % inclinación de la sigmoidea

load dataentrenamiento

eta  = input('Introducir ratio de aprendizaje : ');
beta = input('Introducir momento : ');
etac = input('Introducir ratio de aprendizaje de centro c      : ');
etaa = input('Introducir ratio de aprendizaje de inclinacion a : ');

errormax = input('Introducir el valor maximo del error (%) : ');
errormax = errormax/100;
contmax = input('Introducir el maximo numero de etapas de
aprendizaje:');

outsum2 = sum(dataoutesc.^2);
outsum2 = outsum2';
outsum2total = sum(outsum2);
cont = 1;
erreltotal = 1;
   dw_old = 0;
   dv_old = 0;
   da_old = 0;
   dc_old = 0;

while( (erreltotal > errormax) & (cont < contmax) )
   ersum2 = zeros(ns,1);
   dJdw = 0;
   dJdv = 0;
   dJda = 0;
   dJdc = 0;
   dydw_t = zeros(nm,ns);  % y es la salida de la red neuronal
   dydv_t = zeros(ne,nm);
   dydc_t = zeros(nm,1);
   dyda_t = zeros(nm,1);
   dJdw_t  = zeros(nm,ns);
   dJdv_t  = zeros(ne,nm);
   dJdc_t  = zeros(nm,1);
   dJda_t  = zeros(nm,1);

   yesc_p = dataoutesc(1,1);
   yesc_p = yesc_p';
   for k = 1:ndata-1 %bucle interno son las etapas (nu etapas)
      in_red = [yesc_p
                xesc(k,:)'];
      m = v'*in_red;
      n = 2.0./(1 + exp(-(m-c)./a)) - 1; %Sigmoidea tipo2
      out_red = w'*n;
      outputesc(k,:) = (out_red*facty)';
      dndm = diag((1 - n.*n)./(2*a));

      dydw_s = n;
      dydv_s = in_red*w(:,1)'*dndm;
```

```
        dydc_s = w(:,1) .* ((n.*n-1)./(2.0.*a));
        dyda_s = w(:,1) .* ((n.*n-1).*(m-c)./(2*a.*a));
        jacob = w'*dndm*(v(1:ne-1,:))'; % se calcula el Jacobian
        dydw_t = dydw_s + jacob(1,1).*dydw_t;
        dydv_t = dydv_s + jacob(1,1).*dydv_t;
        dydc_t = dydc_s + jacob(1,1).*dydc_t;
        dyda_t = dyda_s + jacob(1,1).*dyda_t;

        out_des = dataoutesc(k+1,1);
        out_des = out_des';
        er = (out_red - out_des);
        erJ = (out_red - out_des).^1;
        dJdw_t = dJdw_t + erJ(1,1).*dydw_t;
        dJdv_t = dJdv_t + erJ(1,1).*dydv_t;
        dJdc_t = dJdc_t + erJ(1,1).*dydc_t;
        dJda_t = dJda_t + erJ(1,1).*dyda_t;
        ersum2 = ersum2 + er.^2;
        yesc_p = out_red;
    end
        dJdw_t = dJdw_t/ndata;
        dJdv_t = dJdv_t/ndata;
        dJdc_t = dJdc_t/ndata;
        dJda_t = dJda_t/ndata;
        dw = dJdw_t + beta*dw_old;
        dv = dJdv_t + beta*dv_old;
        dc = dJdc_t;
        da = dJda_t;
        w = w - eta*dw;
        v = v - eta*dv;
        c = c - etac*dc;
        a = a - etaa*da;
        dw_old = dw;
        dv_old = dv;
    ersum2total = sum(ersum2);
    cont = cont + 1;
    if ( rem(cont,1) == 0 )
        errorrel(cont/1,:) = sqrt(ersum2'./outsum2');
        errorreltotal(cont/1,1) = sqrt(ersum2total/outsum2total);
        erreltotal = errorreltotal(cont/1,1);
        cont
        erreltotal
    end
end

figure(1);
plot(errorreltotal*100); % ver como baja el error
figure(2);
plot(errorrel*100);
figure(3);
plot(y(:,1),'--b');%Compara salida real con salida de red neuronal
hold on;
plot(outputesc(:,1),'r');
save dataentrenamiento2 factx facty v w a c ne nm ns;
```

Anexo 3.- Simulación Dinámica con el Esquema de Control Inferencial Implementado.