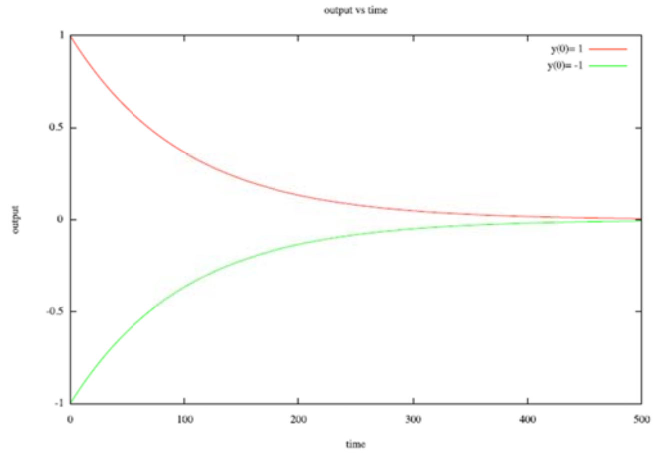
Nodo

```
%% Use initial value of x=1 and x=-1
%% h is the time step for integration
%% t is the length of simulation

function SimpleNode (h, t)
        NumTimeSteps = t/h;

        x = 1;

        SimulateNode(x, 1, h, t);
        hold on;

        x = -1;
        SimulateNode(x, 1, h, t);
        hold off;

end
```




```
%% Use euler to integrate
%% x is the initial value
%% tau is the time constant
%% h is the step size for integration
%% t is the simulation length

function SimulateNode (x, tau, h, t)

        NumTimeSteps = t/h;

        x = zeros(1, NumTimeSteps);
        oldx = x;
        x(1) = oldx;
        tau_value = 1/tau;

        for TStep = 1:NumTimeSteps
                newx = oldx + (h * (tau_value * -oldx));
                x(TStep+1) = newx;
                oldx = newx;
        end

        % Now display
        t = 0:NumTimeSteps;
        max(x);

        % str = sprintf(';x= %g;',x);
        xlabel('time'), ylabel('output'), title('output vs time');
        plot(t, x);

end
```

$$\frac{dy}{dt} = \frac{1}{\tau_i}(-y_i)$$

en discreto es:

(newX-oldX)/h =- tau_value * oldX

Condiciones iniciales oldX=x(1);

Desde ahí calculamos recurrentemente newX que va llenando de valores

x = zeros(1, NumTimeSteps);

Efecto de tau

```
%% Minimal CTRNN – investigates the effect of tau on the system
%% x is the initial value of x
%% h is the time step for integration
%% t is the length of simulation

        function MinimalCTRNN (x, h, t)

        NumTimeSteps = t/h;

        for tau = 0.2:0.2:2.0
                x= zeros(1, NumTimeSteps);
                oldx = x;
                x(1)= oldx;
                tau_value = 1/tau;

                for TStep = 1:NumTimeSteps
                        newx = oldx + (h * (tau_value * -oldx));
                        x(TStep+1) = newx;
                        oldx = newx;
                end

                % Now display
                t = 0:NumTimeSteps;
                max(x);
            % str = sprintf(';T_i= %g;', tau);
                xlabel('time'), ylabel('output'), title('output vs time');
                plot(t, x);
                hold on;
        end
        hold off;

        end
```

$$y(t+1) = y(t) + (h * \frac{1}{\tau_i} * (-y(t))$$

La misma ecuación del caso anterior con

tau_value = 1/tau; donde tau=0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4, 1.6, 1.8, 2

Efecto de la unidad temporal

```
%% Measures the effect of t -> h
%% h is the time step for integration
%% t is the length of simulation

function MinimalCTRNN2 (x, h, t)

        NumTimeSteps = t/h;

        tau = h/2;
                x           = zeros(1, NumTimeSteps);
                oldx        = x;
                X(1)        = oldx;
                tau_value           = 1/tau;

                for TStep = 1:NumTimeSteps
                        newx = oldx + (h * (tau_value * -oldx));
                        X(TStep+1) = newx;
                        oldx = newx;
                end


% Now display
        t = 0:NumTimeSteps;
```

When tau = h/2 the system now oscillates between -1 and 1 during integration steps.

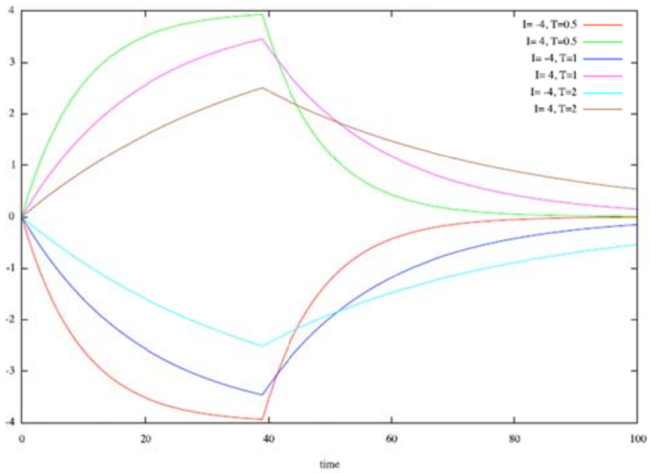
```
        max(x);

        % str = sprintf(';T_i= %g;', tau);

        xlabel('time'), ylabel('output'), title('output  vs  time');
        plot(t, X,  str);
        hold  on;

        tau  = h;
        x = zeros(1, NumTimeSteps);
        oldx = x;
        x(1)= oldx;
        tau_value= 1/tau;

        for TStep = 1:NumTimeSteps
                newx = oldx  + (h  * (tau_value * -oldx));
                x(TStep+1)  = newx;
                oldx  = newx;

end

% Now   display
        t = 0:NumTimeSteps;
        max(X);

        % str = sprintf(';T_i= %g;', tau);

        xlabel('time'), ylabel('output'), title('output  vs  time');
        plot(t, x);
        hold  on;
        hold  off;



end
```

When tau = h the output of the system follows the input, which in this case is zero. So the node 'decays' instantly

Añadir estímulos

```
%% Minimal  CTRNN3
%% Shows the  effect of  varying Tau and input for  a CTRNN  Node
%% h  is  the  time  step  for  integration
%% t  is the  length  of  simulation

function MinimalCTRNN3 (x, h, t)

        NumTimeSteps = t/h;        % Num of integration steps
        HalfTimeStep = 2/h         % halfway point
        tau = [0.5, 1, 2];
        I = [-4, 4];
        t = 0:NumTimeSteps;
        for i=1:3
                for j=1:2
                        x          = zeros(1, NumTimeSteps);
                        oldx       = x;
                        x(1)       =  oldx;
                        tau_value         = 1 /  tau(i);

                        for TStep = 1:NumTimeSteps
                                if (TStep  < HalfTimeStep)
                                        delta_x = tau_value * (-oldx  + I(j));

                                else
```

```
                      delta_x = tau_value * (-oldx);
              %%   I == 0
              end

              newx = oldx  + (h  * delta_x);
              x(TStep+1)  = newx;
              oldx  = newx;

        end

    %  Now display
    %        str = sprintf(';I= %g, T=%g;',  I(j), tau(i))
             xlabel('time'), ylabel('output'),  title('output vs  time');
             plot(t, x);
             hold on;
             end
    end
    hold off;

end
```

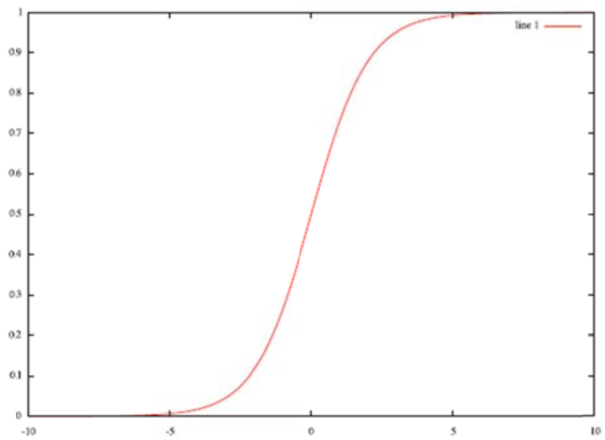


Se muestran los casos de tau = [0.5, 1, 2] combinados con I = [-4, 4] para los casos en que I> 0 (y cuando I=0)

Función de activación (sigmoide)

Used Sigmoid Plot with bias = 0, and gain =1

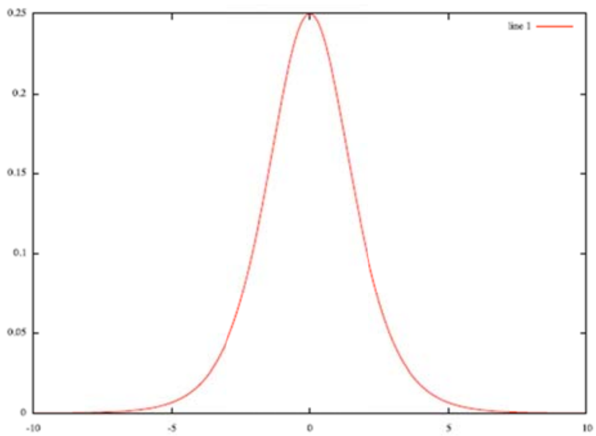
```
% Standard  logistic  activation function

function  s  =  Sigmoid  (x)
    s  =  1;
    s  =  s  /  (1  + exp(-x));
end


%% Plots the  response  of  the  Sigmoid  function
%% b  is  the  bias
%% g  is  the  gain

function   SigmoidPlot (b,  g)
    X        = -10:0.1:10;
    dim      = size(X);
    length  = dim(1,2);
    S        = zeros(1,  length);
```



```
%  Calculate  Sigmoid
        for i=1:length
                 S(i)  = Sigmoid(g  * (X(i) + b));
        end
%  str = sprintf(';g=%g b=%g;',  g,  b);
        plot(X, S);
end
```

```
% Standard  logistic  activation function
% Sigmoid
function  s  =  SigmoidDerivative  (x)
     s  =  exp(-x);
     d  =  (1  +  exp(-x))  ^  2;
     s  =  s  /  d;
end
```

```
%% Plots the  response  of  the  derivative  of  the
%% Sigmoid  function
%% b  is  the  bias
%% g  is  the  gain
function  SigmoidDerivativePlot (b,  g)

     X       = -10:0.1:10;
     dim      = size(X);
     length  = dim(1,2);

     % Calculate  change in  sigmoid
     dS =  zeros(1,  length);

        %%   Plot derivative
        %hold on;

        for i=1:length
        dS(i) = SigmoidDerivative(g *
     (X(i) + b));
        end
        plot (X,  dS);
end
```

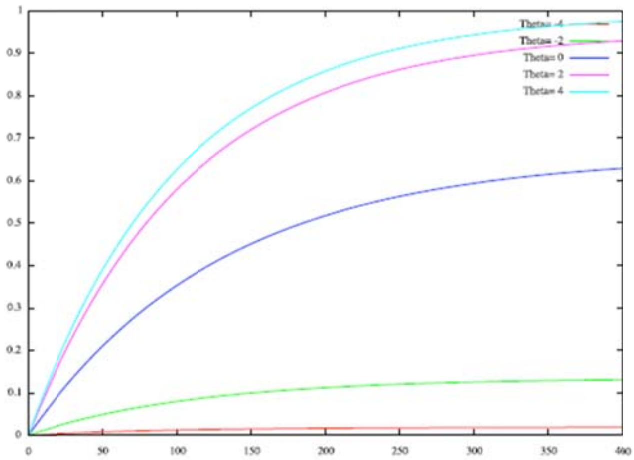


Efecto del umbral

Used SigmoidPlot to show effect of bias on Sigmoid Function.

```
%% Single   recurrent CTRNN node with   Bias
%% x  is  the  initial  value  of  the  node
%% h  is  the  time  step  for  integration
%% t  is  the  length  of  simulation

function  BiasNode  (x,  h,  t)

          NumTimeSteps  =  t/h;                              % Num  of  integration  steps
          tau  =  1;
```

```
for theta = -4:2:4
        X        = zeros(1, NumTimeSteps);
        oldx     = x;
        X(1)     = oldx;
        I        = 0;    %%  no input
        w        = 1;    %%  connection is on but no multiplier

for TStep = 1:NumTimeSteps
        delta_x = -oldx + (w * Sigmoid(oldx + theta)) + I;
        newx = oldx + (h * delta_x);
        X(TStep+1) = newx;
        oldx = newx;
end

% Now display
t = 0:NumTimeSteps;
str = sprintf(';Theta= %g;', theta);
%xlabel('time'), ylabel('output'), title('output vs time');
plot(t, X, str);
hold on;

end
 hold off;

end
```

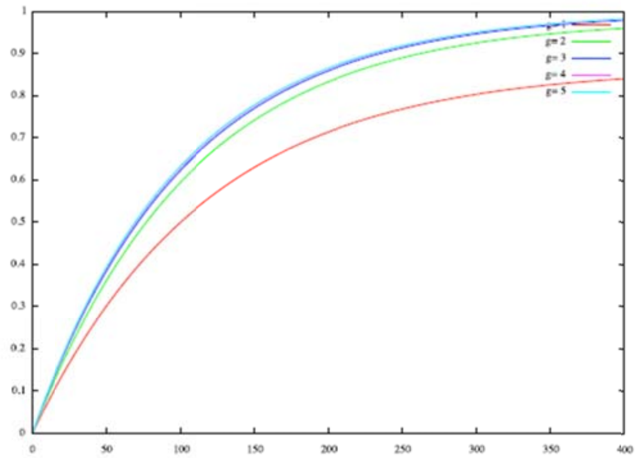


Efecto del factor de escala de ganancia

Used SigmoidPlot to show effect of gain on Sigmoid Function.

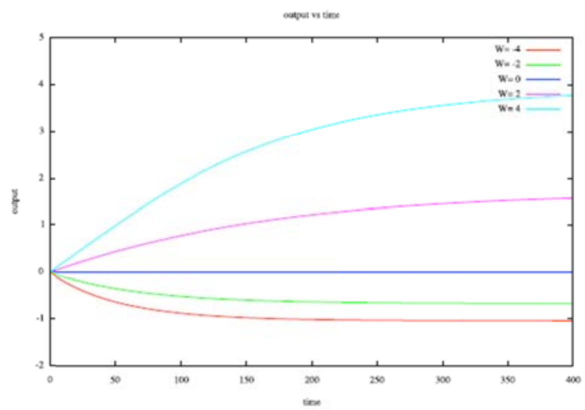
```
%% Single recurrent CTRNN node with Gain
%% x is the initial value of the node
%% h is the time step for integration
%% t is the length of simulation

function GainNode (x, h, t)

        NumTimeSteps = t/h;          % Num of integration steps
        tau = 1;

        for g = -5:5
            X           = zeros(1, NumTimeSteps);
            oldx        = x;
```

```
X(1)           = oldx;
I              = 0;          %% no input
w              = 1;          %% connection is on but no multiplier
theta          = 1;

for TStep = 1:NumTimeSteps
    delta_x = -oldx + (w * Sigmoid(g*(oldx+ theta))) + I;
          newx = oldx + (h * delta_x);
          X(TStep+1) = newx;
          oldx = newx;

  end

          % Now display
          t = 0:NumTimeSteps;
          str = sprintf(';g= %g;', g);
          %xlabel('time'), ylabel('output'), title('output vs time');
          plot(t, X, str);
          hold on;
    end

  hold off;


end
```

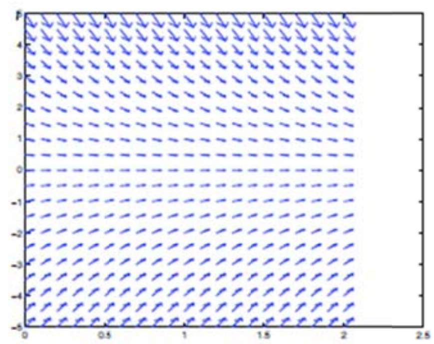


Un modelo completo

```
%% Minimal CTRNN node with recurrent connection
%% x is the initial value
%% h is the time step for integration
%% t is the length of simulation


function RecurrentNode (x, h, t)
        NumTimeSteps = t/h;      % Num of integration steps
        tau = 1;

        for w = -4:2:4
                X          = zeros(1, NumTimeSteps);
                oldx       = x;
                X(1)       = oldx;
                I          = 0;       % No input
                theta      = 0;       % No bias

        for TStep = 1:NumTimeSteps

                delta_x = -oldx + (w * Sigmoid(oldx + theta)) + I;

                newx = oldx + (h * delta_x);
                X(TStep+1) = newx;
                oldx = newx;
 end

                % Now display
                t = 0:NumTimeSteps;
                %str = sprintf(';W= %g;', w);

                %xlabel('time'), ylabel('output'), title('output vs time');
                plot(t, X);
                hold on;
            end
                hold off;
          end
```




```
%% Shows phase portrait for a given weight and bias

        function FullNodePhase (w, bias)

        [T, X] = meshgrid([0:0.1:2], [-5:0.5:5]);
        dT = ones(size(T));
        %dX = -X + (w * ((1-exp(-X)).^-1));

        dX = -X + (w * ((1 + exp(-(X + bias))).^(-1)) );

        quiver(T,X,dT,dX)

end
```



[w = −5, bais = −5]

quiver(X,Y,U,V) es una función de Matlab que dibuja los vectores U, V con flechas en los puntos X, Y. Las matrices X, Y, U, V deben tener el mismo tamaño.

Análisis de una CTRNN como un Sistema dinámico

```
% Plots the equilibria for % I = y - w * sigma (y + bias)
% w is the weight of the connection
% b is the bias

function PlotEquilibria (w, b)

    % I = y - w * sigma (y + bias)

            Y=-20:0.1:20;
            % plot for weight and bias
            str = sprintf(';b=%g;', b);
            xlabel("I");
            ylabel("y");
            plot (Y - (w *((1 + exp(-(Y + b))).^(-1)) ), Y, str);
      end
```

```
% Plots the equilibrium as a function of I
% Will plot for a bias = -5, 0 and 5
% W is the weight of the connection

function PlotEquilibriaRange (w)

    % I = y - w * sigma (y + bias)

    Y=-20:0.1:20;

    % plot for bias = -5 b
    = -5;
    PlotEquilibria(w, -5);
    hold on
    % plot for bias = -0
    PlotEquilibria(w, 0);

    % plot for bias = 5
    PlotEquilibria(w, 5);
    hold off
end

% Plots the equilibrium as a function of I
% where b is the bias
% and w is the weight of the connection

function PlotEquilibria2 (w, bias)

    % I = y - w * sigma (y + bias)
    % y = f(x)

    X=-20:0.1:20;

    % plot for bias
    str = sprintf(';w=%g, b=%g;', w, bias);
    plot (X, X - (w *((1 + exp(-(X + bias))).^(-1)) ), str)

    hold on
    % I = y
```

```
plot(X, X);

end

hold  off

function WeightMesh

    %   I = y - w  * sigma (y  + bias)
    [y, w]  = meshgrid(-20:20, -20:20);
    %i = y - (w *((exp(-y) * (((1 + exp(-(y))).^(-1))^2) )));
    i = y - (w * ((1 + exp(-(y + 0))).^(-1)));
    xlabel('i');
    ylabel('w');
    mesh(i, w,  y);

end
```