```
classdef ALG_SOM < handle
    properties
        xi % matriz de amostras de treinamento
        xin % vetor de entrada do k-ésimo padrão
        w % matriz do pesos
        wn % vetro normalizado de pesos
        eta % taxa de aprendizagem
        gau % Função de vizinhança lateral dos neurônios no espaço de saída
        lx % nº de linhas das matizes de pesos e de treinamento (nº de entrdadas)
        cx % nº de colunas da matriz de amostras de treinamneto (nº de amostras)
        cw % nº de colunas da matriz de pesos (nº de neurônios)
    end
    methods
        function obj = ALG_SOM(n_ent,n_amost, n_neu)
            % Criação da matriz de pesos e de amostaras
            obj.xi = -1*ones(n_ent+1, n_amost);% O +1 é para o bias -1
            obj.w = ones(n_ent+1,n_neu);% matriz de pesos
%             nw = ceil(0.2*n_amost);
%             sxi = 0;
%             for j = 1:n_neu
%                 for i = 1:n_ent
%                     for k = 1:nw
%                         sxi = sxi + obj.xi(i,k);
%                     end
%                     obj.w(i,j) = (sxi/nw)*5*j;
%                 end
%             end
            obj.lx = n_ent+1;% nº de entradas
            obj.cx = n_amost;% nº de amostras
            obj.cw = n_neu;% nº de neurônios
        end

        function nors(obj,entradat,iniw)
            obj.xi(2:obj.lx,:) = entradat;
            if iniw == 'aleat'
            obj.w = 6*rand(obj.lx,obj.cw)-3;% Inicialização aleatória da matriz de↙
pesos
            obj.w(1,:) = -1;
            end
%             if iniw == 'aleat'
%             obj.w = 6*rand(obj.lx,obj.cw)-3;% Inicialização grade quadrada da matriz↙
de pesos
%             obj.w(1,:) = -1;
%             end
%             if iniw == 'aleat'
%             obj.w = 6*rand(obj.lx,obj.cw)-3;% Inicialização grade exagonal da matriz↙
de pesos
%             obj.w(1,:) = -1;
%             end
            % Inicialiazação e Normalização dos vetores de pesos e de treinamento
             for xc = 1:obj.cx
                 obj.xin(:,xc) = obj.xi(:,xc)/norm(obj.xi(:,xc));
             end
             for wc = 1:obj.cw
                 obj.wn(:,wc) = obj.w(:,wc)/norm(obj.w(:,wc));
```

```
            end
            plot(obj.wn(2,:),obj.wn(3,:),'or')
            hold on
            plot(obj.xin(2,1:51),obj.xin(3,1:51),'-g')
            plot(obj.xin(2,52:102),obj.xin(3,52:102),'-b')
            grid
        end
        function treis(obj,txap,nep,r_gau)
            obj.eta = txap; %Taxa de aprendizagem fixa
%             obj.eta = txap; %Taxa de aprendizagem variável
            a=0;
            ind = 0;
            epoca = 1;
            while a==0
                a=0;
                % Cálculo da distância entre a amostra atual e todos os vetores w
                for k = 1:obj.cx
                    for j = 1:obj.cw
                        somatxw = 0;
                        for i = 1:obj.lx
                            somatxw = somatxw + (obj.xin(i,k)-obj.wn(i,j))^2;
                        end
                        dist(j) = sqrt(somatxw);
                    end
                    [mdist, ind] = min(dist); %determinação do neurônio vencedor
                    % Cálculo da distância da vizinhança e atualização
                    for j = 1:obj.cw
                        somatww = 0;
                        for i = 1:obj.lx
                            somatww = somatww + (obj.wn(i,ind)-obj.wn(i,j))^2;
                        end
                        distr = sqrt(somatww);
                        %dp = r_gau*exp-()% Desvio padrão variavel
                        obj.gau = exp(-(distr^2/(2*(r_gau)^2)));
                        obj.wn(:,j) = obj.wn(:,j)+ obj.eta*obj.gau*(obj.xin(:,k)-obj.wn↙
(:,j));
                    end

                end
                if epoca == nep
                    a=1;
                end
                epoca=epoca+1;
            end
            plot(obj.wn(2,:),obj.wn(3,:),'or')
            hold on
            plot(obj.xin(2,1:51),obj.xin(3,1:51),'-g')
            plot(obj.xin(2,52:102),obj.xin(3,52:102),'-b')
            grid
            W = obj.wn;
        end

%         function ind = valid(obj, entradav, pesost)
%             obj.xi(2:obj.lx,:) = entradav;
%             % Normalização dos vetores de pesos e de treinamento
```

```
%             for xc = 1:obj.cx
%                 obj.xin(:,xc) = obj.xi(:,xc)/norm(obj.xi(:,xc));
%             end
%             obj.wn = pesost;
%             plot3(obj.wn(1,:),obj.wn(2,:),obj.wn(3,:),'*r')
%             hold on
%             plot3(obj.xin(1,:),obj.xin(2,:),obj.xin(3,:),'*')
%             grid
%             figure
%             for k = 1:obj.cx
%                     for j = 1:obj.cw
%                         somatxw = 0;
%                         for i = 1:obj.lx
%                             somatxw = somatxw + (obj.xin(i,k)-obj.wn(i,j))^2;
%                         end
%                         dist(j) = sqrt(somatxw);
%                     end
%                     [mdist ind(k)] = min(dist);
%                     if ind == 1
%                          plot3(obj.xin(1,k),obj.xin(2,k),obj.xin(3,k),'*k')
%                     else
%                         plot3(obj.xin(1,k),obj.xin(2,k),obj.xin(3,k),'*r')
%                     end
%                     hold on
%             end
%             plot3(obj.wn(1,:),obj.wn(2,:),obj.wn(3,:),'*y')
%             grid
%         end
    end

end
```