```
function Simulador_celdas(Id)

% Software creado para la simulación de celdas de combustible tipo PEM (PEMFC)
% y celdas de combustible de etanol (DAFC).
% Simulador de curvas de polarización de celdas de combustible tipo PEM y DAFC

%  Modificado
%
%    30 de junio de 2022
%
%  Author(es):
%
%    Luis Blanco-Cocom, Salvador Botello-Rionda
%    S. Ivvan Valdez, L.C. Ordoñez
%

%  ENTRADA para el programa:
%
%  Id = 0,  indica simulación de PEMFC
%  Id = 1,  indica simulación de DAFC

if Id == 0
   Simulador_PEM
else
   Simulador_DAFC
end

disp('Simulación realizada con éxito')

end



function Simulador_PEM

%%% Simulación del modelo macro-homogéneo de Song et. al. 2008

global I_delta lc P F R T C02

F = 96493; % C mol-1
R = 8.315; % JK-1 mol-1
T = 308;   % K
P = 1;     %atm
Ho2 = exp(-666/T +14.1);
xo2 = 0.21;
D02_gdl = 2.396e-1; % cm2 s-1
L_gdl = 100e-2;
e_gdl = 0.3 ;
```

```
Do2eff_gdl = D02_gdl*sqrt(e_gdl*e_gdl*e_gdl);
lc = 11.8e-4;

Ko2 = (1/(R*T))*exp(-666/T +14.1)*0.101325; % adimensional
C0 = P*xo2*0.101325/(R*T);                  % adimensional;

% densidades de corriente
vec_I = [0.001 0.005 0.02 0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5 0.55 0.6 0.65  0.65 0.67 0.7
0.71];

V_cell = zeros(length(vec_I), 1);
act_pot = zeros(length(vec_I), 1);
ohm_pot = zeros(length(vec_I), 1);
C_O2= zeros(length(vec_I), 1);

E_rev = 1.23;
R_ohm = 0.47;

for i=1:length(vec_I)

    I_delta = vec_I(i);

    C02 = (C0 - I_delta*L_gdl/(4*F*Do2eff_gdl)/Ko2 );

    solinit = bvpinit(linspace(0,lc,10),[10, C02, 0.567, 0]);

    options = bvpset('Stats','on','RelTol',1e-10);

    sol = bvp5c(@ivp3, @BVP_bc3, solinit, options);

   t = sol.x;
   y = sol.y;

   V_cell(i) = E_rev - y(3,end) - R_ohm*I_delta/10000;

   act_pot(i) = y(3,end);

   ohm_pot(i) = R_ohm*I_delta/10000;

   C_O2(i) = y(2,1);

   [ E_rev  y(3,end)  R_ohm*I_delta/10000 ]

   pal = ['resultados/song_ord_' num2str(I_delta)];
   pal = [pal '.txt']
   fi3 = fopen(pal, 'w') ;

   lont = length(t);
```

```
    for hh = 1:lont
      fprintf(fi3, '%3.12f %3.12f  %3.12f %3.12f %3.12f \n', t(hh), y(1, hh),  y(2,hh), y(3, hh), y(4,hh));
    end
    fclose(fi3) ;

end

fi = fopen('resultados/polarizacion.txt', 'w') ;
lont = length(vec_I);
for hh = 1:lont
     fprintf(fi, '%3.12f %3.12f  \n', vec_I(hh), V_cell(hh));
end
fclose(fi);

plot(vec_I, V_cell, 'r-');
%axis([0 20 0 1.14])
grid on
title('Densidad de corriente vs voltaje - PEMFC')
xlabel('Densidad de correinte mA cm^{-2}')
ylabel('E_cell - Voltaje')

end

function xdot = ivp3(t,y)

global I_delta lc F R T

xdot=zeros(4,1);

mPt =0.332e-3; %*100*100/(1000); %  debe estar en kg/m3
pPt = 21.500;  %  debe estar en kg/m3
e_N = 0.33;    %0.33;

pC = 2.000;    % debe estar en kg/m3

porPt = 0.8;   % 30%
As = 11*100*100 ; %  cm2/g  (227.79*f*f*f - 158.57*f*f - 201.53*f + 159.5)*1e3;

alpha_c = 1;
alpha_a = 0.5;
CO2_ref = 1.2e-6; % mol/cm3

D02_N = 1.844e-6; % cm2/s
D02_W = 3.032e-5; % cm2/s


e_S = ( (1/pPt) + (1-porPt)/(porPt*pC) )*(mPt/lc);
e_V = 1 - e_N  - e_S;
```

```
D02eff_N = D02_N*sqrt(e_N*e_N*e_N);
D02eff_W = D02_N*sqrt(e_V*e_V*e_V);

aux  = e_N/D02eff_N +e_V/D02eff_W;

Do2_eff = (e_N + e_V)/aux;


kappa = 0.17; % S/cm
aux = e_N;
km_eff = sqrt(aux*aux*aux)*kappa;

sigma = 7.27e2; % S/cm
aux = e_S;
ks_eff = sqrt(aux*aux*aux) * sigma;

Av = mPt*As/lc;

aux = 3.507 - 4001/T;
i0_ref = 10^aux;

didz = Av*i0_ref* ( (y(2)/CO2_ref)* exp((alpha_c*F/(R*T))*y(3)) -
exp(((-1)*alpha_a*F/(R*T))*y(3) ) );

xdot = [
        (-1/(4*F))*didz
        (-1)*y(1)/Do2_eff
        y(4)/km_eff  + (y(4) - I_delta)/ks_eff
        didz
       ];

end

function res = BVP_bc3(ya,yb)

global I_delta C02


res = [
      yb(1) - 0
      ya(2) - C02
      ya(4) - 0
      yb(4) - I_delta
      ];

end

function Simulador_DAFC
```

```
% Simulación de Pramanik et al. 2010

% 1 Joule = Coulomb*Volt
% 1 Coulomb = 1 Amp*seg
% 1 Omh (Omega) = Volt/Amp

i_cell = 0.1:0.1:15;          %% corriente en Acm-2
%i_cell = i_cell./1000;       %% conversión a mAcm-2 ???

N_dim= length(i_cell);
V_cell = zeros(N_dim,1);

%%% Lo necesario para calcular E_cell se divide en 4 partes

E = 0.8; %1.144;          % Voltaje inicial E0

%%%% Parámetros para la parte 1 p1

F = 96500;          % Faraday constant, Cmol^-1
R = 8.314;          % Gas constan, J/molK
T = 70 + 273.15;     % °C -> K
alpha_a = 0.39;      % Anode transfer coefficient at 70 °C
n = 12;             % number of electrons
C_EtOH = 3;         % 0.003 mol/ml assumed same as feed concentrationc of ethanol.
C_H2O =  1;         % water concentration
K_EtOH = 1;         % Coulomb(molcm)^-0.5 s^-1

%%%% Parámetros para la parte 2 p2

lambda = 11;        % sin dimensión
t_m = 0.00145;       % cm       %
R_b = 0.006;        % Omega cm2 = V/A*cm2

%%%% Parámetros para la parte 3 p3

k_d = 4.63e-3;       % cm s-1
C_F_EtOH = C_EtOH;    % mol/ml ethanol feed concentration se cambió a mol/L
rho_H2O= 0.997;       % water density 1.0 g cm-3
M_H2O = 18.01528 ;    % molecular weight of water g/mol;  (kg/m3)         %
lambda_H2O = 3.16;    % sin dimensión
j0 = 0.04;           % 0.04 mAcm-2 --- > Acm-2

%%%% Parámetros para la parte 4 p4

alpha_c = 0.38       % cathode transfer coefficient
l_d = 0.003;         % thickness of carbon paper
C_ob = 0.7;          % concentration of oxygen in water vapor (mol/ml) ¿qué poner?
n2 = 3;             % electrons in the cathode for oxygen
```

```
e_d = 0.834;         % void fraction of carbon paper
D_O2 = 0.31  ;       % bulk diffusivity of oxygen–water vapor (cm 2 s −1 )
T_c = 50 + 273.15; % °C -> K

M = l_d*C_ob/(n2*F*D_O2*e_d^(3.0/2.0));   % Equation (31)

for i=1:N_dim

  % parte 1
  p1 = (R*T/(alpha_a*n*F))*log(i_cell(i)/((C_EtOH^0.25)*(C_H2O^0.25)*K_EtOH))

  % parte 2
  ep = (0.005139*lambda - 0.003260)*exp(1268*( (1.0/303) - (1.0/T) ));
  p2 = i_cell(i)*( t_m/ep + R_b )

  % parte 3
  v_d = (M_H2O*i_cell(i)/(rho_H2O*F))*( 1.0/n + lambda_H2O );   % Eq 23
  e1 = exp(v_d/k_d);
  e2 = i_cell(i)/(n*F*v_d*C_F_EtOH);
  k_f = 1.87e-4*(i_cell(i)/0.003)^0.32;                    % Eq. 16 mass transfer coefficient of ethanol from
feed stream to carbon paper (cm s −1 )
  e3 = exp(v_d/k_d)*( ( (v_d/k_f) + 1.0) - 1.0 );
  cp = i_cell(i)/(j0*(e1 + e2*e3 ));
  p3 = (R*T/(alpha_a*n*F))*log(cp)

  % parte 4
  p4 = R*T_c/(alpha_c*n*F)*log(i_cell(i)/(j0*(1.0 - i_cell(i)*M)))

  V_cell(i) = E - p1 -p2 - p3 - p4;

end


plot(i_cell, V_cell, 'r-');
axis([0 20 0 1.14])
grid on
title('Densidad de corriente vs voltaje - DAFC')
xlabel('Densidad de correinte mA cm^{-2}')
ylabel('E_cell - Voltaje')

end
```

```
function [SOL, suma, V_sim, T_total] = Optimizador_version1(val1)

% Software en desarrollo para la optimización y estimación de parámetros
% Se presenta un algoritmo de estimación de distribución (UMDA^G)
%

%  Modificado
%
%    30 de junio de 2022
%
%  Author(es):
%
%    Luis Blanco-Cocom, Salvador Botello-Rionda
%    S. Ivvan Valdez, L.C. Ordoñez
%

% Entrada:

% val1 = variable que indica el número de archivo a generar.
% Necesita de una función evaludadora, en este caso Song.m

tiempo_inicio = cputime;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%set(gca,'nextplot','replacechildren');
format long;


H = [0.00100000000000   0.974603924015115
   0.005000000000000   0.906780013420890
   0.020000000000000   0.831766532876598
   0.050000000000000   0.779842920214462
   0.100000000000000   0.737359493898511
   0.150000000000000   0.709844821813715
   0.200000000000000   0.688228963154712
   0.250000000000000   0.669619890399399
   0.300000000000000   0.652672758448303
   0.350000000000000   0.636600617036280
   0.400000000000000   0.620842482873290];


vec_I = H(:,1);
V_exp = H(:,2);


%% Parametros del algoritmo
nvars = 4;
```

```
Ngen=200*nvars;      %cantidad de generaciones
Nind=100;      %cantidad de individuos 30 -15
M = 50;
val = 2;
%% dominio de los 4 parámetros

min=[2e-1, 0.2e-3, 21, 0.31];
max=[4.0e-1, 0.4e-3, 22, 0.35];

n=ran(min,max,Nind) ; %n=(iind,2)-->coord y del individuo iind

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%mkdir '../Song' % se cre la carpeta donde se guardarán las soluciones

medias_vars = ['../Song_UMDA_3/sim_mv_' num2str(val1) '_' num2str(val) '.txt']; % se guardan las
mediasy virianzas de las generaciones
fi2 = fopen(medias_vars, 'w')

lN =  length(min);

iter = 1;
norma_DE(1) = 1.01e-2;

while ((iter <= Ngen) && (norma_DE(iter) > 1.0e-12))

   for k=1:Nind % calcula el error cuadratico

      [suma V_sim] = Song(vec_I, V_exp, n(k,:));
      rank(k,:)=[suma suma 0 k n(k,:)]; % la segunda se convierte en probabilidades

   end

   fitness(iter)=1/(sum(rank(:,1)) + 0.0001);

   orden=sortrows(rank) ;

   if iter == 1

      M = Nind;
      aux_M = orden(M,1);

   else

      if M == Nind

         if rem(M,2) == 0
```

```
            cont_i = M/2;
        else

            cont_i = (M+1)/2;
        end
    end

    while (orden(cont_i,1) > aux_M ) && (cont_i > 2)
          cont_i = cont_i - 1;
    end

    if (rand > 0.5) % && (cont_i > 1)
       cont_i = cont_i - 1*floor(rand*50);

    end

    if (norma_DE(iter) < 1e-6)
      if rem(M,2) == 0
          cont_i = M/2;
      else
          cont_i = (M+1)/2;
      end

    end

    if (cont_i < 1)

      cont_i = 6;

    end

    M = cont_i;
    aux_M = orden(M,1);

end

M_vec(iter) = M;


if val == 0

     medias = mean(orden(1:M, 5:(lN+4)) );
     vars = std(orden(1:M, 5:(lN+4)));

elseif val==1
   %orden=sortrows(rank) ;

   aux_orden = ones(Nind,1)*(orden(Nind,1)) - orden(:,1) + 1e-5;
   orden(:,2) = aux_orden /sum(aux_orden) ;
```

```
      aux_orden = ones(M,1)*(orden(M,1)) - orden(1:M,1) + 1e-5;
      orden(1:M,3) = aux_orden /sum(aux_orden) ;                % se calculan proporciones de la
poblacion total

      for vi = 1:lN
         aux_media = orden(1:M, 3).*orden(1:M, (vi+4));
         medias(vi) = sum(aux_media);
         aux_var = orden(1:M, (vi+4)) - ones(M,1)*medias(vi);
         vars(vi) = sqrt( sum(orden(1:M, 3).*aux_var.*aux_var)); % sqrt( (1/(M-
1))*sum(aux_var.*aux_var));
      end

   elseif val == 2

         %orden=sortrows(rank);

         errores_vec = rank(:,1);

         contar = zeros(Nind,1);

         for i=1:Nind
            contar(i)= 0;
            for j=1:Nind
               if orden(j,1) >= orden(i,1)
                  contar(i) = contar(i) + 1;
               end
            end
         end

         orden(:,1) = contar;


         aux_orden =  orden(:,1) - ones(Nind,1)*(orden(1,1)) + 1e-5;
         orden(:,2) = aux_orden /sum(aux_orden) ;

         aux_orden = orden(1:M,1) - ones(1,1)*(orden(M,1)) + 1e-5;
         orden(1:M,3) = aux_orden /sum(aux_orden) ;                % se calculan proporciones de la
poblacion total


         for vi = 1:lN
           aux_media = orden(1:M, 3).*orden(1:M, (vi+4));
           medias(vi) = sum(aux_media);
           aux_var = orden(1:M, (vi+4)) - ones(M,1)*medias(vi);
           vars(vi) = sqrt( (1/(M-1))*sum(aux_var.*aux_var));
         end
```

```
    end
    % generar nuevos individuos


   fobj = ['../Song_UMDA_3/Ind_fobj_' num2str(val1) '_gen_' num2str(iter) '_' num2str(val) '.txt']; % se
guardan las generaciones
   fio = fopen(fobj, 'w');

   for Tg = 1:Nind

        fprintf(fio, '%d ', Tg);

        for gg = 1: lN + 4
          fprintf(fio, '%3.10f ', orden(Tg,gg) );
        end
        %fprintf(fio, '%3.10f \n', rank(Tg,1) );
   end
   fclose(fio);


   % se calculan la nueva generación

   n(1,:)= orden(1, 5:(lN+4));

   for kk = 2:Nind
       for chec = 1: lN
          n(kk,chec) = normrnd(medias(chec), vars(chec));
          %[min(chec) n(kk,chec) max(chec) ]
          while ( (n(kk,chec) < min(chec)) || (n(kk,chec) > max(chec)) )
                n(kk,chec) = normrnd(medias(chec), vars(chec));
          end
       end

    end

    for kk = 1:lN
        vars(kk) = vars(kk)*vars(kk)/(max(kk) - min(kk));  % varianza normalizada
    end

    iter = iter + 1;
    norma_DE(iter) = norm(vars);
    %norma_DE(iter) = norm(std(orden(1:M, 3:(lN+2))));


    for gg = 1: lN
       fprintf(fi2, '%3.10f ', medias(gg) );
    end

    for gg = 1: lN
```

```
        fprintf(fi2, '%3.10f ', vars(gg) );
    end
    fprintf(fi2, '\n ')  ;


end

fclose(fi2);

%iter
[suma V_sim] = Song(vec_I, V_exp, n(1,:));

SOL = n(1,:);

disp('Simulación realizada con éxito');

T_total = cputime - tiempo_inicio;
```