```
% This is a MATLAB program file
% It solves drift diffusion and poisson by propagating in time.
% The equations implemented are suitable for organic LEDs
% Please check that the units and constants are OK
%have fun
%       Nir Tessler
%
clear excitons time

figure(1);
set(1,'Position', [13 369 435 320]);
figure(2);
set(2,'Position', [493 331 484 356]);
figure(3);
set(3,'Position', [ 21 37 435 285 ]);

%parameters and constants

Norm_C = 1e20;
DT = 1e-10; %time interval sec
mup = 1e-5; %hole mobility cm^2/v/s
mun = 1e-6; %electron mobility cm^2/v/s
q = 1.602e-19; % C
eps0 = 8.854e-14; % F/cm
eps=3*eps0;
k = 1.381e-23;
T = 300; % kelvin
kT_q=k*T/q; % eV

p0 = 1e19/Norm_C; %p  at anode cm^-3
nd = 1e19/Norm_C; %n  at cathode cm^-3
Ni=1e+10/Norm_C;
Lang = q*(mup+mun)/eps; %Langevin recombination rate

Vappl=5; %V
Va = 0; %potential at anode
Vc = Vappl; %potential at cathode

%device grid
x1 = 0;
dx = 1e-7; % 1e-7cm = 1nm
x2 = 100e-7; %100e-7cm = 100nm

x = [-dx x1:dx:x2 x2+dx]';
NP = round((x2-x1)/dx)+1;
NNP=NP+2;

%setup sparse matrix for finite difference 1st derivative --> D1*x = x'
D1 = spalloc(NP+2,NP+2,2*NP+4);
D1(1,1) = 0; % point (1,1) is virtual
D1(NP+2,NP+2)=0; %point (NP+2,NP+2) is virtual

for i = 2:NP+1
D1(i,i-1)=-1/(x(i+1)-x(i-1));
D1(i,i+1)=1/(x(i+1)-x(i-1));
end


%D11 in cm
D11 = spalloc(NP+2,NP+2,2*NP+4);
D11(1,1) = 0; % point (1,1) is virtual
D11(NP+2,NP+2)=0; %point (NP+2,NP+2) is virtual
for i = 2:NP+1
D11(i,i)=-1/(x(i+1)-x(i));
D11(i,i+1)=1/(x(i+1)-x(i));
end


%D12 in cm
D12 = spalloc(NP+2,NP+2,2*NP+4);
D12(1,1) = 0; %point (1,1) is virtual
D12(NP+2,NP+2)=0; %point (NP+2,NP+2) is virtual
for i = 2:NP+1
D12(i,i-1)=-1/(x(i)-x(i-1));
```

```
D12(i,i)=1/(x(i)-x(i-1));
end


%setup sparse matrix for finite difference 2nd derivative --> D2*x = x''
D2 = spalloc(NP+2,NP+2,3*NP+6);
D2(1,1)=0; %don't change virtual grid point when calculating p+dp or n+dn
D2(NP+2,NP+2)=0; %don't change virtual grid point when calculating p+dp or n+dn
for i = 2:NP+1
D2(i,i-1)=2*(x(i+1)-x(i))/((x(i)-x(i-1))*(x(i+1)-x(i))*(x(i+1)-x(i-1)));
D2(i,i)=-2/((x(i+1)-x(i))*(x(i)-x(i-1)));
D2(i,i+1)=2*(x(i)-x(i-1))/((x(i)-x(i-1))*(x(i+1)-x(i))*(x(i+1)-x(i-1)));
end

%this sparse matrix also calculates 2nd derivative but also enforces boundary conditions on potential
%if the charge distribution (multiplied by relevant constants) is premulitplied by the inverse of this matrix,
%you can determine the potential and Poisson's Equation is solved
PoisD2 = spalloc(NP+2,NP+2,3*NP+6);
PoisD2(1,1)=1;
PoisD2(2,2)=1; %this applies boundary condition V0 at anode
PoisD2(NP+1,NP+1)=1; %this applies boundary condition Vd at cathode
PoisD2(NP+2,NP+2)=1;
for i = 3:NP
PoisD2(i,i-1)=2*(x(i+1)-x(i))/((x(i)-x(i-1))*(x(i+1)-x(i))*(x(i+1)-x(i-1)));
PoisD2(i,i)=-2/((x(i+1)-x(i))*(x(i)-x(i-1)));
PoisD2(i,i+1)=2*(x(i)-x(i-1))/((x(i)-x(i-1))*(x(i+1)-x(i))*(x(i+1)-x(i-1)));
end


%set initial potential and charge distributions (no charges at beginnning)
p = ones(NP+2,1)*Ni; %.*exp(-(x)/200)+1e-10;
n = ones(NP+2,1)*Ni;

dp = zeros(NP+2,1);
dn = zeros(NP+2,1);
V = zeros(NP+2,1);
for i=1:NP+2
V(i) = -(Va +(Vc-Va)*x(i)/x2);
end


%loop for main program
for loop = 1:200000


%calculate new charge distributions
dp = mup*DT.*(D12*((D11*p))*(kT_q) - (D1*p).*(D1*V) - p.*(D2*V)) - DT.*Lang.*(p.*n-Ni*Ni).*Norm_C ;
dn = mun*DT.*(D12*((D11*n))*(kT_q) + (D1*n).*(D1*V) + n.*(D2*V)) - DT.*Lang.*(p.*n-Ni*Ni).*Norm_C ;
p = p + dp;
n = n + dn;

%set boundary
p(2) = p0; p(1)=p(2); n(NP+1) = nd; n(NP+2)=n(NP+1); % for injecting contacts
n(2) = Ni*Ni/p(2); n(1)=n(2); p(NP+1) = Ni*Ni/n(NP+1); p(NP+2)=p(NP+1); % for collecting contacts



%solve poisson's equation
Q = Norm_C*(q/eps)*(p(3:NP)-n(3:NP)); %notice scaling factor c0 appears here
Q_mat=[Va; Va; Q; Vc; Vc];
V = mldivide(PoisD2,Q_mat); % solves for V*PoisD2=Q_mat

%plot data

if mod(loop,200) == 0
loop
  figure(1);
  semilogy(x(2:NP+1),p(2:NP+1)*Norm_C,x(2:NP+1),n(2:NP+1)*Norm_C);
  title('charge distributions');
  axis([x(2),x(NP+1),1e15,1e20]);
  drawnow;
  figure(2)
  plot(x(2:NP+1),Lang*(p(2:NP+1).*n(2:NP+1)-Ni*Ni)*Norm_C*Norm_C*1e-9);
  title('exciton generation rate distribution cm^-3/ns');
```

```
  axis([x(2),x(NP+1),0,max(Lang*(p(2:NP+1).*n(2:NP+1)-Ni*Ni)*Norm_C*Norm_C*1e-9)]);
  drawnow
  excitons(round(loop/200))=sum(Lang*(p(2:NP+1).*n(2:NP+1)-Ni*Ni)*Norm_C*Norm_C*1e-9);
  time(round(loop/200))=loop*DT*1e6;
  figure(3)
  plot(excitons);
  ylabel('excitons');
  xlabel('time [us]');
 end

end
end
```