ECSE 597: Circuit Simulation and Modelling
Assignment 3

Student Name: Dafne Culha
Student ID: 260785524

Department of Electrical and Computer Engineering
McGill University, Canada
November, 2021

1. *dcsolvealpha.m*

```
function Xdc = dcsolvealpha(Xguess,alpha,maxerr)
% Compute dc solution using newtwon iteration for the augmented system
% G*X + f(X) = alpha*b
% Inputs:
% Xguess is the initial guess for Newton Iteration
% alpha is a paramter (see definition in augmented system above)
% maxerr defined the stopping criterion from newton iteration: Stop the
% iteration when norm(deltaX)<maxerr
% Oupputs:
% Xdc is a vector containing the solution of the augmented system

global G C b DIODE_LIST npnBJT_List


converged = false;

delta_x = 1234567890;

% slides B45/48
while (norm(delta_x) >= maxerr)

    f = f_vector(Xguess);

    disp ('f=');
    disp (f);

    phi = G * Xguess + f - alpha * b;


    J = nlJacobian(Xguess);

    disp ('J=');
    disp(J);


    delta_x = -1 * J \ phi;

    Xguess = Xguess + delta_x;


    disp ('Xdc=');
    disp(Xguess);

  %  if (delta_x >= maxerr)
   %     converged = true;
    %end

end

Xdc = Xguess;
```

2. *dcsolvecont.m*

Vi was set at different values and the following code was run to test this function.

Vi=10;

```
Sedra4_93
Xdc = dcsolvecont(10,1e-6)

Vi=2;
Sedra4_93
Xdc = dcsolvecont(10,1e-6)

Vi=-8;
Sedra4_93
Xdc = dcsolvecont(10,1e-6)
```

At Vi = 10V

```
Xdc =

    1.0000
    3.0702
    3.7895
   10.0000
   -2.0000
   -2.0000
    0.0021
   -0.0021
    0.0000
```

At Vi = 2V

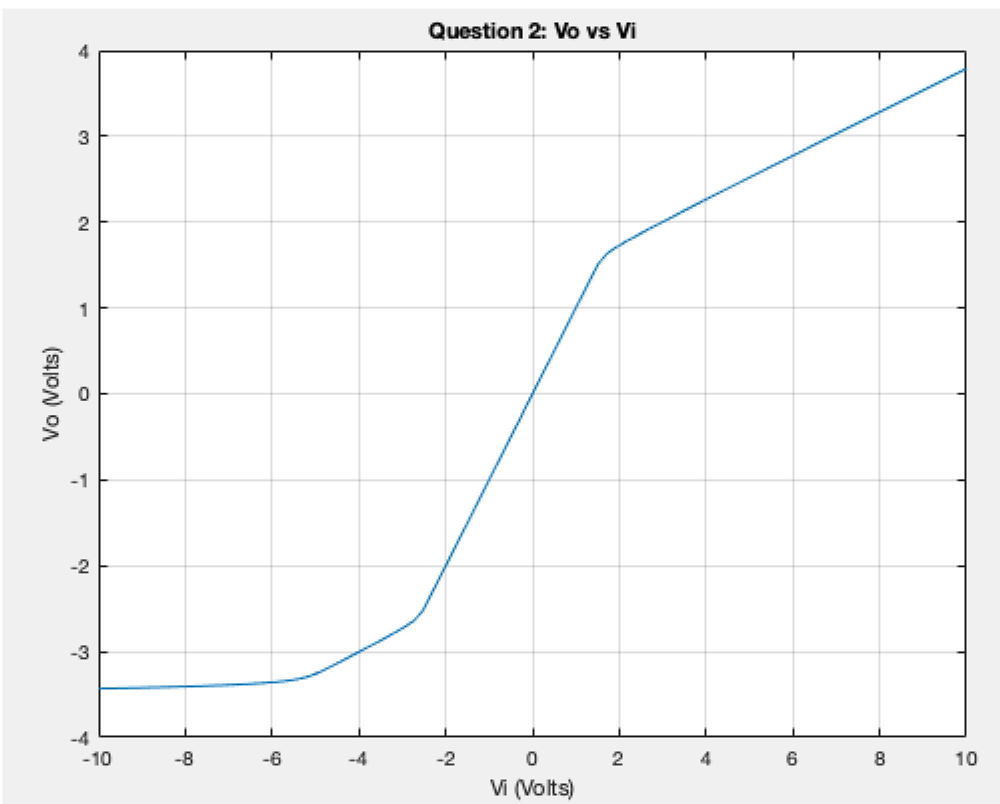
```
Xdc =

    1.0000
    1.0905
    1.7284
    2.0000
   -2.0000
   -2.0000
    0.0001
   -0.0001
    0.0000
```

At Vi = -8V

Xdc =

1.0000
1.0000
-3.4072
-8.0000
-2.6957
-2.0000
-0.0000
0.0015
-0.0015

2. TestBenchDiodeckt4_93.m was run to compute and plot Vo as a function of Vi. Following plot was observed. The values observed in the plot align with DC values obtained in the output node (node 3) in the previous part.



```
function Xdc = dcsolvecont(n_steps,maxerr)
% Compute dc solution using newtwon iteration and continuation method
% (power ramping approach)
% inputs:
% n_steps is the number of continuation steps between zero and one that are
% to be taken. For the purposes of this assigments the steps should be
% linearly spaced (the matlab function "linspace" may be useful).
% maxerr is the stopping criterion for newton iteration (stop iteration
% when norm(deltaX)<maxerr


global G C b DIODE_LIST npnBJT_List

% Slides C00 10/15
sz = length(G);
Xdc = zeros(sz, 1);
%y = linspace(x1,x2,n) generates n points. The spacing between the points is (x2-x1)/(n-1).
steps = linspace(0,1,n_steps);

for i = 1 : n_steps
    Xdc = dcsolvealpha(Xdc, steps(i), maxerr);
end
```

3. *BJT implementation*

1. *Run the BJT_CB.m using the dcsolve.m function you implemented in Assignment 1. Report your observations.*

Without power ramping it keeps iterating forever and doesn't converge although I waited for a long term and set the error tolerance quite high.

2. *Run the BJT_CB.m using the dcsolvecont.m function you implemented in Question II. Report the values of the nodal voltages obtained after using dcsolvecont.m.*

Xdc =

```
  10.0000
   5.3197
   6.0000
   5.2964
  -0.0010
  -0.0006
```

1) `f_vector.m` with BJT

```matlab
function f = f_vector(X)
% Compute the nonlinear vector of the MNA equations as a function of x

global b DIODE_LIST npnBJT_LIST

N = size(b);
f = zeros(N); % Initialize the f vector (same size as b)

NbDiodes = size(DIODE_LIST,2);
NbBJTs = size(npnBJT_LIST,2);

%% Fill in the Fvector for Diodes
for I = 1:NbDiodes
    node_i = DIODE_LIST(I).node1
    node_j = DIODE_LIST(I).node2
    Vt = DIODE_LIST(I).Vt; % Vt of diode (part of diode model)
    Is = DIODE_LIST(I).Is; % Is of Diode (part of diode model)

    if (node_i ~= 0) && (node_j ~= 0)
        v1 = X(node_i); %nodal voltage at anode
        v2 = X(node_j); %nodal voltage at cathode
        diode_current = Is*(exp((v1-v2)/Vt)-1);
        f(node_i) = f(node_i) + diode_current;
        f(node_j) = f(node_j) - diode_current;
    elseif (node_i == 0)
        v2 = X(node_j); %nodal voltage at cathode
        diode_current = Is*(exp((-v2)/Vt)-1);
        f(node_j) = f(node_j) - diode_current;
    elseif (node_j == 0)
        v1 = X(node_i); %nodal voltage at anode
        diode_current = Is*(exp((v1)/Vt)-1);
        f(node_i) = f(node_i) + diode_current;
    end
end

%% Fill in the Fvector for BJTs
for I=1:NbBJTs

     %get Nodes Numbers
     cNode = npnBJT_LIST(I).collectorNode;
     bNode = npnBJT_LIST(I).baseNode;
     eNode = npnBJT_LIST(I).emitterNode;

     % get other parameters
     Vt = npnBJT_LIST(I).Vt;
     Is = npnBJT_LIST(I).Is;
     alphaR = npnBJT_LIST(I).alphaR;
     alphaF = npnBJT_LIST(I).alphaF;


    % if both nodes not grounded

    if(cNode~=0)&&(bNode~=0)&&(eNode~=0) % all nodes present
         % get nodal voltages

         Vbe = X(bNode) -X(eNode);
         Vbc = X(bNode) -X(cNode);
         % diode currents
         If = Is*(exp(Vbe/Vt)-1);
         Ir = Is*(exp(Vbc/Vt)-1);
```

```
        f(cNode) =  f(cNode)  -Ir           +    alphaF*If ;
        f(bNode) =  f(bNode)  +Ir*(1-alphaR) +   If*(1-alphaF);
        f(eNode) =  f(eNode)  +Ir*alphaR    -    If;


    elseif (cNode~=0)&&(bNode==0)&&(eNode~=0) % Base is Grounded

        Vbe = -1*X(eNode);
        Vbc = -1*X(cNode);
        % diode currents
        If = Is*(exp(Vbe/Vt)-1);
        Ir = Is*(exp(Vbc/Vt)-1);

        f(cNode) =  f(cNode)  -Ir           +    alphaF*If ;
        %f(bNode) =  f(bNode)  +Ir*(1-alphaR) +   If*(1-alphaF);
        f(eNode) =  f(eNode)  +Ir*alphaR    -    If;

    elseif (cNode~=0)&&(bNode~=0)&&(eNode==0) % Emitter is Grounded
         % fill this up


         Vbe = X(bNode);
         Vbc = X(bNode) -X(cNode);
         % diode currents
         If = Is*(exp(Vbe/Vt)-1);
         Ir = Is*(exp(Vbc/Vt)-1);

        f(cNode) =  f(cNode)  -Ir           +    alphaF*If ;
        f(bNode) =  f(bNode)  +Ir*(1-alphaR) +   If*(1-alphaF);
        %f(eNode) =  f(eNode)  +Ir*alphaR    -    If;

   elseif (cNode==0)&&(bNode~=0)&&(eNode~=0) % Collector is Grounded
     % fill this up

     % get nodal voltages

     Vbe = X(bNode) -X(eNode);
     Vbc = X(bNode);
     % diode currents

     If = Is*(exp(Vbe/Vt)-1);
     Ir = Is*(exp(Vbc/Vt)-1);


     f(bNode) =  f(bNode)  +Ir*(1-alphaR) +   If*(1-alphaF);
     f(eNode) =  f(eNode)  +Ir*alphaR    -    If;



   elseif(cNode~=0)&&(bNode==0)&&(eNode==0) % Base and  Emitter are grounded
       % get nodal voltages
       Vbe = 0;
       Vbc = -X(cNode);
       % diode currents
       If = Is*(exp(Vbe/Vt)-1);
       Ir = Is*(exp(Vbc/Vt)-1);

       f(cNode) =  f(cNode)  -Ir           +    alphaF*If ;

    end
end %end Forloop for BJTs
```

2) nlJacobian.m with BJT

```
function J = nlJacobian(X)
% Compute the jacobian of the nonlinear vector of the MNA equations as a
% function of X
% input: X is the current value of the unknown vector.
% output: J is the jacobian of the nonlinear vector f(X) in the MNA
% equations. The size of J should be the same as the size of G.

global G DIODE_LIST npnBJT_LIST
N = size(G);
J = zeros(N);
d_diode = zeros(N);
d_BJT = zeros(N);

%% Add the Jacobian for diode --
%copy paste the one you implemented in your previous assignment


NbDiodes = size(DIODE_LIST,2);

if (NbDiodes ~= 0)

for I = 1:NbDiodes
    node_i = DIODE_LIST(I).node1;
    node_j = DIODE_LIST(I).node2;
    v1 = X(node_i); %nodal voltage at anode
    v2 = X(node_j); %nodal voltage at cathode
    Vt = DIODE_LIST(I).Vt; % Vt of diode
    Is = DIODE_LIST(I).Is; % Is of Diode

    if (node_i ~= 0) && (node_j ~= 0)

        %diode_current = Is*(exp((v1-v2)/Vt)-1);
        % derivative of diode current wrt v1
        d_I_v1 = (Is/Vt)*exp((v1-v2)/Vt);
        % derivative of diode current wrt v2
        d_I_v2 = -(Is/Vt)*exp((v1-v2)/Vt);

        % f(node_i) = f(node_i) + diode_current;
        % differentiate f(node_i) wrt v1
        d_diode(node_i, node_i) = d_diode(node_i, node_i) + d_I_v1;
        % f(node_i) = f(node_i) + diode_current;
        % differentiate f(node_i) wrt v2
        d_diode(node_i, node_j) = d_diode(node_i, node_j) + d_I_v2;
        %f(node_j) = f(node_j) - diode_current;
        % differentiate f(node_j) wrt v1
        d_diode(node_j, node_i) = d_diode(node_j, node_i) - d_I_v1;
        %f(node_j) = f(node_j) - diode_current;
        % differentiate f(node_j) wrt v2
        d_diode(node_j, node_j) = d_diode(node_j, node_j) - d_I_v2;

    elseif (node_i == 0)
        % diode_current = Is*(exp((v1-v2)/Vt)-1);
        % derivative of diode current wrt v2
        d_I_v2 = -(Is/Vt)*exp((v1-v2)/Vt);
        %f(node_j) = f(node_j) - diode_current;
        % differentiate f(node_j) wrt v2
```

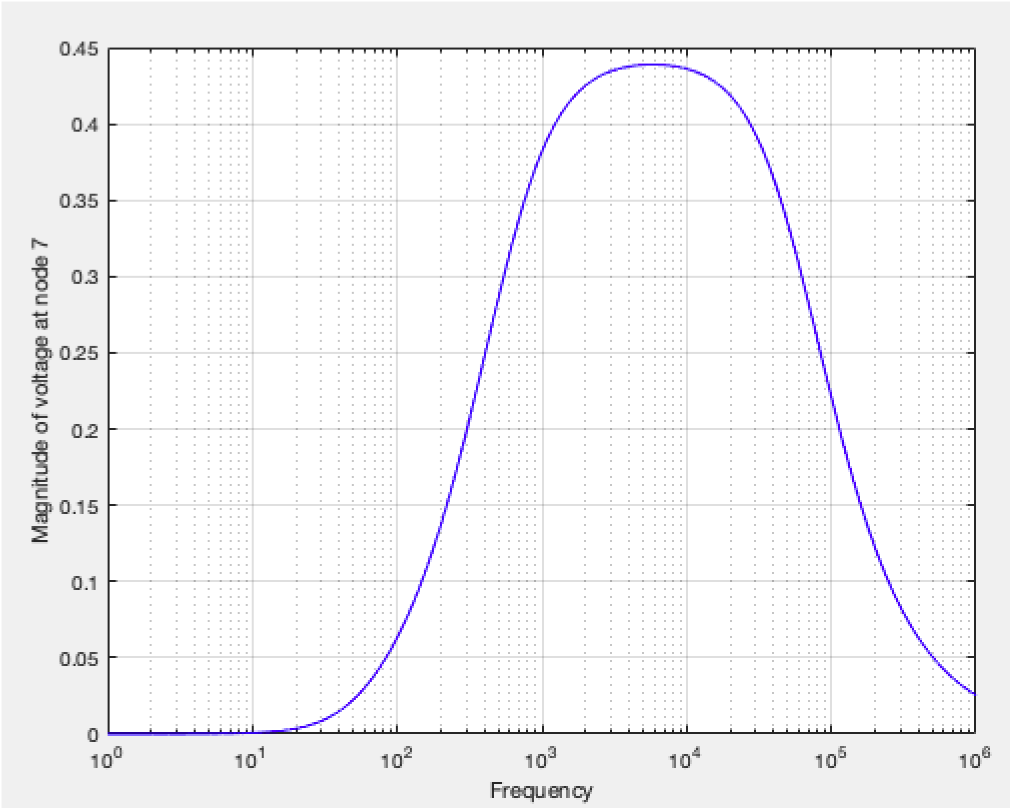
```
        d_diode(node_j, node_j) = d_diode(node_j, node_j) - d_I_v2;

    elseif (node_j == 0)
        % diode_current = Is*(exp((v1-v2)/Vt)-1);
        % derivative of diode current wrt v1
        d_I_v1 = (Is/Vt)*exp((v1-v2)/Vt);
        % f(node_i) = f(node_i) + diode_current;
        % differentiate f(node_i) wrt v1
        d_diode(node_i, node_i) = d_diode(node_i, node_i) + d_I_v1;

    end
end

end


%% Add the Jacobian for BJT

NbBJTs = size(npnBJT_LIST,2);

if (NbBJTs ~= 0)

for I=1:NbBJTs

     %get Nodes Numbers
     cNode = npnBJT_LIST(I).collectorNode;
     bNode = npnBJT_LIST(I).baseNode;
     eNode = npnBJT_LIST(I).emitterNode;

     % get other parameters
     Vt = npnBJT_LIST(I).Vt;
     Is = npnBJT_LIST(I).Is;
     alphaR = npnBJT_LIST(I).alphaR;
     alphaF = npnBJT_LIST(I).alphaF;

     if(cNode~=0)&&(bNode~=0)&&(eNode~=0) % all nodes present

         % get nodal voltages
         Vbe = X(bNode) -X(eNode);
         Vbc = X(bNode) -X(cNode);

         % diode currents
         % If = Is*(exp(Vbe/Vt)-1);
         % derivative of If wrt Vc
         delta_If_c = 0;
         % derivative of If wrt Vb
         delta_If_b = Is/Vt*exp(Vbe/Vt);
         % derivative of If wrt Ve
         delta_If_e = -Is/Vt*exp(Vbe/Vt);

         % Ir = Is*(exp(Vbc/Vt)-1);
         % derivative of Ir wrt Vc
         delta_Ir_c = -Is/Vt*exp(Vbc/Vt);
         % derivative of Ir wrt Vb
         delta_Ir_b = Is/Vt*exp(Vbc/Vt);
         % derivative of Ir wrt Ve
         delta_Ir_e = 0;

         %f(cNode) =  f(cNode)  -Ir            +    alphaF*If ;
         % derivative f(cNode) wrt Vc
         d_BJT(cNode, cNode) = d_BJT(cNode, cNode) - delta_Ir_c + alphaF * delta_If_c;
         % derivative f(cNode) wrt Vb
```

```
        d_BJT(cNode, bNode) = d_BJT(cNode, bNode) - delta_Ir_b + alphaF * delta_If_b;
        % derivative f(cNode) wrt Ve
        d_BJT(cNode, eNode) = d_BJT(cNode, eNode) - delta_Ir_e + alphaF * delta_If_e;

        %f(bNode) =  f(bNode)  +Ir*(1-alphaR) +    If*(1-alphaF);
        d_BJT(bNode,cNode) = d_BJT(bNode, cNode) + delta_Ir_c *(1-alphaR) +
delta_If_c*(1-alphaF);
        d_BJT(bNode,bNode) = d_BJT(bNode, bNode) + delta_Ir_b *(1-alphaR) +
delta_If_b*(1-alphaF);
        d_BJT(bNode,eNode) = d_BJT(bNode, eNode) + delta_Ir_e *(1-alphaR) +
delta_If_e*(1-alphaF);

        %f(eNode) =  f(eNode)  +Ir*alphaR     -    If;
        d_BJT(eNode,cNode) = d_BJT(eNode, cNode) + delta_Ir_c *(alphaR) -    delta_If_c;
        d_BJT(eNode,bNode) = d_BJT(eNode, bNode) + delta_Ir_b *(alphaR) -    delta_If_b;
        d_BJT(eNode,eNode) = d_BJT(eNode, eNode) + delta_Ir_e *(alphaR) -    delta_If_e;

    elseif (cNode~=0)&&(bNode==0)&&(eNode~=0) % Base is Grounded

        Vbe = -1*X(eNode);
        Vbc = -1*X(cNode);


        delta_If_c = 0;
        %delta_If_b = Is/Vt*exp(Vbe/Vt);
        delta_If_e = -Is/Vt*exp(Vbe/Vt);

        delta_Ir_c = -Is/Vt*exp(Vbc/Vt);
        %delta_Ir_b = Is/Vt*exp(Vbc/Vt);
        delta_Ir_e = 0;


        d_BJT(cNode, cNode) = d_BJT(cNode, cNode) - delta_Ir_c + alphaF * delta_If_c;
        d_BJT(cNode, eNode) = d_BJT(cNode, eNode) - delta_Ir_e + alphaF * delta_If_e;

        d_BJT(eNode,cNode) = d_BJT(eNode, cNode) + delta_Ir_c *(alphaR) -    delta_If_c;
        d_BJT(eNode,eNode) = d_BJT(eNode, eNode) + delta_Ir_e *(alphaR) -    delta_If_e;

    elseif (cNode~=0)&&(bNode~=0)&&(eNode==0) % Emitter is Grounded

        % fill this up

        Vbe = X(bNode);
        Vbc = X(bNode) -X(cNode);


        delta_If_c = 0;
        delta_If_b = Is/Vt*exp(Vbe/Vt);
        %delta_If_e = -Is/Vt*exp(Vbe/Vt);

        delta_Ir_c = -Is/Vt*exp(Vbc/Vt);
        delta_Ir_b = Is/Vt*exp(Vbc/Vt);
        %delta_Ir_e = 0;


        d_BJT(cNode, cNode) = d_BJT(cNode, cNode) - delta_Ir_c + alphaF * delta_If_c;
        d_BJT(cNode, bNode) = d_BJT(cNode, bNode) - delta_Ir_b + alphaF * delta_If_b;

        d_BJT(bNode,cNode) = d_BJT(bNode, cNode) + delta_Ir_c *(1-alphaR) +
delta_If_c*(1-alphaF);
        d_BJT(bNode,bNode) = d_BJT(bNode, bNode) + delta_Ir_b *(1-alphaR) +
delta_If_b*(1-alphaF);
```

```
    elseif (cNode==0)&&(bNode~=0)&&(eNode~=0) % Collector is Grounded

        Vbe = X(bNode) -X(eNode);
        Vbc = X(bNode);

        % diode currents

        %delta_If_c = 0;
        delta_If_b = Is/Vt*exp(Vbe/Vt);
        delta_If_e = -Is/Vt*exp(Vbe/Vt);

        %delta_Ir_c = -Is/Vt*exp(Vbc/Vt);
        delta_Ir_b = Is/Vt*exp(Vbc/Vt);
        delta_Ir_e = 0;

        d_BJT(bNode,bNode) = d_BJT(bNode, bNode) + delta_Ir_b *(1-alphaR) +
delta_If_b*(1-alphaF);
        d_BJT(bNode,eNode) = d_BJT(bNode, eNode) + delta_Ir_e *(1-alphaR) +
delta_If_e*(1-alphaF);

        d_BJT(eNode,bNode) = d_BJT(eNode, bNode) + delta_Ir_b *(alphaR) -    delta_If_b;
        d_BJT(eNode,eNode) = d_BJT(eNode, eNode) + delta_Ir_e *(alphaR) -    delta_If_e;


     elseif(cNode~=0)&&(bNode==0)&&(eNode==0) % Base and  Emitter are grounded

         %Vbe = 0;
         Vbc = -X(cNode);
         % diode currents

         delta_If_c = 0
         delta_Ir_c = -1*Is/Vt*exp(Vbc/Vt)

         d_BJT(cNode, cNode) =  d_BJT(cNode, cNode)  -    delta_Ir_c        +
alphaF*delta_If_c ;

     end
end %end Forloop for BJTs

end
J = G + d_BJT + d_diode;
end
```

4. *nlACresponse.m*

1. *Run the BJT_CE.m and plot the gain at node 7 for the frequencies between 0 to 10^6Hz obtained and include it in the assignment.*




```
function r = nonlinear_fsolve(Xdc, fpoints ,out)
%  nonlinear_fsolve(fpoints ,out)
%  Obtain frequency domain response
% global variables G C b bac
% Inputs: fpoints is a vector containing the fequency points at which
%         to compute the response in Hz
%         out is the output node
% Outputs: r is a vector containing the value of
%            of the response at the points fpoint

global  G C b bac

% book page 148
sz = size(fpoints, 2);
r = zeros(1, sz);

for i = 1:sz
    % (G + jwC) * X = b
    %Xdc = (G + 1j * 2 * pi * fpoints(i) * C) \ (b + bac);

    %Xs*(G+Jdc) + CXs = bs

    J = nlJacobian(Xdc);
    Xac = (G + J + 1j * 2 * pi * fpoints(i) * C) \ (bac);

    r(1, i) = Xac(out);
end

end
```