ECSE 597 Assignment 2 Report

TestDCsolve.m

Figure 1 shows the implementation of dcsolve.m, and results of TestDCsolve.m are shown in Figure 2.

```
>> TestDCsolve

Xdc =

    1.0000
    0.2407
   -0.0048


dX =

  Columns 1 through 13

    1.0000    0.0260    0.0260    0.0260    0.0260    0.0259    0.0258    0.0253    0.0238    0.0199    0.0122    0.0036    0.0002

  Column 14

    0.0000
```

Figure 1 - TestDCsolve.m Results

```
function [Xdc dX] = dcsolve(Xguess,maxerr)
% Compute dc solution using newtwon iteration
% input: Xguess is the initial guess for the unknown vector.
%        It should be the correct size of the unknown vector.
%        maxerr is the maximum allowed error. Set your code to exit the
%        newton iteration once the norm of DeltaX is less than maxerr
% Output: Xdc is the correction solution
%         dX is a vector containing the 2 norm of DeltaX used in the
%         newton Iteration. the size of dX should be the same as the number
%         of Newton-Raphson iterations. See the help on the function 'norm'
%         in matlab.
global elementList

[Bdc, Bac] = makeBvector;
G = makeGmatrix;
iteration = 1;
delta_x = zeros(size(Xguess));
Xdc = Xguess;
while (1)
    F = makeFvect(Xdc);
    J = make_nlJacobian(Xdc);
    phi = G * Xdc + F - Bdc;
    dphi = G + J;

    delta_x = ((-1) * dphi)\phi;

    dX(iteration) = norm(delta_x);
    Xdc = Xdc + delta_x;
    if norm(delta_x) < maxerr
        break;
    end
    iteration = iteration + 1;
end


```

Figure 2 - dcsolve Implementation

BJT_CB.m

The implementation of dcsolvealpha.m and dcsolvecont.m are shown in Figure 5 and 6. The results of BJT_CB.m are shown in Figure 3, 4 and 5. The dcsolve.m implemented previously does not guarantee convergence, as shown in Figure 3. In contrast, the dcsolvecont.m converges, and the results are shown in Figure 4 and 5. The implementations of dcsolvealpha.m and dcsolvecont.m are shown in Figure 6 and 7.

```
In BJT_CB (line 30)

Warning: Matrix is singular to working precision.
> In dcsolve (line 26)
In BJT_CB (line 30)

Warning: Matrix is singular to working precision.
> In dcsolve (line 26)
In BJT_CB (line 30)

Warning: Matrix is singular to working precision.
> In dcsolve (line 26)
In BJT_CB (line 30)

Warning: Matrix is singular to working precision.
> In dcsolve (line 26)
In BJT_CB (line 30)
```

Figure 3 - BJT_CB.m DCSolve Result

```
>> BJT_CB

Xdcsolve_cont =

   10.0000
    5.3197
    5.2964
    6.0000
   -0.0010
   -0.0006
```

Figure 4 - BJT_CB.m DCsolve_cont Result (5V)

```
>> BJT_CB

Xdcsolve_cont =

   10.0000
    5.3341
    3.3092
    4.0000
   -0.0010
   -0.0000
```

Figure 5 - BJT_CB.m DCsolve_cont Result (4V)

```matlab
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 elementList
[Bdc, Bac] = makeBvector;
G = makeGmatrix;

delta_x = zeros(size(Xguess));
Xdc = Xguess;
while (1)
    F = makeFvect(Xdc);
    J = make_nlJacobian(Xdc);
    phi = G * Xdc + F - alpha * Bdc;
    dphi = G + J;
    delta_x = ((-1) * dphi)\phi;
    Xdc = Xdc + delta_x;
    if norm(full(delta_x)) < maxerr
        break;
    end
end

```

Figure 6 - dcsolvealpha.m Implementation

```matlab
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 elementList

% size of MNA matrix
alpha = linspace(0,1,n_steps);
n = elementList.n;
Xdc = zeros(n,1);
for step = 1:n_steps
    Xdc = dcsolvealpha(Xdc,alpha(step),maxerr);
end
```

Figure 7 - dcsolvecont.m Implementation

BJT_CE2.m

The results of DC analysis of BJT_CE2.m are shown in Figure 8 and 9. The dcsolve.m does not converge whereas the dcsolvecont.m converges. The implementation of nonlinear_fsolve.m is shown in Figure 10, and the results is shown in Figure 11.

```
In BJT_CE2 (line 57)

Warning: Matrix is singular to working precision.
> In dcsolve (line 26)
In BJT_CE2 (line 57)

Warning: Matrix is singular to working precision.
> In dcsolve (line 26)
In BJT_CE2 (line 57)

Warning: Matrix is singular to working precision.
> In dcsolve (line 26)
In BJT_CE2 (line 57)

Warning: Matrix is singular to working precision.
> In dcsolve (line 26)
In BJT_CE2 (line 57)

Warning: Matrix is singular to working precision.
> In dcsolve (line 26)
In BJT_CE2 (line 57)
```

Figure 8 - BJT_CE2 dcsolve Result

```
>> BJT_CE2

Xdc =

         0
         0
    1.6825
   12.0000
    6.2362
    1.0674
         0
   -0.0053
         0
```

Figure 9 - BJT-CE2 decolvecont Result

```
function r = nonlinear_fsolve(fpoints ,out)

% This function  Obtains frequency response of the nonlinear function
% global variables G C b bac
% Inputs: 1. fpoints is a vector containing the fequency points\
%         2. out is the node name of the output node.

% Outputs: r is a vector containing the value of
%             of the response at the points fpoint

global  elementList

outNodeNumber = getNodeNumber(out);

n = elementList.n;
guess = zeros(n,1);

G = makeGmatrix;
C = makeCmatrix;
[Bdc, Bac] = makeBvector;
Xdc= dcsolvecont(100,1e-6);
Jdc = make_nlJacobian(dcsolvecont(100,1e-6));
[~,f_size] = size(fpoints);
f_responses = zeros(1,f_size);
B_fr = zeros(n,1);

% for m = 1:n
%     if (Bac(m,1) ~= 0)
%         B_fr(m,1) = 1;
%     end
% end

for i = 1:f_size
    A = (G+2*pi*1i*fpoints(1,i)*C+Jdc);
    tmp = A\Bac;
    f_responses(1,i) = tmp(outNodeNumber,1);
end

r = f_responses;
```

Figure 10 - nonlinear_fsolve Implementation

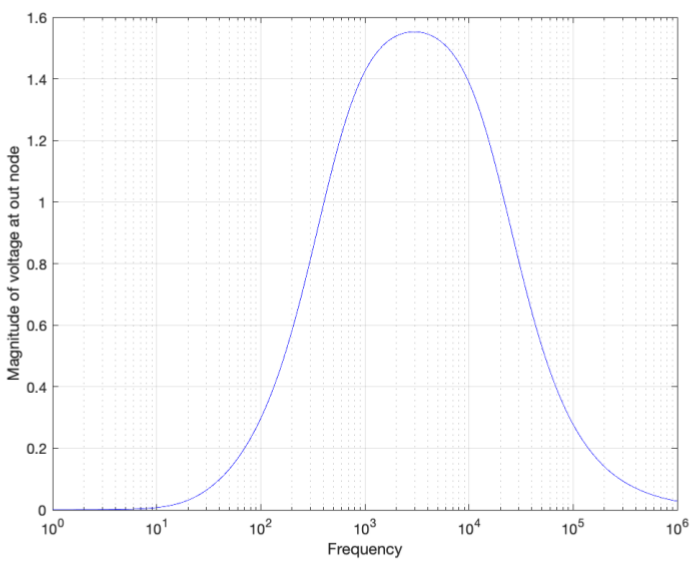

Figure 11 - Frequency Analysis of BJT_CE2