ECSE 597: Circuit Simulations and Modeling

Assignment 2, September 26, 2019

Wenjie Wei, 260685967

1 Circuit Diagram

Figure 1 below shows the circuit described in the file `Circuit_diodeckt1.m`.

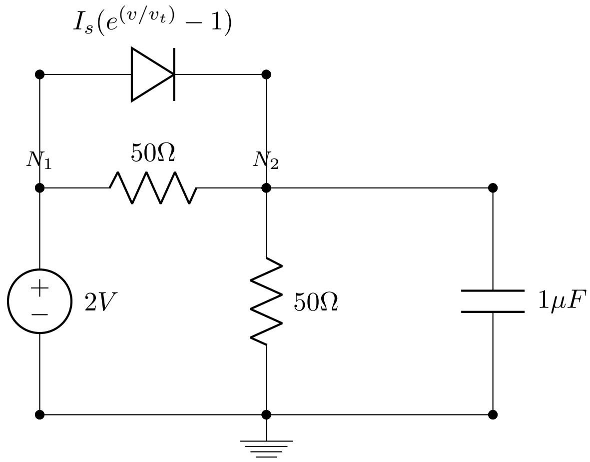

Figure 1: Circuit described in `Circuit_diodeckt1.m`

Where $I_s = 1 \times 10^{-15}A$, $V_t = 26 \times 10^{-3}V$.

2 Simulation Results

The result computed after running the test bench is:

$$V_1 = 2V, \quad V_2 = 1.2245V$$

and the current flowing through the voltage source is calculated to be:

$$I_E = -0.0245A$$

defining that the direction of the current is from Node 1 to ground.

3 Plotting Result

Figure 2 below shows the plotting result of the norm of Δx after each iteration.

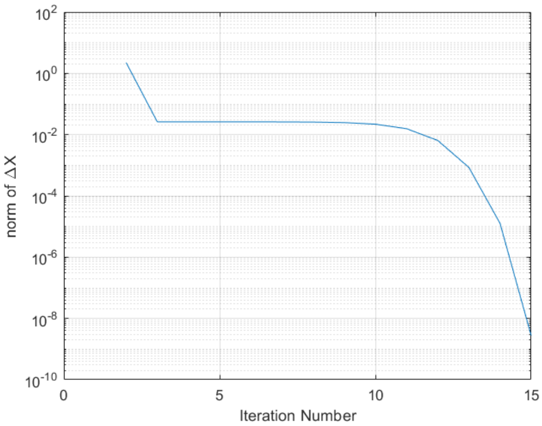

Figure 2: Plotting Results of the Norm of Δx After Newton-Raphson Iterations

A Code Listings

*Listing 1: MATLAB Function Calculating the Jacobian (**nlJacobian.py**).*

```
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.

    % Diode curve: I = Is(exp(V/VT) - 1)
    global G DIODE_LIST

    startNode = DIODE_LIST.node1;
    endNode = DIODE_LIST.node2;
    Is = DIODE_LIST.Is;
    Vt = DIODE_LIST.Vt;

    v1 = X(startNode);
    v2 = X(endNode);

    F = zeros(size(G, 1), size(G, 2));

    F(startNode, startNode) = (Is / Vt) * exp((v1 - v2) / Vt);
    F(startNode, endNode) = -(Is / Vt) * exp((v1 - v2) / Vt);
    F(endNode, startNode) = -(Is / Vt) * exp((v1 - v2) / Vt);
    F(endNode, endNode) = (Is / Vt) * exp((v1 - v2) / Vt);

    J = G + F;
end
```

*Listing 2: MATLAB Function Performing DC Simulation (**dcsolve.py**).*

```
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 G C b DIODE_LIST
```

```
    % Vector contains the result for each iteration. Overwritten for each iteration.
    Xdc = zeros(size(G, 2), 1);

    % Delta X vector for each iteration. Size will be increased for each iteration.
    dX = zeros(size(G, 2), 1);

    % temporary vector containing the results of Is(exp(Vs/Vt) - 1)
    f = zeros(size(G, 2), 1);

    diodeStartNode = DIODE_LIST.node1;
    diodeEndNode = DIODE_LIST.node2;
    Is = DIODE_LIST.Is;
    Vt = DIODE_LIST.Vt;

    converged = false;
    iteration = 0;

    while ~converged
        iteration = iteration + 1;

        % Calculate the Phi vector for each iteration.
        if diodeStartNode ~= 0 && diodeEndNode ~= 0
            % If the diode is not connected to the ground.
            f(diodeStartNode) = Is * (exp((Xguess(diodeStartNode) - Xguess(diodeEndNode)) / Vt) - 1);
            f(diodeEndNode) = -Is * (exp((Xguess(diodeStartNode) - Xguess(diodeEndNode)) / Vt) - 1);
        else
            % If the cathod of the diode is connected to the ground
            f(diodeStartNode) = Is * (exp(Xguess(diodeStartNode) / Vt) - 1);
        end

        % Phi vector containing the temporary result to obtain dX.
        Phi = G * Xguess + f - b;

        % Calculate the Jacobian.
        J = nlJacobian(Xdc);

        dX = [dX (-inv(J) * Phi)];
        Xdc = Xdc + dX(:, iteration + 1);
        Xguess = Xguess + dX(:, iteration + 1);

        % Determine the converge condition.
        % Check if every entry of current iteration meets the threashold.
        % If there is at least one dX > maxerr,
        % Continue the iteration.
        if abs(norm(dX(:, iteration + 1), 2)) < maxerr
            converged = true;
        end
    end

    % Convert dX to 1-by-n matrix containing only the norm of every
    % iteration in order to make the plot.
    tempdx = dX;
    dX = zeros(iteration + 1);
    for i = 1:iteration + 1
        dX(i) = norm(tempdx(:, i), 2);
    end
end
```