CRES Assignment 5

Submitted by: Hutomo Saleh

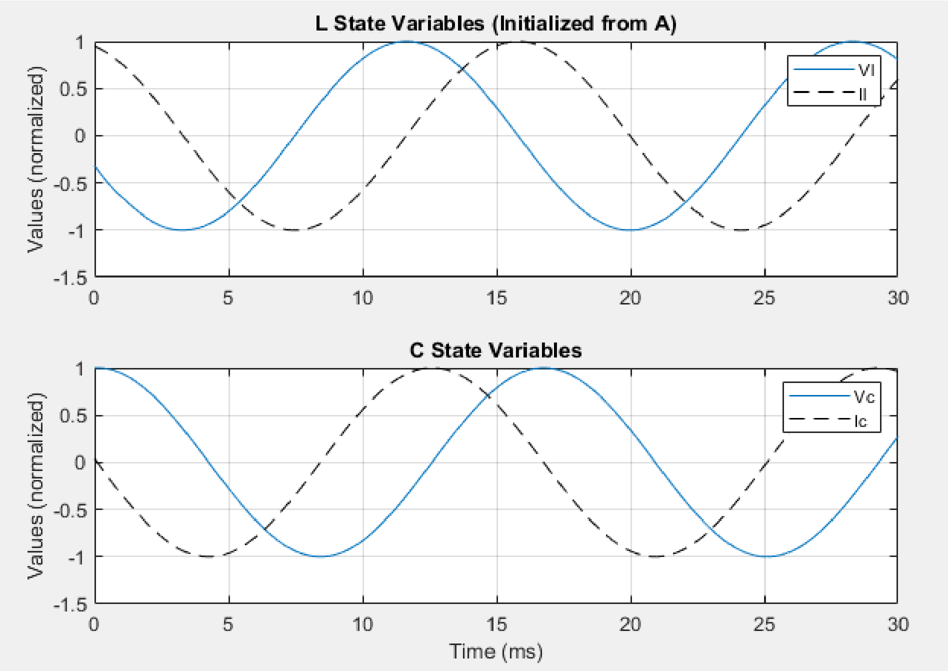

Figure 1: Plot using state variables from A)

C) Discussion:

As shown in Figure 1. Due to the steady state initialized from A), there is no transient seen at the beginning of the curve. In comparison with Figure 2, C) starts with Vc = 0 and iL = 0, which causes a transient in the first few ms, due to the nature of not being able to change instantaneously.

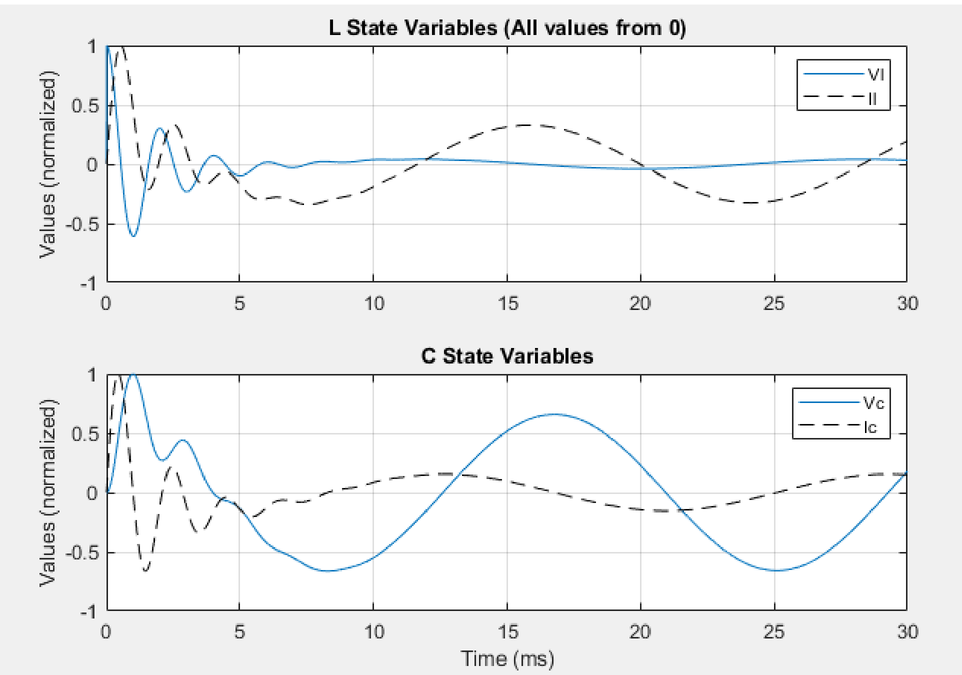

Figure 2: Plot using state variables of 0

Saleh, Hutomo
461980

Matlab Code

```
clc;
close all;
clear all;

%% Discussion on B) and C)
% Due to the steady state initialized from A), there is no transient
% seen at the beginning of the curve. In comparison, C) starts with
% Vc = 0 and iL = 0, which causes a transient in the first few ms,
% due to the nature of not being able to change instantaneously.

%% Variable Initialization
%{
    n: Amount
    val: Value
    mag: Magnitude
    node1 / node2: 1st (start) / 2nd (end) Node
%}

normalized = true;
node.n = 4;
f = 60;

G.val = [1 0.01];
G.node1 = [1 2];
G.node2 = [3 4];
G.n = 2;

L.val = 10e-3;
L.node1 = 1;
L.node2 = 2;
L.n = 1;

C.val = 10e-6;
C.node1 = 2;
C.node2 = 4;
C.n = 1;

V.mag = 230e3;
V.phase = 0;
V.node1 = 3;
V.node2 = 4;
V.n = 1;

I.mag = [];
I.phase = [];
I.node1 = [];
I.node2 = [];
I.n = 0;

%% Calculation - A

% Setup admittance matrix
Yorg = zeros(node.n, node.n); % Preallocate matrix size
G.Y = G.val(1:G.n);
Yorg = updateYMatrix(Yorg, G);
L.Y = 1 / (1j * 2 * pi * f * L.val(1:L.n));
Yorg = updateYMatrix(Yorg, L);
C.Y = 1j * 2 * pi * f * C.val(1:C.n);
Yorg = updateYMatrix(Yorg, C);

V.val = zeros(V.n + 1, 1); % Preallocate Voltage matrix
node.e = zeros(V.n + 1, 1); % Make excitation nodes
```

```matlab

% Voltage Source
for i = 1:V.n
    V.val(i) = V.mag(i) * exp(1j * V.phase(i)); % Get voltage value
    node.e(i) = V.node1(i); % Get excitation nodes
end
node.e(V.n + 1) = node.n; % Insert ground node

node.d = zeros(node.n - length(node.e), 1); % Make dependent nodes
k = 1;
for i = 1:node.n % Loop through nodes
    if(isempty(find(node.e == i, 1))) % If dependent nodes, insert
        node.d(k) = i;
        k = k + 1;
    end
end

% Current source
I.mag = zeros(node.n, 1);
for i = 1:I.n
    I.mag(I.node1(i)) = I.mag(i) * exp(1j * I.phase(i));
    I.mag(I.node2(i)) = -I.mag(I.node1(i));
end

% Make dependent variables (4.12)
Y = Yorg(node.d, node.d);
Yde = Yorg(node.d, node.e);
I.d = I.mag(node.d);
I.mag = I.d - Yde * V.val;

% Solve dependent voltage (4.16)
V.d = Y \ I.mag; % Matrix left division from I.mag * V.d = Ydep

% Calculate branch currents
V1 = V.d(1);
V2 = V.d(2);
I.L = (V1 - V2) / (1j * 2 * pi * f * L.val);
I.C = V2 * 1j * 2 * pi * f * C.val;

% Results
V1 = real(V1);
V2 = real(V2);
I.L = real(I.L);
I.C = real(I.C);

disp("V1: " + V1*1e-3 + " kV");
disp("V2: " + V2*1e-3 + " kV");
disp("iL: " + I.L + " A");
disp("iC: " + I.C + " A");

%% Calculation - B
% Electromagnetic Transients Simulation

for loop = 1:2 % For B) & C)
    %% Initialization
    t.step = 1e-5; % time step
    t.span = 0:t.step:0.03;
    t.len = length(t.span);

    trans.n = (L.n + C.n) * 2; % For V and I of each trans. elements
    trans.val = zeros(t.len, trans.n); % Value of transient elements (V, I)
    V.nodeval = zeros(t.len, node.n); % Voltage of each nodes
```

```
    if (loop==1)
        % Initialize with values from A)]
        window_title = "Plot for B)";
        plot_title = " (Initialized from A)";
        trans.val(1, :) = [V1-V2 I.L V2 I.C]; % Vl, Il, Vc, Ic
        V.nodeval(1, :) = [V1 V2 V.mag 0];
    elseif (loop==2)
        % Initialization for C)
        window_title = "Plot for C)";
        plot_title = " (All values from 0)";
    end

    % Setup admittance matrix (Using trapezoidal method)
    Yorg = zeros(node.n, node.n);
    G.Y = G.val(1:G.n); % No transient effect
    Yorg = updateYMatrix(Yorg, G);
    L.Y = t.step / (2 * L.val(1:L.n)); % Derived (4.23) but w/ inductor
    Yorg = updateYMatrix(Yorg, L);
    C.Y = 2 * C.val(1:C.n) / t.step; % From (4.26)
    Yorg = updateYMatrix(Yorg, C);


    for k = 2:t.len % 1st index already defined
        %% Update voltage values
        V.val = zeros(V.n+1, 1);

        for i = 1:V.n
            V.val(i) = V.mag(i) * cos(2 * pi * f * k * t.step + V.phase(i));
        end

        %% Update current values (according to 4.22-4.28)
        I.mag = zeros(node.n, 1);
        for i = 1:I.n
            I.mag(I.node1(i)) = I.mag(i) * ...
                                cos(2 * pi * f * k * t.step + I.phase(i));
            I.mag(I.node2(i)) = -I.mag(I.node1(i));
        end
        for i = 1:2:L.n % Increments of two due to V and I
            eta = L.Y * trans.val(k-1, i) + trans.val(k-1, i+1);
            I.mag(L.node1(i)) = I.mag(L.node1(i)) - eta;
            I.mag(L.node2(i)) = I.mag(L.node2(i)) + eta;
        end
        for i = 1:2:C.n
            C_index = 2 * L.n + i;
            eta = C.Y * trans.val(k-1, C_index) + trans.val(k-1, C_index+1);
            I.mag(C.node1(i)) = I.mag(C.node1(i)) + eta;
            I.mag(C.node2(i)) = I.mag(C.node2(i)) - eta;
        end

        %% Dependant nodes taken from A)
        Y = Yorg(node.d, node.d);
        I.d = I.mag(node.d);
        Yde = Yorg(node.d, node.e);
        I.mag = I.d - Yde * V.val;

        V.d = Y \ I.mag; % Solve dependent node voltage

        %% Update node voltages
        n = length(node.d); % Insert dependent values
        for i=1:n
            V.nodeval(k, node.d(i)) = V.d(i);
```

```
        end
        V.nodeval(k, 3) = V.val(1);
        V.nodeval(k, 4) = 0.0;

        %% Calculate transient values
        for i = 1:2:L.n
            eta = L.Y * trans.val(k-1, i) + trans.val(k-1, i+1); % (4.27)
            V_L = V.nodeval(k, L.node1(i)) - V.nodeval(k, L.node2(i));
            I_L = L.Y * V_L + eta;

            trans.val(k, i) = V_L;
            trans.val(k, i+1) = I_L;
        end
        for i = 1:2:C.n
            C_index = 2 * L.n + i;
            eta = C.Y * trans.val(k-1, C_index)+ trans.val(k-1, C_index+1);
            V_C = V.nodeval(k, C.node1(i)) - V.nodeval(k, C.node2(i));
            I_C = C.Y * V_C - eta;

            trans.val(k, C_index) = V_C;
            trans.val(k, C_index+1) = I_C;
        end
    end

    %% Normalize values
    y_label = 'Voltage [V] and Current [A]';
    if (normalized == true)
        y_label = 'Values (normalized)';
        for i = 1:node.n-1
            V.nodeval(:, i) = V.nodeval(:, i) / max(V.nodeval(:, i));
        end
        for i = 1:trans.n
            trans.val(:, i) = trans.val(:, i) / max(trans.val(:, i));
        end
    end

    %% Plot
    figure('name', window_title);
    subplot(211);
    x_plot = t.span*1e3;
    plot(x_plot, trans.val(:, 1), x_plot, trans.val(:, 2), 'k--');
    grid on;
    legend('Vl', 'Il');
    title('L State Variables' + plot_title);
    ylabel(y_label);

    subplot(212);
    plot(x_plot, trans.val(:, 3), x_plot, trans.val(:, 4), 'k--');
    grid on;
    legend('Vc', 'Ic');
    title('C State Variables');
    xlabel('Time (ms)');
    ylabel(y_label);

end

%% Functions
function Y_out = updateYMatrix(Y_in, Y)
    for i = 1:Y.n
        % Diagonal elements(same index): positive. Else negative.
        Y_in(Y.node1(i), Y.node1(i)) = Y_in(Y.node1(i), Y.node1(i)) + Y.Y(i);
        Y_in(Y.node2(i), Y.node2(i)) = Y_in(Y.node2(i), Y.node2(i)) + Y.Y(i);
        Y_in(Y.node1(i), Y.node2(i)) = Y_in(Y.node1(i), Y.node2(i)) - Y.Y(i);
        Y_in(Y.node2(i), Y.node1(i)) = Y_in(Y.node2(i), Y.node1(i)) - Y.Y(i);
    end
    Y_out = Y_in;
end
```