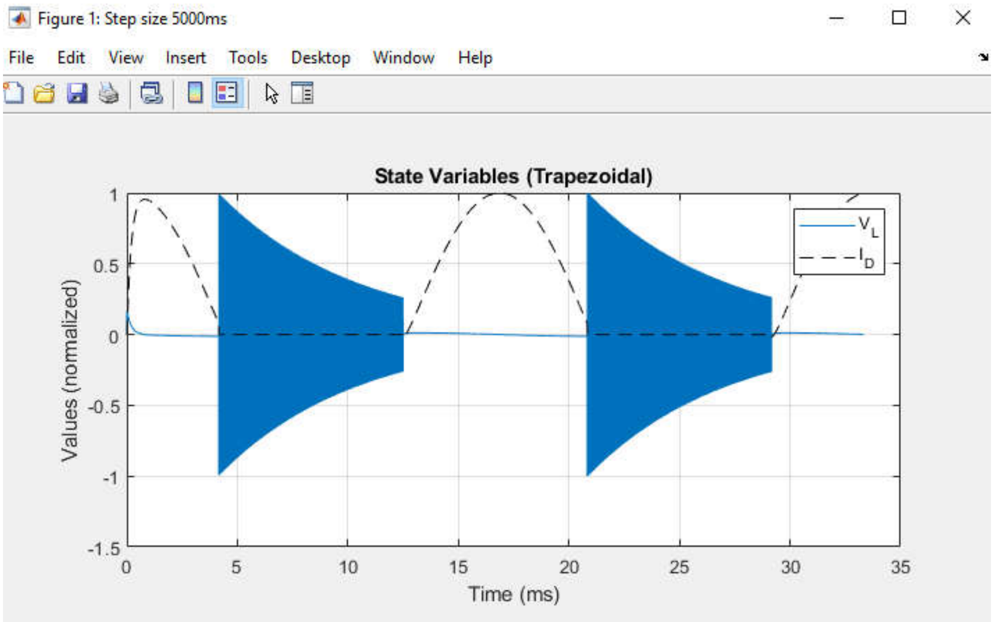
Saleh, Hutomo
461980

CRES Assignment 6

Submitted by: Hutomo Saleh

The simulation using trapezoidal method returns a heavily distorted result. The oscillation during the blocking of the diode is due to the history variable of the trapezoidal method.



The derivation of the backward-euler method is as follows:

$$\frac{V(k) - V(k-1)}{\tau} = \lambda\, i(k)$$

$$i(k) = \underbrace{\frac{1}{\lambda\tau}}_{G} \left(V(k) - V(k-1)\right)$$

$$= \frac{1}{\lambda\tau} V(k) - \frac{1}{\lambda\tau} V(k-1)$$

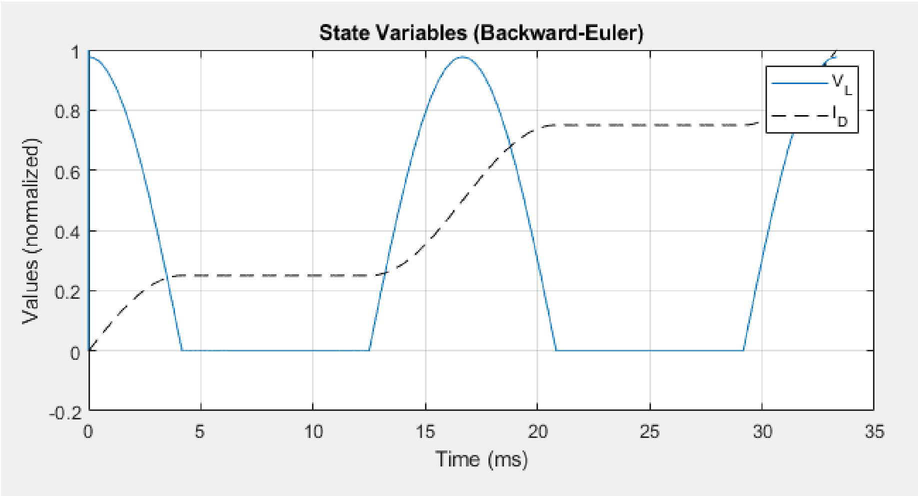
$$= G \cdot V(k) - \eta \qquad [1.0]$$

$$\Rightarrow G = \frac{1}{\lambda\tau} \quad \text{and} \quad \eta = G \cdot V(k-1)$$

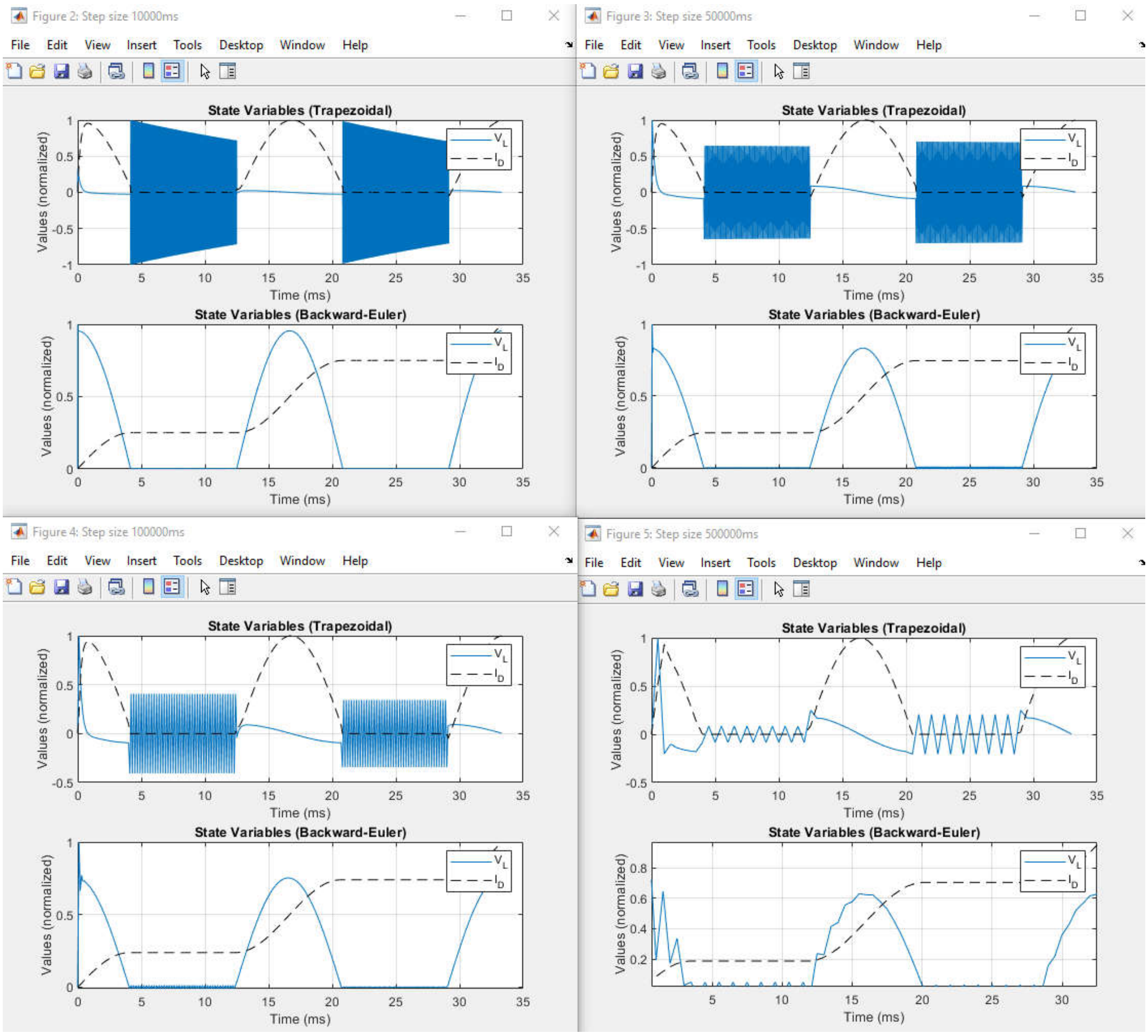
$$\text{For } L \Rightarrow G_L = \frac{\tau}{L}$$

$$C \Rightarrow G_C = \frac{C}{\tau}$$

The backward-Euler method shows that the voltage across the inductor does not oscillate, unlike the result using the trapezoidal method.



Generally, the accuracy of the approximation will increase with smaller time step sizes. This is why with successively larger time step size, the oscillation as well as distortions will be seen more clearly.



For C), the chosen step sizes are 5µs, 10µs, 50µs, 0.1ms and 0.5ms. It is shown in the figures that with increasing step size, the oscillation or distortion in the trapezoidal method becomes more spaced. At the largest step size of 0.5ms it is even shown that using backward euler-method results in oscillation, this is probably due to the step size nearing the

MATLAB Code

The whole code will be shown at the end of this document. This part will only show the important changes applied to the existing code from assignment 5.

The values of the diode admittance is separated into two parts for ON and OFF.

```

D.val.off = 1e-6;
D.val.on = 1e3;
D.node1 = 2;
D.node2 = 3;
D.n = 1;

```

The value of the diode will alternate depending on the value of voltage source. As shown below:

```
        for i = 1:V.n
            V.val(i) = V.mag(i) * cos(2 * pi * f * k * t.step + V.phase(i));
            %% Change value of Diode depending on Voltage
            if V.val(i) < 0
                % disp("Voltage is minus")
                D.Y = D.val.off(1:D.n);
            else
                % disp("Voltage is positive")
                D.Y = D.val.on(1:D.n);
            end
            node.e(i) = V.node1(i); % Get excitation nodes
        end
```

When applying the backward-Euler method, the admittance calculation of inductor must be changed according to the derivation showed in [1.0].

```
        L.Y = t.step / (loop * L.val(1:L.n)); % Derived (4.23) but w/ inductor
        C.Y = loop * C.val(1:C.n) / t.step; % From (4.26)
```

```
        %% 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);
            if loop == 1 % backward-euler
                eta = L.Y * trans.val(k-1, i);
            end
            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);
            if loop == 1 % backward-euler
               eta = C.Y * trans.val(k-1, C_index);
            end
            I.mag(C.node1(i)) = I.mag(C.node1(i)) + eta;
            I.mag(C.node2(i)) = I.mag(C.node2(i)) - eta;
        end
```

Saleh, Hutomo
461980

Full MATLAB Code

```
clc;
close all;
clear all;

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

D.val.off = 1e-6;
D.val.on = 1e3;
D.node1 = 2;
D.node2 = 3;
D.n = 1;

G.val = 0.2;
G.node1 = 1;
G.node2 = 4;
G.n = 1;

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

C.val = [];
C.node1 = [];
C.node2 = [];
C.n = 0;

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;

normalized = true;
node.n = D.n + G.n + L.n + C.n + 1;
f = 60;

%% Calculation - A & B
% Diode Model Simulation using Trapezoidal & Backward-Euler Method
step_sizes = [5e-6 1e-5 5e-5 1e-4 5e-4];
% step_sizes = [5e-6];


```

```
for step = 1:length(step_sizes)
    %% Initialization
    t.step = step_sizes(step); % time step
    t.span = 0:t.step:(2/60);
    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
    plot_title = ["(Backward-Euler)" "(Trapezoidal)"];

    for loop = 1:2

        %{
            Loop 1: Backward-Euler
            Loop 2: Trapezoidal Method
        %}

        % Setup admittance values
        G.Y = G.val(1:G.n); % No transient effect
        D.Y = D.val.on(1:D.n); % No transient effect
        L.Y = t.step / (loop * L.val(1:L.n)); % Derived (4.23) but w/ inductor
        C.Y = loop * C.val(1:C.n) / t.step; % From (4.26)

        for k = 2:t.len % 1st index already defined
            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));
                %% Change value of Diode depending on Voltage
                if V.val(i) < 0
                    % disp("Voltage is minus")
                    D.Y = D.val.off(1:D.n);
                else
                    % disp("Voltage is positive")
                    D.Y = D.val.on(1:D.n);
                end
                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
            l = 1;
            for i = 1:node.n % Loop through nodes
                if(isempty(find(node.e == i, 1))) % If dependent nodes, insert
                    node.d(l) = i;
                    l = l + 1;
                end
            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);
                if loop == 1 % backward-euler
                    eta = L.Y * trans.val(k-1, i);
                end
                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;
```

Saleh, Hutomo
461980

```
            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);
                if loop == 1 % backward-euler
                    eta = C.Y * trans.val(k-1, C_index);
                end
                I.mag(C.node1(i)) = I.mag(C.node1(i)) + eta;
                I.mag(C.node2(i)) = I.mag(C.node2(i)) - eta;
            end

            % Setup admittance matrix
            Ymat = zeros(node.n, node.n);
            Ymat = updateYMatrix(Ymat, G);
            Ymat = updateYMatrix(Ymat, D);
            Ymat = updateYMatrix(Ymat, L);
            Ymat = updateYMatrix(Ymat, C);

            Y = Ymat(node.d, node.d);
            I.d = I.mag(node.d);
            Yde = Ymat(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

```

Saleh, Hutomo
461980

```matlab

        if loop == 2
            y_plot_TV = trans.val(:, 1);
            y_plot_TI = trans.val(:, 2);
        else
            y_plot_BV = trans.val(:, 1);
            y_plot_BI = trans.val(:, 2);
        end
    end

    %% Plot
    fig_title = "Step size " + sprintf("%.0f", (t.step*1e9)) + 'ms';
    figure("name", fig_title);
    subplot(211)
    x_plot = t.span*1e3;
    plot(x_plot, y_plot_TV, x_plot, y_plot_TI, 'k--');
    grid on;
    legend('V_L', 'I_D');
    title('State Variables (Trapezoidal)');
    ylabel(y_label);
    xlabel('Time (ms)');

    subplot(212)
    plot(x_plot, y_plot_BV, x_plot, y_plot_BI, 'k--');
    grid on;
    legend('V_L', 'I_D');
    title('State Variables (Backward-Euler)');
    ylabel(y_label);
    xlabel('Time (ms)');
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
```