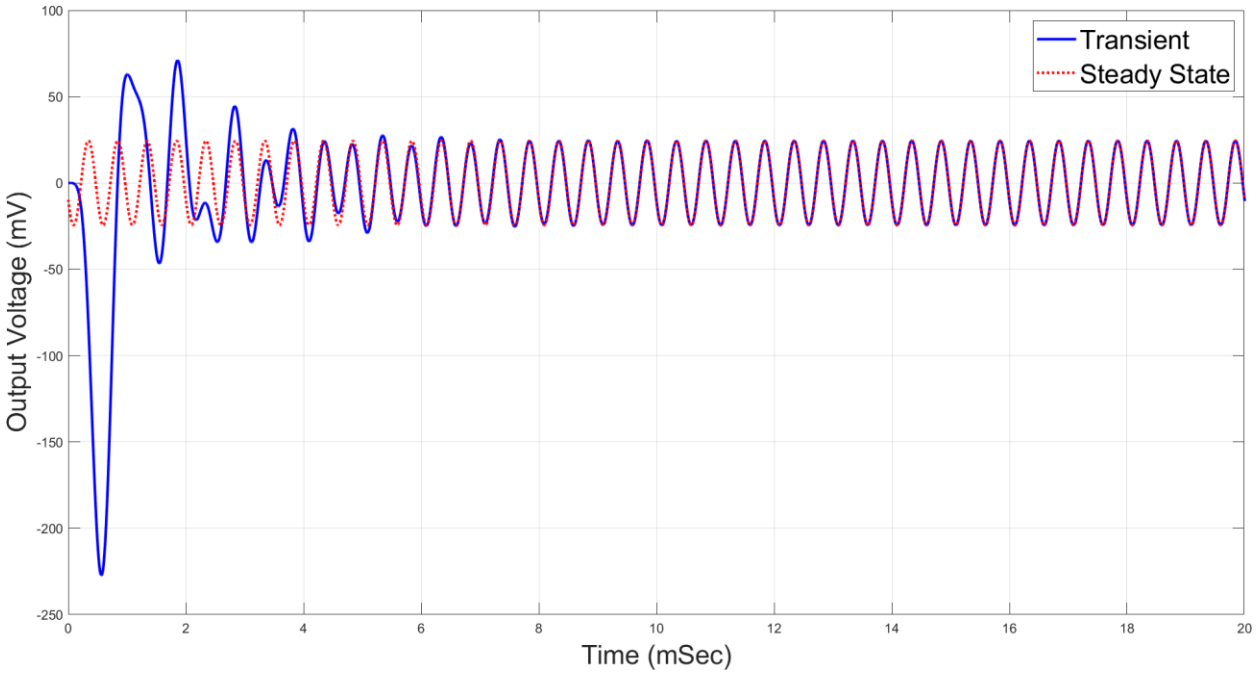
Bryan Jay
260738764

ECSE 472: Assignment 5b

1- **Backward Euler**. Please see Appendix A for the Matlab code.

In the diagram, we can observe that the transient plot begins in a state scrambled state where it was inconsistent oscillations around the steady state. Initially the amplitude of the transient is high as it fluctuates and as more time passes and the system progresses the system falls into steady state. After about 6 ms the system is in steady state and we get the predicted output voltage which oscillates approximately between -25mV to 25mV.

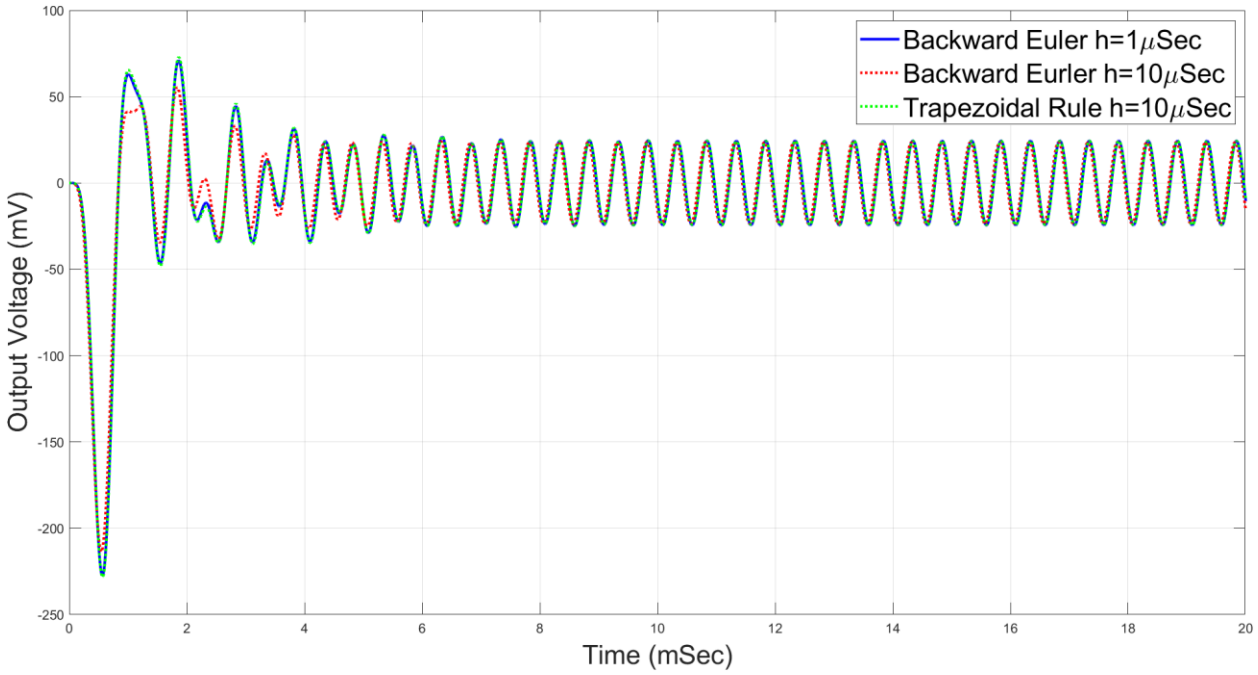
Transient Response vs Steady State plot



2- **Trapezoidal Rule**. Please refer to Appendix B for the Matlab code.

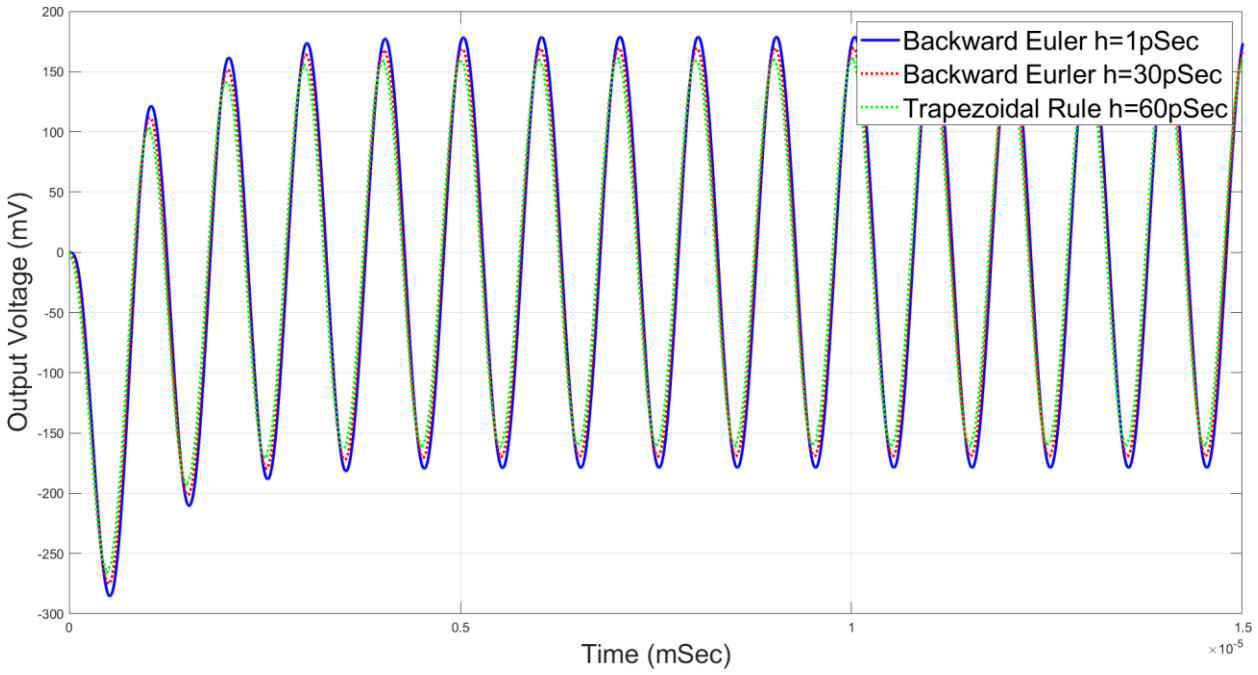
In this diagram we see a comparison between the Trapezoidal method and the Backward Euler method. We can observe that both methods reach a steady state after a certain time period. However, we can see that the trapezoidal method with an h = 10 µs follows the backward Euler with an h = 1 µs. Since the error of Euler's method depends on the step size, then the trapezoidal method can perform better since it can emulate the same plot as the backward Euler with a larger step size. This relates back to the stability condition since we can assume that at h = 10 µs we are approaching the backward Euler stability condition. Therefore, we can conclude that the trapezoidal rule has a **higher stability condition** since it can tolerate a higher step size and perform at a similar level as the backward Euler.

Trapezoidal Rule vs Backward Euler Method



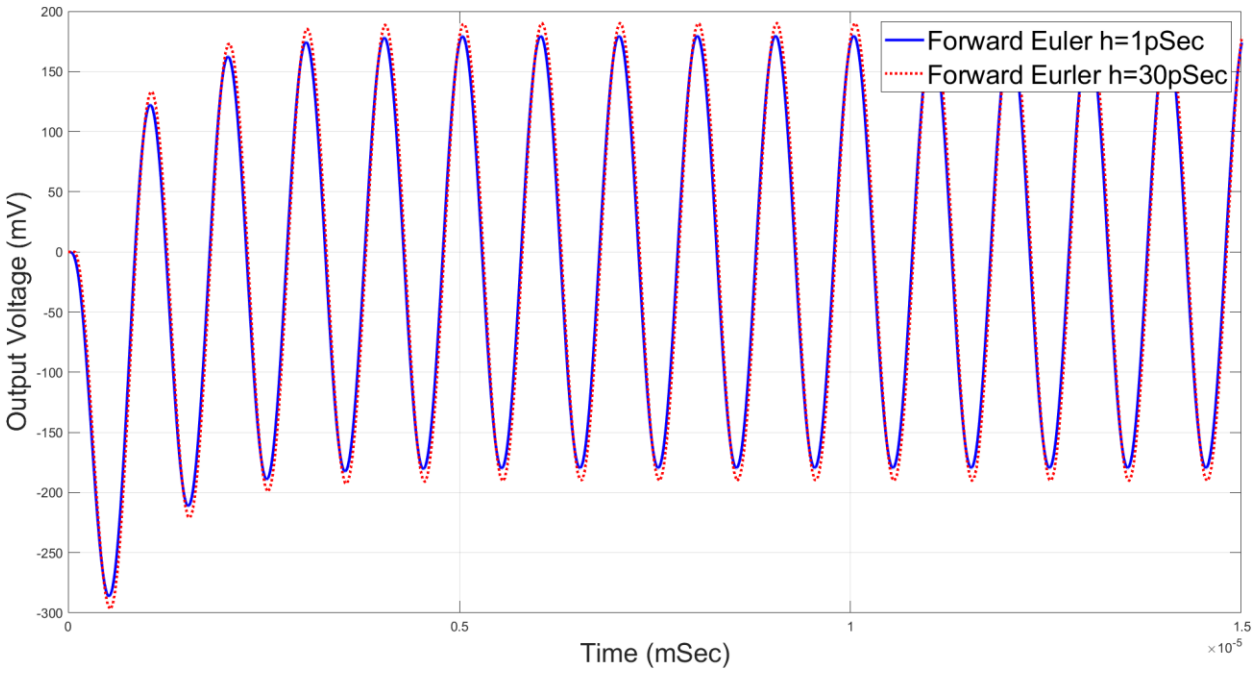
3- **Forward Euler**. Please refer to Appendix C for the Matlab code.

Trapezoidal Rule vs Backward Euler Method

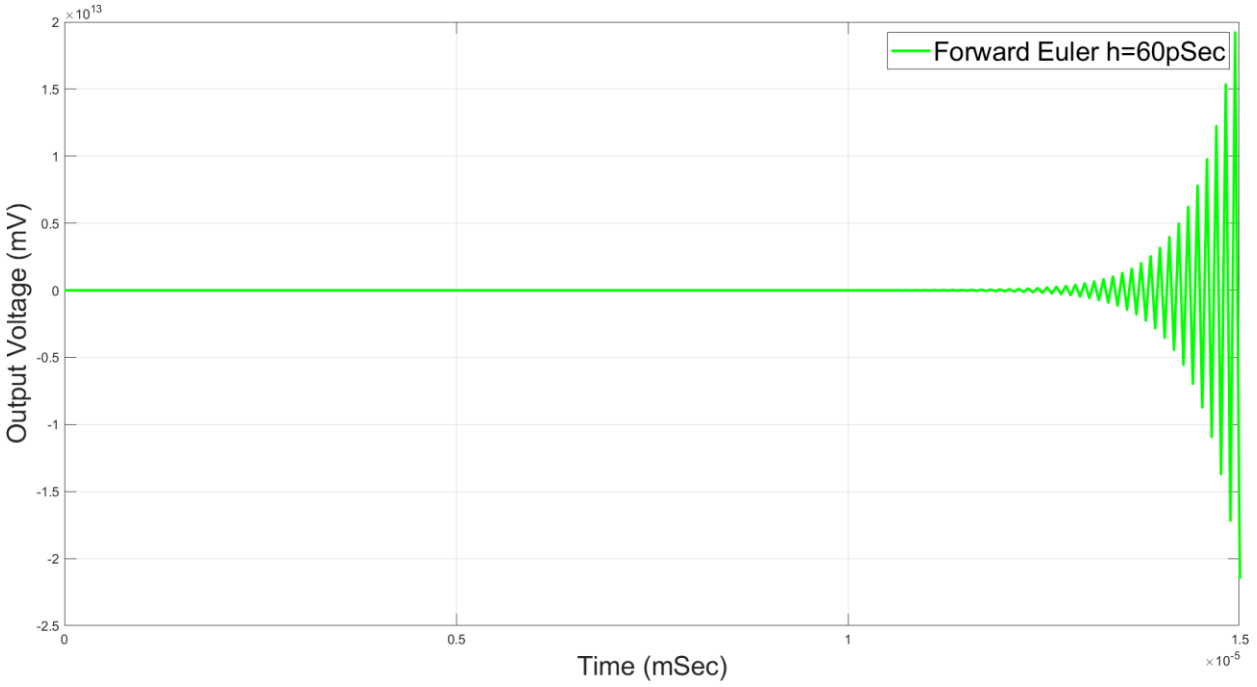


Forward Euler Method

Bryan Jay
260738764



Forward Euler Method h = 60ps



Comparing the outputted graphs, the main observation we can see from the transient system for the 3 methods is again the difference in the stability conditions. We observe that the Forward Euler and Backward Euler method has the lowest stability condition in comparison to the Trapezoidal rule. Additionally, both Euler methods have similar stability conditions as their graphs resemble quite similarly at the given step sizes. Lastly it is evident that with h = 60ps we have reached a point in the system where the transient system never reaches steady state so we can conclude that we have breached its stability condition.

Bryan Jay
260738764

Matlab Code for eigenvector calculation:

```
global C G
eigen = eig(-inv(C)*G)
```

The eigenvalues of the system correspond to the poles of the system therefore the given system has the following poles.

-3.5321e10	-2.3473e10	-1.0000e10	-0.1206e10

Knowing the poles of the system we can calculate the required step size to fulfill the stability condition. We will apply each poll into the following equation derived in the previous assignment based on the forward Euler condition.

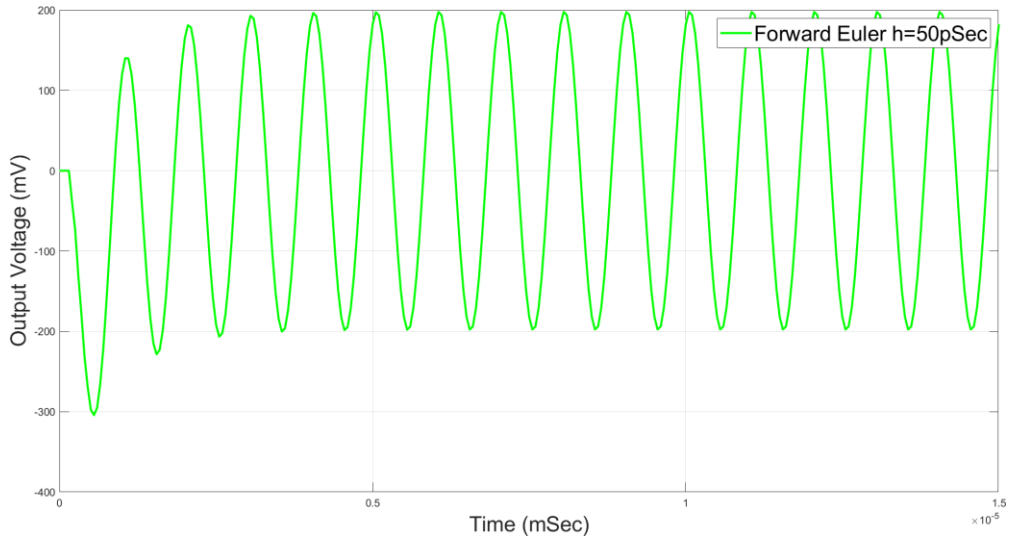
$$|1 + h\lambda| < 1$$

We get h <

5.6624e-11	8.5204e-11	2.0000e-10	1.6582e-09

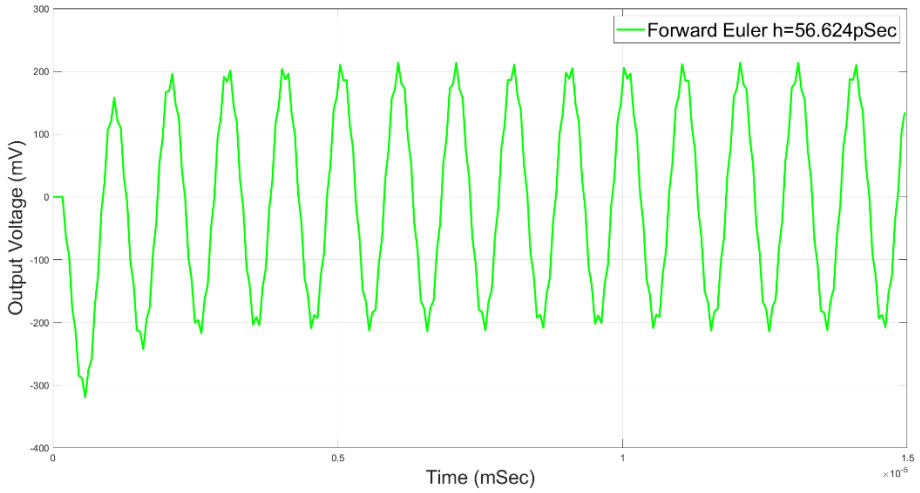
So, our stability condition is h < **5.6624e-11**. Through the simulation before we can observe that our stability condition is indeed **5.6624e-11**. As we set h below the condition the system is stable, and we get the desired output voltage corresponding to steady state and even at the stability condition we get the overall structure but there is a little bit of noise. Lastly when we set h to 57ps which is above the stability condition we can see that the system never reaches steady state.

Forward Euler Method h = 50ps

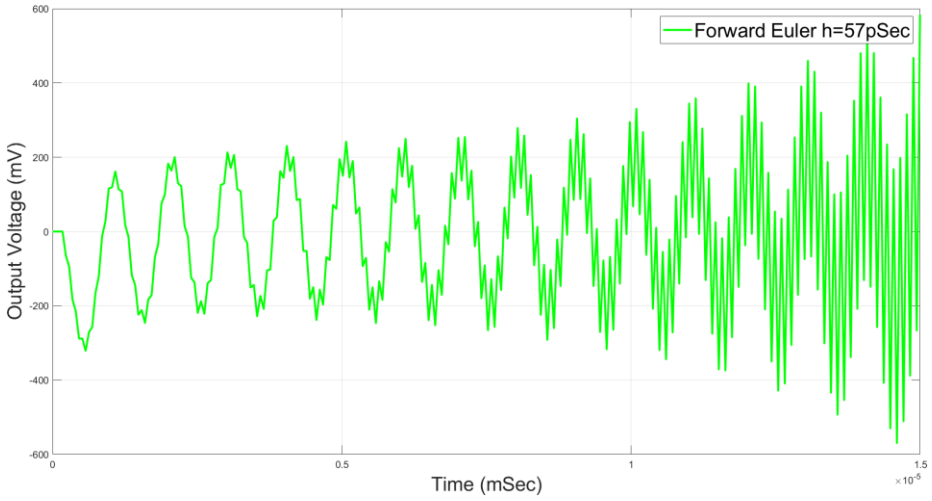


Forward Euler Method at the stability condition h = 56.634ps

Bryan Jay
260738764

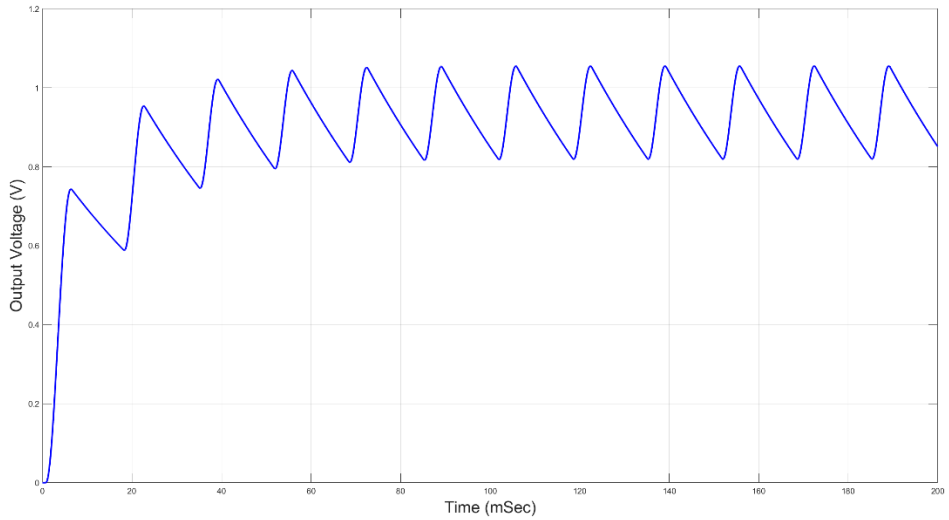


Forward Euler Method slightly above the stability condition h = 57ps



4- Nonlinear Circuits. Please refer to Appendix D for the corresponding Matlab code.

Non-linear Backward Euler Method



Bryan Jay
260738764

Appendix

Appendix A: Question 1

```
function [tpoints,r] = transient_beuler(t1,t2,h,out)
% [tpoints,r] = beuler(t1,t2,h,out)
% Perform transient analysis for LINEAR Circuits using Backward Euler
% assume zero initial condition.
% Inputs:  t1 = starting time point (typically 0)
%          t2 = ending time point
%          h  = step size
%          out = output node
% Outputs  tpoints = are the time points at which the output
%                    was evaluated
%          r       = value of the response at above time points
% plot(tpoints,r) should produce a plot of the transient response

global G C b

% tpoints is a vector containing all the timepoints from t1 to t2 with step
% size h
tpoints = t1:h:t2;

r = zeros(1,length(tpoints));

% assuming zero for IC
x_n = zeros(size(b));

for i=1:length(tpoints)-1
    x_n = inv(G + C / h) * (BTime(tpoints(i+1)) + (C / h) * x_n);

    r(i+1) = x_n(out);
end
```

Appendix B: Question 2

```
function [tpoints,r] = transient_trapez(t1,t2,h,out)
% [tpoints,r] = Transient_trapez(t1,t2,h,out)
% Perform Transient Analysis using the Trapezoidal Rule for LINEAR
% circuits.
% assume zero initial condition.
% Inputs:  t1 = starting time point (typically 0)
%          t2 = ending time point
%          h  = step size
%          out = output node
% Outputs  tpoints = are the time points at which the output
%                    was evaluated
%          r       = value of the response at above time points
% plot(tpoints,r) should produce a plot of the transient response

global G C b

x0 = zeros(size(G,2),1);

tpoints = t1:h:t2;

r = zeros(1,length(tpoints));

% assuming zero for IC
x_n = zeros(size(b));

for i=1:length(tpoints)-1
    x_n = inv(G + 2 * C / h) * (BTime(tpoints(i+1)) + BTime(tpoints(i)) + (2 * C / h -
G) * x_n);
```

```
    r(i+1) = x_n(out);
end
```

Appendix C: Question 3

```
function [tpoints,r] = transient_feuler(t1,t2,h,out)
% [tpoints,r] = Transient_feuler(t1,t2,h,out)
% Perform Transient analysis for LINEAR circuit using Forward Euler
% This function assumes the C matrix is invertible and will not work
% for circuits where C is not invertible.
% assume zero initial condition.
% Inputs:  t1 = starting time point (typically 0)
%          t2 = ending time point
%          h  = step size
%          out = output node
% Outputs  tpoints = are the time points at which the output
%                    was evaluated
%          r       = value of the response at above time points
% plot(tpoints,r) should produce a plot of the transient response


global G C b

x0 = zeros(size(G,2),1);

tpoints = t1:h:t2;

r = zeros(1,length(tpoints));

% assuming zero for IC
x_n = x0;

for i=1:length(tpoints)-1
    x_n = inv(C / h) * (BTime(tpoints(i)) + (C / h - G) * x_n);

    r(i+1) = x_n(out);
end
```

Appendix D: Question 4

```
function [tpoints,r] = nl_transient_beuler(t1,t2,h,out)
% [tpoints,r] = beuler(t1,t2,h,out)
% Perform transient analysis for NONLINEAR Circuits using Backward Euler
% Assume zero initial condition.
% Inputs:  t1 = starting time point (typically 0)
%          t2 = ending time point
%          h  = step size
%          out = output node
% Outputs  tpoints = are the time points at which the output
%                    was evaluated
%          r       = value of the response at above time points
% plot(tpoints,r) should produce a plot of the transient response

global G C b

tpoints = t1:h:t2;

r = zeros(1,length(tpoints));
maxerr = 10^(-6);

% assuming zero for IC
x_n = zeros(size(b));
x_n1 = zeros(size(b));
```

Bryan Jay
260738764

```
for i=1:length(tpoints)-1
    delta_x = intmax();

    while delta_x >= maxerr
        f = f_vector(x_n1);
        phi = f + (G + C / h) * x_n1 - BTime(tpoints(i+1)) - (C / h) * x_n;

        % Get the Jacobian matrix
        J = nlJacobian(x_n1);
        phi_d = G + C/h + J;

        % get delta_x matrix
        delta_x_m = -1 * inv(phi_d) * phi;

        % caclulate the new point to test and get the normal of delta_x
        x_n1 = x_n1 + delta_x_m;
        delta_x = norm(delta_x_m);
    end
    r(i+1) = x_n1(out);
    x_n = x_n1;
end
```