ECSE 597 Report

Q1

```
function [tpoints, y]= BE_method(tEnd,h, outNode)
% This function uses BACKWARD EULER method to compute the transient reponse
% of the circuit.
%Inputs: 1. tEnd:  The simulation starts at time = 0s  and ends at time =
%                  tEND s.
%        2. h: length  of step size.
%        3. outNode: is the node for which the transient is required.
%Output:  1. tpoints: list of time points.
%         2. y:  is the transient response at output node.
%
%Note: The function stub provided above is just an example. You can modify the
%      in function in any fashion.
%--------------------------------------------------------------------------

global elementList


tpoints = 0:h:tEnd;
Gmat = makeGmatrix;
Cmat = makeCmatrix;
out_NodeNumber = getNodeNumber(outNode) ;
[row,~] = size(Gmat);
X_n = zeros(row);
% x_n+1 = x_n + hx_n+1_dot
% x_n+1_dot = (x_n+1 - x_n)/h
% Gx_n+1 + C
for I=1:length(tpoints)
    % make Bvector at time tpoints(I)
    Btr = makeBt(tpoints(I));

    % you can write your code here
    X_n = inv((Gmat+Cmat/h))*(Btr + (Cmat/h)*X_n);
    y(I) = X_n(out_NodeNumber);
end


end
```

Figure 1 - Backward Euler

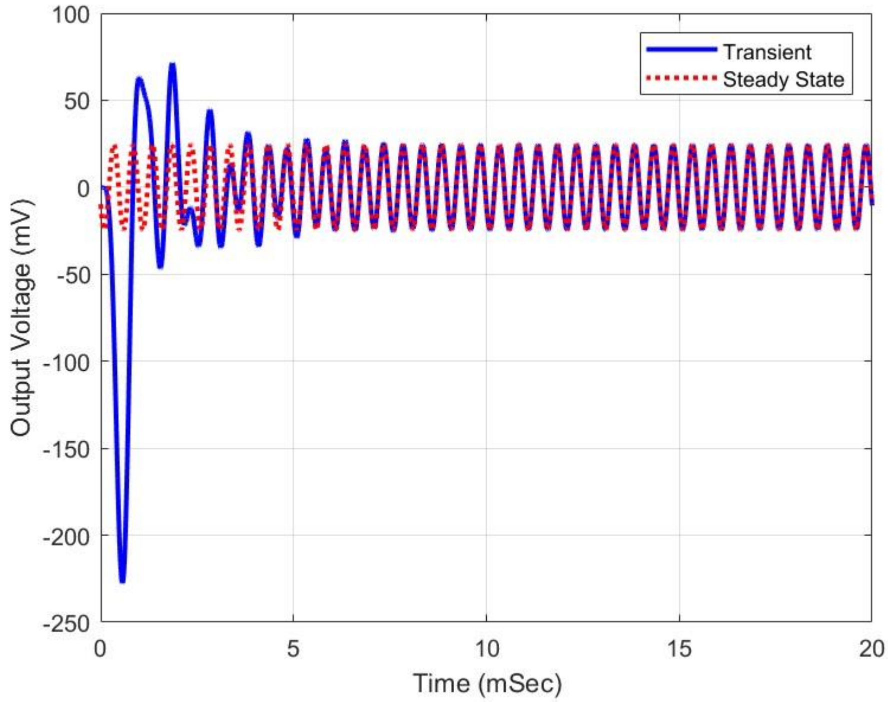

Figure 2 - BE Simulation

As shown in Figure 2, the transient response gradually converges to the steady state response, which is as expected.

Q2

```
function [tpoints,y]= Trapezoidal_method(tEnd,h, outNode)
% This function uses Trapezoidal method to compute the transient reponse
% of the circuit.
%Inputs: 1. tEnd:  The simulation starts at time = 0s  and ends at
%                  time = tEND s.
%        2. h: length  of step size.
%        3. outNode: is the node for which the transient is required.
%Output:  1.y:  is the transient response at output Node.
%
%Note: The function stub provided above is just an example. You can modify the
%      in function in any fashion.
%---------------------------------------------------------------------------

global elementList

out_NodeNumber = getNodeNumber(outNode) ;

tpoints = 0:h:tEnd;

Gmat = makeGmatrix;
Cmat = makeCmatrix;


[row,~] = size(Gmat);
X_n = zeros(row);
y(1) = 0;
for I=2:(length(tpoints))
    % make Bvector at time tpoints(I)
    Btr = makeBt(tpoints(I-1));
    Btr_1 = makeBt(tpoints(I));
    % you can write your code here
    X_n = inv((Gmat+(2/h)*Cmat))*(((2/h)*Cmat-Gmat)*X_n+Btr + Btr_1);
    y(I) = X_n(out_NodeNumber);
end


end
```

Figure 3 - Trapezoidal Rule

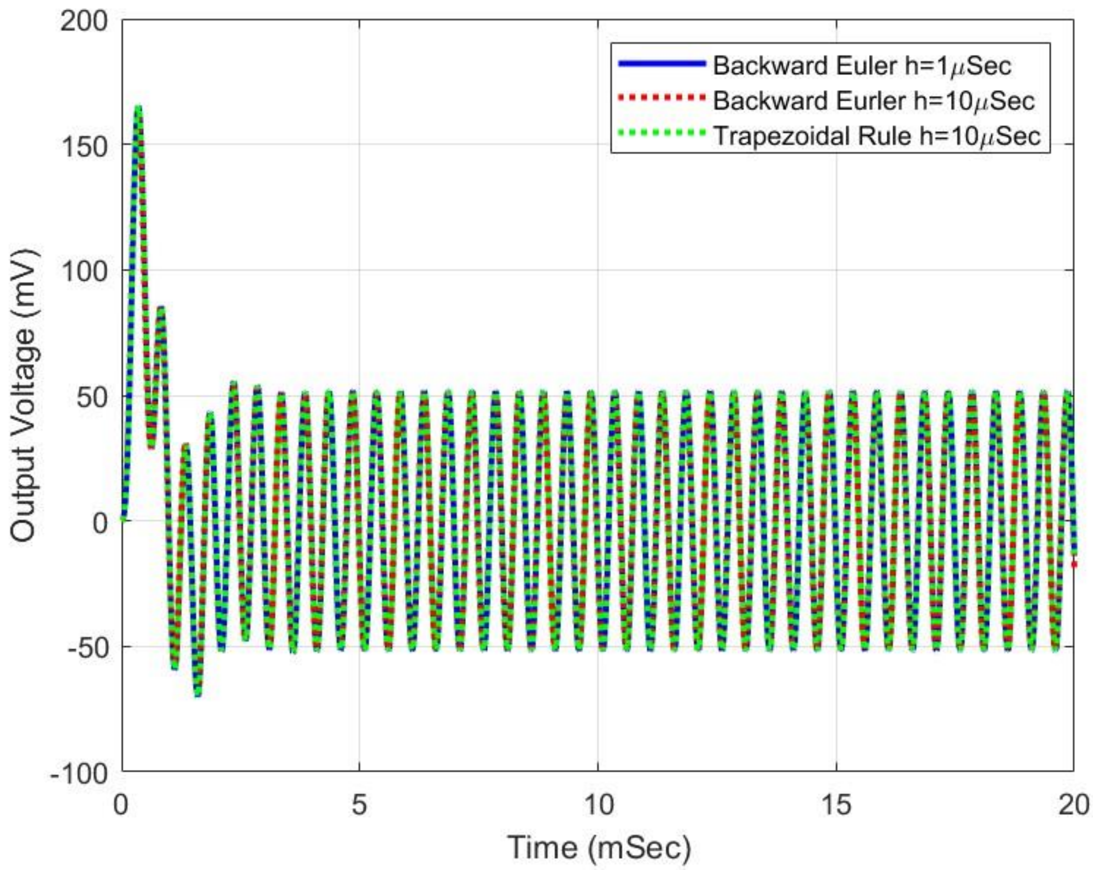

Figure 4 - TR Simulation

As shown in Figure 4, it can be deduced that both BE and TR methods can generates the same results and are both stable in most cases.

Q3

```
function [tpoints, y]= FE_method(tEnd,h, outNode)
% This function uses FORWARD EULER method to compute the transient reponse
% of the circuit.
%Inputs: 1. tEnd:  The simulation starts at time = 0s  and ends at
%                  time = tEND s.
%        2. h: length  of step size.
%        3. outNode: is the node for which the transient is required.
%Output:  1.y:  is the transient response at outNode.
%
%
%Note: The function stub provided above is just an example. You can modify the
%      in function in any fashion.
%--------------------------------------------------------------------------

global elementList

out_NodeNumber = getNodeNumber(outNode) ;
tpoints = 0:h:tEnd;

Gmat = makeGmatrix;
Cmat = makeCmatrix;

[row,~] = size(Gmat);
X_n = zeros(row);
y(1) = 0;
for I=2:(length(tpoints))
    % make Bvector at time tpoints(I)
    Btr = makeBt(tpoints(I-1));
    X_n = inv((Cmat/h))*(Btr-(Gmat-Cmat/h)*X_n);
    % you can write your code here
    y(I) = X_n(out_NodeNumber);
end


end
```

Figure 6 - Forward Euler

```

Gmat = makeGmatrix;
Cmat = makeCmatrix;

poles = eig((-1) * Cmat\Gmat)

```

Figure 5 - Pole Calculation

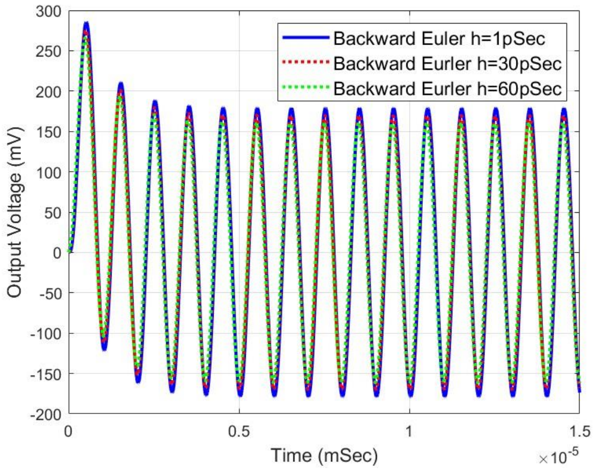

Figure 10 - BE Simulation Q3

```
poles =

   1.0e+10 *

  -3.5321
  -2.3473
  -1.0000
  -0.1206
```

Figure 9 - Pole Values

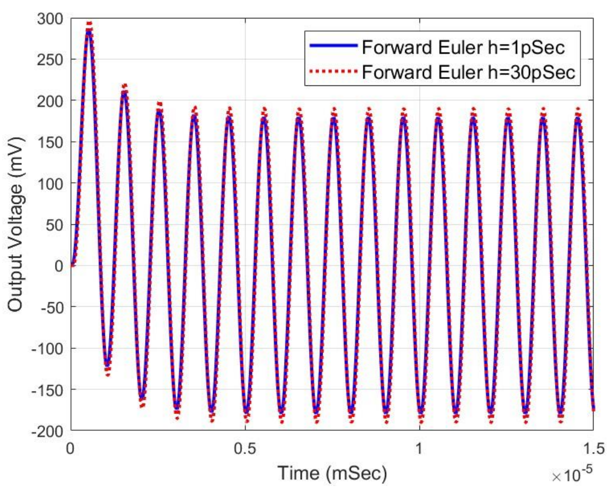

Figure 8 - FE Simulation (1 ps & 30 ps)

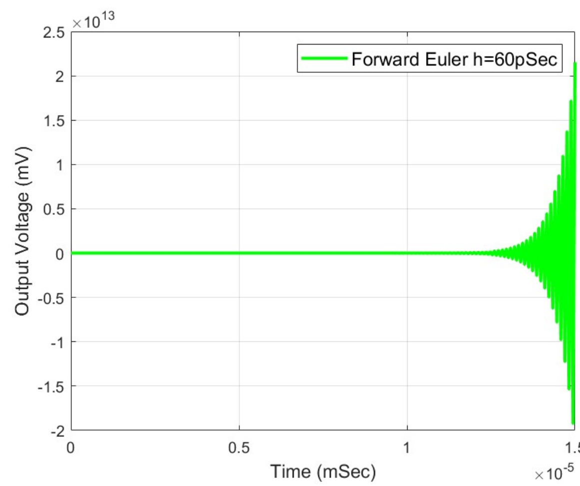

Figure 7 - FE Simulation (60 ps)

As shown in Figure 7 and 9, FE method is stable when step size h is 1 ps and 30 ps, and the results are the same as the BE method.

According to the property of FE method mentioned in class, the value of poles and the step size need to satisfy the condition of $-2 < h\lambda < 0$ to have a stable simulation. By examine the poles with the eigen value function as shown in Figure 6, where the values are shown in Figure 8, the step size needs to be considered for the most negative pole of $-3.5321e10$. In this case, the step size h needs to be smaller than 56.624 ps to have a stable simulation for FE. This is verified in Figure 10, where the simulation diverges when the step size is 60 ps.

Q4

```
function [D,S] = sens_perturbation_method(fpoints,eleNames,outNode)
% find elements
global elementList
flag = 0
ori_Gmat = makeGmatrix;
ori_Cmat = makeCmatrix;
out_NodeNumber = getNodeNumber(outNode);
ori_r = fsolve( fpoints ,outNode, ori_Gmat, ori_Cmat);
for m = 1:length(eleNames)
    element = eleNames(m);
    Gmat = ori_Gmat;
    Cmat = ori_Cmat;
    flag = 0;
    for I=1:elementList.Resistors.numElements
        name = elementList.Resistors.Name(I);
        if strcmp(name,element)
            nodes = elementList.Resistors.nodeNumbers(I,:);
            g = 1/( elementList.Resistors.value(I));
            g_delta = (1.0000001/( elementList.Resistors.value(I)));
            delta_lamda = 0.0000001/( elementList.Resistors.value(I));
            lamda = g;
            if(nodes(1)~=0) && (nodes(2)~=0)
                Gmat(nodes(1),nodes(1)) = Gmat(nodes(1),nodes(1))  - g + g_delta;
                Gmat(nodes(1),nodes(2)) = Gmat(nodes(1),nodes(2)) + g - g_delta;
                Gmat(nodes(2),nodes(1)) = Gmat(nodes(2),nodes(1)) + g - g_delta;
                Gmat(nodes(2),nodes(2)) = Gmat(nodes(2),nodes(2)) - g + g_delta;
            elseif (nodes(1)==0) && (nodes(2)~=0)

                Gmat(nodes(2),nodes(2)) = Gmat(nodes(2),nodes(2)) - g + g_delta;

            elseif (nodes(1)~=0) && (nodes(2)==0)
                Gmat(nodes(1),nodes(1)) = Gmat(nodes(1),nodes(1))  - g + g_delta;
            end
            flag = 1;
            break;
        end

    end
```

Figure 11 - Perturbation Method (Part A)

```
    if flag == 0
        for I=1:elementList.Capacitors.numElements
            name = elementList.Capacitors.Name(I);
            if strcmp(name,element)
                nodes = elementList.Capacitors.nodeNumbers(I,:);
                c= elementList.Capacitors.value(I);
                c_delta= elementList.Capacitors.value(I)*1.0000001;
                delta_lamda = elementList.Capacitors.value(I)*0.0000001;
                lamda = c;
                if(nodes(1)~=0) && (nodes(2)~=0)
                    Cmat(nodes(1),nodes(1)) = Cmat(nodes(1),nodes(1))  - c + c_delta;
                    Cmat(nodes(1),nodes(2)) = Cmat(nodes(1),nodes(2)) + c - c_delta;
                    Cmat(nodes(2),nodes(1)) = Cmat(nodes(2),nodes(1)) + c - c_delta;
                    Cmat(nodes(2),nodes(2)) = Cmat(nodes(2),nodes(2)) - c + c_delta;
                elseif (nodes(1)==0) && (nodes(2)~=0)

                    Cmat(nodes(2),nodes(2)) = Cmat(nodes(2),nodes(2)) - c + c_delta;

                elseif (nodes(1)~=0) && (nodes(2)==0)
                    Cmat(nodes(1),nodes(1)) = Cmat(nodes(1),nodes(1))  - c + c_delta;
                end
                break;
            end
        end
    end

    % now we have the changed G and C

    delta_r = fsolve( fpoints ,outNode, Gmat, Cmat) - ori_r;
    D(:,m) = delta_r/delta_lamda;
    for I = 1:length(fpoints)
        S(I,m) = D(I,m)*(lamda/ori_r(I));
    end
end

end
```

Figure 12 - Perturbation (Part B)

Q5

```
function [D,S] = sens_differentiation_method(fpoints,eleNames,out)
global elementList
flag = 0;
ori_Gmat = makeGmatrix;
ori_Cmat = makeCmatrix;
f_r_v = f_solve_vector(fpoints, ori_Gmat, ori_Cmat);
out_NodeNumber = getNodeNumber(out) ;
lamda = 0;
for m = 1:length(eleNames)
    delta_Gmat = zeros(size(ori_Gmat));
    delta_Cmat = zeros(size(ori_Cmat));
    element = eleNames(m);
    flag = 0;
    for I=1:elementList.Resistors.numElements
        name = elementList.Resistors.Name(I);
        lamda = 1/elementList.Resistors.value(I);
        if strcmp(name,element)
            nodes = elementList.Resistors.nodeNumbers(I,:);
            if(nodes(1)~=0) && (nodes(2)~=0)
                delta_Gmat(nodes(1),nodes(1)) = 1;
                delta_Gmat(nodes(1),nodes(2)) = -1;
                delta_Gmat(nodes(2),nodes(1)) = -1;
                delta_Gmat(nodes(2),nodes(2)) = 1;
            elseif (nodes(1)==0) && (nodes(2)~=0)

                delta_Gmat(nodes(2),nodes(2)) = 1;

            elseif (nodes(1)~=0) && (nodes(2)==0)
                delta_Gmat(nodes(1),nodes(1)) = 1;
            end
            flag = 1;
            break;
        end

    end
```

Figure 13 - Differentiation Method (Part A)

```
    if flag == 0
        for I=1:elementList.Capacitors.numElements
            name = elementList.Capacitors.Name(I);
            lamda = elementList.Capacitors.value(I);
            if strcmp(name,element)
                nodes = elementList.Capacitors.nodeNumbers(I,:);
                c= elementList.Capacitors.value(I);
                c_delta= elementList.Capacitors.value(I)*1.01;
                if(nodes(1)~=0) && (nodes(2)~=0)
                    delta_Cmat(nodes(1),nodes(1)) = 1;
                    delta_Cmat(nodes(1),nodes(2)) = -1;
                    delta_Cmat(nodes(2),nodes(1)) = -1;
                    delta_Cmat(nodes(2),nodes(2)) = 1;
                elseif (nodes(1)==0) && (nodes(2)~=0)

                    delta_Cmat(nodes(2),nodes(2)) = 1;

                elseif (nodes(1)~=0) && (nodes(2)==0)
                    delta_Cmat(nodes(1),nodes(1)) = 1;
                end
                break;
            end
        end
    end

    % now we have the changed delta_G and delta_C -> dA/dlamda = delta_G +
    % jw*delta_C
    % loop over all frequency, select outnode

    for I = 1:length(fpoints)
        A = (ori_Gmat+2*pi*fpoints(I)*1i*ori_Cmat);
        delta_abs = A\((-1)*(delta_Gmat+2*pi*fpoints(I)*1i*delta_Cmat)*transpose(f_r_v(I,:)));
        D(I,m) = delta_abs(out_NodeNumber);
        S(I,m) = D(I,m)*(lamda/f_r_v(I,out_NodeNumber));
    end
end
```

Figure 14 - Differentiation Method (Part B)

Q6

```
function [D,S] = sens_adjoint_method(fpoints,eleNames,out)
global elementList
flag = 0;
ori_Gmat = makeGmatrix;
ori_Cmat = makeCmatrix;
f_r_v = f_solve_vector(fpoints, ori_Gmat, ori_Cmat);
out_NodeNumber = getNodeNumber(out) ;
lamda = 0;
d = zeros(length(ori_Gmat));
d(out_NodeNumber) = 1;

for I = 1:length(fpoints)
    A = (ori_Gmat+2*pi*fpoints(I)*1i*ori_Cmat);
    A_f(I,:,:) = d/A; % precalculate Xat for all frequency
end

for m = 1:length(eleNames)
    delta_Gmat = zeros(size(ori_Gmat));
    delta_Cmat = zeros(size(ori_Cmat));
    element = eleNames(m);
    flag = 0;
    for I=1:elementList.Resistors.numElements

        % access nodes Numbers of the Resistor
        %nodes of a I^{th} element are located in Row I of the nodeNumbers
        %field
        name = elementList.Resistors.Name(I);
        lamda = 1/elementList.Resistors.value(I);
        if strcmp(name,element)
            nodes = elementList.Resistors.nodeNumbers(I,:);

            % get the conductance for the resisitor
            % resistance is stored in the field named value
            if(nodes(1)~=0) && (nodes(2)~=0)
                delta_Gmat(nodes(1),nodes(1)) = 1;
                delta_Gmat(nodes(1),nodes(2)) = -1;
                delta_Gmat(nodes(2),nodes(1)) = -1;
                delta_Gmat(nodes(2),nodes(2)) = 1;
            elseif (nodes(1)==0) && (nodes(2)~=0)

                delta_Gmat(nodes(2),nodes(2)) = 1;

            elseif (nodes(1)~=0) && (nodes(2)==0)
                delta_Gmat(nodes(1),nodes(1)) = 1;
            end
            flag = 1;
            break;
        end
    end
```

Figure 15 - Adjoint Method (Part A)

```
    if flag == 0
        for I=1:elementList.Capacitors.numElements
            name = elementList.Capacitors.Name(I);
            lamda = elementList.Capacitors.value(I);
            if strcmp(name,element)
                nodes = elementList.Capacitors.nodeNumbers(I,:);
                c= elementList.Capacitors.value(I);
                c_delta= elementList.Capacitors.value(I)*1.01;
                if(nodes(1)~=0) && (nodes(2)~=0)
                    delta_Cmat(nodes(1),nodes(1)) = 1;
                    delta_Cmat(nodes(1),nodes(2)) = -1;
                    delta_Cmat(nodes(2),nodes(1)) = -1;
                    delta_Cmat(nodes(2),nodes(2)) = 1;
                elseif (nodes(1)==0) && (nodes(2)~=0)

                    delta_Cmat(nodes(2),nodes(2)) = 1;

                elseif (nodes(1)~=0) && (nodes(2)==0)
                    delta_Cmat(nodes(1),nodes(1)) = 1;
                end
                break;
            end
        end
    end

    for I = 1:length(fpoints)
        delta_abs = A_f(I,:,:)*(delta_Gmat+2*pi*fpoints(I)*1i*delta_Cmat)*transpose(f_r_v(I,:));
        D(I,m) = delta_abs;
        S(I,m) = D(I,m)*(lamda/f_r_v(I,out_NodeNumber));
    end

end
```

Figure 16 - Adjoint Method (Part B)

Q7

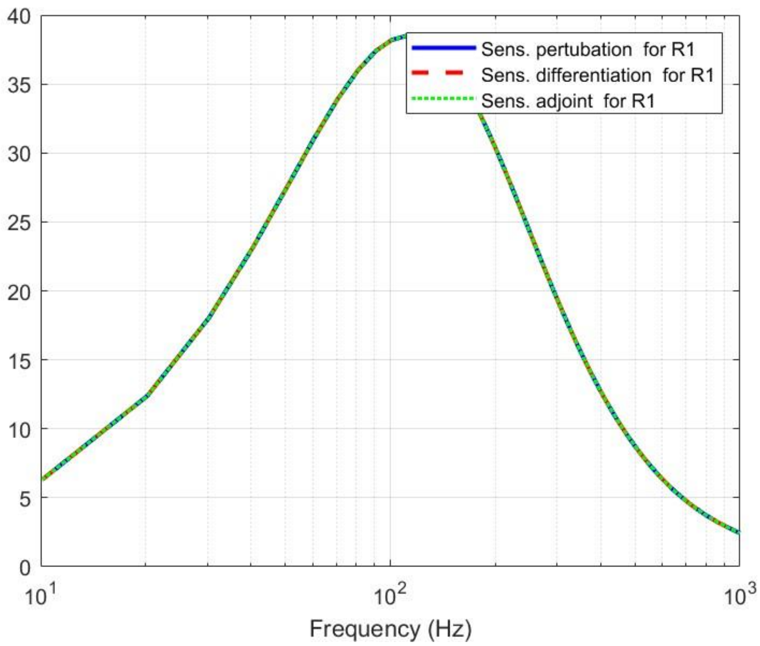

Figure 18 - R1 Absolute

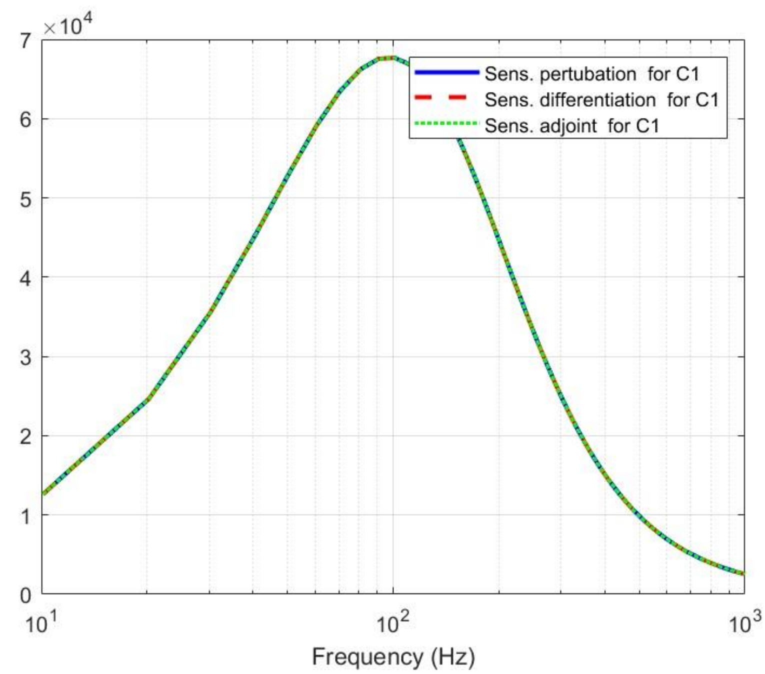

Figure 17 - C1 Absolute

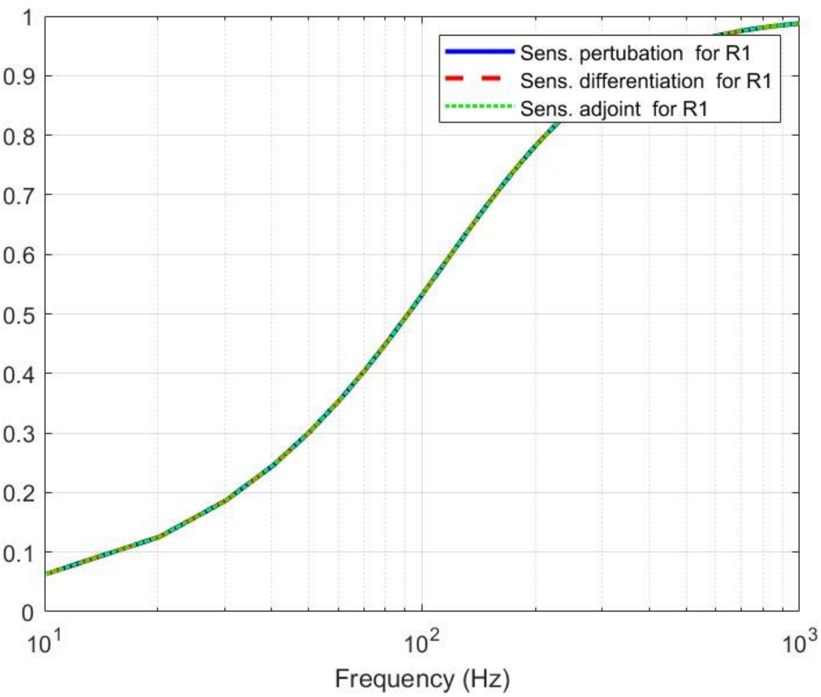

Figure 20 - R1 Relative

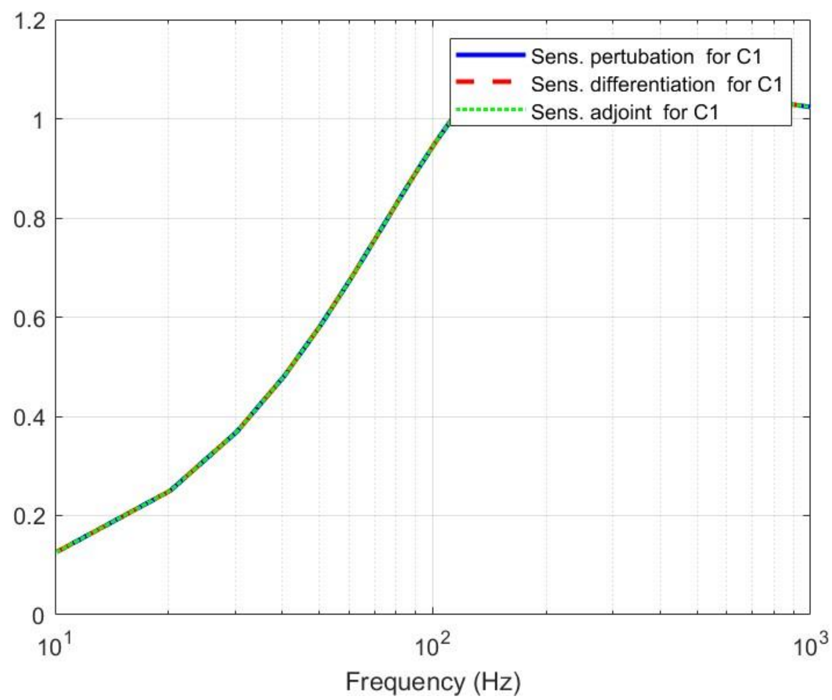

Figure 19 - C1 Relative