

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

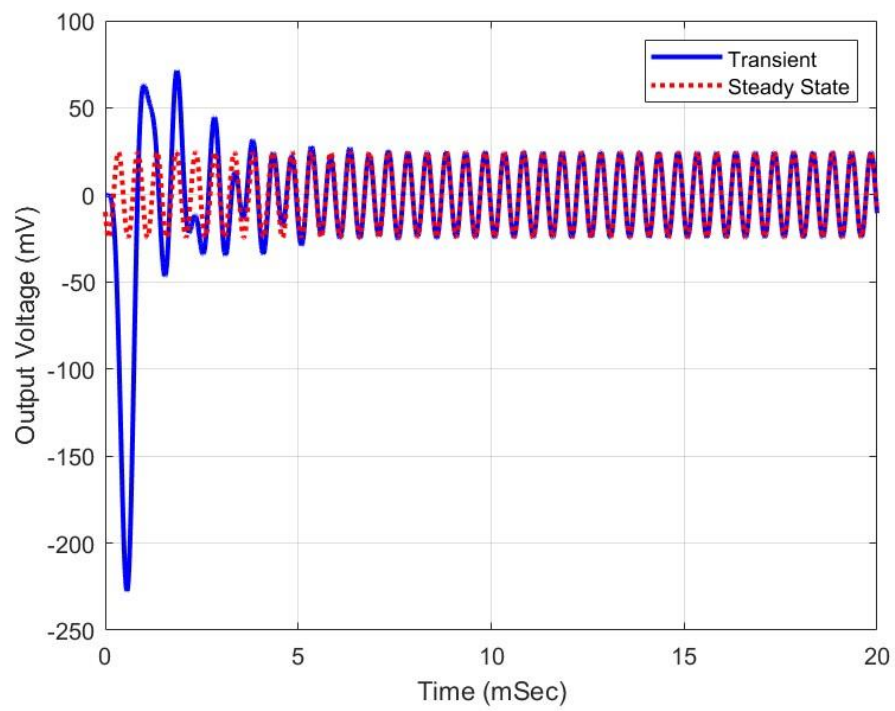


Figure 2 - BE Simulation

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

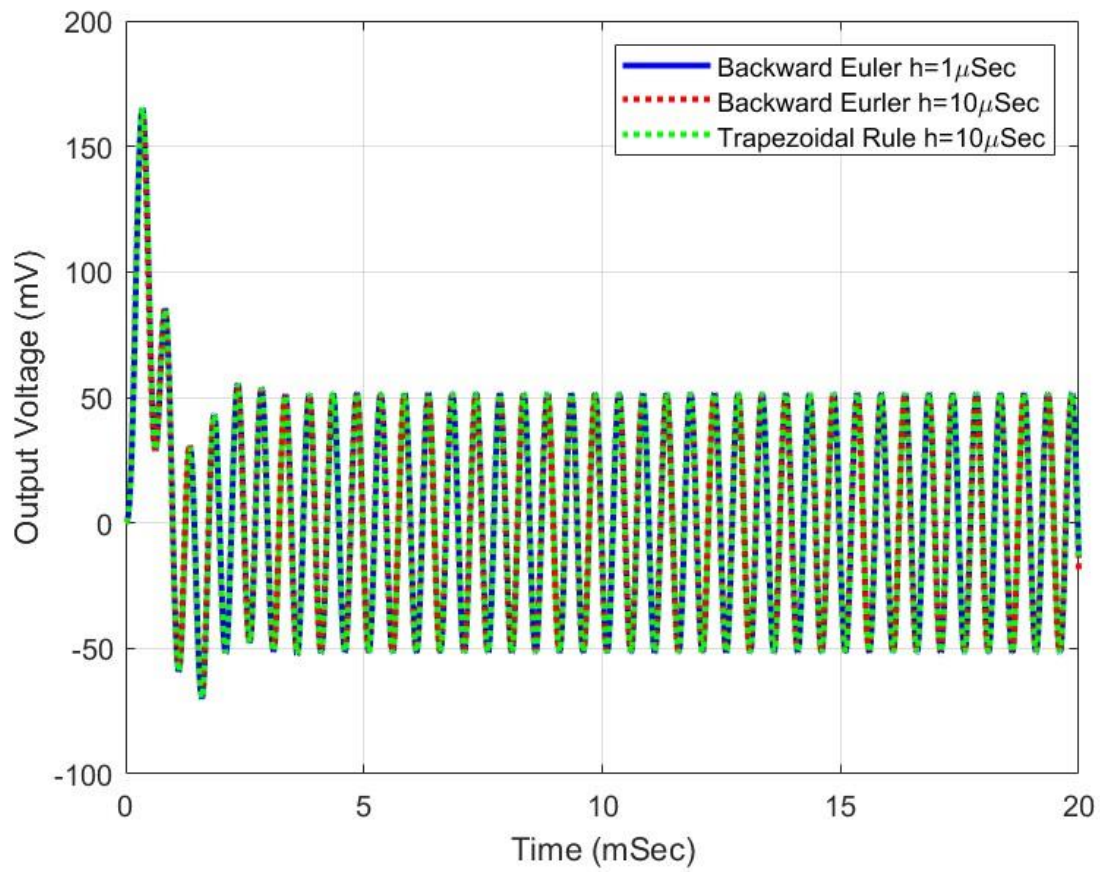


Figure 4 - TR Simulation

As shown in Figure 4, it can be deduced that both BE and TR methods can generate 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

```
63
64     Gmat = makeGmatrix;
65     Cmat = makeCmatrix;
66
67     poles = eig((-1) * Cmat\Gmat)
68
```

Figure 5 - Pole Calculation

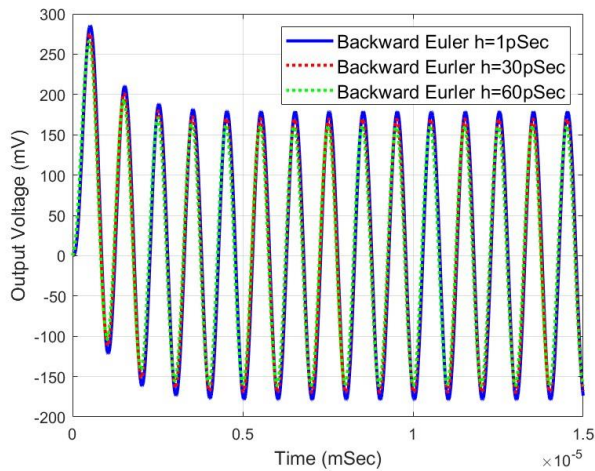


Figure 10 - BE Simulation Q3

```
poles =
    1.0e+10 *
    -3.5321
    -2.3473
    -1.0000
    -0.1206
```

Figure 9 - Pole Values

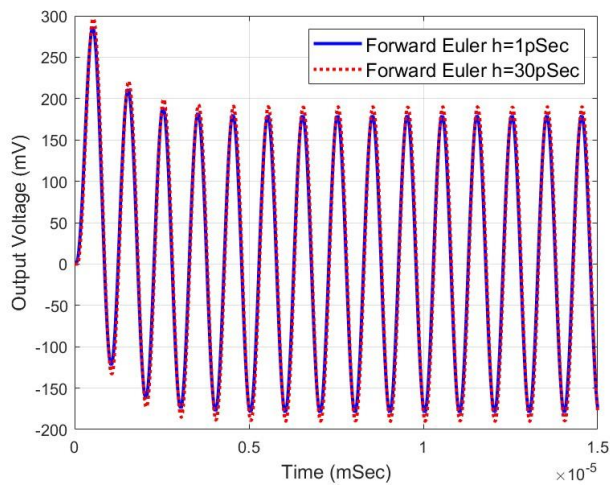


Figure 8 - FE Simulation (1 ps & 30 ps)

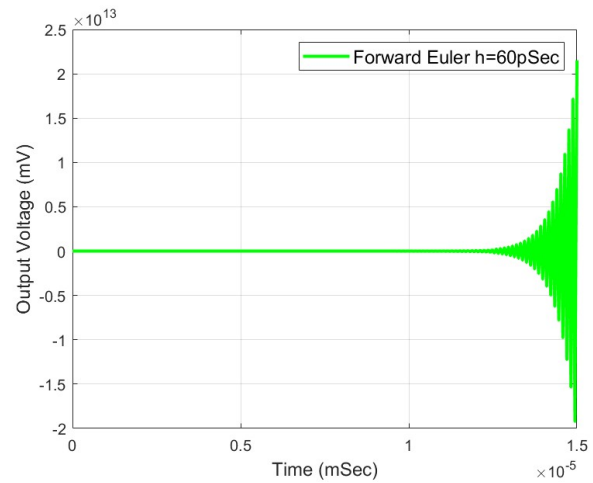


Figure 7 - FE Simulation (60 ps)

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

```

1 function [D,S] = sens_perturbation_method(fpoints,eleNames,outNode)
2 % find elements
3 global elementList
4 flag = 0
5 ori_Gmat = makeGmatrix;
6 ori_Cmat = makeCmatrix;
7 out_NodeNumber = getNodeNumber(outNode);
8 ori_r = fsolve( fpoints ,outNode, ori_Gmat, ori_Cmat);
9 for m = 1:length(eleNames)
10     element = eleNames(m);
11     Gmat = ori_Gmat;
12     Cmat = ori_Cmat;
13     flag = 0;
14     for I=1:elementList.Resistors.numElements
15         name = elementList.Resistors.Name(I);
16         if strcmp(name,element)
17             nodes = elementList.Resistors.nodeNumbers(I,:);
18             g = 1/( elementList.Resistors.value(I));
19             g_delta = (1.0000001/( elementList.Resistors.value(I)));
20             delta_lamda = 0.0000001/( elementList.Resistors.value(I));
21             lamda = g;
22             if(nodes(1)~=0) && (nodes(2)~=0)
23                 Gmat(nodes(1),nodes(1)) = Gmat(nodes(1),nodes(1)) - g + g_delta;
24                 Gmat(nodes(1),nodes(2)) = Gmat(nodes(1),nodes(2)) + g - g_delta;
25                 Gmat(nodes(2),nodes(1)) = Gmat(nodes(2),nodes(1)) + g - g_delta;
26                 Gmat(nodes(2),nodes(2)) = Gmat(nodes(2),nodes(2)) - g + g_delta;
27             elseif (nodes(1)==0) && (nodes(2)~=0)
28
29                 Gmat(nodes(2),nodes(2)) = Gmat(nodes(2),nodes(2)) - g + g_delta;
30
31             elseif (nodes(1)~=0) && (nodes(2)==0)
32                 Gmat(nodes(1),nodes(1)) = Gmat(nodes(1),nodes(1)) - g + g_delta;
33             end
34             flag = 1;
35             break;
36         end
37     end
38 end

```

Figure 11 - Perturbation Method (Part A)

```

39     if flag == 0
40         for I=1:elementList.Capacitors.numElements
41             name = elementList.Capacitors.Name(I);
42             if strcmp(name,element)
43                 nodes = elementList.Capacitors.nodeNumbers(I,:);
44                 c= elementList.Capacitors.value(I);
45                 c_delta= elementList.Capacitors.value(I)*1.0000001;
46                 delta_lamda = elementList.Capacitors.value(I)*0.0000001;
47                 lamda = c;
48                 if(nodes(1)~=0) && (nodes(2)~=0)
49                     Cmat(nodes(1),nodes(1)) = Cmat(nodes(1),nodes(1)) - c + c_delta;
50                     Cmat(nodes(1),nodes(2)) = Cmat(nodes(1),nodes(2)) + c - c_delta;
51                     Cmat(nodes(2),nodes(1)) = Cmat(nodes(2),nodes(1)) + c - c_delta;
52                     Cmat(nodes(2),nodes(2)) = Cmat(nodes(2),nodes(2)) - c + c_delta;
53                 elseif (nodes(1)==0) && (nodes(2)~=0)
54
55                     Cmat(nodes(2),nodes(2)) = Cmat(nodes(2),nodes(2)) - c + c_delta;
56
57                 elseif (nodes(1)~=0) && (nodes(2)==0)
58                     Cmat(nodes(1),nodes(1)) = Cmat(nodes(1),nodes(1)) - c + c_delta;
59                 end
60                 break;
61             end
62         end
63     end
64
65     % now we have the changed G and C
66
67     delta_r = fsolve( fpoints ,outNode, Gmat, Cmat) - ori_r;
68     D(:,m) = delta_r/delta_lamda;
69     for I = 1:length(fpoints)
70         S(I,m) = D(I,m)*(lamda/ori_r(I));
71     end
72 end
73
74 end
75

```

Figure 12 - Perturbation (Part B)

Q5

```

1 function [D,S] = sens_differentiation_method(fpoints,eleNames,out)
2 global elementList
3 flag = 0;
4 ori_Gmat = makeGmatrix;
5 ori_Cmat = makeCmatrix;
6 f_r_v = f_solve_vector(fpoints, ori_Gmat, ori_Cmat);
7 out_NodeNumber = getNodeNumber(out) ;
8 lamda = 0;
9 for m = 1:length(eleNames)
10     delta_Gmat = zeros(size(ori_Gmat));
11     delta_Cmat = zeros(size(ori_Cmat));
12     element = eleNames(m);
13     flag = 0;
14     for I=1:elementList.Resistors.numElements
15         name = elementList.Resistors.Name(I);
16         lamda = 1/elementList.Resistors.value(I);
17         if strcmp(name,element)
18             nodes = elementList.Resistors.nodeNumbers(I,:);
19             if(nodes(1)~=0) && (nodes(2)~=0)
20                 delta_Gmat(nodes(1),nodes(1)) = 1;
21                 delta_Gmat(nodes(1),nodes(2)) = -1;
22                 delta_Gmat(nodes(2),nodes(1)) = -1;
23                 delta_Gmat(nodes(2),nodes(2)) = 1;
24             elseif (nodes(1)==0) && (nodes(2)~=0)
25
26                 delta_Gmat(nodes(2),nodes(2)) = 1;
27
28             elseif (nodes(1)~=0) && (nodes(2)==0)
29                 delta_Gmat(nodes(1),nodes(1)) = 1;
30             end
31             flag = 1;
32             break;
33         end
34     end
35 end

```

Figure 13 - Differentiation Method (Part A)

```

36     if flag == 0
37         for I=1:elementList.Capacitors.numElements
38             name = elementList.Capacitors.Name(I);
39             lamda = elementList.Capacitors.value(I);
40             if strcmp(name,element)
41                 nodes = elementList.Capacitors.nodeNumbers(I,:);
42                 c= elementList.Capacitors.value(I);
43                 c_delta= elementList.Capacitors.value(I)*1.01;
44                 if(nodes(1)~=0) && (nodes(2)~=0)
45                     delta_Cmat(nodes(1),nodes(1)) = 1;
46                     delta_Cmat(nodes(1),nodes(2)) = -1;
47                     delta_Cmat(nodes(2),nodes(1)) = -1;
48                     delta_Cmat(nodes(2),nodes(2)) = 1;
49                 elseif (nodes(1)==0) && (nodes(2)~=0)
50
51                     delta_Cmat(nodes(2),nodes(2)) = 1;
52
53                 elseif (nodes(1)~=0) && (nodes(2)==0)
54                     delta_Cmat(nodes(1),nodes(1)) = 1;
55                 end
56                 break;
57             end
58         end
59     end
60
61     % now we have the changed delta_G and delta_C -> dA/dlamda = delta_G +
62     % jw*delta_C
63     % loop over all frequency, select outnode
64
65     for I = 1:length(fpoints)
66         A = (ori_Gmat+2*pi*fpoints(I)*1i*ori_Cmat);
67         delta_abs = A\((-1)*(delta_Gmat+2*pi*fpoints(I)*1i*delta_Cmat)*transpose(f_r_v(I,:));
68         D(I,m) = delta_abs(out_NodeNumber);
69         S(I,m) = D(I,m)*(lamda/f_r_v(I,out_NodeNumber));
70     end
71 end

```

Figure 14 - Differentiation Method (Part B)

Q6

```

1 function [D,S] = sens_adjoint_method(fpoints,eleNames,out)
2 global elementList
3 flag = 0;
4 ori_Gmat = makeGmatrix;
5 ori_Cmat = makeCmatrix;
6 f_r_v = f_solve_vector(fpoints, ori_Gmat, ori_Cmat);
7 out_NodeNumber = getNodeNumber(out) ;
8 lamda = 0;
9 d = zeros(length(ori_Gmat));
10 d(out_NodeNumber) = 1;
11
12 for I = 1:length(fpoints)
13     A = (ori_Gmat+2*pi*fpoints(I)*1i*ori_Cmat);
14     A_f(I,,:) = d/A; % precalculate Xat for all frequency
15 end
16
17 for m = 1:length(eleNames)
18     delta_Gmat = zeros(size(ori_Gmat));
19     delta_Cmat = zeros(size(ori_Cmat));
20     element = eleNames(m);
21     flag = 0;
22     for I=1:elementList.Resistors.numElements
23
24         % access nodes Numbers of the Resistor
25         %nodes of a I^{th} element are located in Row I of the nodeNumbers
26         %field
27         name = elementList.Resistors.Name(I);
28         lamda = 1/elementList.Resistors.value(I);
29         if strcmp(name,element)
30             nodes = elementList.Resistors.nodeNumbers(I,:);
31
32             % get the conductance for the resisitor
33             % resistance is stored in the field named value
34             if(nodes(1)~=0) && (nodes(2)~=0)
35                 delta_Gmat(nodes(1),nodes(1)) = 1;
36                 delta_Gmat(nodes(1),nodes(2)) = -1;
37                 delta_Gmat(nodes(2),nodes(1)) = -1;
38                 delta_Gmat(nodes(2),nodes(2)) = 1;
39             elseif (nodes(1)==0) && (nodes(2)~=0)
40
41                 delta_Gmat(nodes(2),nodes(2)) = 1;
42
43             elseif (nodes(1)~=0) && (nodes(2)==0)
44                 delta_Gmat(nodes(1),nodes(1)) = 1;
45             end
46             flag = 1;
47             break;
48         end
49     end

```

Figure 15 - Adjoint Method (Part A)

```

50 if flag == 0
51     for I=1:elementList.Capacitors.numElements
52         name = elementList.Capacitors.Name(I);
53         lamda = elementList.Capacitors.value(I);
54         if strcmp(name,element)
55             nodes = elementList.Capacitors.nodeNumbers(I,:);
56             c= elementList.Capacitors.value(I);
57             c_delta= elementList.Capacitors.value(I)*1.01;
58             if(nodes(1)~=0) && (nodes(2)~=0)
59                 delta_Cmat(nodes(1),nodes(1)) = 1;
60                 delta_Cmat(nodes(1),nodes(2)) = -1;
61                 delta_Cmat(nodes(2),nodes(1)) = -1;
62                 delta_Cmat(nodes(2),nodes(2)) = 1;
63             elseif (nodes(1)==0) && (nodes(2)~=0)
64
65                 delta_Cmat(nodes(2),nodes(2)) = 1;
66
67             elseif (nodes(1)~=0) && (nodes(2)==0)
68                 delta_Cmat(nodes(1),nodes(1)) = 1;
69             end
70             break;
71         end
72     end
73 end
74
75 for I = 1:length(fpoints)
76     delta_abs = A_f(I, :, :)*(delta_Gmat+2*pi*fpoints(I)*1i*delta_Cmat)*transpose(f_r_v(I, :));
77     D(I,m) = delta_abs;
78     S(I,m) = D(I,m)*(lamda/f_r_v(I,out_NodeNumber));
79 end
80
81 end

```

Figure 16 - Adjoint Method (Part B)

Q7

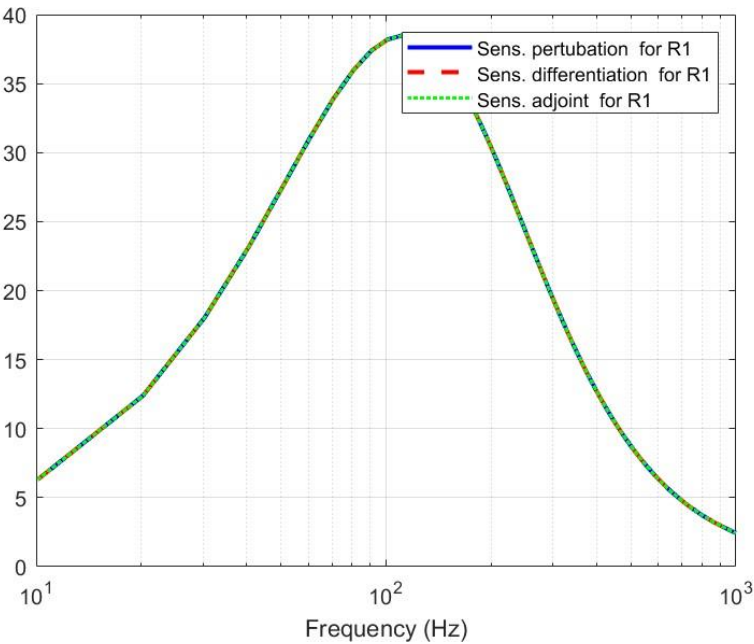


Figure 18 - R1 Absolute

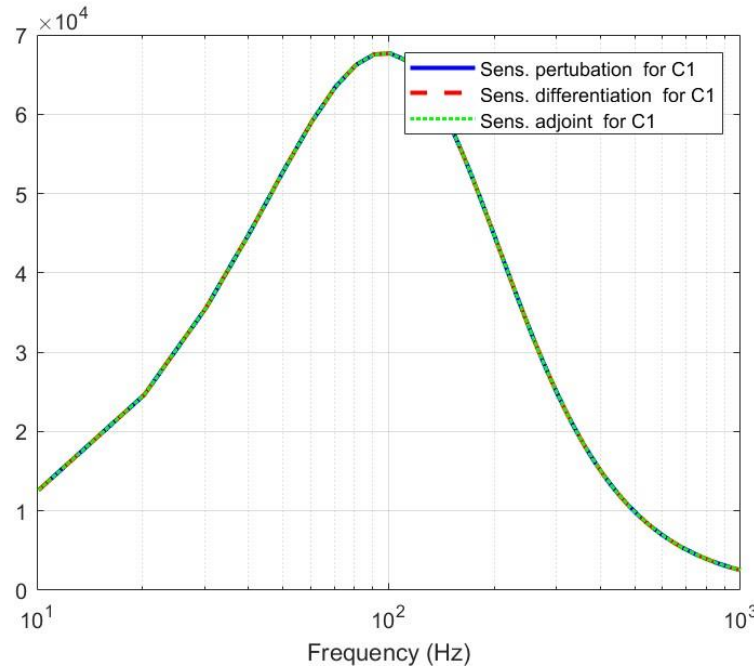


Figure 17 - C1 Absolute

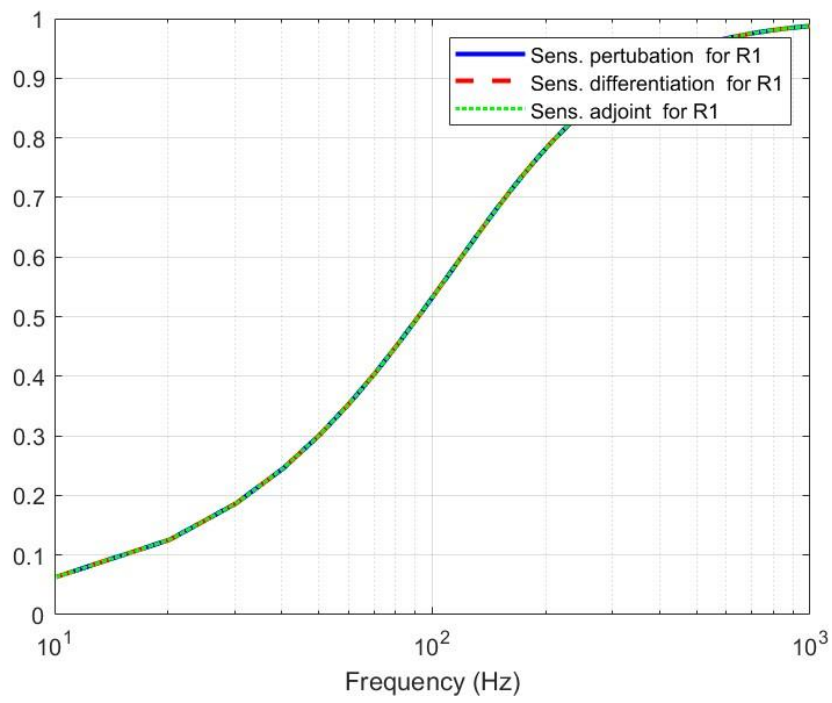


Figure 20 - R1 Relative

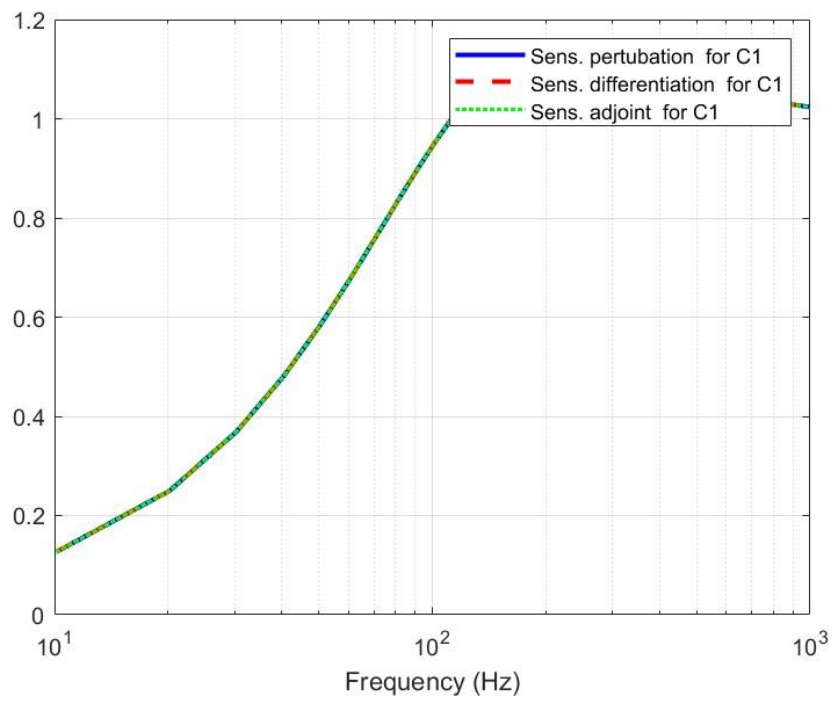


Figure 19 - C1 Relative

