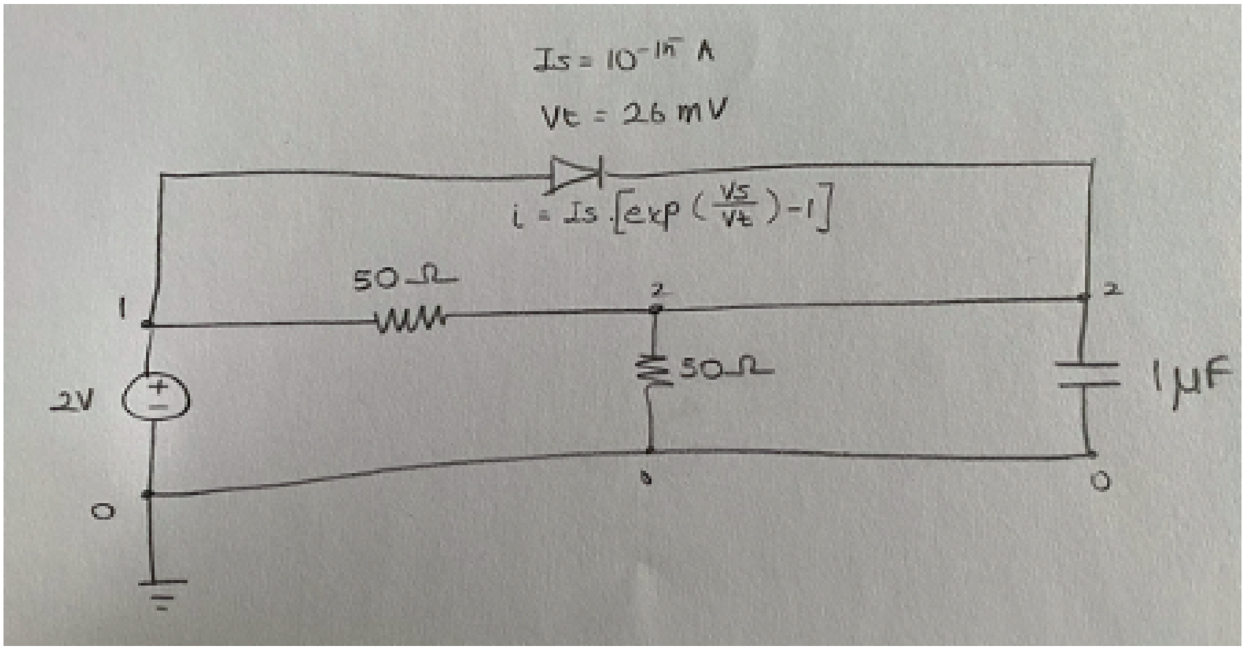
ECSE 597: Circuit Simulation and Modelling
Assignment 1

Student Name: Dafne Culha
Student ID: 260785524

Department of Electrical and Computer Engineering
McGill University, Canada
October, 2021

1. Deliverable 1

a-) The schematic of the circuit in Circuit_diodeckt1.m



b-) DC values computed after running test bench

The Xdc vector was returned as

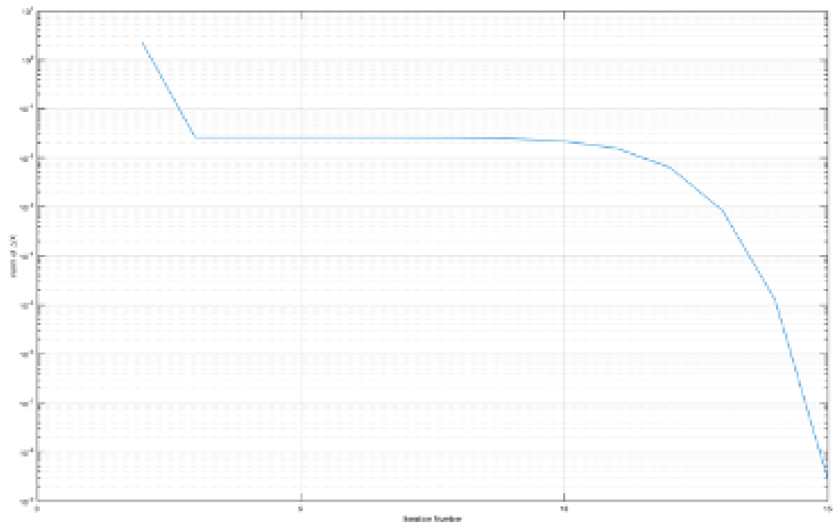
Xdc =

    2.0000
    1.2245
    -0.0245

This corresponds to a V1 = 2.0000 V, V2 = 1.2245 and I1 = -0.0245 A.

c-) the figure that was plotted when you ran the test bench

The norm of dX was plotted for each iteration.



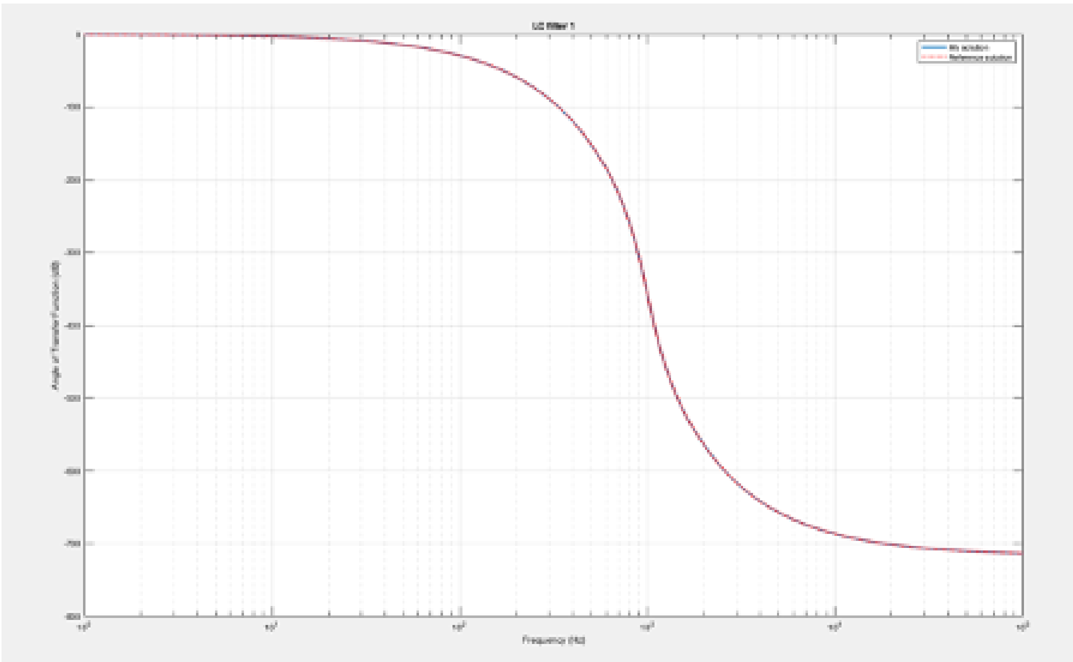
d-) the matlab code for nlJacobian.m and dcsolve.m

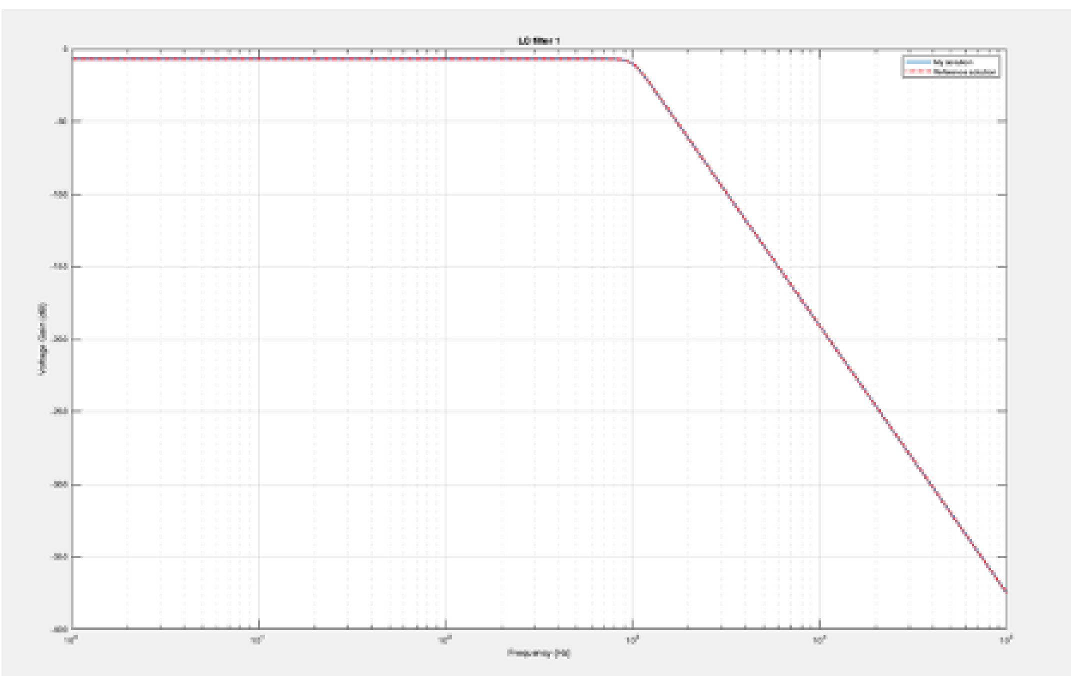
See Appendix A.

2. Deliverable 3

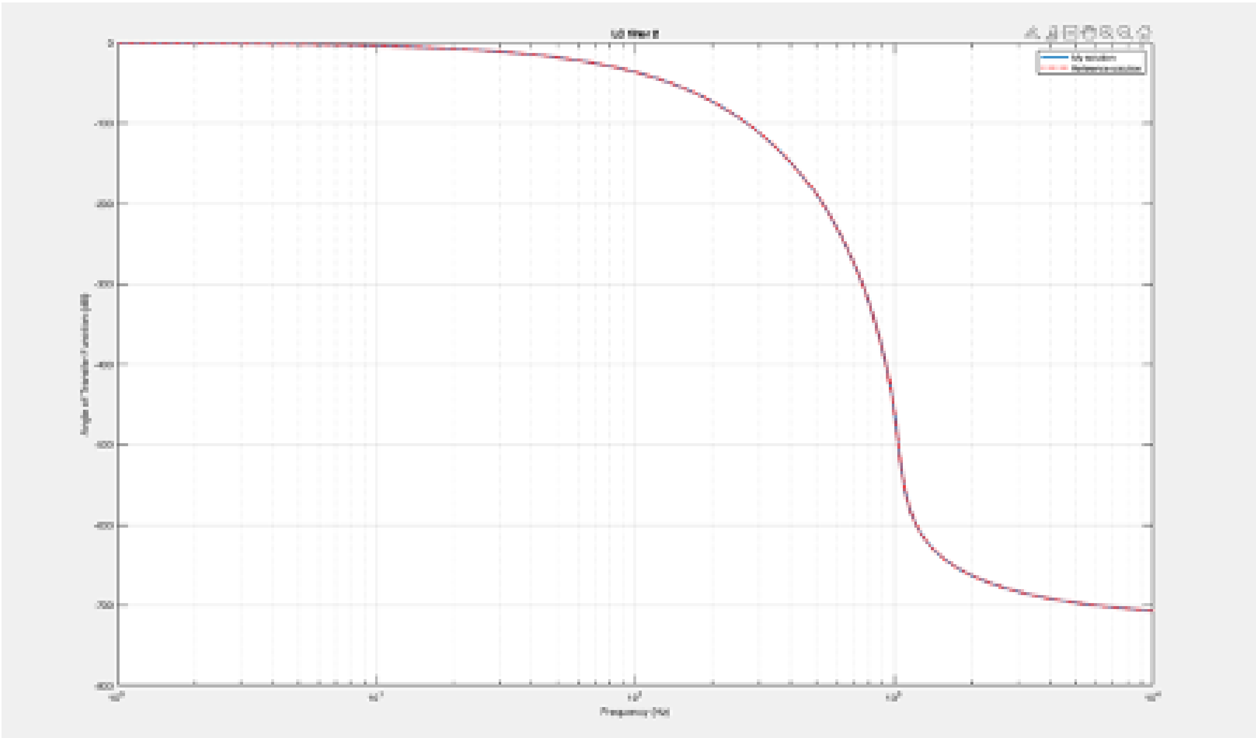
a-) Simulation results of all benchmark functions

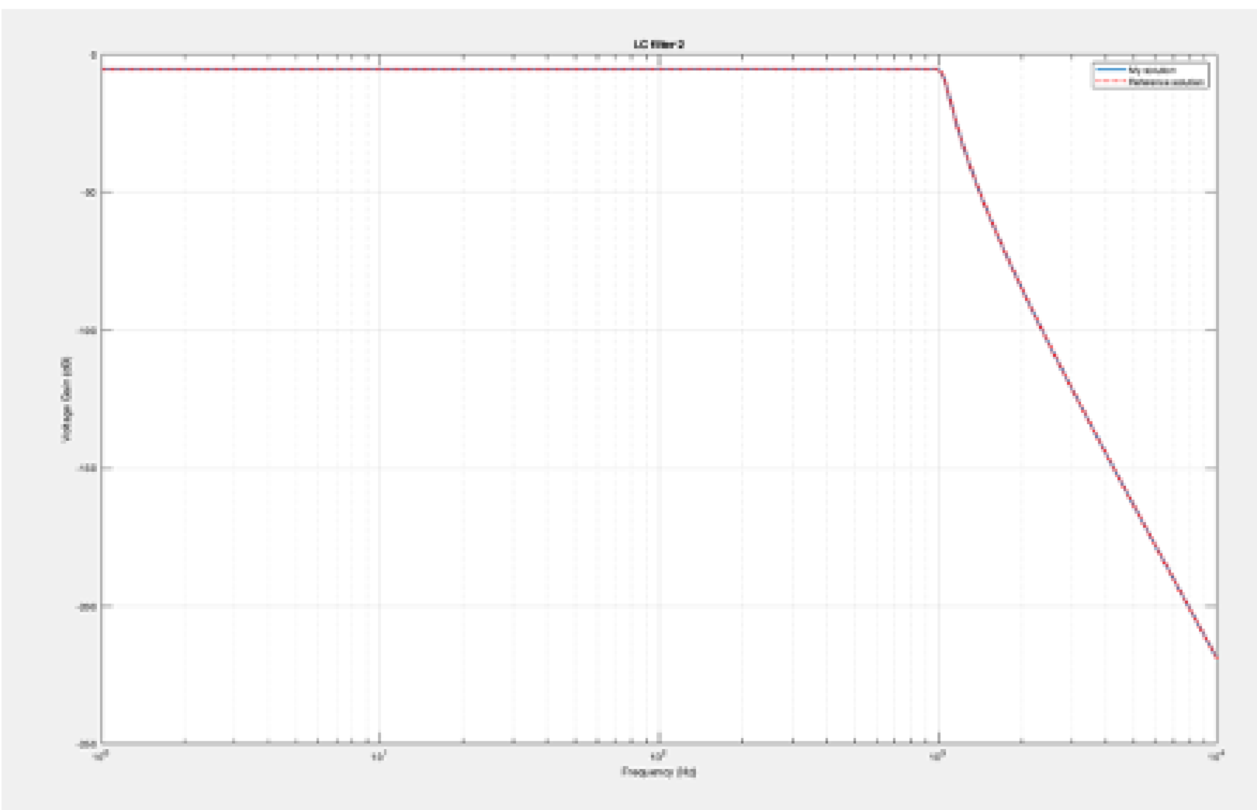
1-) LC Filter 1



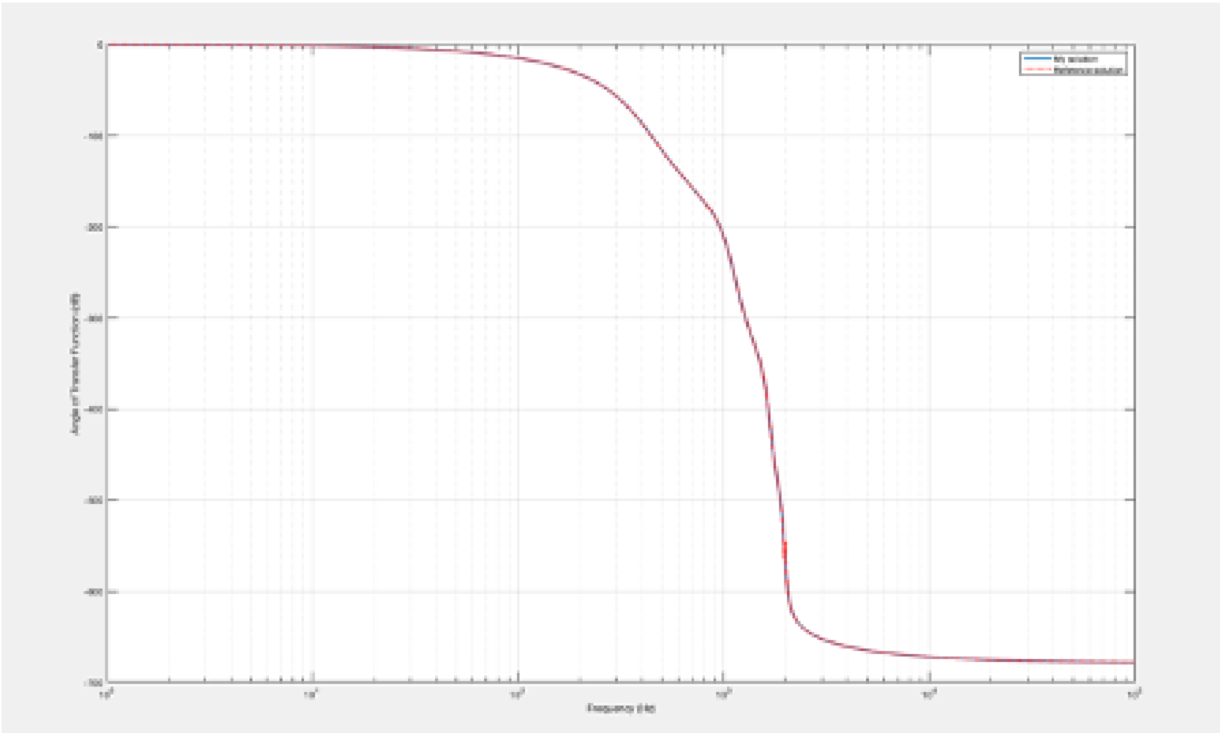


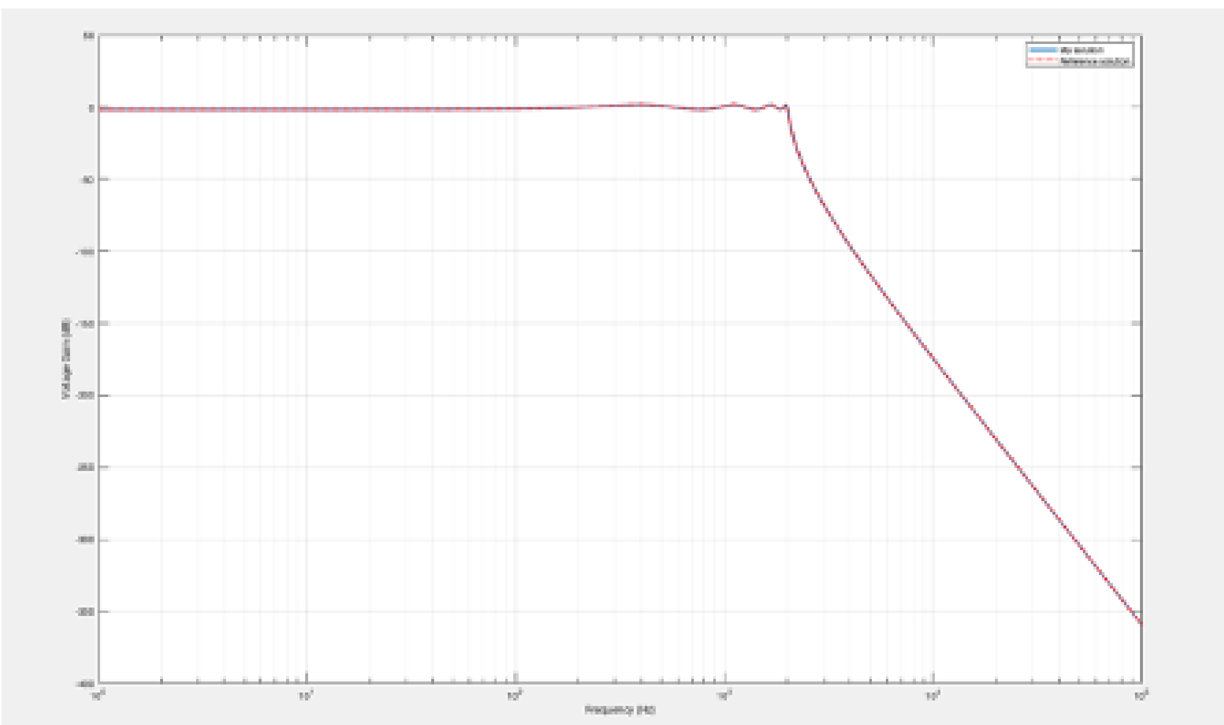
2-) LC Filter 2



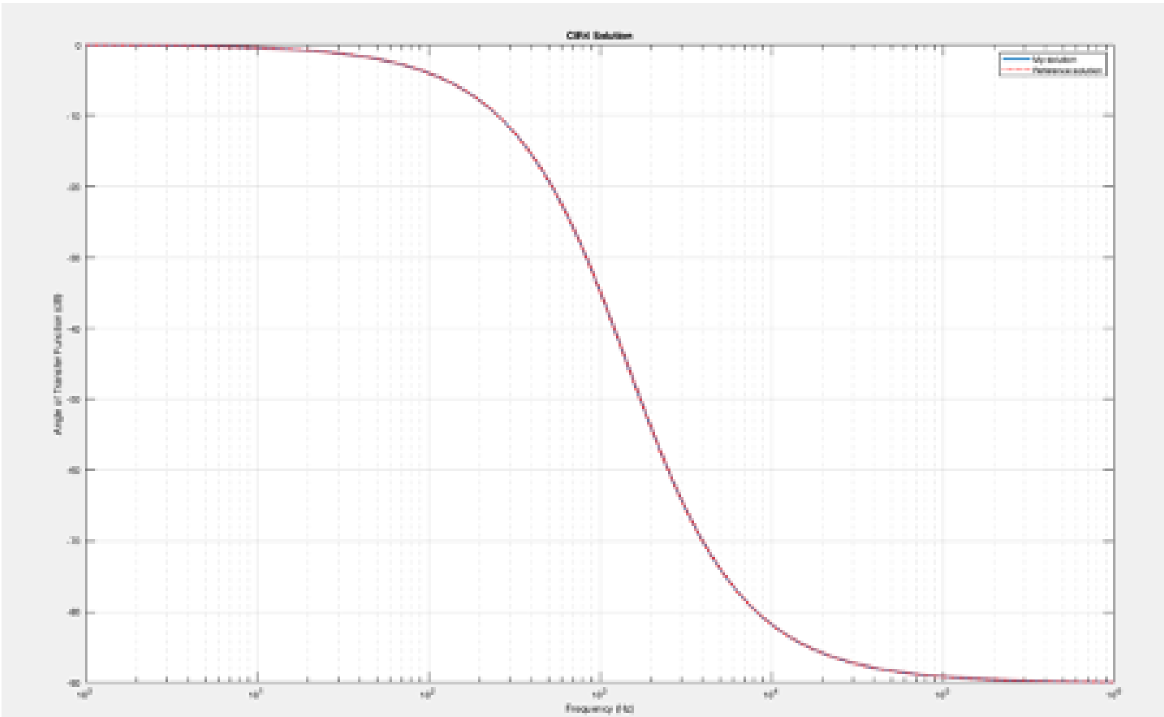


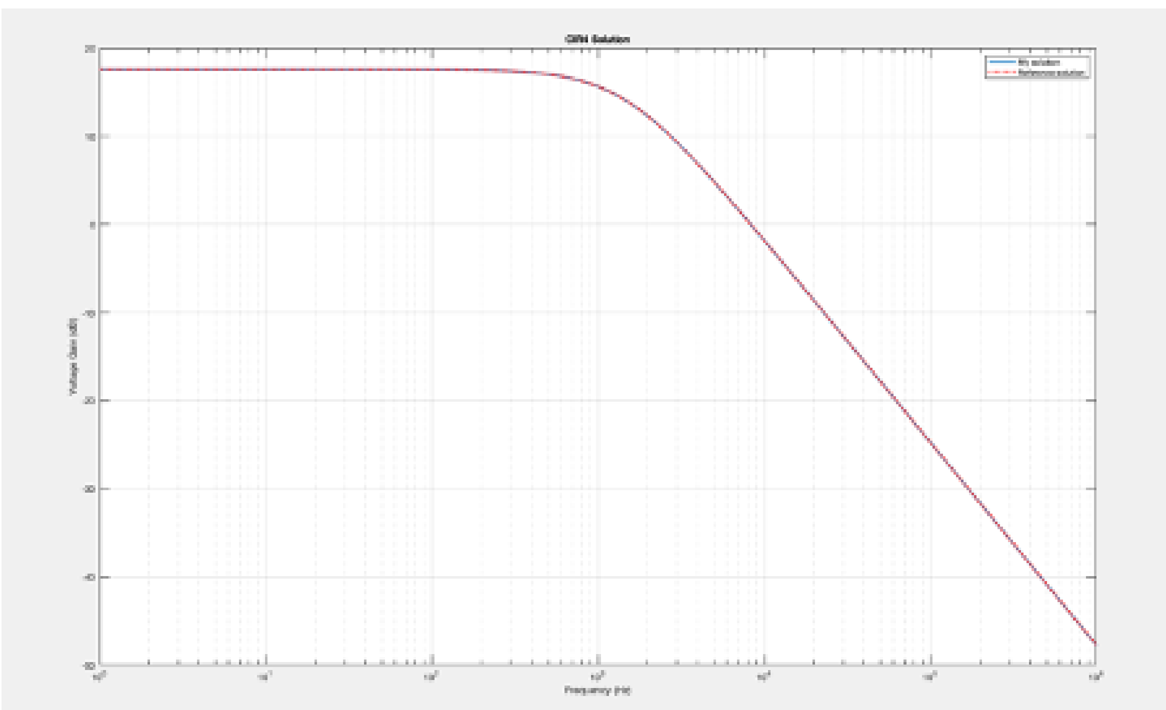
3-) LC Filter 3



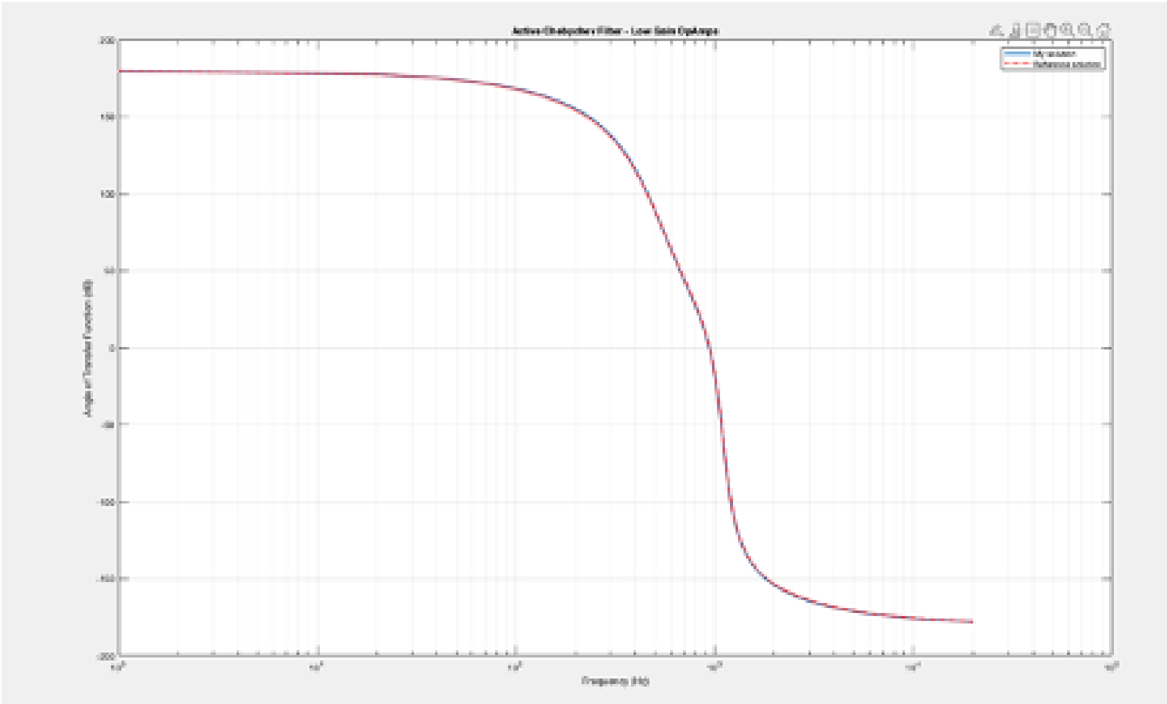


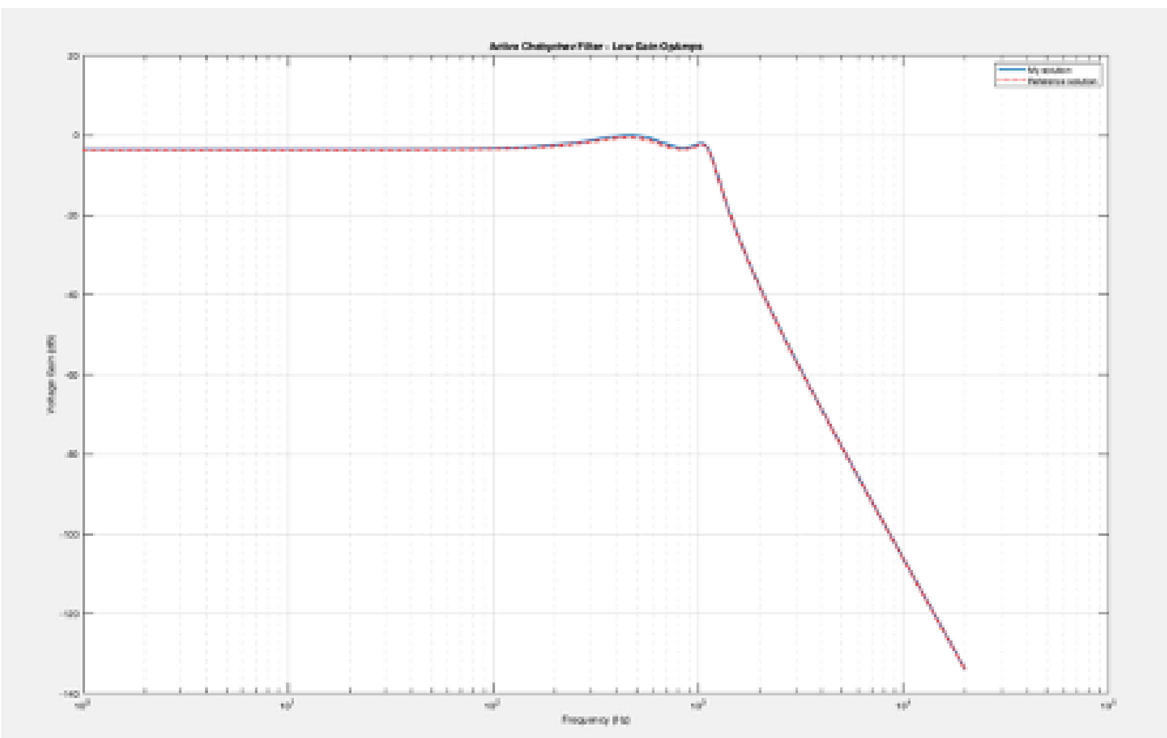
4-) CIR 4



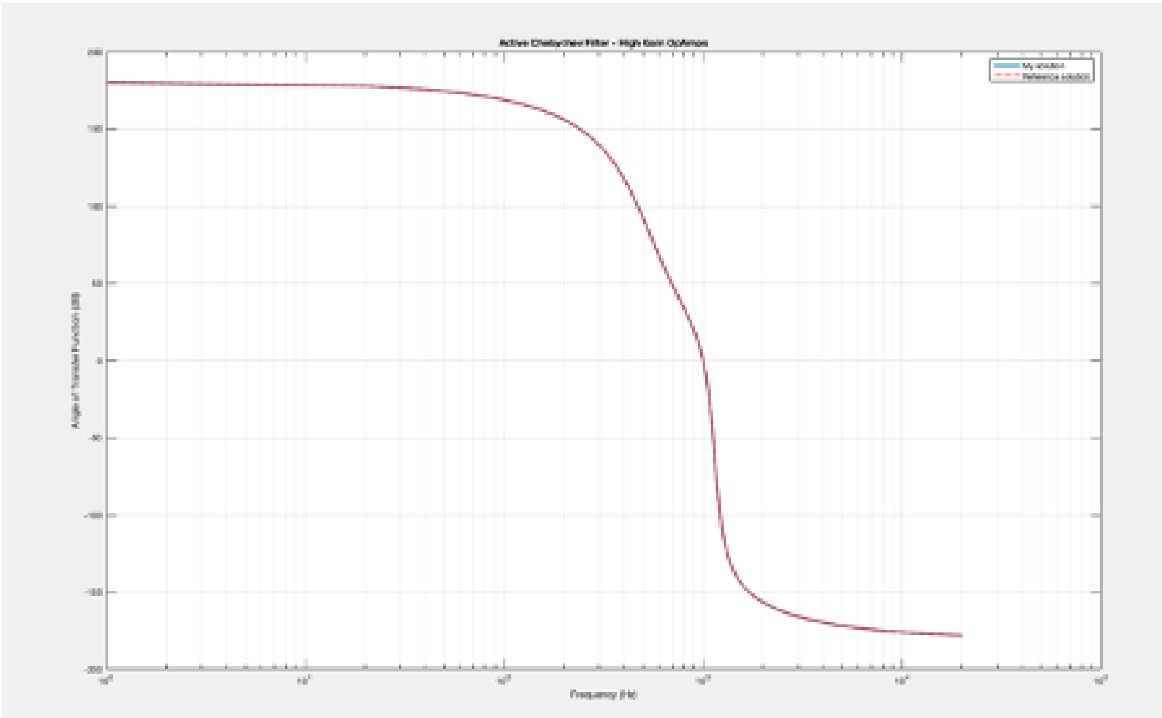


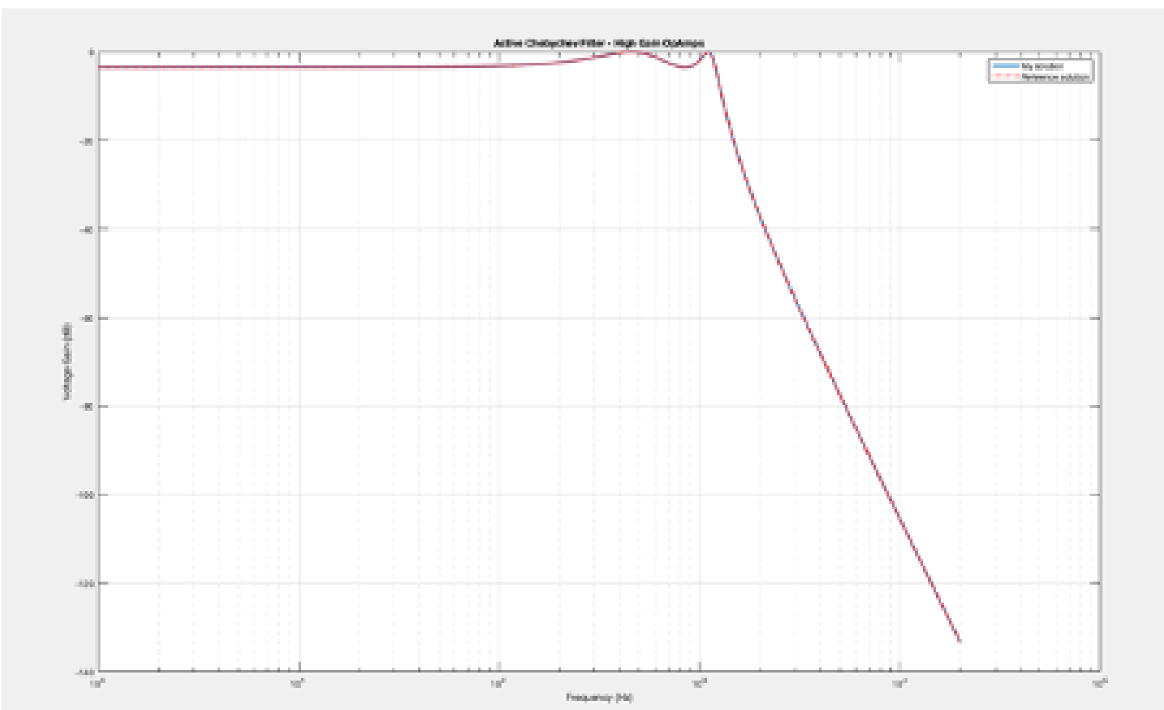
5-) Active Chebychev Filter Low Gain OpAmp





6-) Active Chebychev Filter High Gain OpAmp





b-) matlab code for the functions you have written

See Appendix B.

Appendix A

1. nlJacobian.m

```
function J = nlJacobian(X)
% Compute the jacobian of the nonlinear vector of the MNA equations as a
% function of X
% input: X is the current value of the unknown vector.
% output: J is the jacobian of the nonlinear vector f(X) in the MNA
% equations. The size of J should be the same as the size of G.

global G DIODE_LIST


node_i = DIODE_LIST.node1;
node_j = DIODE_LIST.node2;
Is = DIODE_LIST.Is;
Vt = DIODE_LIST.Vt;

Vi = X(node_i);
Vj = X(node_j);

% size of matrix
sz = size (G, 1)

% contribution to jacobian
A = zeros(sz, sz);

% ii
A(node_i, node_i) = (Is / Vt) * exp((Vi - Vj) / Vt);
% ij
A(node_i, node_j) = -(Is / Vt) * exp((Vi - Vj) / Vt);
% ji
A(node_j, node_i) = -(Is / Vt) * exp((Vi - Vj) / Vt);
% jj
A(node_j, node_j) = (Is / Vt) * exp((Vi - Vj) / Vt);

J = G + A;

end
```

2. dcsolve.m

```
function [Xdc dX] = dcsolve(Xguess,maxerr)
% Compute dc solution using newtwon iteration
% input: Xguess is the initial guess for the unknown vector.
%        It should be the correct size of the unknown vector.
%        maxerr is the maximum allowed error. Set your code to exit the
%        newton iteration once the norm of DeltaX is less than maxerr
% Output: Xdc is the correction solution
%         dX is a vector containing the 2 norm of DeltaX used in the
%         newton Iteration. the size of dX should be the same as the number
%         of Newton-Raphson iterations. See the help on the function 'norm'
%         in matlab.


global G C b DIODE_LIST

node_i = DIODE_LIST.node1;
node_j = DIODE_LIST.node2;
Is = DIODE_LIST.Is;
Vt = DIODE_LIST.Vt;


sz = size (G, 1);
```

```matlab
%Xdc is the vector of correction solution
Xdc = zeros(sz, 1);

% dX is a vector containing the 2 norm of DeltaX used in the
% newton Iteration. the size of dX should be the same as the number
% of Newton-Raphson iterations.
dX = zeros(sz, 1);

% f_x vector
f = zeros(sz, 1);

% boolean to end the while loop
error_tolerable = false;
iteration = 0;

while (~error_tolerable)
    iteration = iteration + 1;
    % f_x = Is*(exp(Vs/Vt) - 1) for each row
    % if both nodes not grounded
    if ((node_i ~= 0) && (node_j ~= 0))
        Vs = Xguess(node_i) - Xguess(node_j)
        f(node_i) = Is * (exp(Vs / Vt) - 1);
        f(node_j) = -Is * (exp(Vs / Vt) - 1);

    % node j grounded
    elseif ((node_i ~= 0) && (node_j ==0))
        Vs = Xguess(node_i)
        f(node_i) = Is * (exp(Vs / Vt) - 1);

    % node i grounded
    else
        Vs = -Xguess(node_j)
        f(node_j) = -Is * (exp(Vs / Vt) - 1);
    end

    % phi_x = G * x + f_x - Bdc
    phi = G * Xguess + f - b;

    % Jacobian vector phi'
    d_phi = nlJacobian(Xdc);

    %dX = - inv(phi') * phi;
    dX = [dX (- d_phi \ phi)];

    % X_new = X_old + dX
    Xguess = Xguess + dX(:, iteration + 1);

    % X_new = X_old + dX
    Xdc = Xdc + dX(:, iteration + 1);

    % update norms
    norms = zeros (iteration +1);
    for i=1: (iteration+1)
        norms(i) = norm(dX(:, i), 2);
    end

    % dX <= max error -> end loop.
    if abs(norm(dX(:, iteration + 1), 2)) <= maxerr
        error_tolerable = true;
    end
end

dX = norms;
end
```

Appendix B

1. ind.m

```
function ind(n1,n2,val)
          % ind(n1,n2,val)
          % Add stamp for inductor to the global circuit representation
          % Inductor connected between n1 and n2
          % The indjuctance is val in Henry
          % global G
          % global C
          % global b
          % Date: 02/10/2021

     % define global variables
     global G
     global b
     global C

     d = size(G,1);       %current size of the MNA
     xr = d+1;            %new row

     b(xr)=0;             %add new row
                                % Matlab automatically increases the size of a matrix if you use an index
                                % that is bigger than the current size.
     G(xr,xr)=0;        %add new row/column
     C(xr,xr)=0;        %add new row/column



     if (n1~=0)
        G(n1,xr) = 1;
        G(xr,n1) = 1;
     end

     if (n2~=0)
        G(n2,xr) = -1;
        G(xr,n2) = -1;
     end

     C(xr,xr) = -val;

end
```

2. vcvs.m

```
function vcvs(nd1,nd2,ni1,ni2,val)
% vcvs(nd1,nd2,ni1,ni2,val)
% Add stamp for a voltage controlled voltage source
% to the global circuit representation
% val is the gain of the vcvs
% ni1 and ni2 are the controlling voltage nodes
% nd1 and nd2 are the controlled voltage nodes
% The relation of the nodal voltages at nd1, nd2, ni1, ni2 is:
% Vnd1 - Vnd2 = val*(Vni1 - Vni2)


global G
global b
global C

sz = size(G,1);        %current size of the MNA
xr = sz + 1;             %new row
b(xr)=0;              %add new row
                                % Matlab automatically increases the size of a matrix if you use an index
                                % that is bigger than the current size.
G(xr,xr)=0;         %add new row/column
C(xr,xr)=0;         %add new row/column


if (nd1 ~= 0)
    G(nd1, xr) = 1;
    G(xr, nd1) = 1;
end


if (nd2 ~= 0)
    G(nd2, xr) =  -1;
    G(xr, nd2) = -1;
end

if (ni1 ~= 0)
    G(xr, ni1) = -val;
end

if ni2 ~= 0
    G(xr, ni2) = val;
end
end
```

3. vccs.m

```
function vccs(nd1,nd2,ni1,ni2,val)
% vccs(nd1,nd2,ni1,ni2,val)
% Add stamp for voltage controlled current source
% to the global circuit representation
% ni1 and ni2 are the controlling voltage nodes
% the controlled current source is between nd1 and nd2
% The controlled current (from nd1 to nd2) is val*(Vni1-Vni2)


global G

if (nd1 ~= 0)

    if (ni1 ~= 0)
        G(nd1, ni1) = G(nd1, ni1) + val;
    end

    if (ni2 ~= 0)
        G(nd1, ni2) = G(nd1, ni2) - val;
    end
end

if (nd2 ~= 0)

    if (ni1 ~= 0)
        G(nd2, ni1) = G(nd2, ni1) - val;
    end

    if (ni2 ~= 0)
        G(nd2, ni2) = G(nd2, ni2) + val;
    end
end
end
```

4. fsolve.m

```
function r = fsolve(fpoints, out)
%  fsolve(fpoints, out)
%  Obtain frequency domain response
% global variables G C b
% Inputs: fpoints is a vector containing the frequency points at which
%         to compute the response in Hz
%         out is the output node
% Outputs: r is a vector containing the value of
%            of the response at the points fpoint


% define global variables
global G C b

sz = size(fpoints, 2);
r = zeros(1, sz);

for i = 1:sz
    % (G + jwC) * X = b
    X = (G + 1j * 2 * pi * fpoints(i) * C) \ b;
    r(1, i) = X(out);
end

end
```