Lab 2

Part 1 (prelab):

Solve_circuit.m

```
%__Part 3__
matrices_size = nodes_number + numel(V_Names);

% Z matrix
unit_matrix = cell(matrices_size, 1);
for i = 1:1:numel(unit_matrix)
    unit_matrix{i} = ['0'];
end
z = unit_matrix;

% Stamping I sources
for I = 1:1:numel(I_Names)
    current_node_1 = str2double(I_Node_1(I));
    current_node_2 = str2double(I_Node_2(I));
    current_name = I_Names{I};
    if current_node_1 ~= 0
        z{current_node_1} = [z{current_node_1} '-' current_name];
    end
    if current_node_2 ~= 0
        z{current_node_2} = [z{current_node_2} '+' current_name];
    end
end

% Stamping V sources
for V = 1:1:numel(V_Names)
    z{nodes_number + V} = [V_Names{V}];
end
Z = cellfun(@str2sym, z);
```

```
% X matrix
x = cell(matrices_size, 1);
for node = 1:1:nodes_number
    x{node} = ['V_' num2str(node)];
end
for V = 1:1:numel(V_Names)
    x{nodes_number + V} = ['I_' V_Names{V}];
end
X = cellfun(@str2sym, x);


% A matrix - G part
G = repmat(unit_matrix(1:nodes_number), 1, nodes_number);
for R = 1:1:numel(R_Names)
    current_node_1 = str2double(R_Node_1(R));
    current_node_2 = str2double(R_Node_2(R));
    current_name = R_Names{R};
    if current_node_1 ~= 0
        G{current_node_1, current_node_1} = [G{current_node_1, current_node_1} '+1/' current_name];
    end
    if current_node_2 ~= 0
        G{current_node_2, current_node_2} = [G{current_node_2, current_node_2} '+1/' current_name];
    end
    if current_node_1 ~= 0 && current_node_2 ~= 0
        G{current_node_1, current_node_2} = [G{current_node_1, current_node_2} '-1/' current_name];
        G{current_node_2, current_node_1} = [G{current_node_2, current_node_1} '-1/' current_name];
    end
end
```

```
% B matrix
B = repmat(unit_matrix, 1, numel(V_Names));
for V = 1:1:numel(V_Names)
    current_node_1 = str2double(V_Node_1(V));
    current_node_2 = str2double(V_Node_2(V));
    if current_node_1 ~= 0
        B{current_node_1, V} = ['1'];
    end
    if current_node_2 ~= 0
        B{current_node_2, V} = ['-1'];
    end
end

% C matrix
C = B.';

% A matrix (final)
a = [G; C(:,1:nodes_number)];
a = [a B];
A = cellfun(@str2sym, a);


%__Part 4__
% Symbolic solution
symbolic_ans = A \ Z;
```

```
%__Part 4__
% Symbolic solution
symbolic_ans = A \ Z;

% Assign numerical values
for R = 1:1:numel(R_Names)
    eval([R_Names{R} ' = ' num2str(R_Values{R}) ';']);
end

for V = 1:1:numel(V_Names)
    eval([V_Names{V} ' = ' num2str(V_Values{V}) ';']);
end

for I = 1:1:numel(I_Names)
    eval([I_Names{I} ' = ' num2str(I_Values{I}) ';']);
end

% Substitute symbolic values with numerical
numeric_ans = subs(symbolic_ans);

% Print the results
for i = 1:1:numel(symbolic_ans)
    fprintf('%s = %f\n', char(X(i)), double(numeric_ans(i)));
end

end
```

Solve_Circuit_Example.m

C: > Users > DELL > Desktop > Siemens AMS 2025 > lab2 > C Solve_Circuit_Example.m

```
% cleaning the workspace, and cmd window
clear all;
clc;

% running the first SPICE netlist
fprintf('the first netlist:\n');
[sum,num]=Solve_Circuit('circuit_1.txt');



fprintf('the second netlist:\n');
% add a line here to run the second netlist
[sum,num]=Solve_Circuit('circuit_2.txt');
```

LTSpice output for circuit 1:

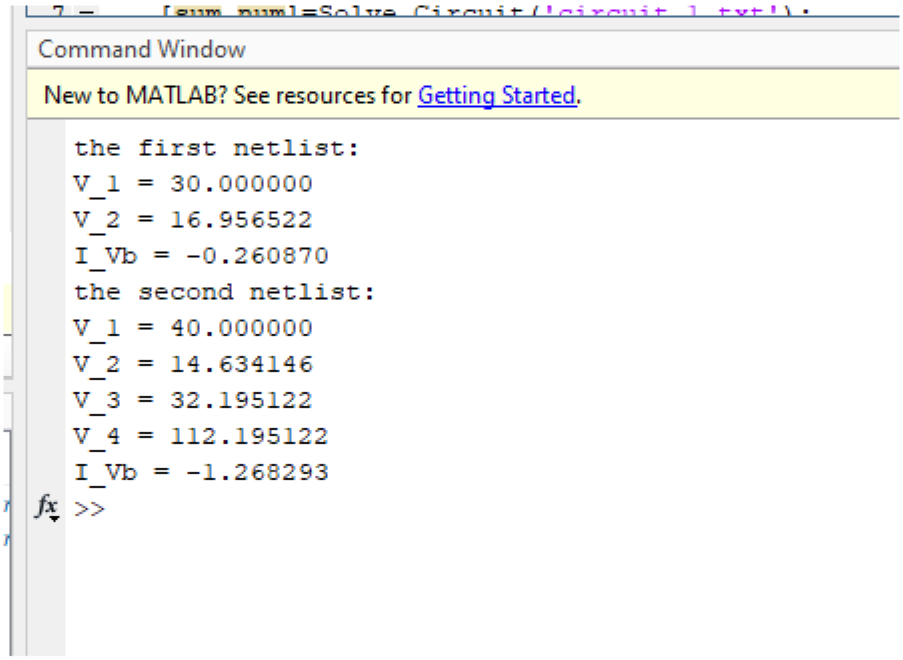
```
       --- Operating Point ---

V(2):           16.9565        voltage
V(1):           30             voltage
I(R3):          1.69565        device_current
I(Is):          2              device_current
I(R1):          0.26087        device_current
I(R2):          0.565217       device_current
I(Vb):          -0.26087       device_current
```

LTSpice output for circuit 2 :

C:\Users\DELL\Desktop\Siemens AMS 2023\lab2\circuit_2.cir

```
       --- Operating Point ---

V(2):           14.6341        voltage
V(4):           112.195        voltage
V(1):           40             voltage
V(3):           32.1951        voltage
I(R3):          -1             device_current
I(R6):          0.804878       device_current
I(Vb):          -1.26829       device_current
I(R1):          1.26829        device_current
I(R2):          -0.195122      device_current
I(R4):          1.46341        device_current
I(Is):          1              device_current
```

Matlab results :


Command Window

New to MATLAB? See resources for Getting Started.

```
the first netlist:
V_1 = 30.000000
V_2 = 16.956522
I_Vb = -0.260870
the second netlist:
V_1 = 40.000000
V_2 = 14.634146
V_3 = 32.195122
V_4 = 112.195122
I_Vb = -1.268293
>>
```

For circuit 1:

Result	My simulator	LTSpice
V_1	30 V	30 V
V_2	16.956522 V	16.9565 V
I(Vb)	-0.26087 A	-0.26087 A

For circuit 2:

Result	My simulator	LTSpice
V_1	40 V	40 V
V_2	14.634146 V	14.6341 V
V_3	32.195122 V	32.1951 V
V_4	122.195122	122.195 V
I(Vb)	-1.268293 A	-1.26829 A

We can see that both simulators give the same results with slight more numerical precision for our simulator .

Part 2:

Parameters

L = 1mH , C= 1uF , f= $\frac{1}{2*pi*\sqrt{(l*c)}}$ = 5KHz nearly

damping ratio = R/2 *$\sqrt{\frac{l}{c}}$

Underdamped ⇨ R<63.24

Critically damped ⇨ R=63.24

Overdamped ⇨ R>63.24

Modified code with adding L , C stamping and AC Analysis

Solve_AC_Circuit.m

```
function [symbolic_ans, numeric_ans ,frequencies] = Solve_AC_Circuit(netlist_directory, points_per_decade , start_freq , stop_freq)

%{
Part 1: reading the netlist
Part 2: parsing the netlist
Part 3: creating the matrices
Part 4: solving the matrices
%}

frequencies = logspace(log10(start_freq), log10(stop_freq), round(log10(stop_freq/start_freq) * points_per_decade) + 1);


%loading netlist
raw_netlist = fopen(netlist_directory);
raw_netlist = fscanf(raw_netlist, '%c');

%Deleting multiple spaces, etc. using regular expressions
netlist = regexprep(raw_netlist,' *',' ');
netlist = regexprep(netlist,' I','I');
netlist = regexprep(netlist,' R','R');
netlist = regexprep(netlist,' V','V');
netlist = regexprep(netlist,' C','C');
netlist = regexprep(netlist,' L','L');
netlist = regexp(netlist,'[^\n]*','match');

%__Part 2__
[R_Node_1, R_Node_2, R_Values, R_Names] = ParseNetlist(netlist, 'R');
[V_Node_1, V_Node_2, V_Values, V_Names] = ParseNetlist(netlist, 'V');
[I_Node_1, I_Node_2, I_Values, I_Names] = ParseNetlist(netlist, 'I');
[C_Node_1, C_Node_2, C_Values, C_Names] = ParseNetlist(netlist, 'C');
[L_Node_1, L_Node_2, L_Values, L_Names] = ParseNetlist(netlist, 'L');
```

```
%%%%%%%%%%%%%%%%add C L STAMP IN ALOOP WITH FREQUENCIES%%%%%%%%%%%%%%%%%
% A matrix - G part
for freq_x = 1:numel(frequencies)

            current_freq = frequencies(freq_x) ;
            current_omega = 2 * pi * current_freq ;
            G = repmat(unit_matrix(1:nodes_number), 1, nodes_number);
            %% R stamp %%
            for R = 1:1:numel(R_Names)
                current_node_1 = str2double(R_Node_1(R));
                current_node_2 = str2double(R_Node_2(R));
                current_name = R_Names{R};
                if current_node_1 ~= 0
                    G{current_node_1, current_node_1} = [G{current_node_1, current_node_1} '+1/' current_name];
                end
                if current_node_2 ~= 0
                    G{current_node_2, current_node_2} = [G{current_node_2, current_node_2} '+1/' current_name];
                end
                if current_node_1 ~= 0 && current_node_2 ~= 0
                    G{current_node_1, current_node_2} = [G{current_node_1, current_node_2} '-1/' current_name];
                    G{current_node_2, current_node_1} = [G{current_node_2, current_node_1} '-1/' current_name];
                end
            end


            for C = 1:numel(C_Names)
                current_node_1 = str2double(C_Node_1(C));
                current_node_2 = str2double(C_Node_2(C));
                current_name = C_Names{C};
                admittance = ['1i*' num2str(current_omega) '*' current_name];
```

```
            for C = 1:numel(C_Names)
                if current_node_1 ~= 0
                    G{current_node_1, current_node_1} = [G{current_node_1, current_node_1} '+' admittance];
                end
                if current_node_2 ~= 0
                    G{current_node_2, current_node_2} = [G{current_node_2, current_node_2} '+' admittance];
                end
                if current_node_1 ~= 0 && current_node_2 ~= 0
                    G{current_node_1, current_node_2} = [G{current_node_1, current_node_2} '-' admittance];
                    G{current_node_2, current_node_1} = [G{current_node_2, current_node_1} '-' admittance];
                end
            end


            for L = 1:numel(L_Names)
                current_node_1 = str2double(L_Node_1(L));
                current_node_2 = str2double(L_Node_2(L));
                current_name = L_Names{L};
                admittance = ['1/(1i*' num2str(current_omega) '*' current_name ')'];
                if current_node_1 ~= 0
                    G{current_node_1, current_node_1} = [G{current_node_1, current_node_1} '+' admittance];
                end
                if current_node_2 ~= 0
                    G{current_node_2, current_node_2} = [G{current_node_2, current_node_2} '+' admittance];
                end
                if current_node_1 ~= 0 && current_node_2 ~= 0
                    G{current_node_1, current_node_2} = [G{current_node_1, current_node_2} '-' admittance];
                    G{current_node_2, current_node_1} = [G{current_node_2, current_node_1} '-' admittance];
                end
            end
```

```
            % B matrix
            B = repmat(unit_matrix, 1, numel(V_Names));
            for V = 1:1:numel(V_Names)
                current_node_1 = str2double(V_Node_1(V));
                current_node_2 = str2double(V_Node_2(V));
                if current_node_1 ~= 0
                    B{current_node_1, V} = ['1'];
                end
                if current_node_2 ~= 0
                    B{current_node_2, V} = ['-1'];
                end
            end

            % C matrix
            C = B.';

             %% Rebuild full A matrix
                a = [G; C(:,1:nodes_number)];
                a = [a B];
                A = cellfun(@str2sym, a);

                %% Solve at this frequency
                current_numeric = subs(A \ Z);
                numeric_ans(freq_x, :) = double(current_numeric);

end
```

Solve_AC_Circuit_Example

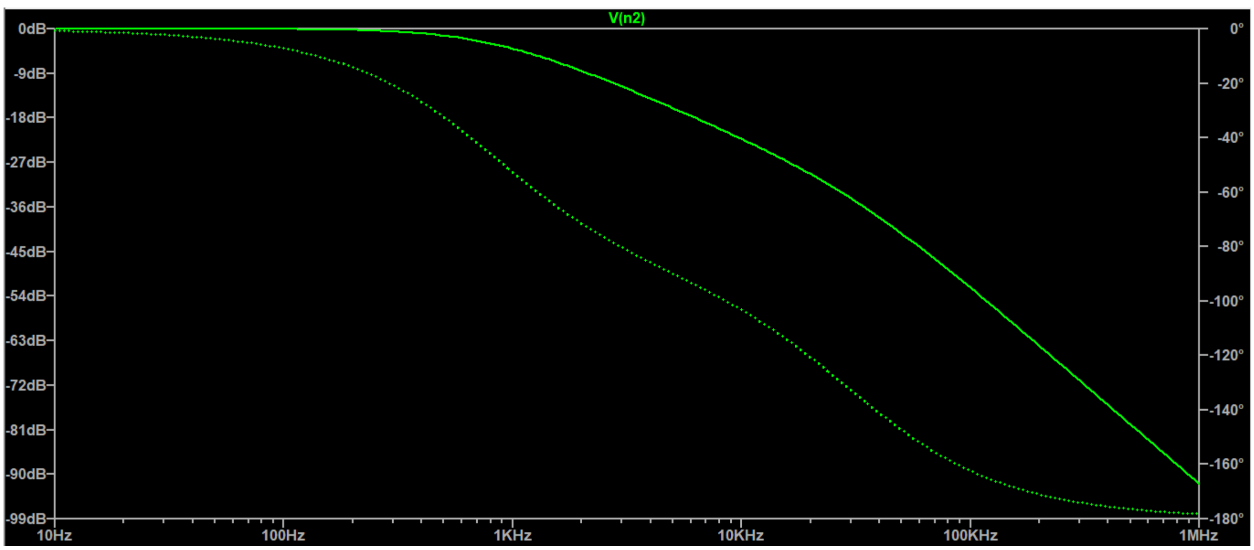
```
Solve_AC_Circuit_Example.m
% Cleaning the workspace and command window
clear all;
clc;

% Frequency sweep parameters
start_freq = 10;          % Hz
stop_freq = 1e6;          % 1 MHz
points_per_decade = 100;

% Run first SPICE netlist
fprintf('The first netlist:\n');
[sym1, num1, freq1] = Solve_AC_Circuit('RLC_OD.txt', points_per_decade, start_freq, stop_freq);

% Choose which variable to plot (e.g., output node V_2)
% You may change the index based on your node of interest
output_index = 3;

% Plot magnitude and phase for circuit 1
figure;
subplot(2,1,1);
semilogx(freq1, 20*log10(abs(num1(:, output_index))), 'b', 'LineWidth', 1.5);
xlabel('Frequency (Hz)');
ylabel('|V_{out}| (dB)');
title('Circuit 1: Magnitude Response');
grid on;

subplot(2,1,2);
semilogx(freq1, angle(num1(:, output_index)) * 180/pi, 'r', 'LineWidth', 1.5);
xlabel('Frequency (Hz)');
ylabel('Phase (degrees)');
title('Circuit 1: Phase Response');
grid on;
```

RLC Overdamped :

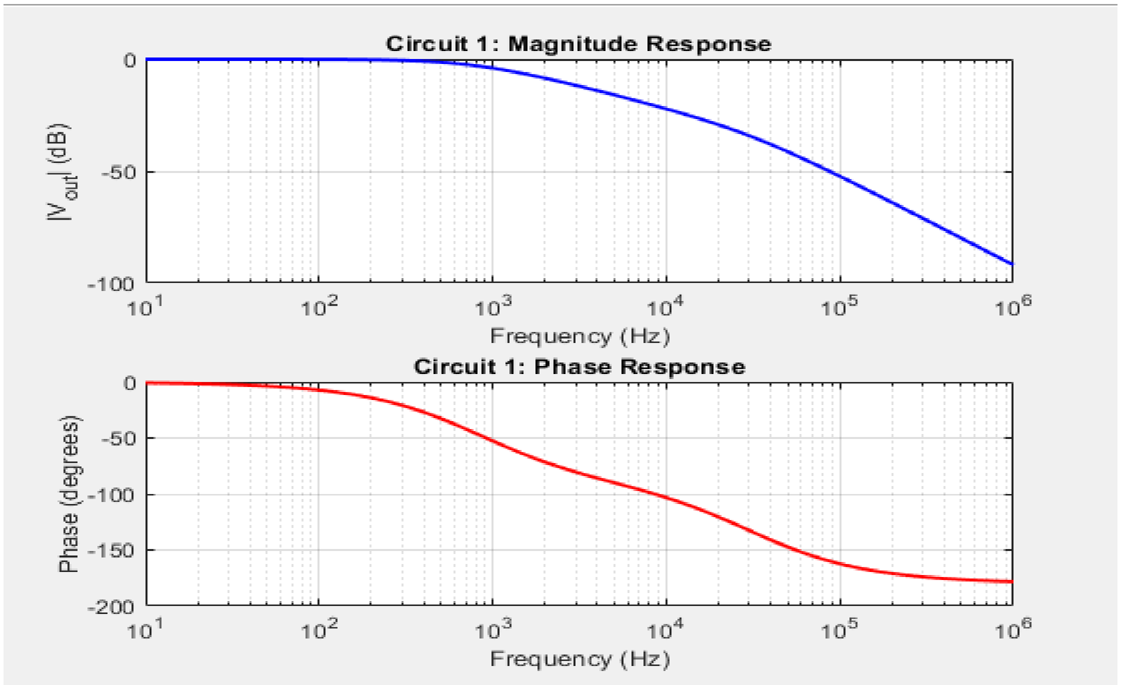
RLC_OD.txt

```
* RLC Low-Pass Filter - Overdamped
Vin in 0 AC 1
R1 in n1 200 ; over damped
L1 n1 n2 1m
C1 n2 0 1u
.ac dec 100 10 1Meg
.plot ac mag(V(n2)) phase(V(n2))
.end
```

LTSPICE



Matlab results:

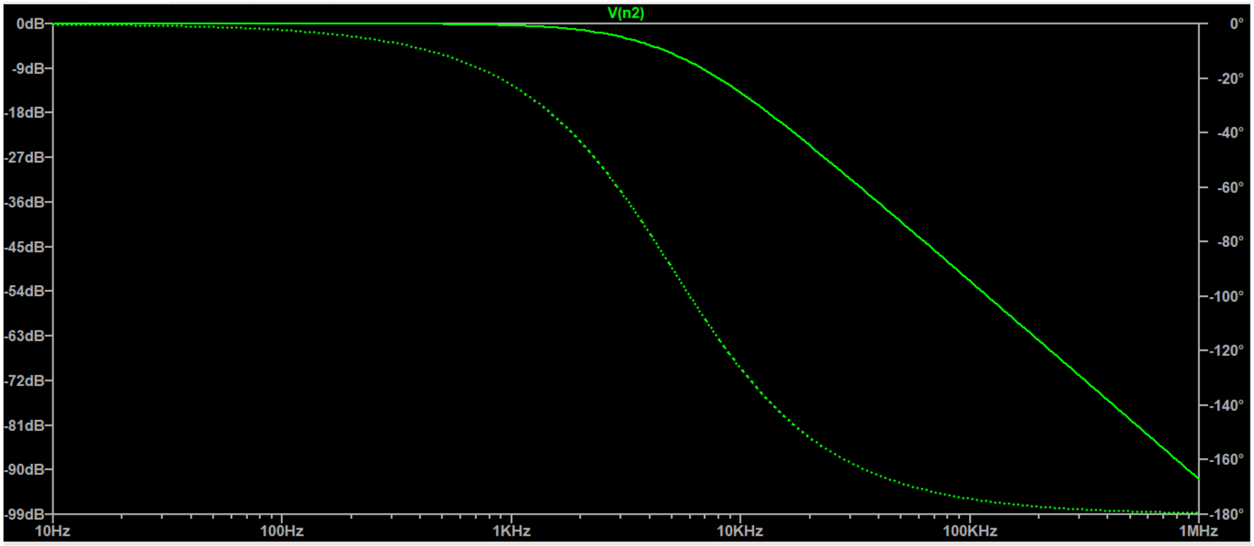


RLC Critically Damped :

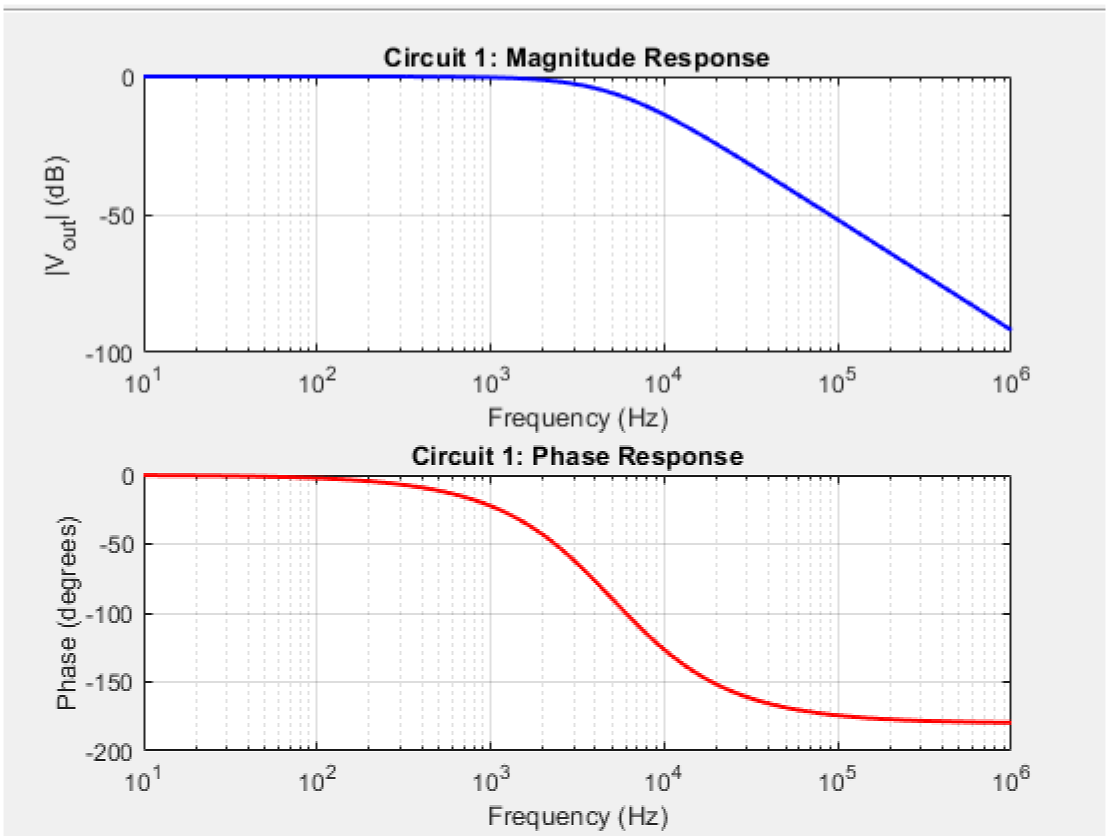
RLC_CD.txt

```
* RLC Low-Pass Filter - criticallydamped
Vin in 0 AC 1
R1 in n1 63.24 ; critically damped
L1 n1 n2 1m
C1 n2 0 1u
.ac dec 100 10 1Meg
.plot ac mag(V(n2)) phase(V(n2))
.end
```

LTSpice



Matlab results :

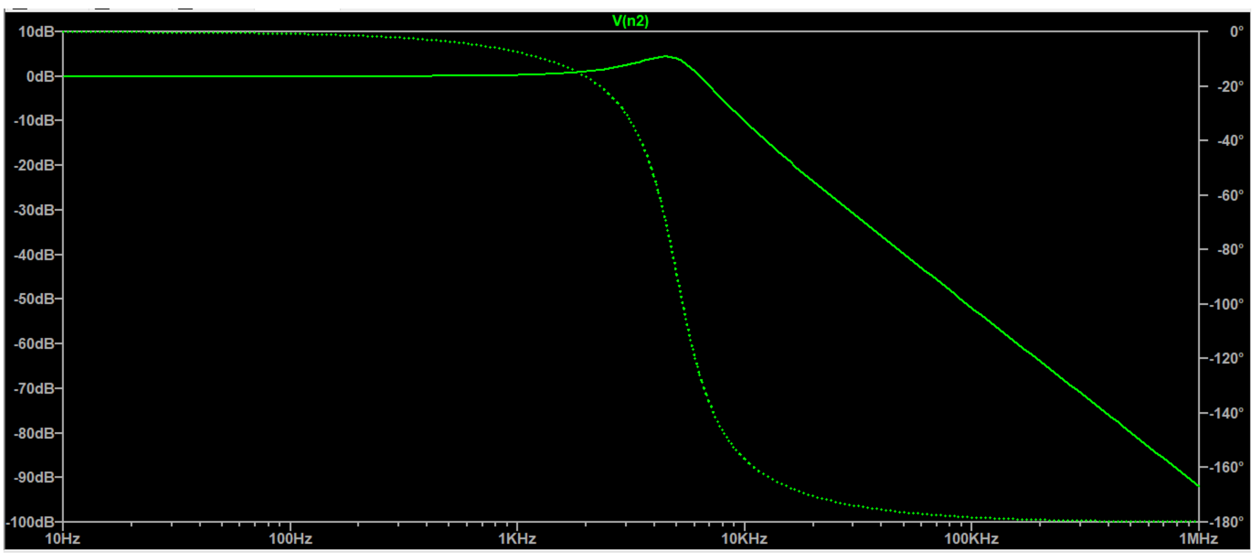


RLC Underdamped :

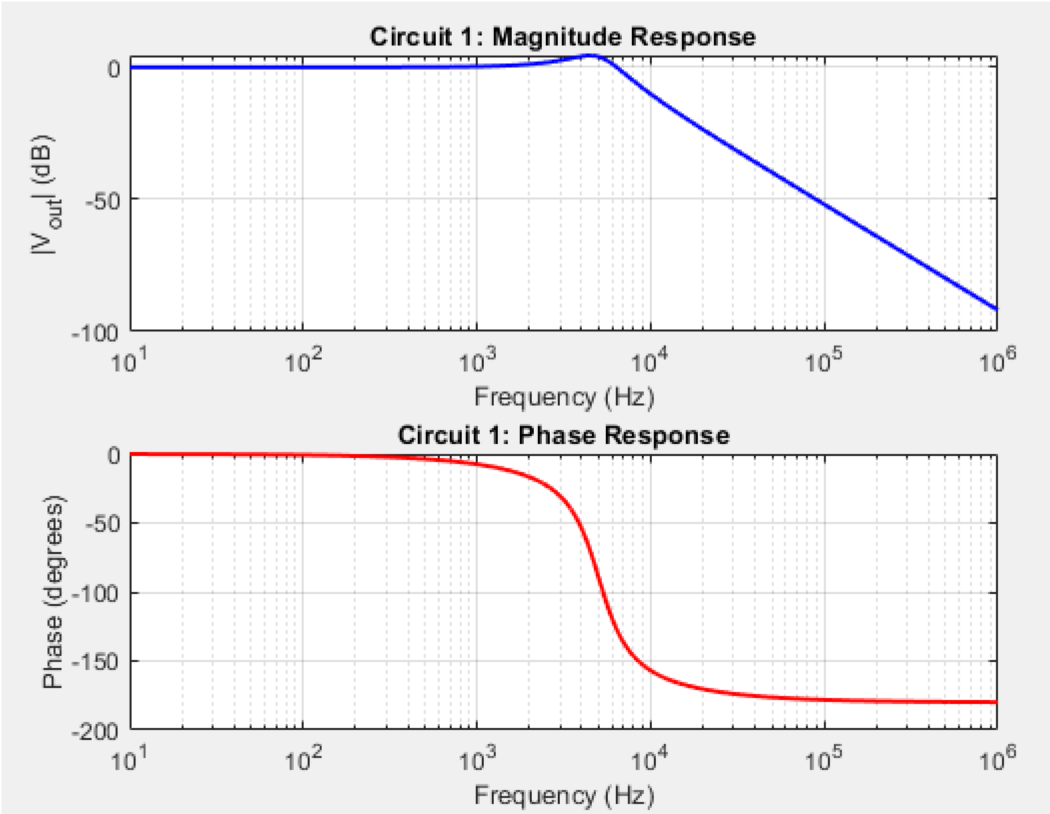
RLC_UD.txt

```
* RLC Low-Pass Filter - Underdamped
Vin in 0 AC 1
R1 in n1 20    ; under damped
L1 n1 n2 1m
C1 n2 0 1u
.ac dec 100 10 1Meg
.plot ac mag(V(n2)) phase(V(n2))
.end
```

LTSpice



Matlab results :



Comment :

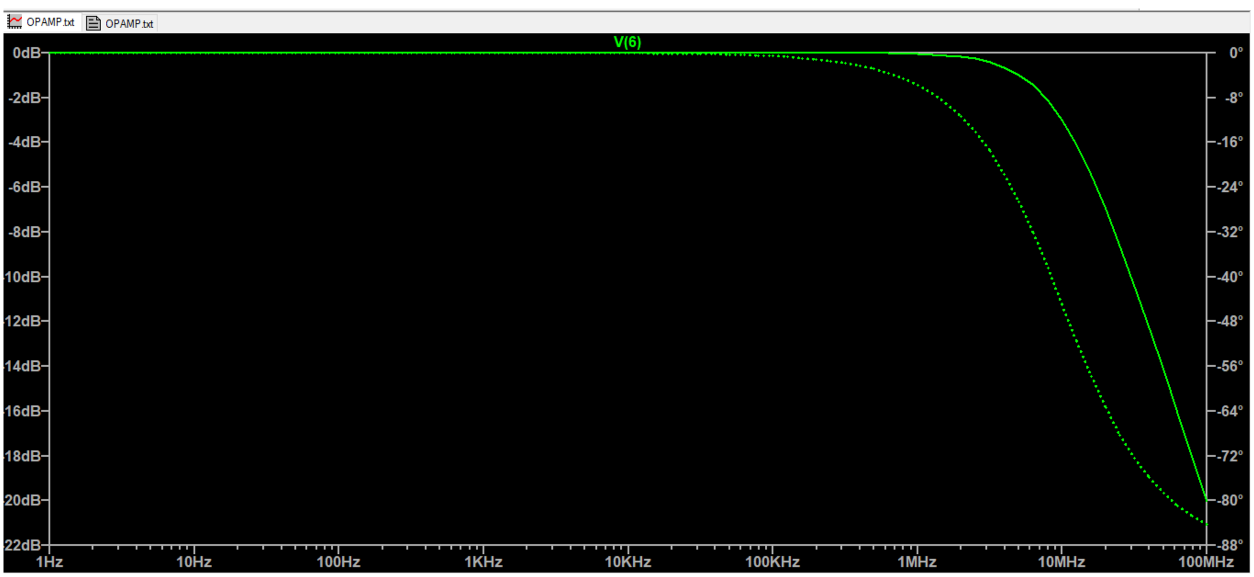
We can see how similar our simulator's results to those results from LTSpice and that confirms the correct implementation of AC analysis done by our matlab code after adding support for capacitors and inductors

Part 3 (OPAMP) :

Netlist :

```
*** opamp subcircuit ***
.subckt opamp 1 2 3
* VCVS with gain 10000
Eopamp 1 0 4 0 1
Gp 0 4 2 3 0.6289                  ; A0 = Gp * Rp
* Redundant current sources to avoid errors
Iopen1 2 0 0A
Iopen2 3 0 0A
* Dominant pole model
Rp 4 0 15.9k
Cp 4 0 10n
.ends opamp

*** Circuit Setup ***
* AC source: signal applied to non-inverting input
Vsig 5 0 AC 1

* Unity gain feedback: connect OUT to inverting input
XOP 6 5 6 opamp

*** AC Analysis ***
.AC DEC 10 1 100Meg                 ; Sweep from 1 Hz to 100 MHz

.END
```

LTSpice :



Parse_netlist.m updates:

```
ParseNetlist.m
function [Node_1 Node_2 Node_3 Node_4 Values Names] = ParseNetlist(netlist, instances_key)

%{
Part 1: We loop on the netlist lines to search for the given instances' key
For each hit (instance found) we save the fist, and second nodes, value,
and name
Part 2: Evaluating prefixes
%}

Node_1 = {};
Node_2 = {};
Node_3 = {};
Node_4 = {};
Values = {};
Names  = {};
```

```
    if line(1) == upper(instances_key)
        %Split the line at spaces
        splitted_line = strsplit(line);
        %Remove the empty cells due to strsplit function
        splitted_line = splitted_line(~cellfun('isempty',splitted_line));
        %Splitted_line = 'Name' 'Node_1' 'Node_2' 'Value'
        %Append each cell to its vector
        if length(splitted_line) > 4
        Node_1 = [Node_1 splitted_line(2)];
        Node_2 = [Node_2 splitted_line(3)];
        Node_3 = [Node_3 splitted_line(4)];
        Node_4 = [Node_4 splitted_line(5)];
        Values = [Values splitted_line(6)];
        Names = [Names splitted_line(1)];
        else
        Node_1 = [Node_1 splitted_line(2)];
        Node_2 = [Node_2 splitted_line(3)];
        Node_3 = [Node_3 ""];
        Node_4 = [Node_4 ""];
        Values = [Values splitted_line(4)];
        Names = [Names splitted_line(1)];
        end
    end
end
```

Code updates to add support for VCCS and VCVS :

```
% Deletion of multiple spaces, etc. using regular expressions
netlist = regexprep(raw_netlist,' *',' ');
netlist = regexprep(netlist,' I','I');
netlist = regexprep(netlist,' R','R');
netlist = regexprep(netlist,' V','V');
netlist = regexprep(netlist,' C','C');
netlist = regexprep(netlist,' L','L');
netlist = regexprep(netlist,' E','E');
netlist = regexprep(netlist,' G','G');
netlist = regexp(netlist,'[^\n]*','match');

%__Part 2: Netlist Parsing__
[R_Node_1, R_Node_2, ~, ~, R_Values, R_Names] = ParseNetlist(netlist, 'R');
[V_Node_1, V_Node_2, ~, ~, V_Values, V_Names] = ParseNetlist(netlist, 'V');
[I_Node_1, I_Node_2, ~, ~, I_Values, I_Names] = ParseNetlist(netlist, 'I');
[C_Node_1, C_Node_2, ~, ~, C_Values, C_Names] = ParseNetlist(netlist, 'C');
[L_Node_1, L_Node_2, ~, ~, L_Values, L_Names] = ParseNetlist(netlist, 'L');
[E_Node_1, E_Node_2, E_Node_3, E_Node_4, E_Values, E_Names] = ParseNetlist(netlist, 'E');
[G_Node_1, G_Node_2, G_Node_3, G_Node_4, G_Values, G_Names] = ParseNetlist(netlist, 'G');

% Counting nodes
nodes_list = [R_Node_1 R_Node_2 V_Node_1 V_Node_2 I_Node_1 I_Node_2 ...
              C_Node_1 C_Node_2 L_Node_1 L_Node_2 E_Node_1 E_Node_2 E_Node_3 E_Node_4 ...
              G_Node_1 G_Node_2 G_Node_3 G_Node_4];
valid_nodes = ~cellfun('isempty', nodes_list);
nodes_number = max(str2double(nodes_list(valid_nodes)));
```

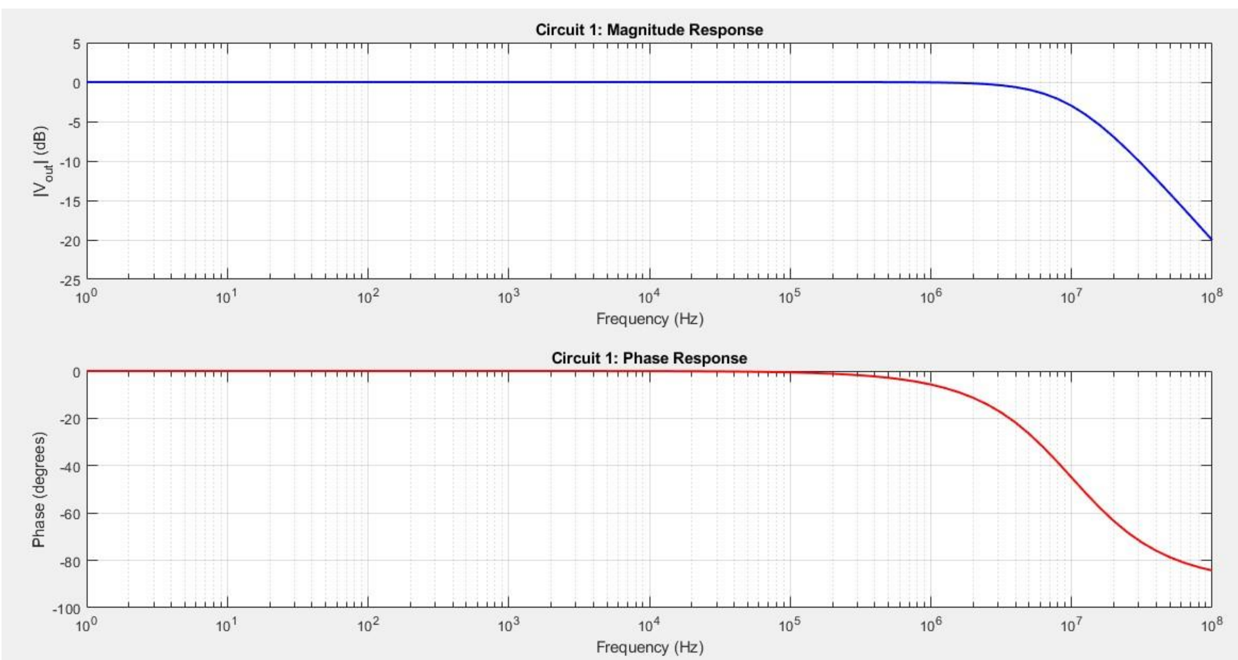
```
% Stamp current sources
for I = 1:numel(I_Names)
    current_node_1 = str2double(I_Node_1{I});
    current_node_2 = str2double(I_Node_2{I});
    current_name = I_Names{I};
    if current_node_1 ~= 0
    z{current_node_1} = [z{current_node_1} '+' current_name];
    end
    if current_node_2 ~= 0
        z{current_node_2} = [z{current_node_2} '-' current_name];
    end
end

% Stamp voltage sources
for V = 1:numel(V_Names)
    z{nodes_number + V} = V_Names{V};
end

% Stamp VCVS (E sources)
for E = 1:numel(E_Names)
    z{nodes_number + numel(V_Names) + E} = '0';
end
```

```matlab
    % Stamp VCCS (G sources)
for Gs = 1:numel(G_Names)
    out_node1 = str2double(G_Node_1{Gs});
    out_node2 = str2double(G_Node_2{Gs});
    in_node1 = str2double(G_Node_3{Gs});
    in_node2 = str2double(G_Node_4{Gs});
    value = sym(G_Names{Gs}); % This is gm

    % Current flows from out_node1 to out_node2, controlled by (V_in_node1 - V_in_node2)
    % KCL at out_node1: -gm*(Vin1 - Vin2) ...
    if out_node1 ~= 0
        if in_node1 ~= 0
            G(out_node1, in_node1) = G(out_node1, in_node1) - value; % -gm
        end
        if in_node2 ~= 0
            G(out_node1, in_node2) = G(out_node1, in_node2) + value; % +gm
        end
    end

    % KCL at out_node2: +gm*(Vin1 - Vin2) ...
    if out_node2 ~= 0
        if in_node1 ~= 0
            G(out_node2, in_node1) = G(out_node2, in_node1) + value; % +gm
        end
        if in_node2 ~= 0
            G(out_node2, in_node2) = G(out_node2, in_node2) - value; % -gm
        end
    end
end
```

```matlab
% B matrix (voltage sources)
B = sym(zeros(nodes_number, numel(V_Names) + numel(E_Names)));

% Stamp independent voltage sources
for V = 1:numel(V_Names)
    node1 = str2double(V_Node_1{V});
    node2 = str2double(V_Node_2{V});
    if node1 ~= 0
        B(node1, V) = 1;
    end
    if node2 ~= 0
        B(node2, V) = -1;
    end
end

% Stamp VCVS (E sources) in B matrix
for E = 1:numel(E_Names)
    out_node1 = str2double(E_Node_1{E});
    out_node2 = str2double(E_Node_2{E});
    col = numel(V_Names) + E;

    if out_node1 ~= 0
        B(out_node1, col) = 1;
    end
    if out_node2 ~= 0
        B(out_node2, col) = -1;
    end
end
```

```
    % Stamp VCVS (E sources) in C matrix
for E = 1:numel(E_Names)
    out_node1 = str2double(E_Node_1{E}); % Added from your B matrix stamping
    out_node2 = str2double(E_Node_2{E}); % Added from your B matrix stamping
    in_node1 = str2double(E_Node_3{E});
    in_node2 = str2double(E_Node_4{E});
    gain = sym(E_Names{E});
    row = numel(V_Names) + E;

    % Constraint equation: V_out1 - V_out2 - gain*(V_in1 - V_in2) = 0
    if out_node1 ~= 0
        C(row, out_node1) = 1; % Coefficient for V_out1
    end
    if out_node2 ~= 0
        C(row, out_node2) = -1; % Coefficient for V_out2
    end
    if in_node1 ~= 0
        C(row, in_node1) = C(row, in_node1) + gain; % Coefficient for V_in1 (from -gain*V_in1)
    end
    if in_node2 ~= 0
        C(row, in_node2) = C(row, in_node2) - gain; % Coefficient for V_in2 (from +gain*V_in2)
    end
end
```

Matlab results :



Comment :

We can see how similar our simulator’s results to those results from LTSpice and that confirms the correct implementation of AC analysis done by our matlab code after adding support for VCCS and VCVS to represent the behavioral model of Opamp .