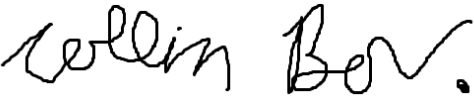



ECEN 3714-----Network Analysis

Spring 2023

Lab Final

Final Report (Pre-lab + Post-lab)

Name of the group members	
Name (print): Collin Bovenshchen	Name (print): Jacob Erwin
Signature: 	Signature: 

1. Introduction

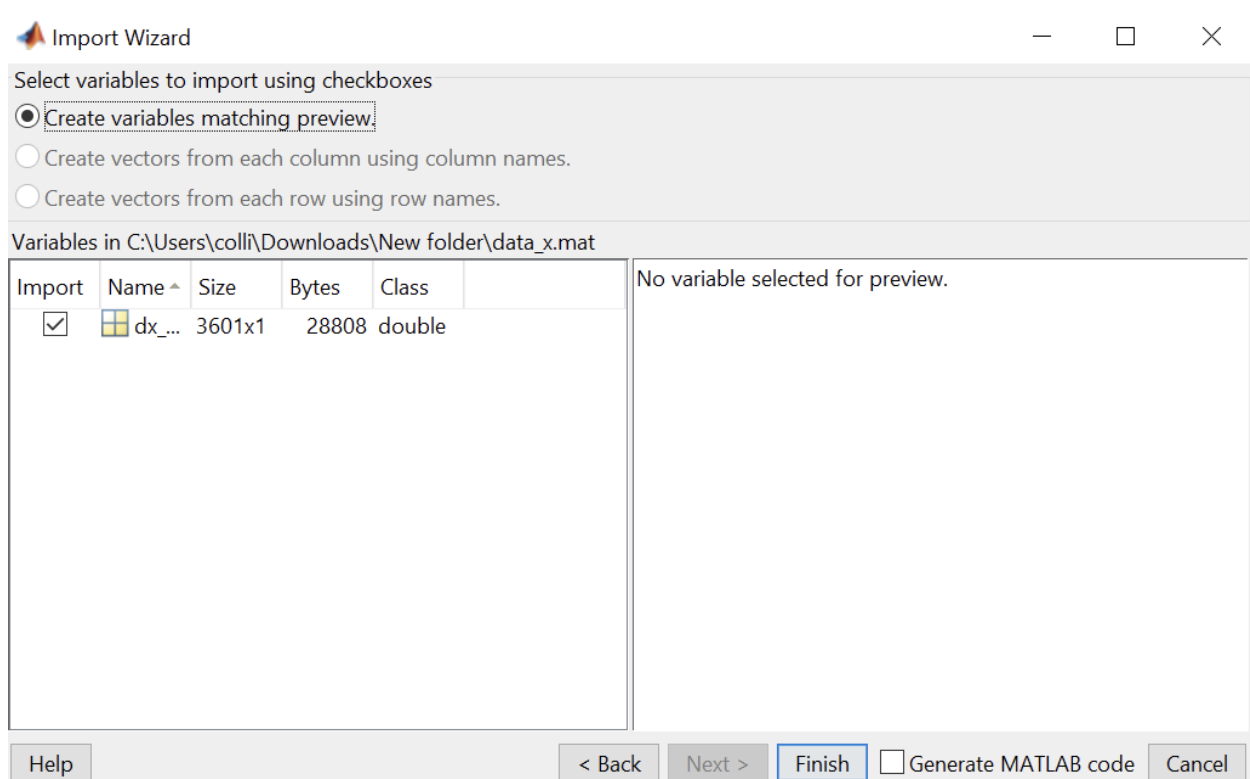
To procure the needed objectives from this lab, many prerequisites are needed. This includes:

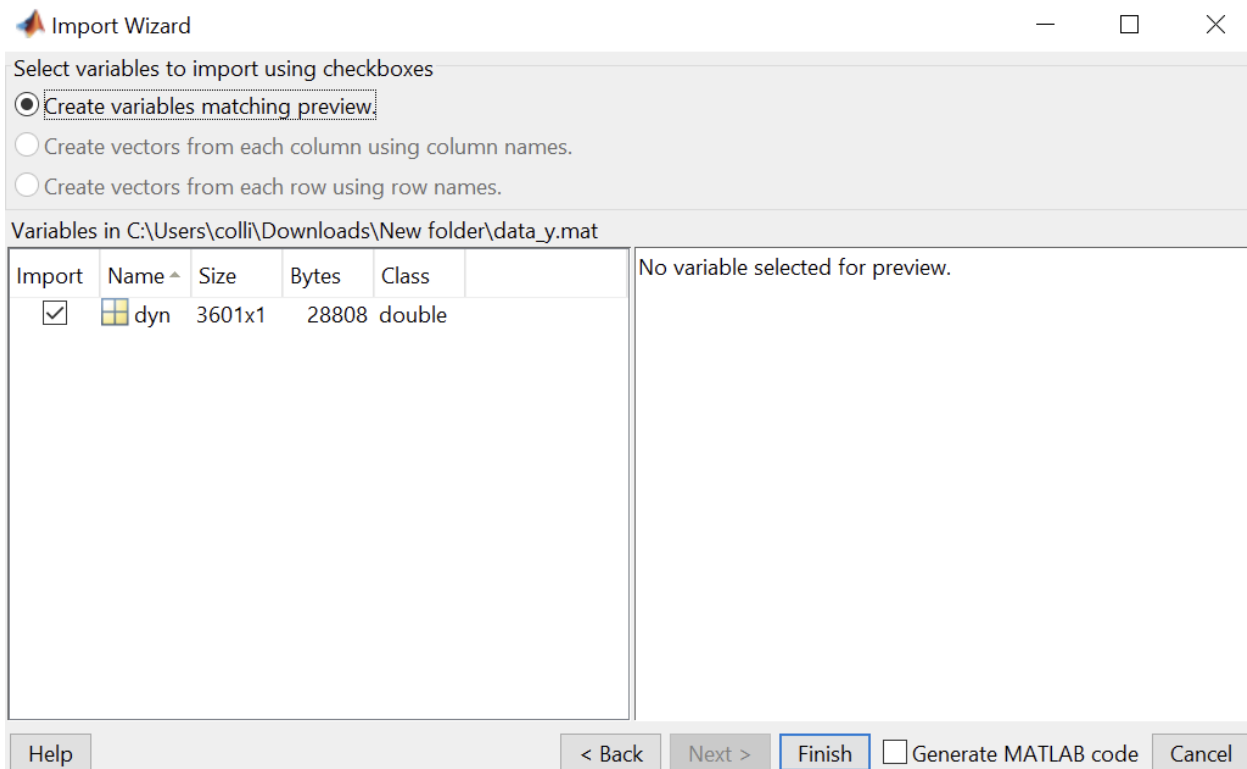
- A Transfer Function of the Raw Data
- Gain of the Transfer Function
- Corner Frequencies of the Transfer Function
- Circuit containing Low Pass, High Pass, and Gain Control filter



These can all be found within each part of the Lab, using various circuits and mathematical principles.

2. Loading Data Files

Double clicking on the uncompressed files present in the rar file allows for MATLAB to recognise the data present in these files, and through the Import Wizard, can bring this data into MATLAB.



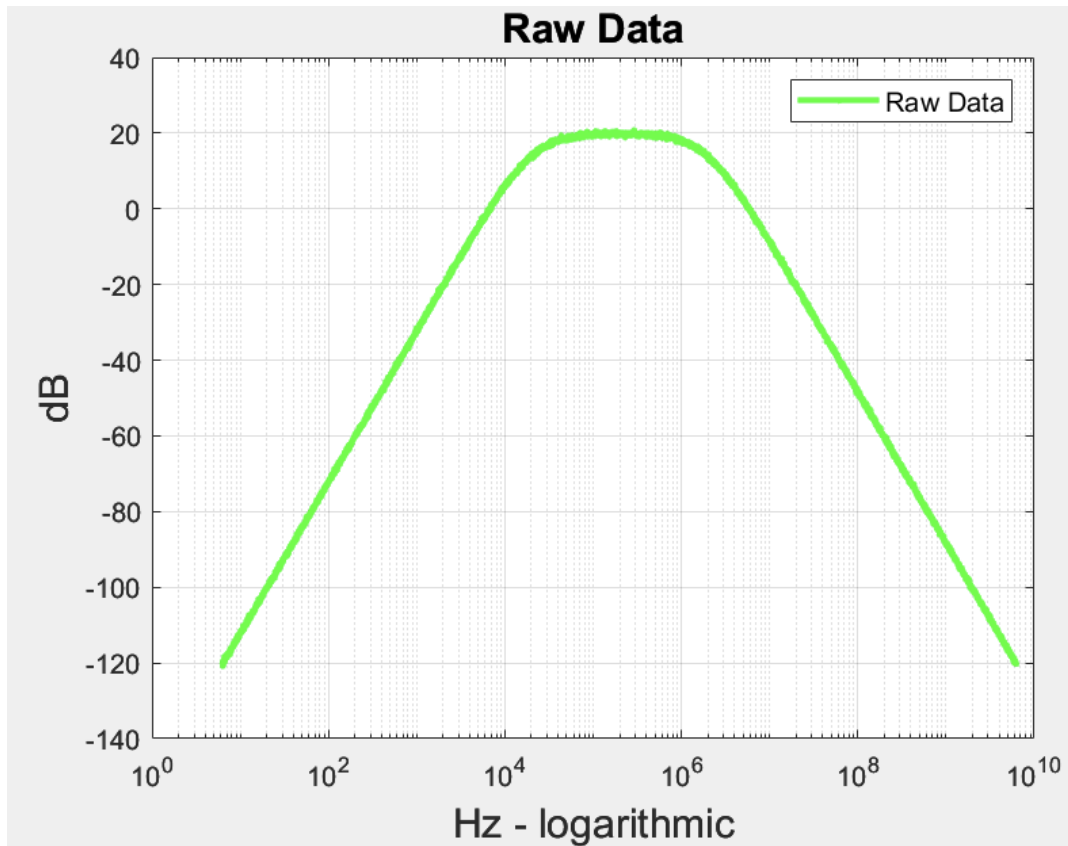


Name ▲	Value
 dx_omega	3601x1 double
 dyn	3601x1 double

Another method to import the data is using the `importdata()` function in MATLAB. This allows the data to be imported through the program, without having to run the previous import wizard every time MATLAB is started.

3. Plotting Raw Data

To plot the raw data in a way beneficial to determining the Transfer function, a few changes must be made to the data. One of which is displaying the data on the x axis to a logarithmic scale, as this is more accurate to a true transfer function in the $j\omega$ domain. The output data on the y axis must be converted to decimals as well. This can all be done through different MATLAB functions, such as `db()` and `semilogx()`. The following graphic demonstrates how the Raw Data is plotted in a graphical format, with the horizontal axis being in a logarithmic decade.



The previous graph was created through the following MATLAB code:

```
clc; close all; clearvars
% Instantiation of external variables
dx_omega = importdata("data_x.mat");
dyn = importdata("data_y.mat");
dbout = db(dyn);

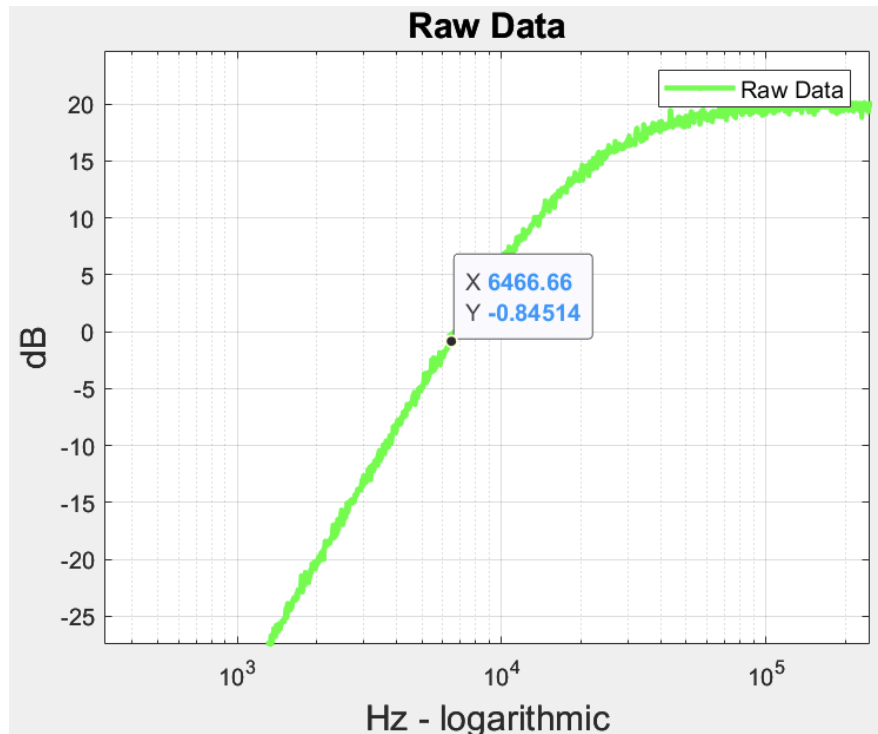
% Output Data to Graph
semilogx(dx_omega,dbout,'- .','LineWidth',2, 'Color', 'Green'), grid on; hold on;

% Format Graph
xlabel('Hz - logarithmic', 'fontsize', 15);
ylabel('dB', 'fontsize', 15);
title('Raw Data', 'fontsize', 15);
legend('Raw Data', 'fontsize', 10);
```

4. The Transfer Function Estimation

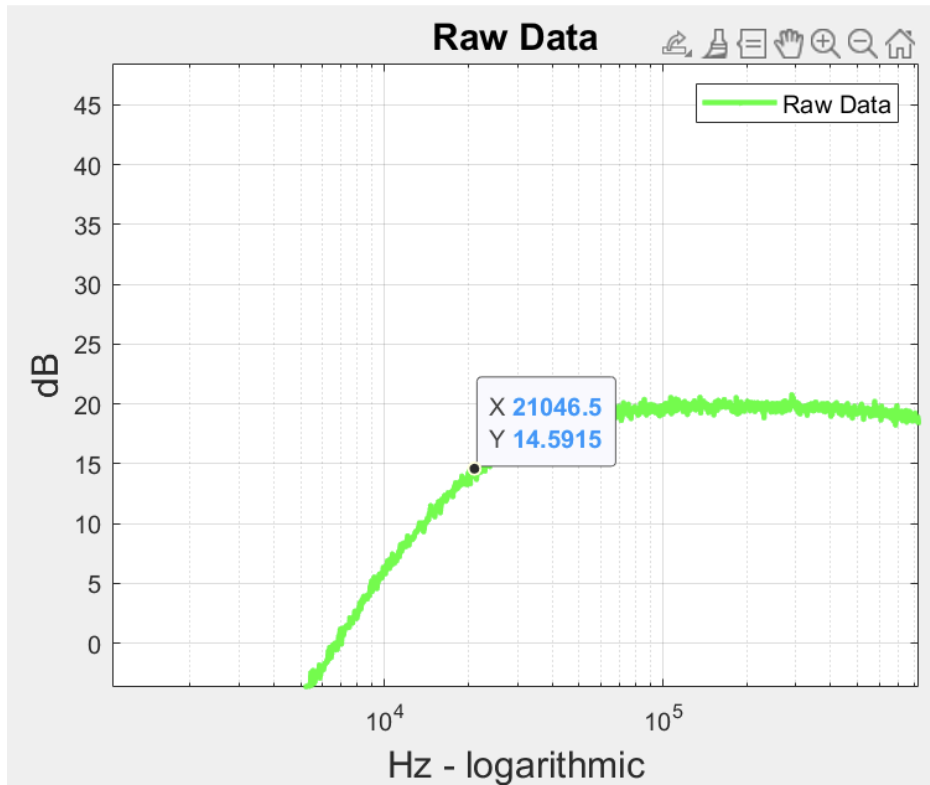
To estimate a transfer function from the raw data, some properties must be known from the data. First, the first point of intersection at 0 on the x axis must be found. The point on the y axis (about 6500) marks the constant that will modify the first part of the transfer function, being a straight line in the positive direction. This first line has a slope of 40 decibels per decade, so the function must be squared to match this.

$$\left(j\frac{\omega}{6500}\right)^2$$



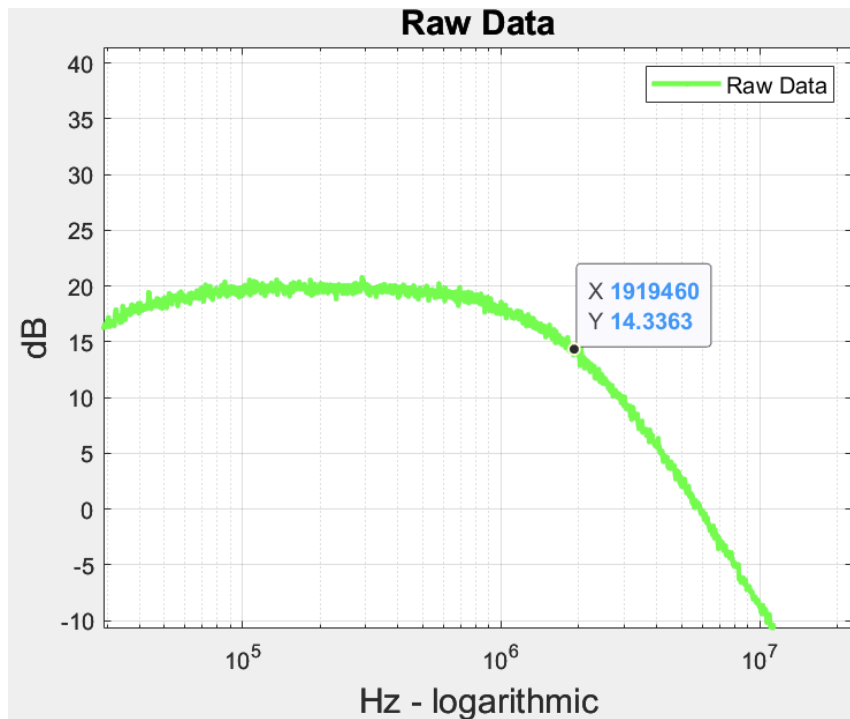
Next, the lower corner frequency (Will be used later in the high pass filter) can be found by determining where the dB value is approximately -3dB from the rising edge of the data. To achieve a straight line at this area, an equal but opposite function must be applied to cancel out the previous function.

$$\frac{1}{\left(1-j\frac{\omega}{21000}\right)^2}$$



The same principles as the previous function can be applied to the falling edge of the function. This value will be used in the Low-Pass filter

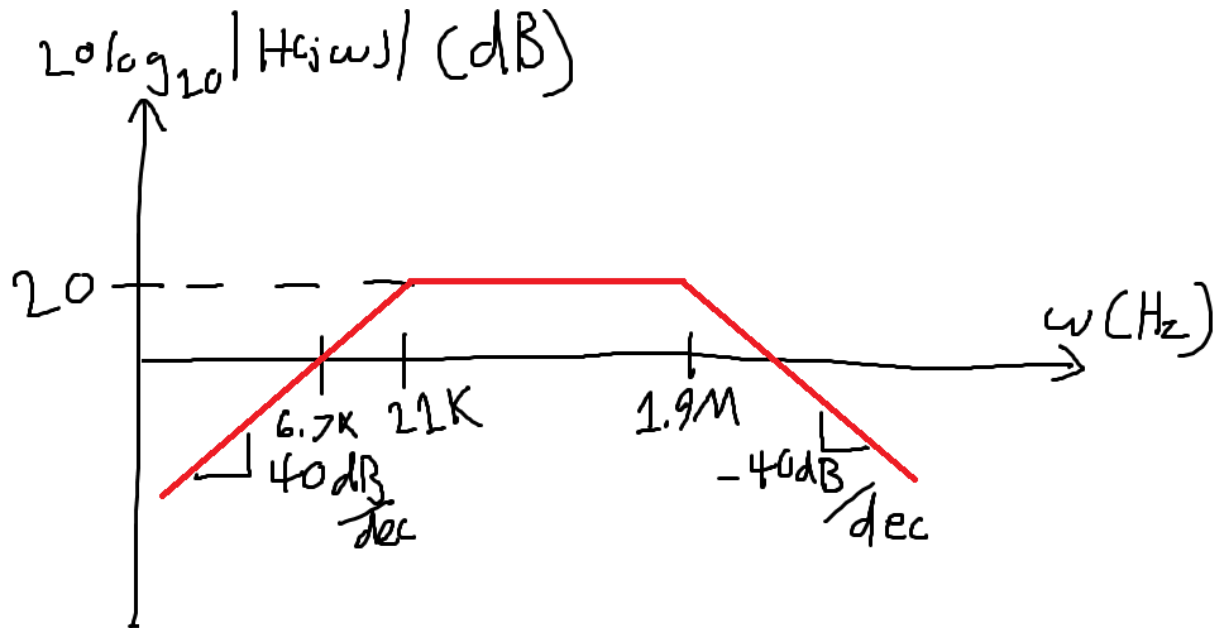
$$\frac{1}{(1 - j\frac{\omega}{1900000})^2}$$



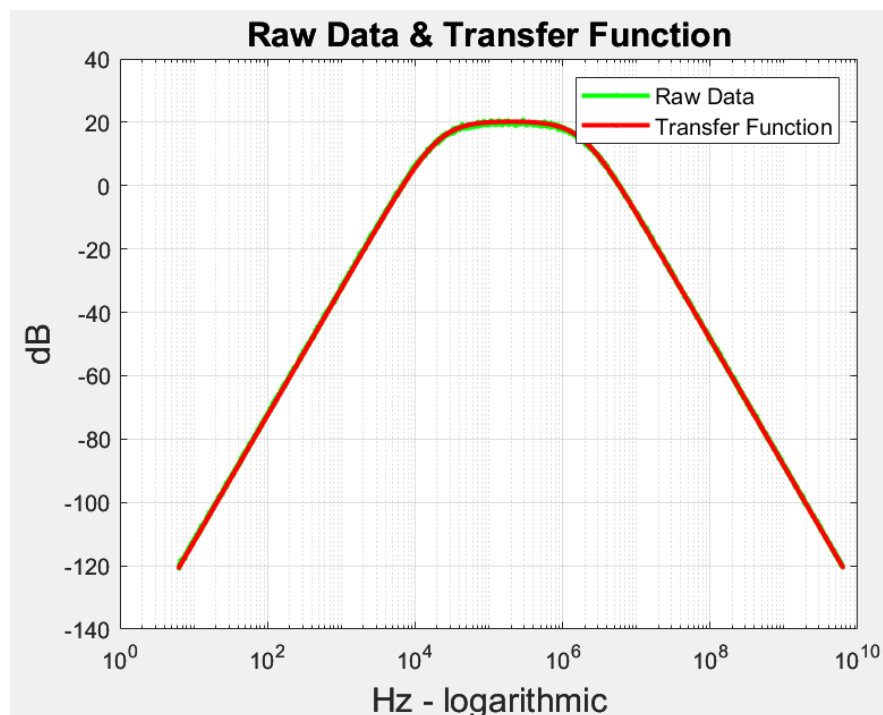
Multiplying all of these values together will create a transfer function that will closely match the data in value.

$$H(j\omega) = \left(j\frac{\omega}{6500}\right)^2 \frac{1}{(1-j\frac{\omega}{21000})^2} \frac{1}{(1-j\frac{\omega}{1900000})^2}$$

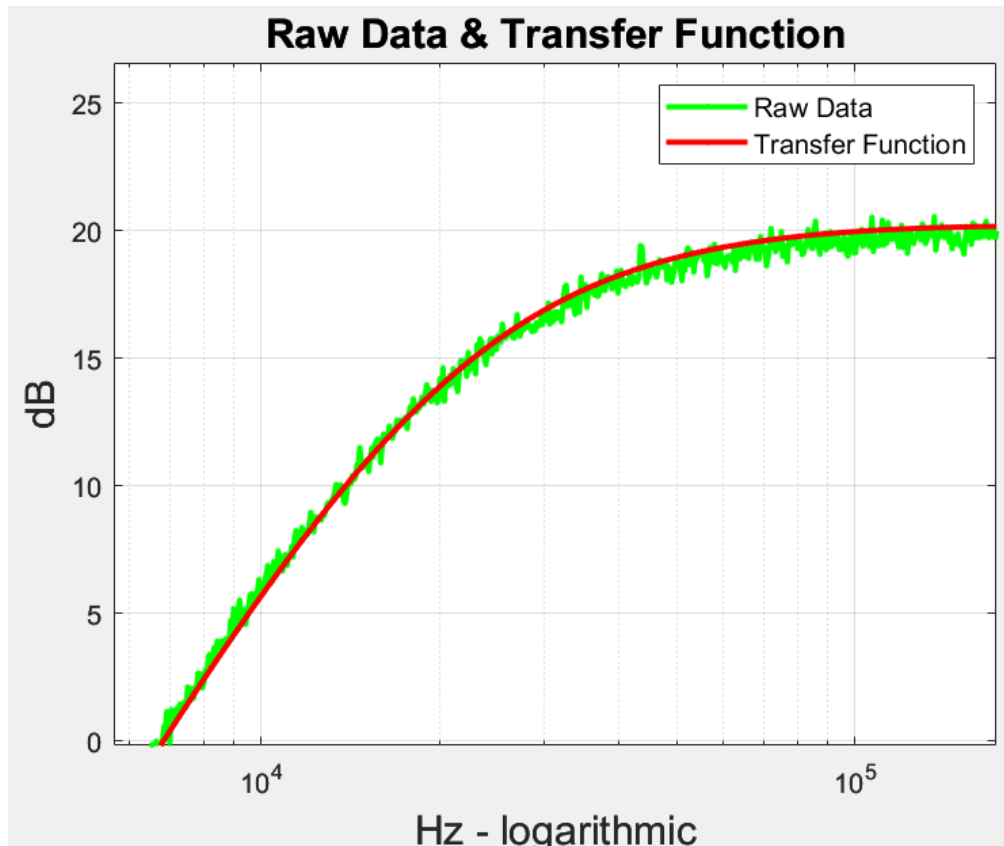
With straight-line approximations, the above function is depicted as such:



Putting both the Raw Data and Transfer Function together in a MATLAB graph, the resulting Transfer Function can be more closely compared to that of the Raw Data. As shown below, the two functions match closely.



To look closer, the MATLAB window can be shrunk down, viewing only a specific portion of both functions. With a smaller window, it is also possible to see that the Raw Data contains a considerable amount of noise. This is due to the experimental nature of it, being likely drawn from a physical circuit with imperfections and irregularities.



This graph was generated using this MATLAB program:

```
clc; close all; clearvars
% Instantiation of external variables
dx_omega = importdata("data_x.mat");
dyn = importdata("data_y.mat");
f = logspace(0,9,1000);
w = 2*pi*f;

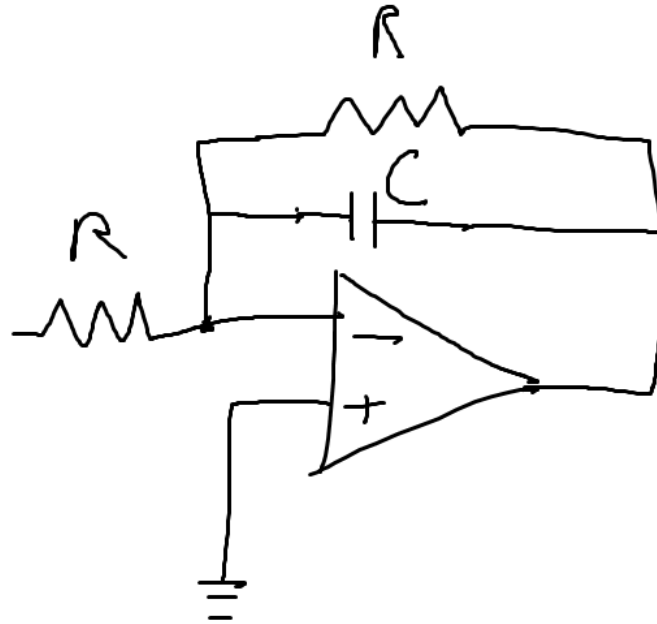
% Instantiation of Transfer Function
functOut = db(((1i*w)/6500).^2) ...
    .*((1-(1i*w)/21000).^2) ...
    .*((1-(1i*w)/1900000).^2));
dbout = db(dyn);

% Output Data to Graph
semilogx(dx_omega,dbout,'- .','LineWidth',2, 'Color', 'Green'), grid on; hold on;
semilogx(w, functOut,'- .','lineWidth',2, 'Color', 'Red');

% Format Graph
xlabel('Hz - logarithmic', 'fontsize', 15);
ylabel('dB', 'fontsize', 15);
title('Raw Data & Transfer Function', 'fontsize', 15);
legend('Raw Data', 'Transfer Function', 'PSpice Data','fontsize', 10);
```

5. Circuit Design and Simulation in PSpICE

To bring the transfer function to a physical circuit, a basic design must be constructed. To cut off the higher frequencies, a low pass filter will be used. To achieve the second-order 40 dB decrease present within the transfer function and raw data plot, two low pass filters (similar to the one pictured below) can be cascaded into one another to achieve this.



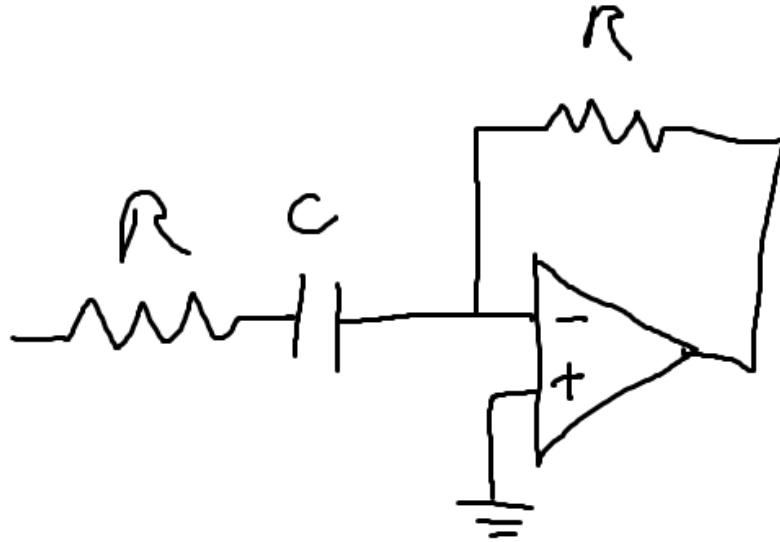
The desired resistor value can be found by using an arbitrary capacitor value found within our lab kit, then used in the below equation.

$$R = \frac{1}{2\pi f_{\text{Higher Corner}} C} = \frac{1}{2\pi \times 1.9\text{MHz} \times 0.01\mu\text{F}} \approx 8.38\Omega$$

This calculated value of the resistor can then be rounded to meet the value of a commonly used resistor.

$$R_{\text{Kit}} = 8.2\Omega$$

For a high pass filter, the same principles can be applied. A single high pass filter is cascaded with an identical filter to achieve an increase in 40 dB/decade.



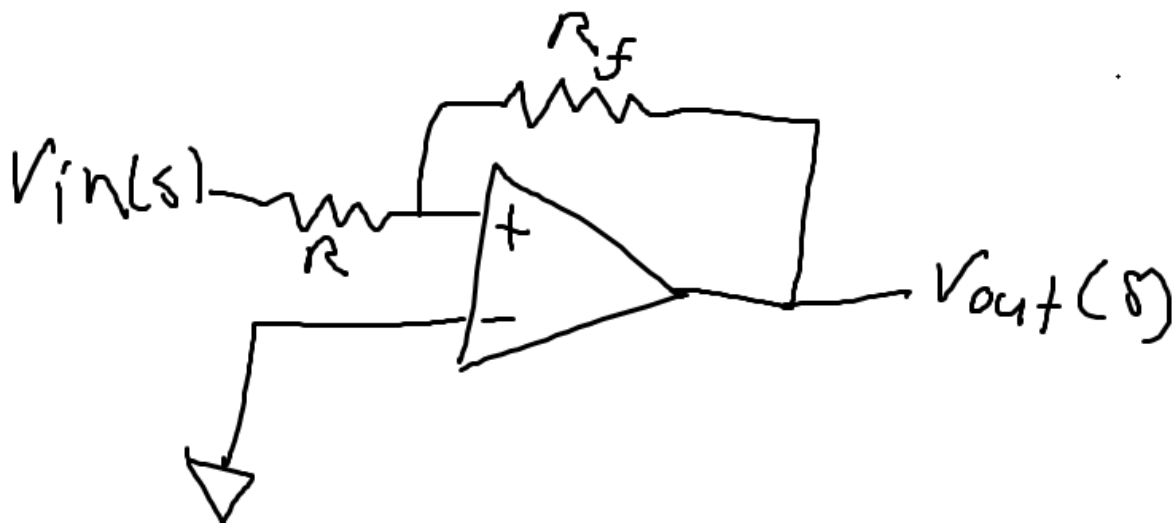
The same equation as before can be used to find the resistor value, and another arbitrary capacitor can be selected.

$$R = \frac{1}{2\pi f_{\text{Lower Corner}} C} = \frac{1}{2\pi \times 21\text{kHz} \times 0.1\mu\text{F}} \approx 75.79\Omega$$

This can then be rounded to get a value found in our kit.

$$R_{\text{Kit}} = 75\Omega$$

To control the gain of the filter, a fifth Op-Amp circuit is required. A regular gain filter is depicted below:



This circuit uses the below equation:

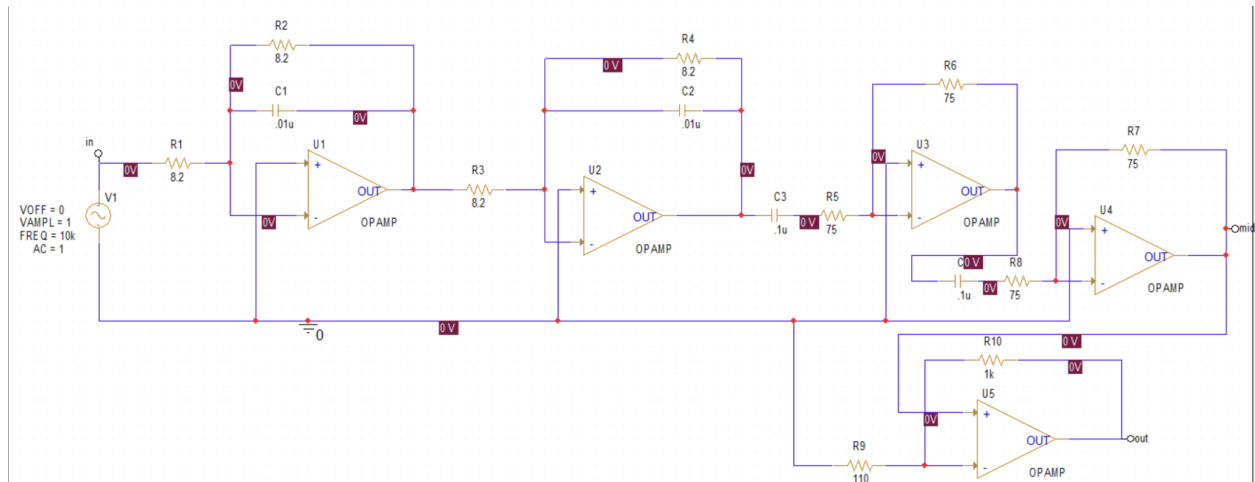
$$Gain = 1 + \frac{R_f}{R}$$

R_f can be assumed as an arbitrary resistor value. The gain is assumed to be 10, as the transfer function is moved up 20 dB.

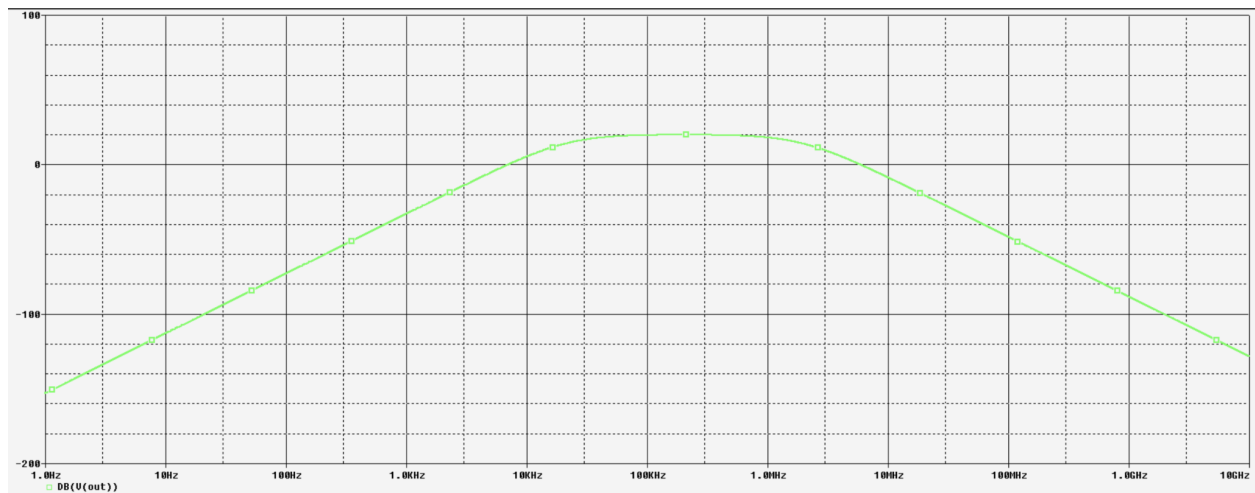
$$R = \frac{R_f}{Gain-1} = \frac{(1k\Omega)}{(10)-1} \approx 111.1\Omega$$

$$R_{Kit} = 110\Omega$$

These filters can all be cascaded into one another to create the desired band pass filter.



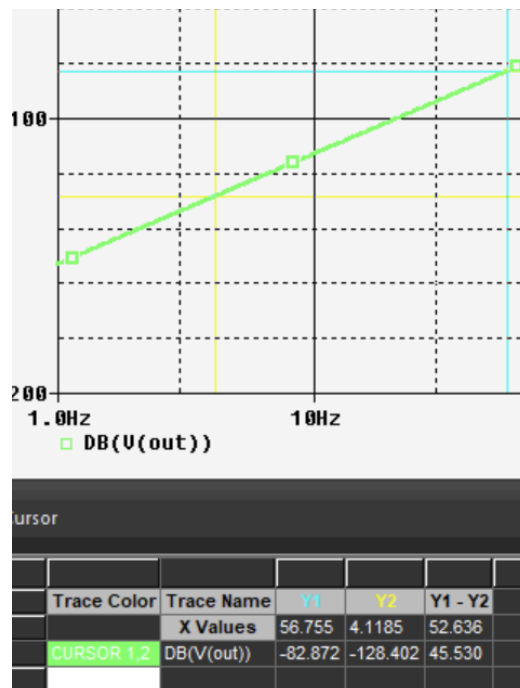
Various simulations can now be conducted through this circuit. Through PSpice, a bode plot can be generated, with the proper logarithmic decade and decibel measurements present, similar to the MATLAB data and Transfer Function.



PSpice allows for function analysis, and through this the Highpass/Lowpass cutoffs and Max dB value can be derived.

Cutoff_Highpass_3dB(V(out))	31.87550k
Cutoff_Lowpass_3dB(V(out))	1.29214meg
Max(DB(V(out)))	19.88955

Taking two points on the graph, it can be analyzed that the slope of the left and right end of the graph respectively are approximately 40 dB per decade and -40 dB per decade.



To find the slope from these points, the slope formula can be used. the x coordinates need the log of them to be taken, as they are logarithmically scaled. The following slope results to be around 40 dB per decade.

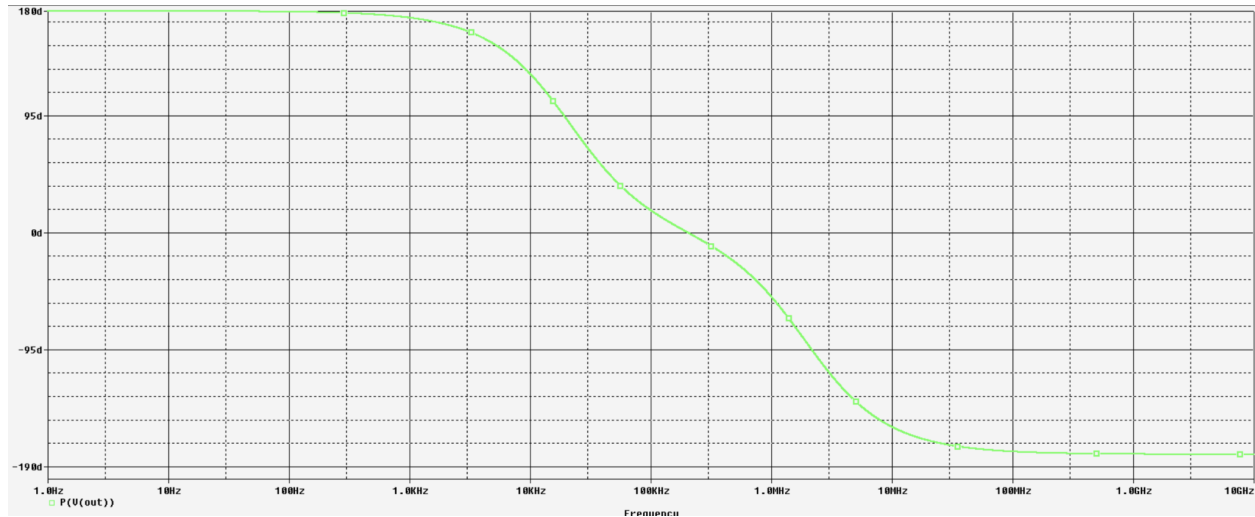
$$\frac{Y1-Y2}{X1-X2} = \frac{45.530}{\log(56.755)-\log(4.1185)} \approx 39.96 \frac{dB}{dec} \approx 40 \frac{dB}{dec}$$

The second set of points were placed on the falling side of the slope. The final slope was calculated using the same method as before. Both slopes confirmed the previous notions about the circuit graph and how it would behave.

Trace Color	Trace Name	Y1	Y2	Y1 - Y2
	X Values	269.774M	39.995M	229.780M
CURSOR 1,2	DB(V(out))	-65.602	-32.462	-33.140

$$\frac{Y1-Y2}{X1-X2} = \frac{-33.140}{\log(269.774M) - \log(39.995M)} \approx -39.97 \frac{dB}{dec} \approx -40 \frac{dB}{dec}$$

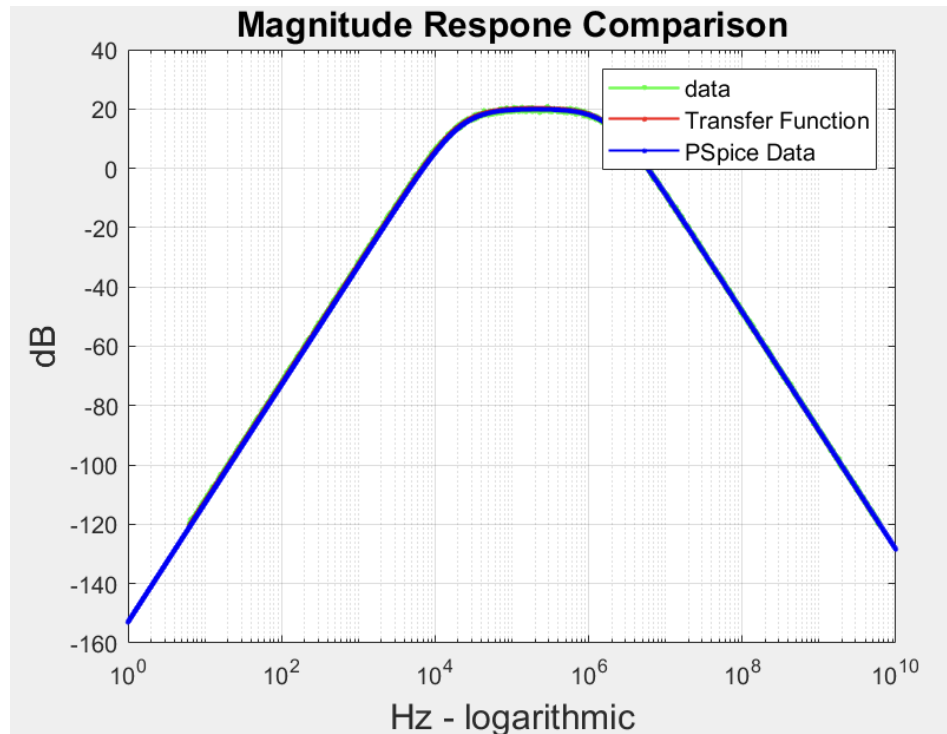
The phase response can also be derived in PSpice by using the P() function. This generates the below phase response graph.



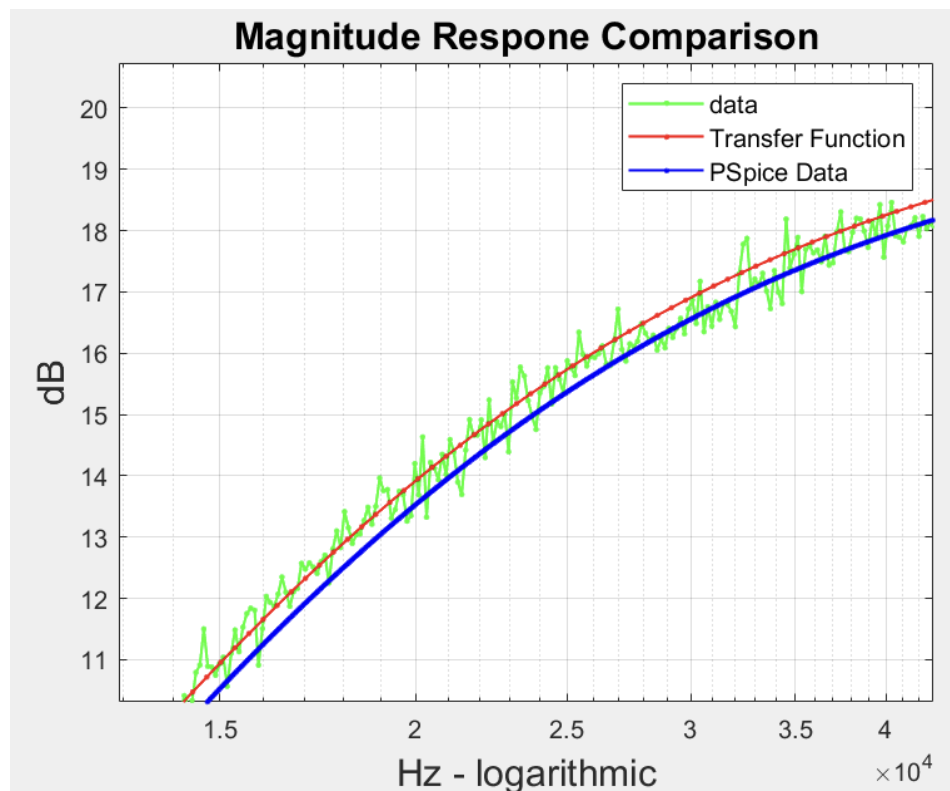
6. Comparison of Magnitude Responses

To bring the PSpice Data into MATLAB, it must first be exported into a csv file. It then can be imported by using the readtable() function in MATLAB. In this current state, the data is not readable by MATLAB. To convert it, the data needs to be set to an array using the table2array() function. Then, each column of the table can be set to both the X and Y axis of the graph, allowing it to be plotted like the raw data.

```
TempData = readtable("PSpicePhase.csv");
TempData2 = table2array(TempData);
PSX = TempData2(:,1);
PSY = TempData2(:,2);
```



To view the discretion of these values, taking a smaller part of the function greatly improves the visibility of these differences. The values are all in an acceptable range of each other, and are extremely close.



To generate this graph, a similar program to the last two programs was used, only adding in the code to utilize the PSpice data:

```
clc; close all; clearvars
% Instantiation of external variables
dx_omega = importdata("data_x.mat");
dyn = importdata("data_y.mat");
TempData = readtable("PSpiceData.csv");
% Converts the table to an array
TempData2 = table2array(TempData);
% Splits the array to 2 functions
PSX = TempData2(:,1);
PSY = TempData2(:,2);
f = logspace(0,9,1000);
w = 2*pi*f;

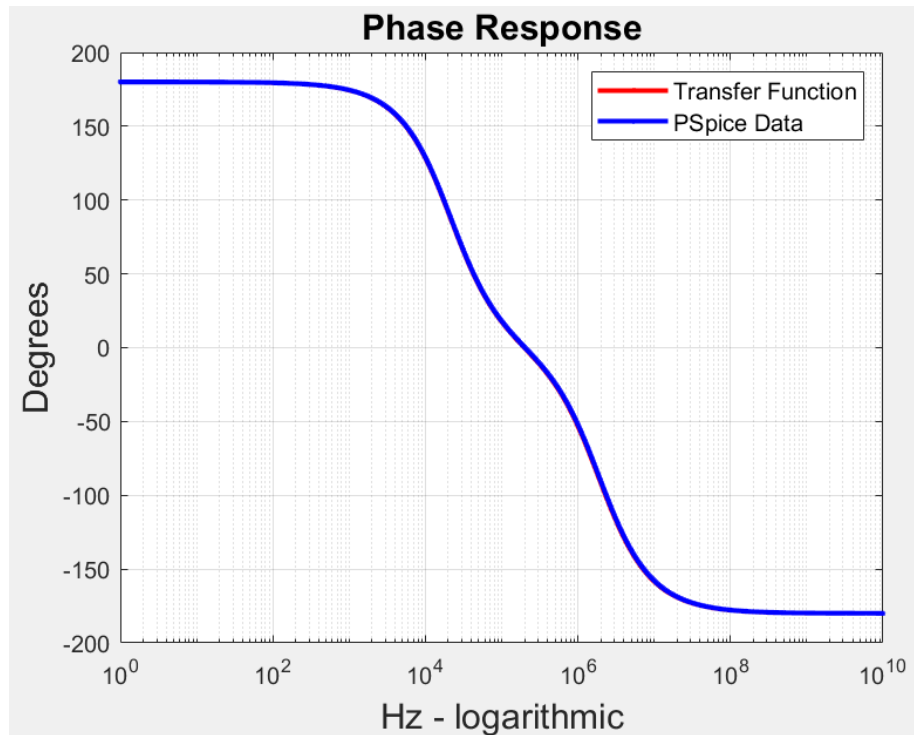
% Instantiation of Transfer Function
functOut = db((((1i*w)/6500).^2) ...
    .*((1-(1i*w)/21000).^(-2)) ...
    .*((1-(1i*w)/1900000).^(-2)));
dbout = db(dyn);

%Plot Data to Graph
semilogx(dx_omega,dbout,'- .','LineWidth',1, 'Color', 'Green'), grid on; hold on;
semilogx(w, functOut,'- .','lineWidth',1, 'Color', 'Red');
semilogx(PSX, PSY,'- .','lineWidth',1, 'Color', 'Blue');

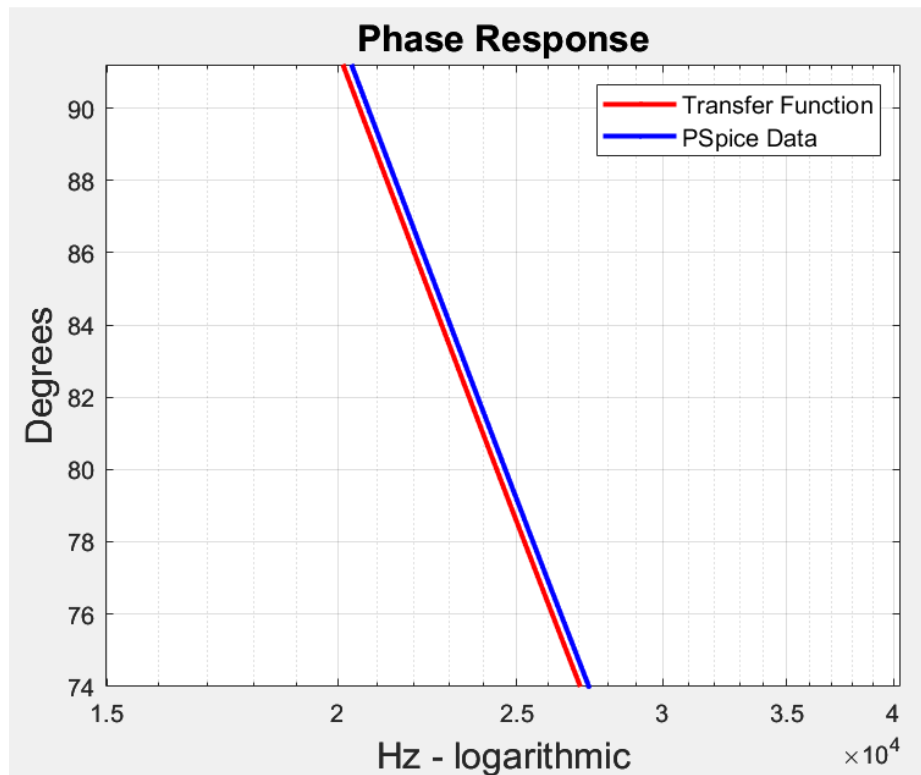
%Format Graphs
xlabel('Hz - logarithmic', 'fontsize', 15);
ylabel('dB', 'fontsize', 15);
title('Magnitude Response Comparison', 'fontsize', 15);
legend('data', 'Transfer Function', 'PSpice Data','fontsize', 10);
```

7. Comparison of Phase Responses

The data points from the PSpice phase response can be exported in the same way as the magnitude response, and plotted the same in MATLAB. However, plotting the transfer function required some tweaking. To do so, the angle() function is used to convert the data to its angle, and then it must be multiplied by 180 divided by pi as well. Due to how close these functions are, it is difficult to differentiate between the two.



Zooming the view closer shows the difference of these functions:



The following code generated the MATLAB Phase Response Graphs:


```

clc; close all; clearvars
% Instantiation of external variables
dx_omega = importdata("data_x.mat");
dyn = importdata("data_y.mat");
TempData = readtable("PSpicePhase.csv");
% Converts the table to an array
TempData2 = table2array(TempData);
% Splits the array to 2 functions
PSX = TempData2(:,1);
PSY = TempData2(:,2);
f = logspace(0,9,1000);
w = f;

% Instantiation of Transfer Function
functOut = (((1i*w)/6500).^2) ...
    .*((1-(1i*w)/21000).^2) ...
    .*((1-(1i*w)/1900000).^2);
dbout = db(dyn);

%Plot Data to Graph
semilogx(w, angle(functOut)*-180/pi,'- .','lineWidth',2, 'Color', 'Red'); grid on; hold on
semilogx(PSX, PSY,'- .','lineWidth',2, 'Color', 'Blue');

%Format Graph
xlabel('Hz - logarithmic', 'fontsize', 15);
ylabel('Degrees', 'fontsize', 15);
title ('Phase Response', 'fontsize', 15);
legend('Transfer Function', 'Spice Data','fontsize', 10);

```

8. Statement on Group Work

The work was completed as a group with the following contributions from Collin Bovenschen and Jacob Erwin:

Collin Bovenschen - 50%

Jacob Erwin - 50%

9. Discussion

This lab provided a deeper understanding of second order filters and their responses. Many techniques from previous labs were called upon to complete the final project. While the equations from previous labs were helpful, it was important to understand that the corner frequency varied slightly in second order low and high pass filters and so corner frequencies were used instead when constructing the circuits. Overall we were able to produce a circuit that mirrored the raw data with low deviation under simulation. Next steps could include replicating the circuit in the lab and collecting raw data of our own and comparing it. One discrepancy within the plots is the PSpice data versus the transfer function. In theory, it is possible to create a circuit with the exact same output as the transfer function. However, this requires the value of many of the resistors to be exactly precise to a long repeating decimal, and these values are not found within our lab kit. Therefore, the exact transfer function will not be converted into the circuit.