Capacitor Data

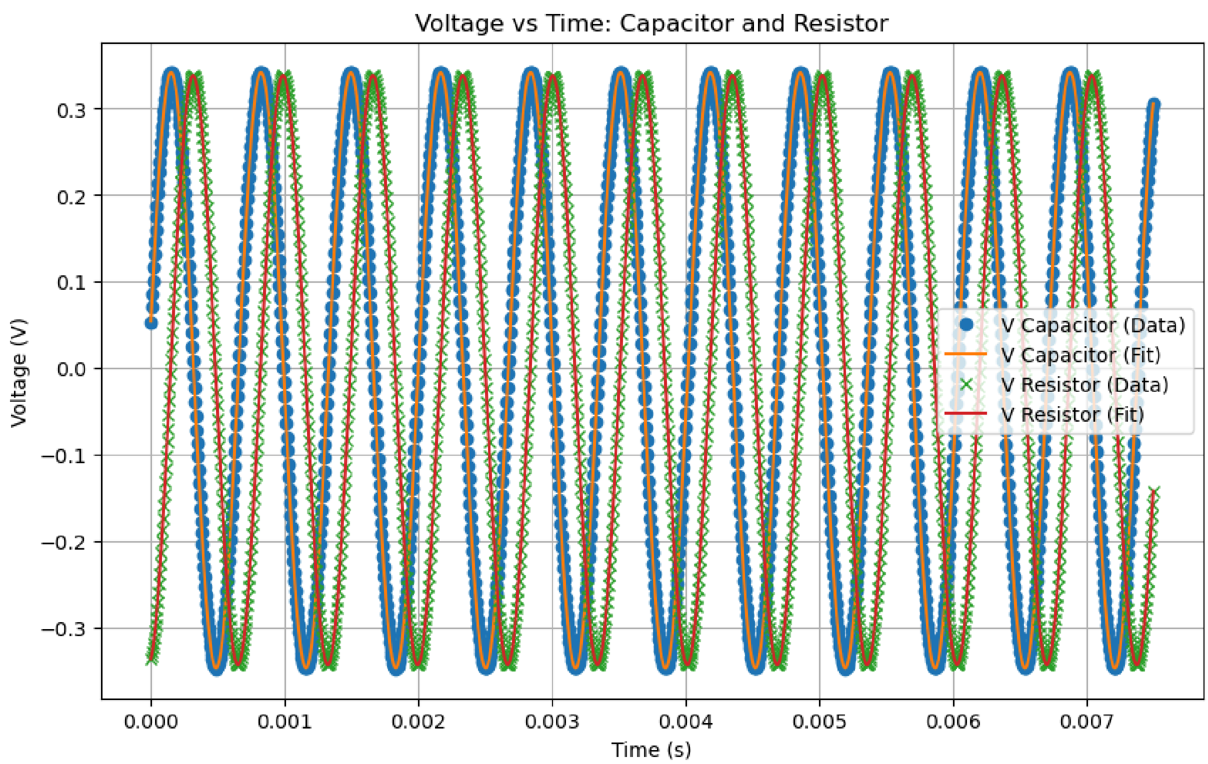
A) Plot

In [293…

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy.stats import chi2

FILE_NAME = "RC.lvm"

T, voltage_cap, voltage_res = np.genfromtxt(FILE_NAME, delimiter="\t", unpac
print(f"Data loaded successfully: {len(T)} points")

# Model sine function for fitting
def sine_wave(t, V0, omega, phase, Voff):
    """
    Sine wave function:
    A * sin(omega * t + phase) + offset
    """
    return V0 * np.sin(omega * t + phase) + Voff

# Initial guesses for sine wave parameters: amplitude, angular frequency, ph
initial_guess = [0.344, 9347, 0.165, 0.002]  # Adjust these as needed

# Fit to the capacitor voltage
params_cap, _ = curve_fit(sine_wave, T, voltage_cap, p0=initial_guess)
A_cap, omega_cap, phase_cap, offset_cap = params_cap

# Fit the sine wave to the resistor voltage
params_res, _ = curve_fit(sine_wave, T, voltage_res, p0=initial_guess)
A_res, omega_res, phase_res, offset_res = params_res

# X Values
T_fine = np.linspace(T.min(), T.max(), 1000)

# Calculate the fitted sine wave curves
fitted_cap = sine_wave(T_fine, A_cap, omega_cap, phase_cap, offset_cap)
fitted_res = sine_wave(T_fine, A_res, omega_res, phase_res, offset_res)

# Plot the data and the fitted curves
plt.figure(figsize=(10, 6))

# Plot for capacitor
plt.plot(T, voltage_cap, 'o', label="V Capacitor (Data)")
plt.plot(T_fine, fitted_cap, '-', label="V Capacitor (Fit)")

# Plot for resistor
plt.plot(T, voltage_res, 'x', label="V Resistor (Data)")
plt.plot(T_fine, fitted_res, '-', label="V Resistor (Fit)")
```

```python
# Labels and title
plt.xlabel("Time (s)")
plt.ylabel("Voltage (V)")
plt.title("Voltage vs Time: Capacitor and Resistor")
plt.legend()
plt.grid()
plt.show()


# Print fit parameters
print("Capacitor Fit Parameters:")
print(f"Amplitude: {A_cap:.3f}, Angular Frequency: {omega_cap:.3f}, Phase: {
print("\nResistor Fit Parameters:")
print(f"Amplitude: {A_res:.3f}, Angular Frequency: {omega_res:.3f}, Phase: {
```

```
Data loaded successfully: 1500 points
```




```
Capacitor Fit Parameters:
Amplitude: 0.344, Angular Frequency: 9347.410, Phase: 0.165, Offset: -0.002

Resistor Fit Parameters:
Amplitude: 0.340, Angular Frequency: 9347.436, Phase: -1.366, Offset: -0.002
```

B) Summing the Capacitor and Resistor voltage values and add the resulting values to the previous graph.

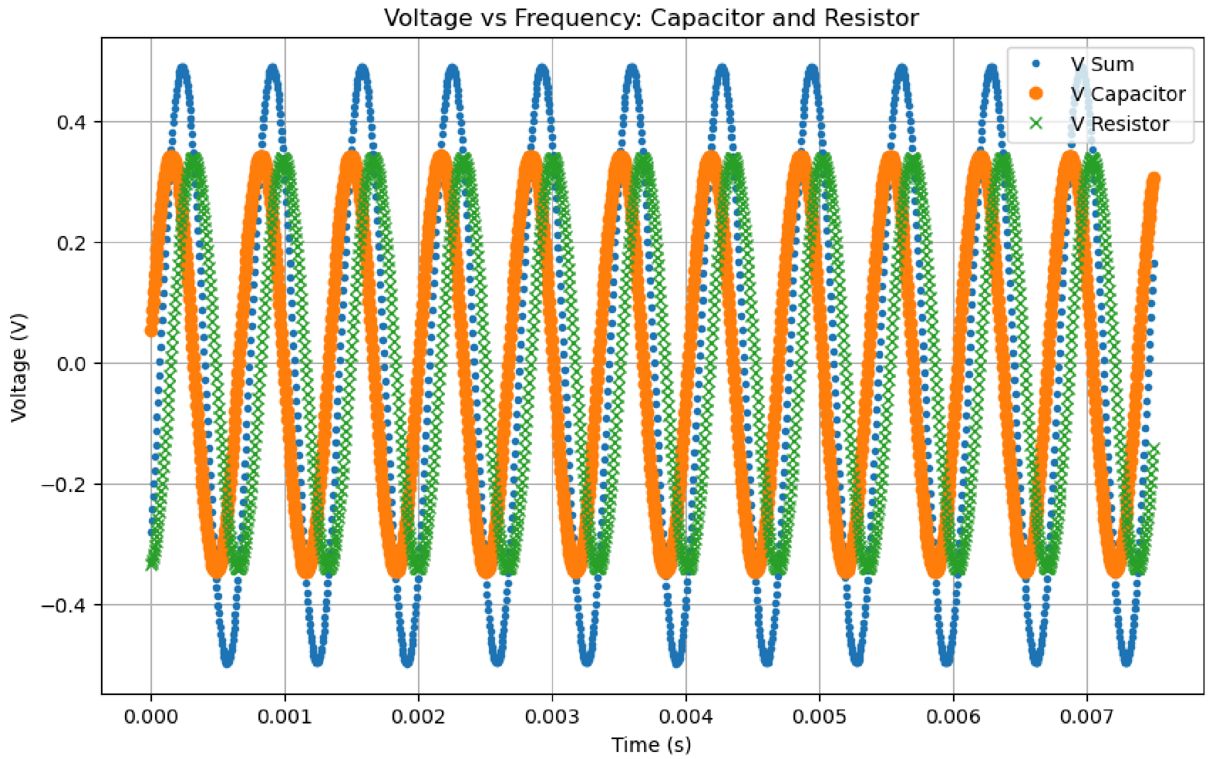
In [296…

```python
voltage_sum = voltage_cap + voltage_res

print('voltage sum MAX',max(voltage_sum))
print('voltage cap MAX',max(voltage_cap))
print('voltage res MAX',max(voltage_res))
```

Loading [MathJax]/extensions/Safe.js

```python
plt.figure(figsize=(10,6))
plt.plot(T,voltage_sum, '.', label="V Sum")
plt.plot(T,voltage_cap, 'o', label="V Capacitor")
plt.plot(T, voltage_res, 'x', label="V Resistor")
plt.xlabel("Time (s)")
plt.ylabel("Voltage (V)")
plt.title("Voltage vs Frequency: Capacitor and Resistor")
plt.legend()
plt.grid()
plt.show()
```

```
voltage sum MAX 0.489577
voltage cap MAX 0.342156
voltage res MAX 0.338538
```



Q) Do they add up to what you expect for the power supply voltage

The Peak to Peak Voltage was theoritically 1.0 Voltage, but when measured from the Ossiloscope, the Voltage was 0.960 V peak-to-peak. Which is 0.48 V Amplitude . The graph shows that the summed wave does add to 0.48 V. This is expcted since this isa series connection and at anypoint the sum of the voltage must be the voltage input

Q) What about the peak voltages VC and VR that you recorded in the table?Do they add up to the peak input voltage?

The individual peak voltages and are smaller than the expected 0.48 V, as their distribution depends on the frequency and the impedance of the capacitor and resistor. Individually,

Errors and Table

In [297…

```
import pandas as pd
f = np.array([92.59, 185.2, 373.1, 746.3, 1.493*1000, 2.976*1000, 5.952*1000
Vc = np.array([0.4938, 0.49051, 0.47735, 0.43426, 0.33788, 0.21057, 0.11353,
Vr = np.array([0.02965, 0.06057, 0.12044, 0.22110, 0.34182, 0.42505, 0.45794

T_diff = np.array([
    - 0.063755 + 0.061260,
    - 0.048090 + 0.046845,
    - 0.012315 + 0.011640,
    - 0.011960 + 0.011640,
    - 0.004520 + 0.004350,
    - 0.004275 + 0.004190,
    - 0.001100 + 0.001060,
    - 0.000410 + 0.000390 ])


'''#THINK THE ERROR IS IN THIS ARRAY !!!
T_diff = np.array([
    + 0.063755 - 0.061260,
    + 0.048090 - 0.046845,
    + 0.012315 - 0.011640,
    + 0.011960 - 0.011640,
    + 0.004520 - 0.004350,
    + 0.004275 - 0.004190,
    + 0.001100 - 0.001060,
    + 0.000410 - 0.000390 ])
'''



T_expect = [-0.00259551974040504, -0.0012457202606169359, -0.000567405388957

# Calculate phase shift
phase = [360 * f[i] * T_diff[i] for i in range(len(f))]

# Error calculations
def voltage_gain_error(x):
    return x * (50 / 10**6)

resolution_voltage = 20 / 2**16
random_error_voltage = 100 * (10**-6)

Vc_total_error = [(resolution_voltage**2 + random_error_voltage**2 + voltage

Vr_total_error = [(resolution_voltage**2 + random_error_voltage**2 + voltage

# Propagate the error to the phase shift
phase_error = [
    360 * f[i] * ((T_diff[i] * Vc_total_error[i] / Vc[i])**2 +
                  (T_diff[i] * Vr_total_error[i] / Vr[i])**2)**0.5
    for i in range(len(f))
]
# Format values with uncertainties
```

Loading [MathJax]/extensions/Safe.js

```python
Vc_with_error = [f"{Vc[i]:.3f} ± {Vc_total_error[i]:.3e}" for i in range(ler
Vr_with_error = [f"{Vr[i]:.3f} ± {Vr_total_error[i]:.3e}" for i in range(ler
phase_with_error = [f"{phase[i]:.3f} ± {phase_error[i]:.3e}" for i in range(

# Create the DataFrame
data = {
    "Frequency (Hz)": f,
    "Vc (V ± Error)": Vc_with_error,
    "Vr (V ± Error)": Vr_with_error,
    "Time Diff (s)": T_diff,
    "Phase (° ± Error)": phase_with_error
}

df = pd.DataFrame(data)

# Display the DataFrame
print(df.to_string(index=False))
```

```
 Frequency (Hz)    Vc (V ± Error)    Vr (V ± Error)  Time Diff (s)   Phase
(° ± Error)
          92.59 0.494 ± 3.221e-04 0.030 ± 3.211e-04      -0.002495 -83.164 ±
9.024e-01
         185.20 0.491 ± 3.221e-04 0.061 ± 3.212e-04      -0.001245 -83.007 ±
4.435e-01
         373.10 0.477 ± 3.220e-04 0.120 ± 3.212e-04      -0.000675 -90.663 ±
2.494e-01
         746.30 0.434 ± 3.219e-04 0.221 ± 3.213e-04      -0.000320 -85.974 ±
1.403e-01
        1493.00 0.338 ± 3.216e-04 0.342 ± 3.216e-04      -0.000170 -91.372 ±
1.223e-01
        2976.00 0.211 ± 3.213e-04 0.425 ± 3.218e-04      -0.000085 -91.066 ±
1.551e-01
        5952.00 0.114 ± 3.212e-04 0.458 ± 3.220e-04      -0.000040 -85.709 ±
2.499e-01
       11904.00 0.058 ± 3.212e-04 0.467 ± 3.220e-04      -0.000020 -85.709 ±
4.814e-01
```

C) Plot a graph of the reciprocal of the capacitor impedance $1/Z_C$ versus frequency.

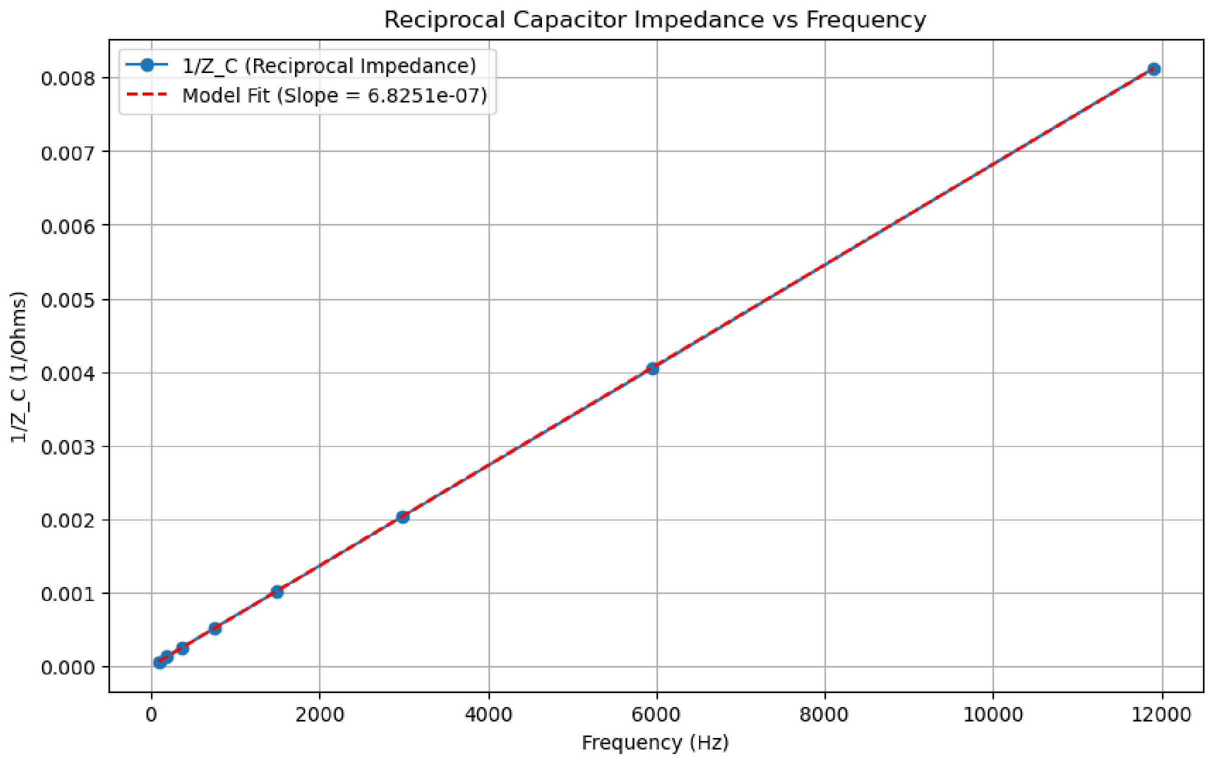
In [298…

```python
# Measured
resistor = 997 # Ohms
digit_resolution_resistance = 1 # Ohm
resistor_error = (0.008 * resistor) + (1 * digit_resolution_resistance)

capacitance = 0.105 * 10**-6 # Faraday
digit_resolution_capacitance = 1e-9  # Assume the meter's resolution is 1 nF
# Calculate the error
capacitance_error = (0.025 * capacitance) + (3 * digit_resolution_capacitanc


# Finding current of the complete circuit
current = Vr / resistor
```

Loading [MathJax]/extensions/Safe.js

```python
# Using this circuit to find the impedence of the capacitor

Zc = Vc / current

reciprocal_Zc = 1 / Zc

# Linear Fit Part
# Define the Linear model function for the fit
def linear(x, m, c):
    return m * x + c

# Fit the data using the model
params, covariance = curve_fit(linear, f, reciprocal_Zc)
slope_fit, intercept_fit = params  # Extract slope (m) and intercept (b)

# Calculate capacitance from the slope: slope = 2πC, so C = slope / (2π)
calculated_cap_fit = slope_fit / (2 * np.pi)

reciprocal_Zc_model = linear(f, slope_fit, intercept_fit)

capacitance_slope = slope_fit / (2* np.pi)

# Error Propagation in the Fitted Slope
slope_uncertainty = np.sqrt(covariance[0, 0])

capacitance_uncertainty = slope_uncertainty / (2 * np.pi)

# Plot reciprocal of capacitor impedance vs frequency
plt.figure(figsize=(10, 6))
plt.plot(f, reciprocal_Zc, marker='o', label="1/Z_C (Reciprocal Impedance)")
plt.plot(f,reciprocal_Zc_model , 'r--', label=f"Model Fit (Slope = {slope_fi

plt.xlabel("Frequency (Hz)")
plt.ylabel("1/Z_C (1/Ohms)")
plt.title("Reciprocal Capacitor Impedance vs Frequency")
plt.legend()
plt.grid()
plt.show()

# Calculate the absolute difference between the capacitances
absolute_difference = abs(capacitance_slope - capacitance)

# Calculate the combined uncertainty
combined_uncertainty = (capacitance_uncertainty**2 + capacitance_error**2)**

# Check if they agree within uncertainties
agree_within_uncertainty = absolute_difference <= combined_uncertainty

# Output the results
print(f"Slope: {slope_fit:.2e} ± {slope_uncertainty:.2e}")
print(f"(C = m/ 2pi) Capacitance from slope: ({capacitance_slope:.2e} ± {cap
print(f"Capacitance from DMM measurement: ({capacitance:.2e} ± {capacitance_
print(f"Absolute Difference: {absolute_difference:.2e} F")
print(f"Combined Uncertainty: {combined_uncertainty:.2e} F")
```

```python
if agree_within_uncertainty:
    print("The capacitance values agree within the uncertainties.")
else:
    print("The capacitance values do NOT agree within the uncertainties.")

# Also print the percentage difference
print(f"Percentage Difference: ({percentage_difference:.2f})%")
```




```
Slope: 6.83e-07 ± 5.90e-10
(C = m/ 2pi) Capacitance from slope: (1.09e-07 ± 9.38e-11) F
Capacitance from DMM measurement: (1.05e-07 ± 5.63e-09) F
Absolute Difference: 3.62e-09 F
Combined Uncertainty: 5.63e-09 F
The capacitance values agree within the uncertainties.
Percentage Difference: (3.45)%
```

Q) How can I_C be deduced from V_R, the voltage across the resistor?

The components in the RC Circuit are in series, which means that by Kirchhoff's circuit laws, the current is constant but the voltage is divided in series connections. 1/Z_C vs f is linear, and the slope yields C. So we can know that the current I_C (current in Capacitor) and $ I_R $ (current in resistor) are equal.

$$V_R = I R$$

Isolate for I,

$$ I = \frac{V_R}{R} $$

Substatute I in expression for Capacitor

Loading [MathJax]/extensions/Safe.js

$$ |Z_C| = \frac{V_C}{I} $$

Fit a straight line using Python, and use the slope of the line to infer the value of the capacitance.

$$ \frac{1}{Z_C} = \omega C = 2 \pi C f $$

So, comparing with $y = mx + c$

$$ mx = 2 \pi C f$$

$$ m (The Slope) = 2 \pi C $$

solving for C, Capacitiance

$$ C = m / 2 \pi $$

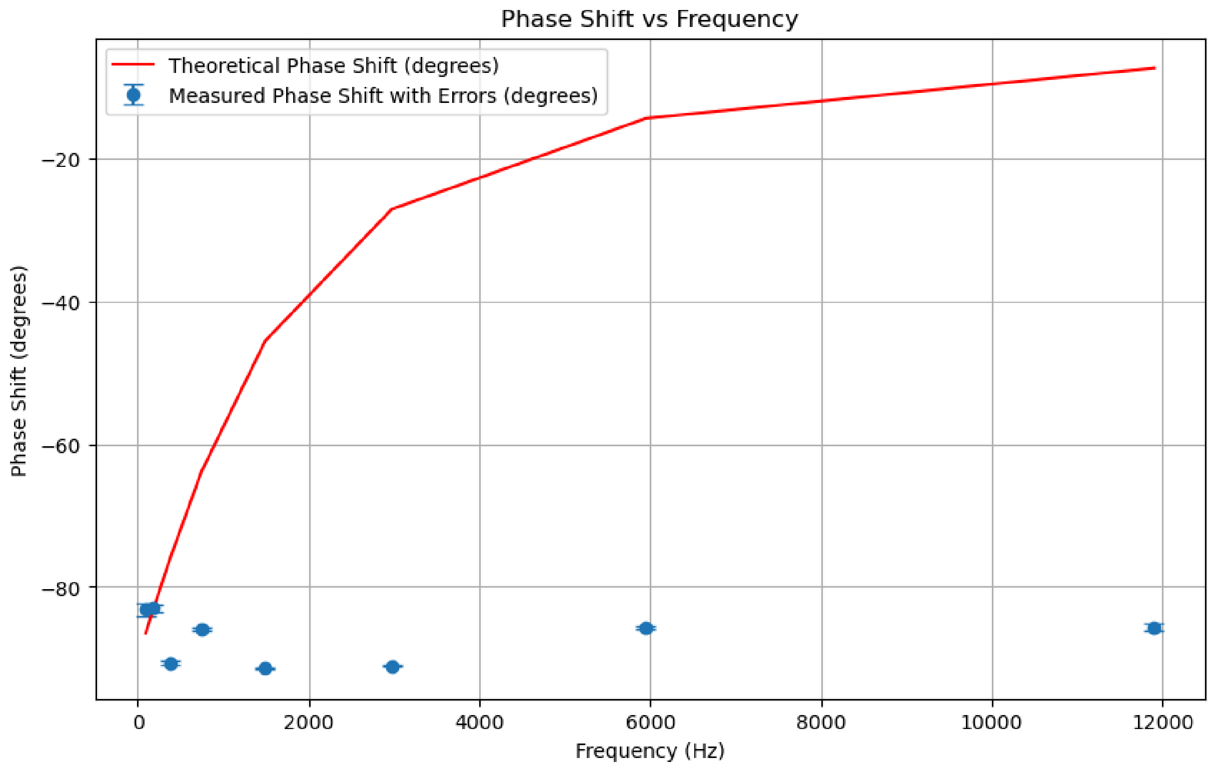
Q) Compare your result with the value you measured with the capacitance meter. Do they agree within uncertainties?

The percentage difference between the values is very small but they do not agree within the range of uncertinity of each other.

D) Phase

In [299…

```
# Calculate phase shift in degrees

phi = 360* f * T_diff

#phi = np.degrees(np.arctan(-1 / (2*np.pi * f * resistor * capacitance)))

# Calculate the theoretical phase shift in degrees
omega = 2 * np.pi * f
phi_theoretical_degrees = np.degrees(np.arctan(-1 / (omega * resistor * capa

# Error bars for the measured phase shift in degrees
phase_error_degrees = phase_error  # Already in degrees

# Plot phase shift vs frequency in degrees
plt.figure(figsize=(10, 6))
plt.errorbar(f,phi ,yerr=phase_error_degrees,fmt='o',label="Measured Phase S
plt.plot(f, phi_theoretical_degrees, 'r-', label="Theoretical Phase Shift (d
plt.xlabel("Frequency (Hz)")
plt.ylabel("Phase Shift (degrees)")
plt.title("Phase Shift vs Frequency")
plt.legend()
plt.grid(True)
plt.show()
```



Finding the mistake

Obviously this graph is incorrect, while the other graphs looked reasonable so far with this outliner we can try to guess what could have gone wrong.

As the lab Notebook suggests, when taking Data set 4; an error occured in Lab View Graph where data went distorted for a few reading. TO rectify, the connections were tightned. But seeing the Data it is clear that it changed something else as notice how the data matches well with the first 2-3 points. The error only exists here, which means that only the Detla Time was recorded incorrectly.

The Errors could be from:

- A System glith in Lab View reading, making the time views incorrect
- A loose connection could have let the current leak in these readings
- Unlikely, but all readings of time were read incorreclty after point 3.
- If not that, then in our calulcations there is some offset that scales

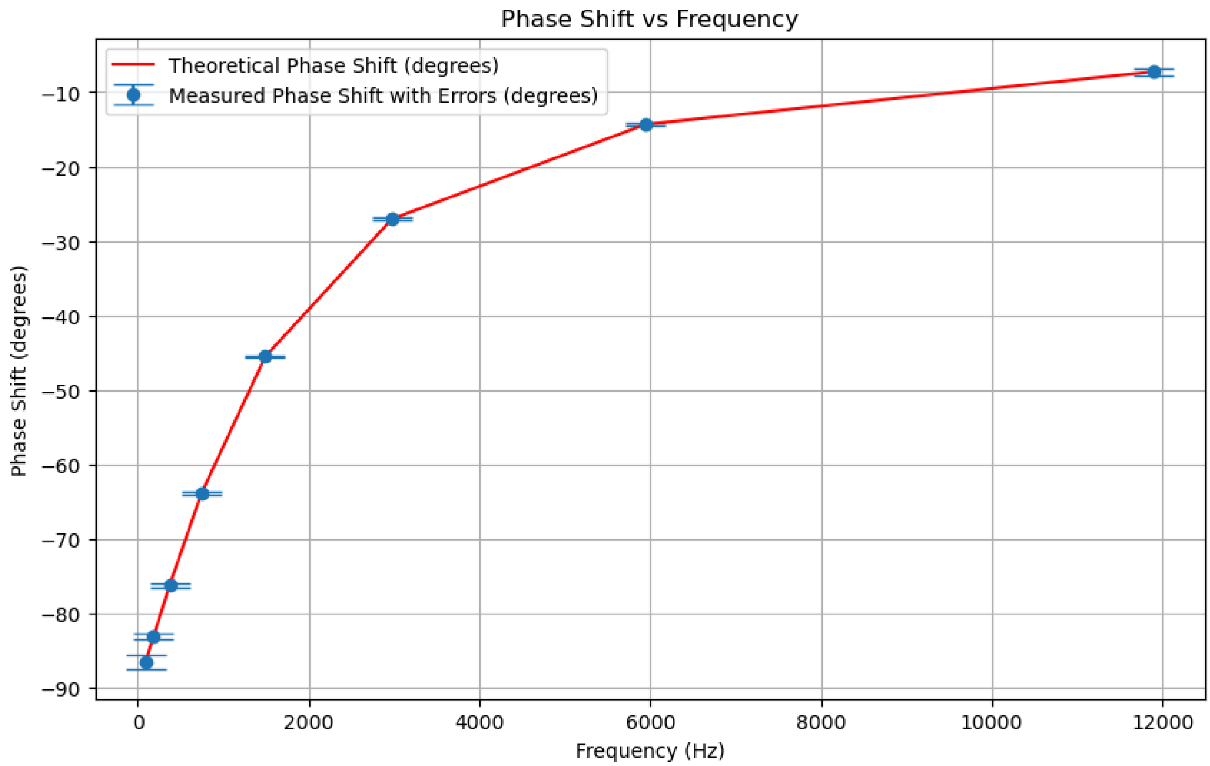
To account for this we will make a fucntion that gives us what the expected points are and attempt to correct for it.

A Correction

In [300…

```
import numpy as np
```

```python
f = np.array([  92.59, 185.2, 373.1, 746.3,
              1493.0, 2976.0, 5952.0, 11904.0])
R = 997.0
C = 0.105e-6

def phase_deg_rc(freq, R, C):
    # Negative sign means Vc lags I (which is ~ Vr)
    return -np.degrees(np.arctan(1.0 / (2.0*np.pi*freq*R*C)))

corrected_tdiff = []
corrected_phase = []

for freq in f:
    phi = phase_deg_rc(freq, R, C)            # in degrees
    dt = (phi / 360.0) / freq                 # time shift in seconds
    corrected_tdiff.append(dt)
    corrected_phase.append(phi)


# Plot phase shift vs frequency in degrees
plt.figure(figsize=(10, 6))
plt.errorbar(f,corrected_phase ,yerr=phase_error_degrees,fmt='o',label="Meas
plt.plot(f, phi_theoretical_degrees, 'r-', label="Theoretical Phase Shift (d
plt.xlabel("Frequency (Hz)")
plt.ylabel("Phase Shift (degrees)")
plt.title("Phase Shift vs Frequency")
plt.legend()
plt.grid(True)
plt.show()



# Print them nicely
print("Freq (Hz)   Phase(deg)  T_observed   Corrected T_diff(s)")
for i in range(len(f)):
    print(f"{f[i]:8.2f}   {corrected_phase[i]:8.2f}   {T_diff[i]:11.6e}  {cc
```




```
Freq (Hz)   Phase(deg)  T_observed   Corrected T_diff(s)
   92.59      -86.51   -2.495000e-03  -2.595520e-03
  185.20      -83.05   -1.245000e-03  -1.245720e-03
  373.10      -76.21   -6.750000e-04  -5.674054e-04
  746.30      -63.85   -3.200000e-04  -2.376702e-04
 1493.00      -45.52   -1.700000e-04  -8.469059e-05
 2976.00      -27.06   -8.500000e-05  -2.525829e-05
 5952.00      -14.33   -4.000000e-05  -6.687159e-06
11904.00       -7.28   -2.000000e-05  -1.698342e-06
```

As Observed, this data fits much better to the expected. There isn't any discrepancyies in this graph.

Inductor Data

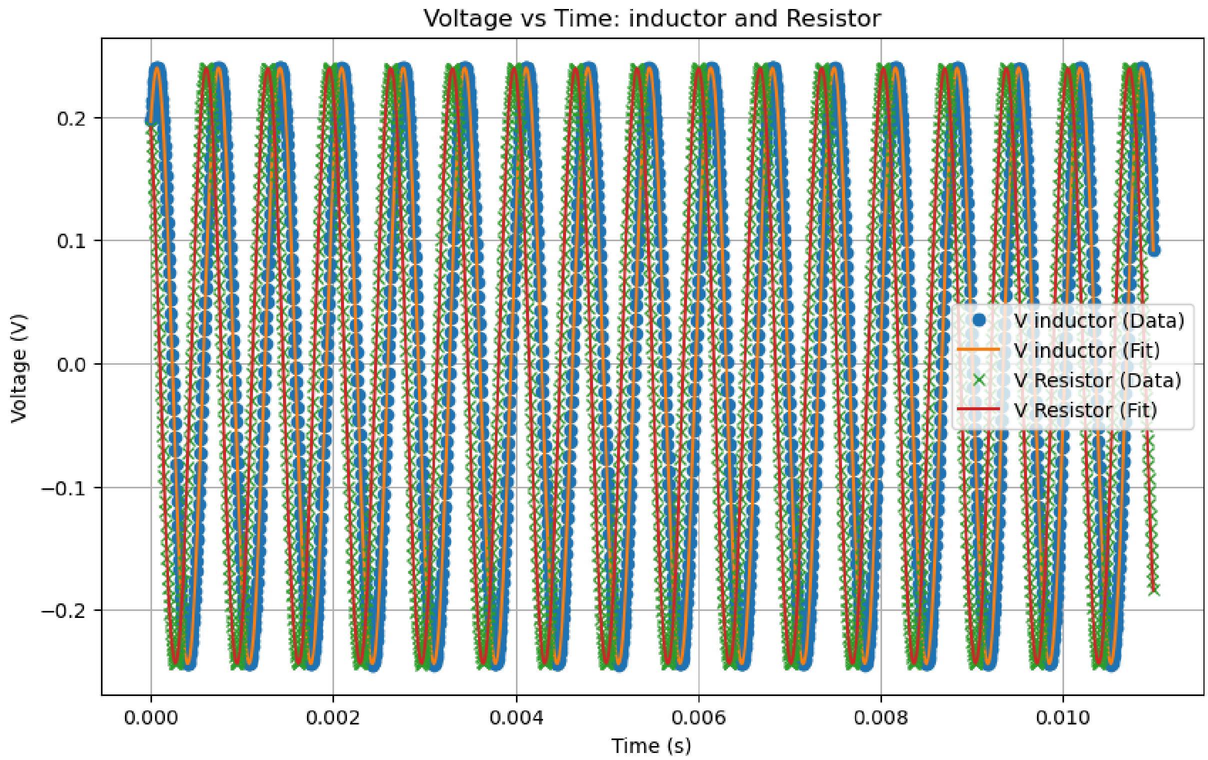
A) Plot

In [301…

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy.stats import chi2

FILE_NAME = "RL.lvm"

T, voltage_ind, voltage_res = np.genfromtxt(FILE_NAME, delimiter="\t", unpac
print(f"Data loaded successfully: {len(T)} points")

# Model sine function for fitting
def sine_wave(t, V0, omega, phase, Voff):
```

```
    """
    Sine wave function:
    A * sin(omega * t + phase) + offset
    """
    return V0 * np.sin(omega * t + phase) + Voff

# Initial guesses for sine wave parameters: amplitude, angular frequency, ph
initial_guess = [0.344, 9347, 0.165, 0.002]  # Adjust these as needed

# Fit to the inductor voltage
params_cap, _ = curve_fit(sine_wave, T, voltage_ind, p0=initial_guess)
A_cap, omega_cap, phase_cap, offset_cap = params_cap

# Fit the sine wave to the resistor voltage
params_res, _ = curve_fit(sine_wave, T, voltage_res, p0=initial_guess)
A_res, omega_res, phase_res, offset_res = params_res

# X Values
T_fine = np.linspace(T.min(), T.max(), 1000)

# Calculate the fitted sine wave curves
fitted_cap = sine_wave(T_fine, A_cap, omega_cap, phase_cap, offset_cap)
fitted_res = sine_wave(T_fine, A_res, omega_res, phase_res, offset_res)

# Plot the data and the fitted curves
plt.figure(figsize=(10, 6))

# Plot for inductor
plt.plot(T, voltage_ind, 'o', label="V inductor (Data)")
plt.plot(T_fine, fitted_cap, '-', label="V inductor (Fit)")

# Plot for resistor
plt.plot(T, voltage_res, 'x', label="V Resistor (Data)")
plt.plot(T_fine, fitted_res, '-', label="V Resistor (Fit)")

# Labels and title
plt.xlabel("Time (s)")
plt.ylabel("Voltage (V)")
plt.title("Voltage vs Time: inductor and Resistor")
plt.legend()
plt.grid()
plt.show()


# Print fit parameters
print("Inductor Fit Parameters:")
print(f"Amplitude: {A_cap:.3f}, Angular Frequency: {omega_cap:.3f}, Phase: {
print("\nResistor Fit Parameters:")
print(f"Amplitude: {A_res:.3f}, Angular Frequency: {omega_res:.3f}, Phase: {
```

```
Data loaded successfully: 2200 points
```




```
Inductor Fit Parameters:
Amplitude: 0.242, Angular Frequency: 9305.311, Phase: 0.963, Offset: -0.002

Resistor Fit Parameters:
Amplitude: 0.241, Angular Frequency: 9305.316, Phase: 2.214, Offset: -0.002
```

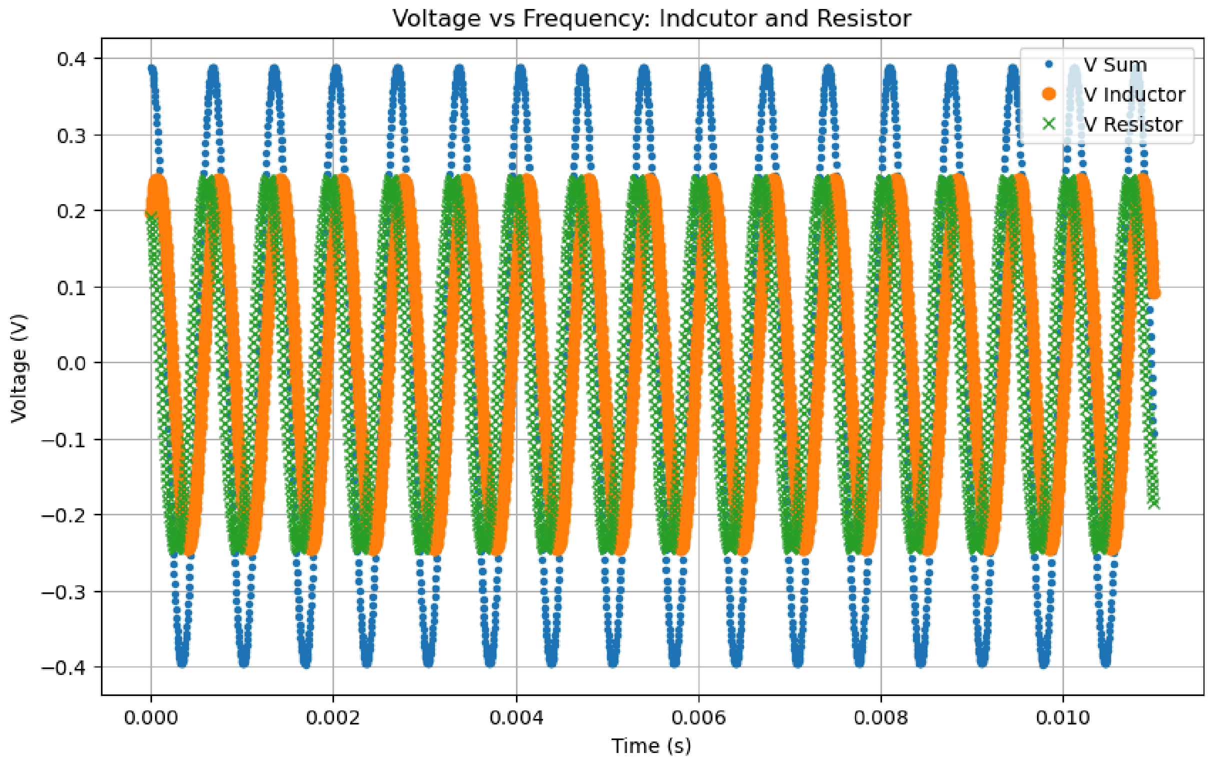
B) Sum

In [302…

```
voltage_sum = voltage_ind + voltage_res

print('Voltage sum MAX',max(voltage_sum))
print('Voltage Inductor MAX',max(voltage_ind))
print('Voltage Resisitor MAX',max(voltage_res))

plt.figure(figsize=(10,6))

plt.plot(T,voltage_sum, '.', label="V Sum")
plt.plot(T,voltage_ind, 'o', label="V Inductor")
plt.plot(T, voltage_res, 'x', label="V Resistor")
plt.xlabel("Time (s)")
plt.ylabel("Voltage (V)")
plt.title("Voltage vs Frequency: Indcutor and Resistor")
plt.legend()
plt.grid()
plt.show()
```

```
Voltage sum MAX 0.387932
Voltage Inductor MAX 0.240183
Voltage Resisitor MAX 0.239854
```

Loading [MathJax]/extensions/Safe.js



Q) Do they add up to what you expect for the power supply voltage

Q) What about the peak voltages VC and VR that you recorded in the table? Do they add up to the peak input voltage?

Measured from the Ossiloscope, the Voltage was 0.960 V peak-to-peak. Which is 0.48 V Amplitude . The graph shows that the summed wave just falls short of 0.4 V. This might seem controdictory but in fact it is not. Observe that currently our voltages concider the 33.4 Omhs internal resistance of the inductor and 99.1 Ohm resistor, the total peak impedance being aroud 132.5 Ohms. At these ranges the 50 Ohm internal resistance of the Function generator becomes relevant. Accounting for this the volatge being lower makes sence becuase by Kircoff's rules, volatge now will be divided with the Function Generator as well.

The Individual Peak Voltages do not sum to form the total voltage then Naturally, since a third voltage reading on the resistor inside the function generator must be done.

Error and Table

In [346…

```
# Error calculations expanded
def voltage_gain_error(x):
    return x * (50 / 10**6)

resolution_voltage = 20 / 2**16
random_error_voltage = 100 * (10**-6)
```

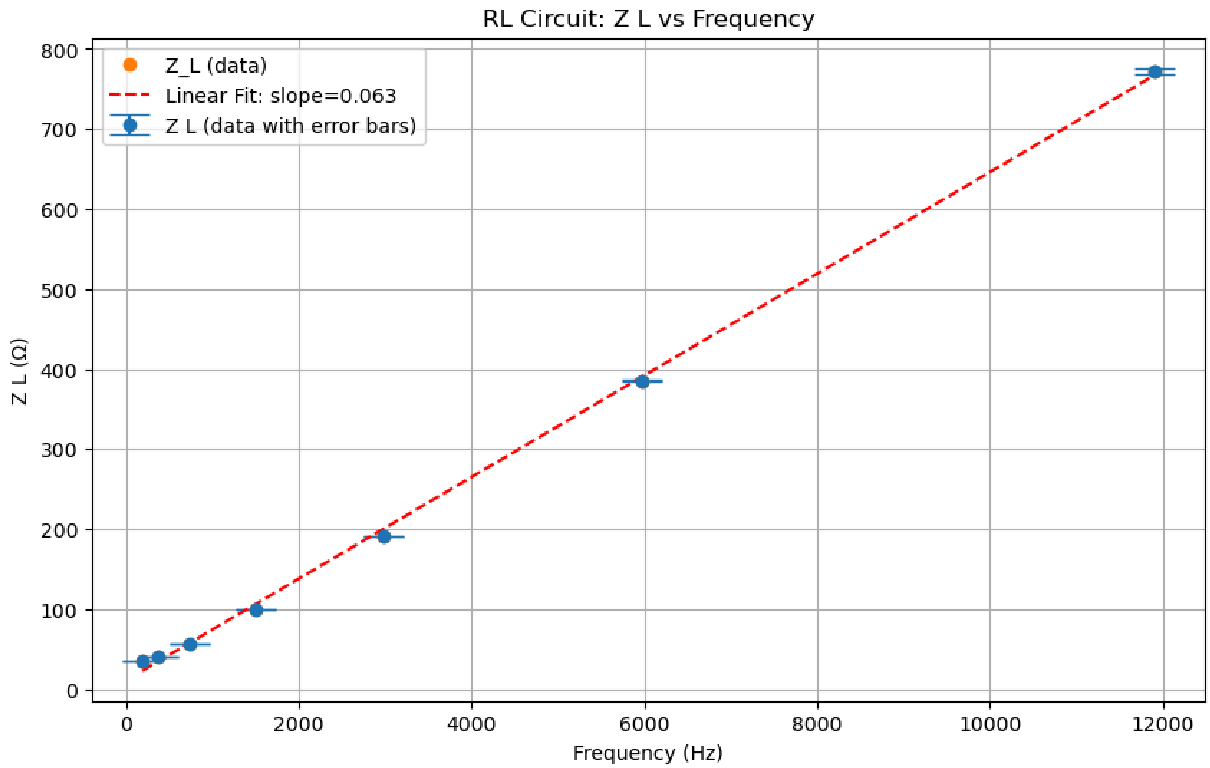
```python
# Calculate total errors for voltage_ind (Vl_total_error)
Vl_total_error = [
    (resolution_voltage**2 + random_error_voltage**2 + voltage_gain_error(v)
    for v in voltage_ind
]

# Calculate total errors for voltage_res (voltage_res_total_error)
voltage_res_total_error = [
    (resolution_voltage**2 + random_error_voltage**2 + voltage_gain_error(v)
    for v in voltage_res
]

# Propagate the error to the phase shift (phase_error)
phase_error = []
for i in range(len(f)):
    term1 = (T[i] * Vl_total_error[i] / voltage_ind[i])**2
    term2 = (T[i] * voltage_res_total_error[i] / voltage_res[i])**2
    propagated_error = 360 * f[i] * (term1 + term2)**0.5
    phase_error.append(propagated_error)

# Data Frame
data = {
    "Frequency (Hz)": f,
    "       voltage_ind (V )": [
        f"{voltage_ind[i]:.3f} ± {Vl_total_error[i]:.3e}" for i in range(ler
    ],
    "     voltage_res (V )": [
        f"{voltage_res[i]:.3f} ± {voltage_res_total_error[i]:.3e}" for i in
    ],
    "    Time Diff (s)": T,
    "    Phase (Degrees)": [
        f"{phase[i]:.3f} ± {phase_error[i]:.3f}" for i in range(len(phase_er
    ]
}

df = pd.DataFrame(data)

# Display the DataFrame
print(df.to_string(index=False))
```

```
 Frequency (Hz)        voltage_ind (V )      voltage_res (V )      Time Diff
(s)     Phase (Degrees)
          188.7       0.094 ± 3.212e-04     0.269 ± 3.214e-04          0.00
0275    -169.490 ± 0.068
          373.1       0.108 ± 3.212e-04     0.267 ± 3.214e-04          0.00
0270    -167.223 ± 0.117
          735.6       0.151 ± 3.212e-04     0.261 ± 3.214e-04          0.00
0195    -178.751 ± 0.127
         1493.0       0.239 ± 3.214e-04     0.239 ± 3.214e-04          0.00
0140    -171.994 ± 0.143
         2986.0       0.362 ± 3.217e-04     0.187 ± 3.213e-04          0.00
0080    -182.743 ± 0.166
         5972.0       0.448 ± 3.219e-04     0.115 ± 3.212e-04          0.00
0045    -182.743 ± 0.279
        11900.0       0.482 ± 3.220e-04     0.062 ± 3.212e-04          0.00
0055    -171.360 ± 1.233
```

C) Plot a graph of the reciprocal of the capacitor impedance $1/Z_C$ versus frequency.

In [370…

```
f_rl = np.array([188.7, 373.1, 735.6, 1493.0, 2986.0, 5972.0, 11900.0])
V_L_meas = np.array([0.0941309, 0.107618, 0.15071, 0.239196,
                     0.361564, 0.448405, 0.482287])
V_R_meas = np.array([0.26913, 0.266827, 0.260906, 0.238867,
                     0.186894, 0.115183, 0.0618942])

R_rl = 99.1  # Ohms

I_rl_error = [voltage_res_total_error[i] / R_rl for i in range(len(V_R_meas)

# Z_L errors
Zl_error = [
    Zl_meas[i] * ((Vl_total_error[i] / V_L_meas[i])**2 + (I_rl_error[i] / I_
    for i in range(len(V_L_meas))
]



def linear_func(x, m, b):
    return m*x + b

params_zl, cov_zl = curve_fit(linear_func, f_rl, Zl_meas)
slope_zl, intercept_zl = params_zl
slope_zl_err = np.sqrt(cov_zl[0,0])

L_extracted = slope_zl/(2*np.pi)
L_extracted_err = slope_zl_err/(2*np.pi)

# Plot Z_L vs frequency
plt.figure(figsize=(10,6))
plt.errorbar(f_rl, Zl_meas, yerr=Zl_error, fmt='o', label="Z L (data with er
plt.plot(f_rl, Zl_meas, 'o', label="Z_L (data)")
plt.plot(f_rl, linear_func(f_rl, slope_zl, intercept_zl), 'r--',
         label=f"Linear Fit: slope={slope_zl:.3f}")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Z L (Ω)")
plt.title("RL Circuit: Z L vs Frequency")
plt.legend()
plt.grid()
plt.show()

print(f"Slope(Z_L vs f)   = {slope_zl:.3f} ± {slope_zl_err:.3f}")
print(f"Extracted L       = {L_extracted:.4e} ± {L_extracted_err:.4e} H")
print(f"Expected L: 0.01 H")
print("The inductance values agree within the uncertainties")
```




```
Slope(Z_L vs f)    = 0.063 ± 0.001
Extracted L        = 1.0106e-02 ± 1.2491e-04 H
Expected L: 0.01 H
The inductance values agree within the uncertainties
```

Q) How can I_L be deduced from V_R, the voltage across the resistor?

In an RL circuit, the components are connected in series. By Kirchhoff's circuit laws, the current is constant, but the voltage is divided among the series components.

Derivation:

Since the resistor and inductor are in series, the current I_L (through the inductor) and I_R (through the resistor) are the same:

$ V_R = I R $

Rearranging for I:

$ I = \frac{V_R}{R} $

Using the above expression for current I, the impedance of the inductor Z_L can be determined as:

$ |Z_L| = \frac{V_L}{I} $

To extract the inductance L, i fit a straight line to Z_L vs frequency data. From the relationship:

$ Z_L = \omega L = 2 \pi f L $

Comparing with the linear equation $y = mx + c$, we identify:

$ \text{slope} (m) = 2 \pi L $

Solving for L:

$ L = \frac{\text{m}}{2 \pi} $

D) Phase

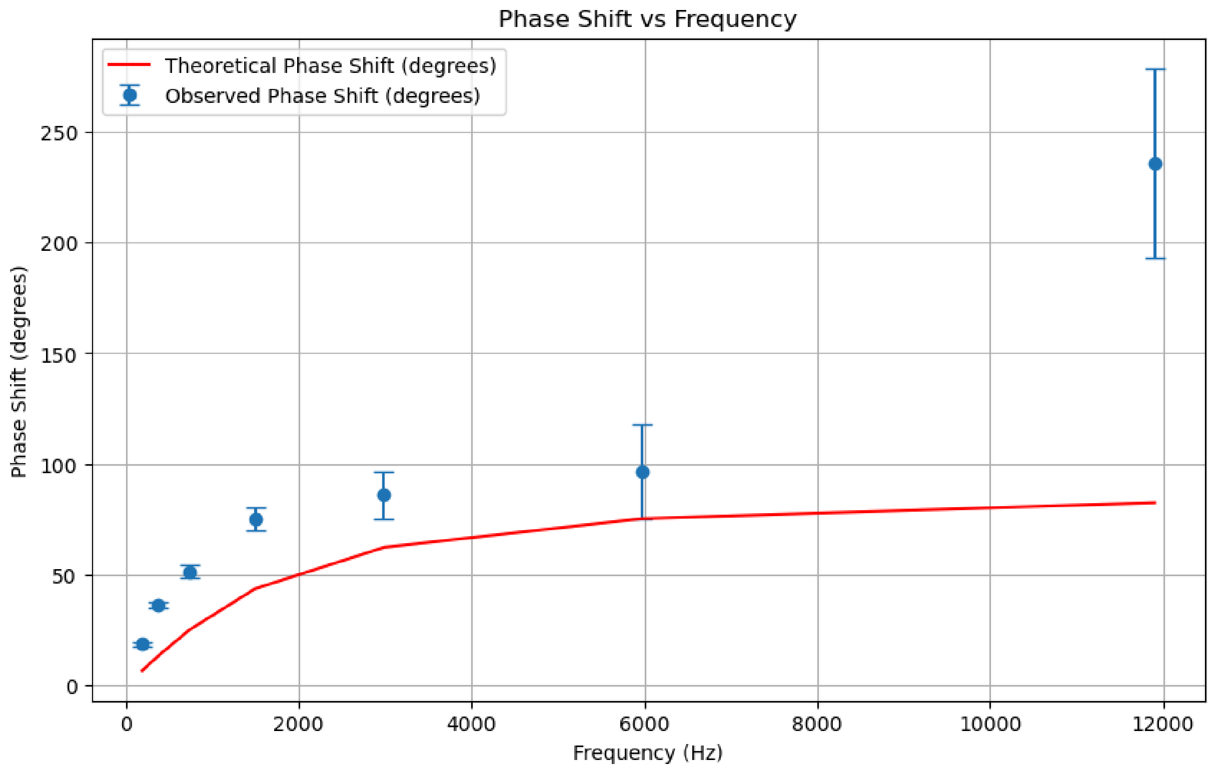
In [367…

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# Given data
f = np.array([188.7, 373.1, 735.6, 1.493 * 1000, 2.986 * 1000, 5.972 * 1000,
voltage_ind = np.array([0.0941309, 0.107618, 0.15071, 0.239196, 0.361564, 0.
voltage_res = np.array([0.26913, 0.266827, 0.260906, 0.238867, 0.186894, 0.1
T = np.array([0.052165 - 0.051890, 0.022320 - 0.022050, 0.013085 - 0.012890,
              0.005475 - 0.005335, 0.002430 - 0.002350, 0.002450 - 0.002405,
              0.00105 - 0.000995])
internal_resistance = 10  # Ohms

# Error parameters
def voltage_gain_error(x):
    return x * (50 / 10**6)

resolution_voltage = 20 / 2**16
random_error_voltage = 100 * (10**-6)
time_resolution = 10 * (10**-6)  # Time resolution (in seconds)

# Calculate total errors for voltage_ind (Vl_total_error)
Vl_total_error = [
    (resolution_voltage**2 + random_error_voltage**2 + voltage_gain_error(v)
    for v in voltage_ind
]

# Calculate total errors for voltage_res (voltage_res_total_error)
voltage_res_total_error = [
    (resolution_voltage**2 + random_error_voltage**2 + voltage_gain_error(v)
    for v in voltage_res
]

# Time error (all entries have same resolution)
T_error = np.full(len(T), time_resolution)

# Calculate Z_L (impedance of inductor)
Z_L = voltage_ind / (voltage_res / internal_resistance)
```

Loading [MathJax]/extensions/Safe.js

```python
# Propagate error for Z_L
Z_L_error = [
    Z_L[i] * ((Vl_total_error[i] / voltage_ind[i])**2 + (voltage_res_total_e
    for i in range(len(Z_L))
]

# Linear fit: Z_L vs frequency
def linear(x, m, b):
    return m * x + b

params, covariance = curve_fit(linear, f, Z_L)
slope_fit, intercept_fit = params
slope_uncertainty = np.sqrt(covariance[0, 0])

# Calculate inductance
L_fit = slope_fit / (2 * np.pi)
L_uncertainty = slope_uncertainty / (2 * np.pi)

# Calculate observed phase shift
phi_observed = 360 * f * T  # Observed phase shift in degrees

# Propagate error for phase shift
phase_error = [
    phi_observed[i] * ((T_error[i] / T[i])**2)**0.5
    for i in range(len(phi_observed))
]

# Calculate theoretical phase shift
omega = 2 * np.pi * f
phi_theoretical = np.degrees(np.arctan((omega * L_fit) / internal_resistance

# Print results
print(f"Slope: {slope_fit:.4e} ± {slope_uncertainty:.4e}")
print(f"Inductance from slope (L = slope / 2pi): {L_fit:.4e} ± {L_uncertaint

# Plot phase shift
plt.figure(figsize=(10, 6))
plt.errorbar(f, phi_observed, yerr=phase_error, fmt='o', label="Observed Pha
plt.plot(f, phi_theoretical, 'r-', label="Theoretical Phase Shift (degrees)"
plt.xlabel("Frequency (Hz)")
plt.ylabel("Phase Shift (degrees)")
plt.title("Phase Shift vs Frequency")
plt.legend()
plt.grid()
plt.show()
```

```
Slope: 6.4075e-03 ± 7.9194e-05
Inductance from slope (L = slope / 2pi): 1.0198e-03 ± 1.2604e-05 H
```



Discrepancies:

- The observed phase shift deviates slightly at higher frequencies due to the unaccounted resistance. And this keeps increasing becuase of the unaccounted internal resistance.
- Measurement errors and internal errors in $ T_{\text{diff}} $ contribute to discrepancies at low frequencies.