In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
from UliEngineering.EngineerIO import format_value
from si_prefix import si_format
import plecs_helper as helper
%matplotlib
%matplotlib inline

# Imports and setup
from pint import UnitRegistry
from scipy.signal import find_peaks
from scipy.optimize import fsolve

# pandas display using scientific notation
# pd.set_option('display.float_format', lambda x: f'{x:.3e}')

# use pint
units = UnitRegistry()
units.default_format = "~P.2f"

def to_decibels(arr, dc_gain = 0):
    return np.log10(np.abs(arr)) * 20 + dc_gain
```

```
Using matplotlib backend: TkAgg
```

Independent Steps

Hi Beat! I found the things wrong with my original analysis. Because the frequency ω is measured in Hertz, I thought it was acceptable to use the impedance equation $Z = \frac{1}{j\omega C}$ for a capacitor. However, the factor of 2π in the equation is actually important for converting Hertz *into* radians per second so that the relationship becomes valid. The correct equation is $Z = \frac{1}{2\pi j\omega C}$. This fixed a vertical and horizontal offset of my Z_c bode plot.

The other thing wrong was that I used the equation $\mathrm{dB} = 20 \cdot \ln(a)$ instead of the correct $\mathrm{dB} = 20 \cdot \log_{10}(a)$. This is a particularly pernicious mistake because numpy uses the function `log` for ln. This is why I was seeing slopes for Z_c which were a factor of three steeper than expected from the equation.

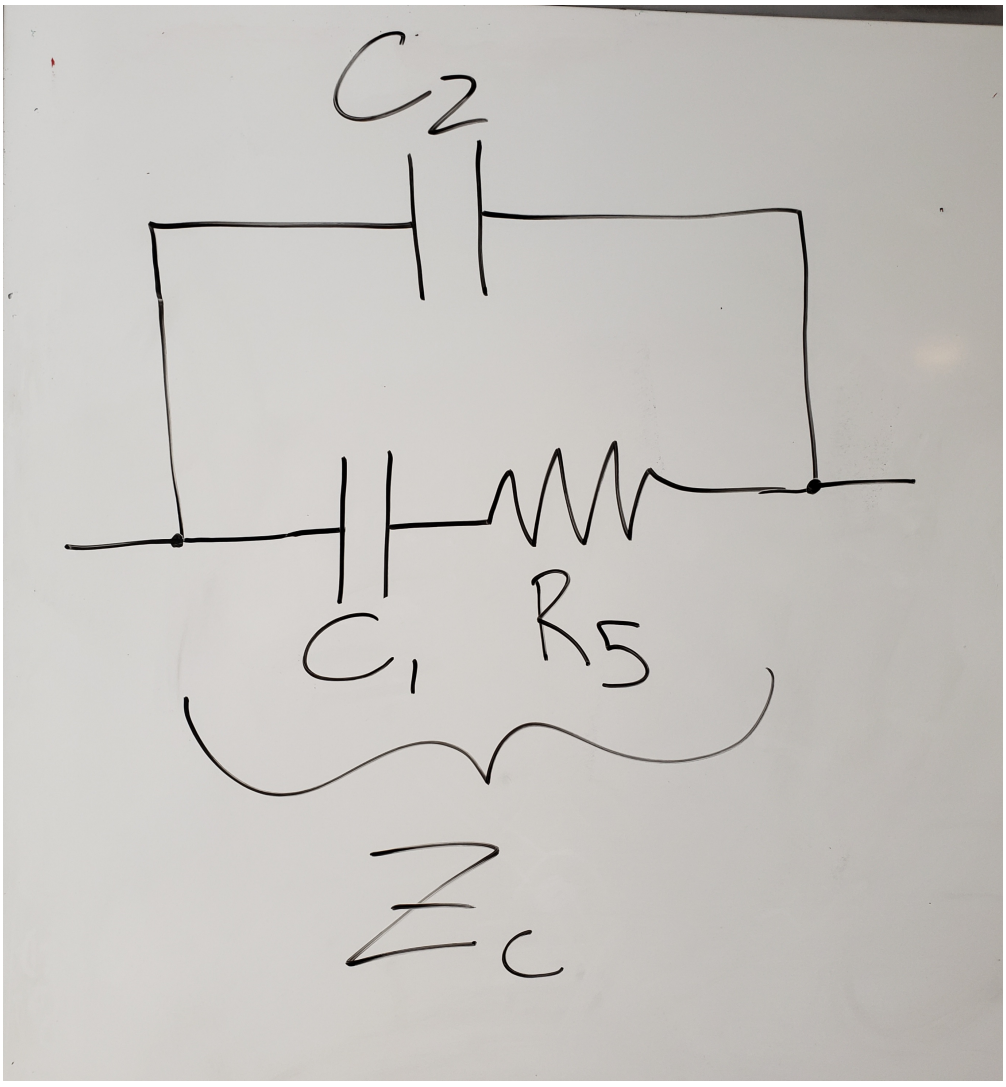
I think that wrestling with this for many hours over the weekend and last night counts as significant independent steps. I had to look up old ISIM notes to derive the op-amp gain equation and puzzle over whether there were any other transfer functions or unaccounted for gains elsewhere in the circuit. I feel like I understand our hardware circuit and schematic better as a result.

Lab 10

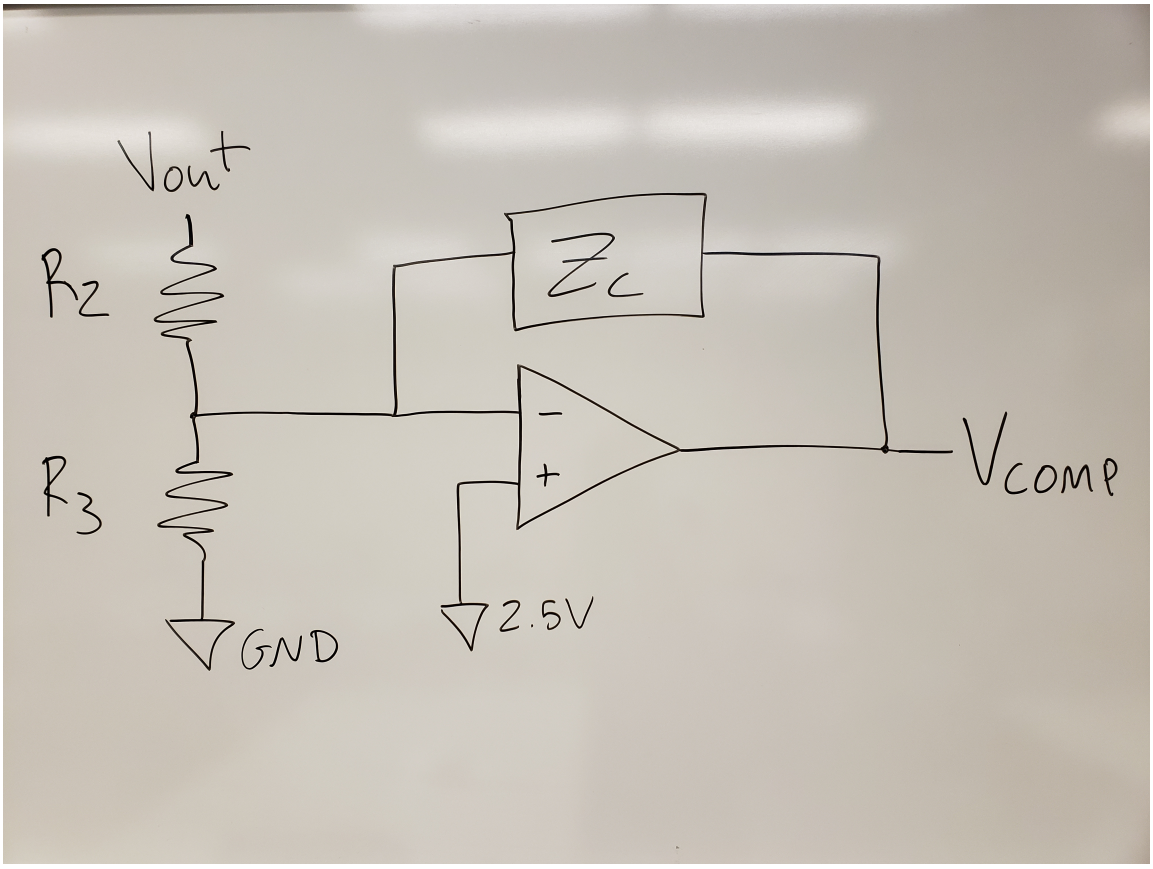
Ian Eykamp

As expected, the output voltage jumped to a value of 9.60V and an input current of 1.30A after reaching steady state (very quickly on a human timescale). This is because the voltage divider `R2 = 4.3kOhm`, `R3 = 1.5kOhm` creates a middle node voltage of 2.5V when the input voltage is 9.67V.

For Lab 10, we used the provided feedback components `C1 = 47nF`, `R5 = 510Ohm`, and `C2 = 10nF`, with `C2` in parallel with `(C1 + R5)` in series.



The impedance value of Zc is obtained directly by substituting in $Z = R$ for the resitor and $Z = \frac{1}{2\pi j\omega C}$ for the capacitors. By parallel and series addition, we have $Z_c = \frac{1}{(\frac{1}{(R_5 + \frac{1}{2\pi j\omega C_1})} + \frac{1}{(\frac{1}{2\pi j\omega C_2})})}$.



The op-amp is configured as an inverting amplifier with gain $\frac{Z_c}{R_2}$, referenced to 2.5V. The bottom resistor `R3` does not contribute to the transfer function, but it does affect the DC `V-` voltage. Fortunately, when Vout = 10V, the `R3` - `R4` voltage divider is also referenced to 2.5V, so the amplifier will work as expected without any offset.

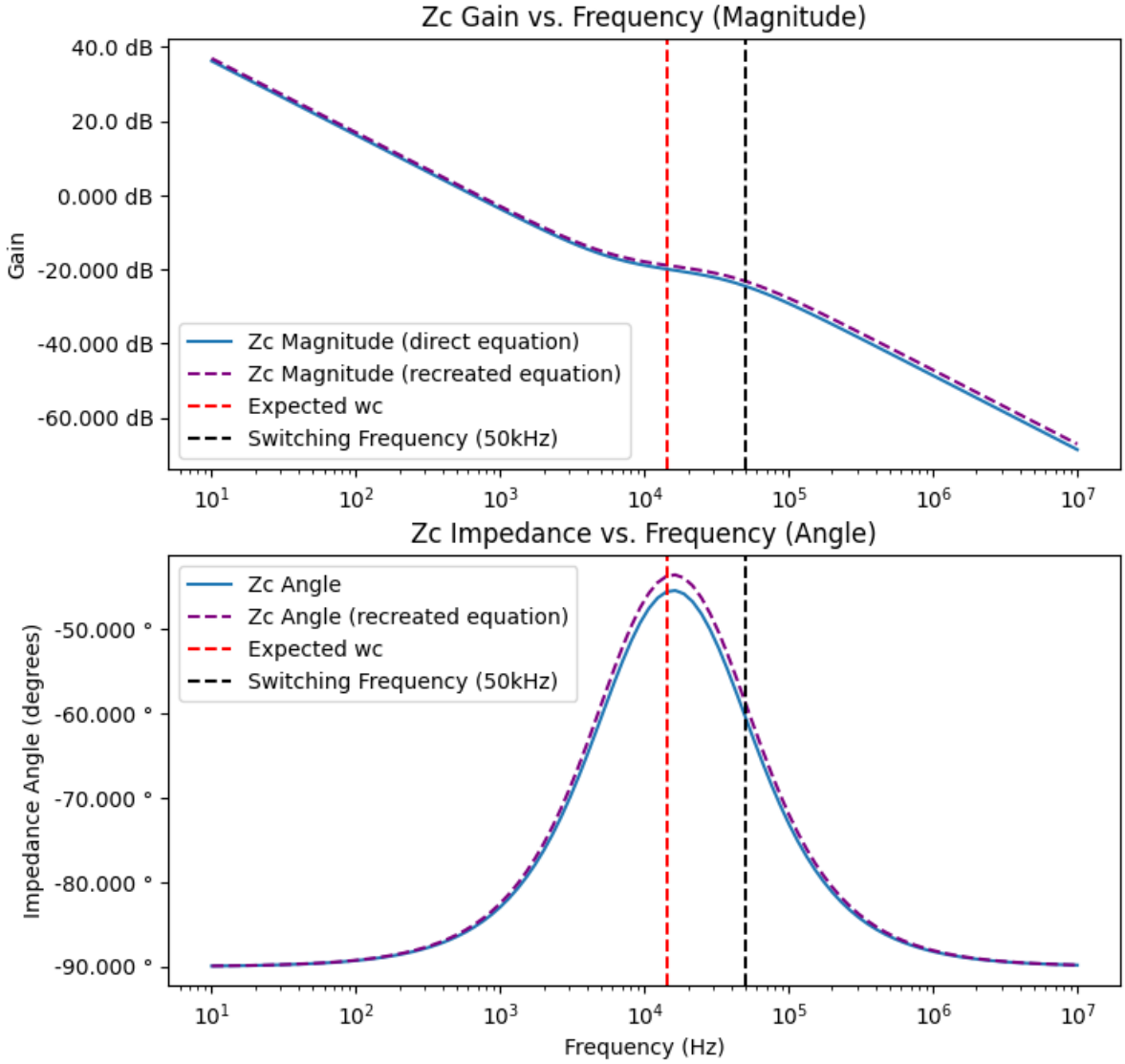
The resulting bode plot indicates Zc has an integrator (pole at $\omega = 0Hz$) as well as a pole-zero combination above the switching frequency (zero at $\omega = 40kHz$, pole at $\omega = 250kHz$). The dashed line shows a transfer function I recreated with these characteristics; it matches nearly perfectly with the direct equation.

In [ ]:
```
R2 = 4.3 # * units.kiloohm
R3 = 1.5 # * units.kiloohm
Vtarget = 2.5 * (R2 + R3) / R3 # * units.volt
print(f"The expected output voltage given the voltage divider resistors is: {si_for

C1 = 47e-9 # * units.nanofarad
C2 = 10e-9 # * units.nanofarad
R5 = 510 # * units.ohm
w = np.logspace(1, 7, 100) # * units.hertz
def get_Zc(my_w):
    # my_Zc = 1 / (1 / (R5 + 1 / (1j * my_w * C1)) + 1 / (1 / (1j * my_w * C2))) /
    my_Zc = 1 / (1 / (R5 + 1 / (2 * np.pi * 1j * my_w * C1)) + 1 / (1 / (2 * np.pi
    return my_Zc
Zc = get_Zc(w) # direct equation
Zc2 = (2 * np.pi * 1j * w / (40e3) + 1) / ((2 * np.pi * 1j * w / (1.9e7)) * (2 * np

expected_wc = 1 / (R5 * np.sqrt(C1 * C2)) / 2 / np.pi # to get it into Hz
```

```
print(f"Lead element center voltage (ω_c): {si_format(expected_wc, precision = 2)}H

fig, (ax1, ax2) = plt.subplots(nrows = 2, ncols = 1, sharex = False, sharey = False
helper.axes_labels("", "Hz", "Gain", "dB", title = "Zc Gain vs. Frequency (Magnitud
ax1.semilogx(w, to_decibels(Zc), linestyle = "solid", label = "Zc Magnitude (direct
ax1.semilogx(w, to_decibels(Zc2), linestyle = "dashed", color = "purple", label = "
ax1.axvline(x = expected_wc, linestyle = "dashed", color = "red", label = "Expected
ax1.axvline(x = 50e3, linestyle = "dashed", color = "black", label = "Switching Fre
ax1.legend(loc = "lower left")

helper.axes_labels("Frequency (Hz)", "Hz", "Impedance Angle (degrees)", "°", title
ax2.semilogx(w, np.angle(Zc, deg = True), linestyle = "solid", label = "Zc Angle")
ax2.semilogx(w, np.angle(Zc2, deg = True), linestyle = "dashed", color = "purple",
ax2.axvline(x = expected_wc, linestyle = "dashed", color = "red", label = "Expected
ax2.axvline(x = 50e3, linestyle = "dashed", color = "black", label = "Switching Fre
ax2.legend(loc = "upper left")
# ax2.set_ylim(-100, -80)
```

```
The expected output voltage given the voltage divider resistors is: 9.67 V
Lead element center voltage (ω_c): 14.39 kHz
```

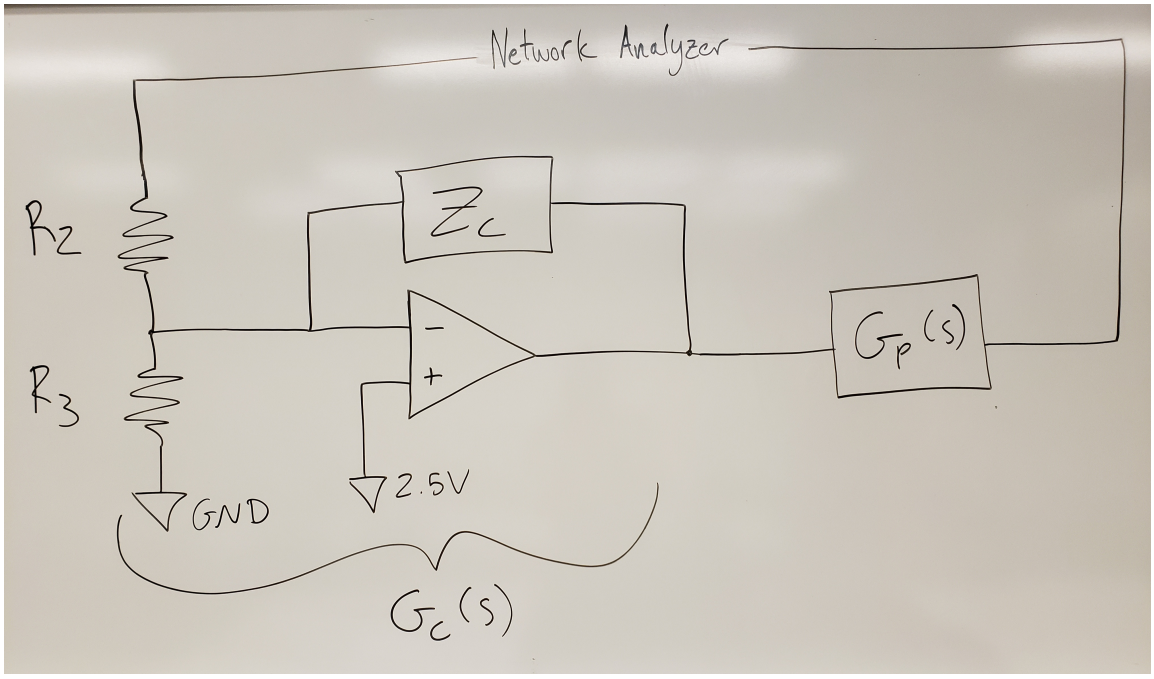
Out[ ]: `<matplotlib.legend.Legend at 0x255043d69d0>`



The Zc network consists of an integrator and a lead element centered around $\omega_c = 14.4kHz$, which matches the value derived from the components by $\omega_c = \frac{1}{R_5\sqrt{(C1 \cdot C2)}}$.

On the magnitude plot, this appears as a never-ending increase in magnitude towards low frequencies and a slight horizontal jog at ω_c. On the angle plot, this appears as a constant -90° phase shift with a large phase boost centered around ω_c. This will help increase the phase margin at ω_c, which is chosen to be the cross-over frequency of the compensator.

Bode Plots



In [ ]:
```
# Import data
open_loop_bode = pd.read_csv("bode_plots/open-loop.csv")
closed_loop_bode = pd.read_csv("bode_plots/closed-loop.csv")
print(open_loop_bode.columns)
```

```
Index(['Frequency (Hz)', 'Channel 1 Magnitude (dB)',
       'Channel 2 Magnitude (dB)', 'Channel 2 Phase (deg)'],
      dtype='object')
```

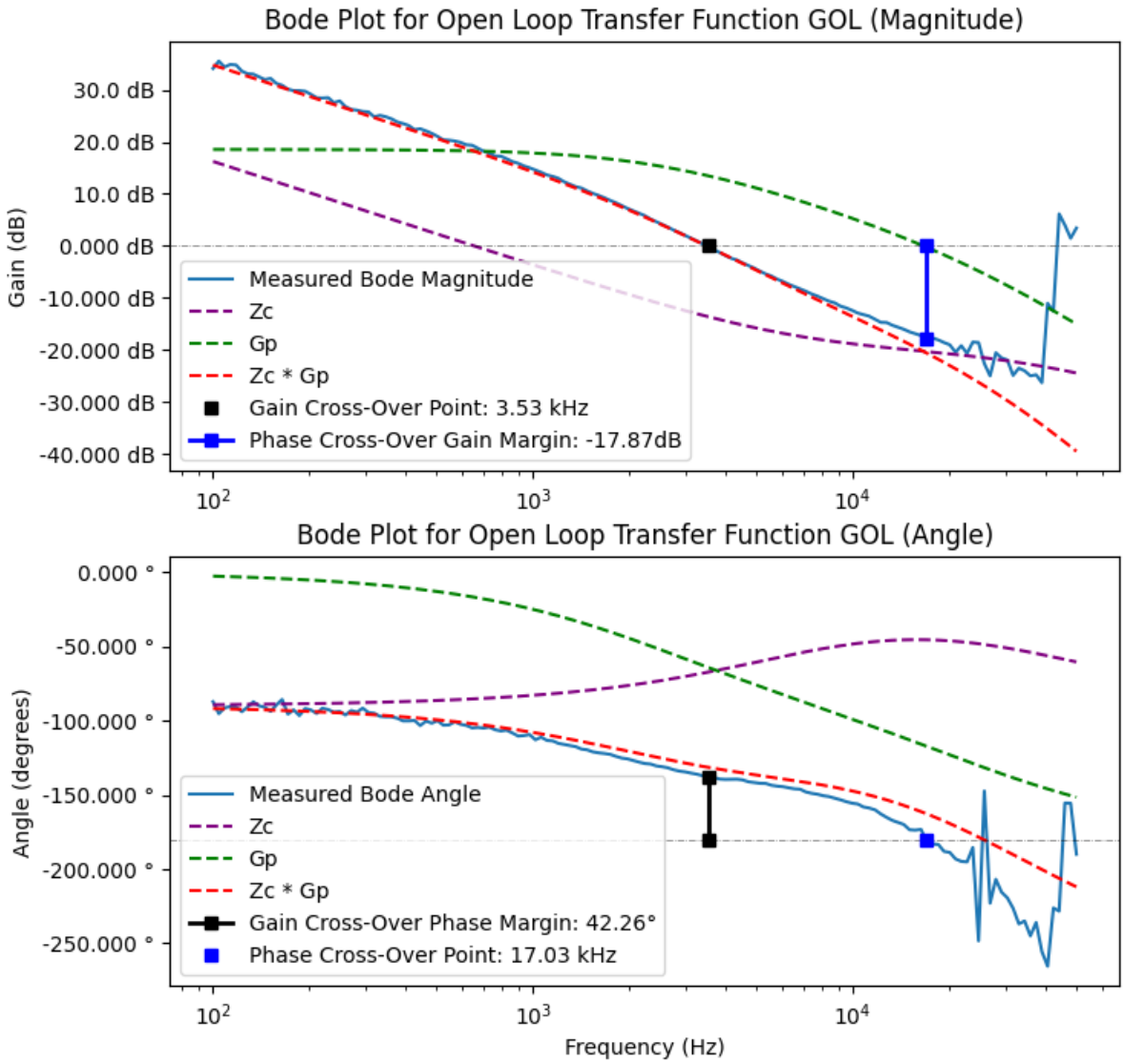
Open-Loop Transfer Function (GOL)

In [ ]:
```
w = open_loop_bode["Frequency (Hz)"]
Zc = get_Zc(w)
dc_gain = 18.58 # dB
# G = 1 / ((w * 1j / (1.8e3)) + 1) / ((w * 1j / (2e4)) + 1) * 10**(dc_gain / 20) #
G = 1 / ((w * 1j / 2.4e3) + 1) / ((w * 1j / 2.4e4) + 1) * 10**(dc_gain / 20) # opti

open_loop_gc = open_loop_bode[(open_loop_bode["Channel 2 Magnitude (dB)"] < 0) & (o
open_loop_gc_freq = open_loop_gc["Frequency (Hz)"]
open_loop_gc_phase_margin = 180 - np.abs(open_loop_gc["Channel 2 Phase (deg)"])
print(f"Open-loop gain cross-over frequency: {si_format(open_loop_gc_freq, precisio
```

```
print(f"Open-loop gain cross-over phase margin: {np.round(open_loop_gc_phase_margin

open_loop_pc = open_loop_bode[(open_loop_bode["Channel 2 Phase (deg)"] < -180)].ilo
open_loop_pc_freq = open_loop_pc["Frequency (Hz)"]
open_loop_pc_gain_margin = open_loop_pc["Channel 2 Magnitude (dB)"]
print(f"Open-loop phase cross-over frequency: {si_format(open_loop_pc_freq, precisi
print(f"Open-loop phase cross-over gain margin: {np.round(open_loop_pc_gain_margin,


fig, (ax1, ax2) = plt.subplots(nrows = 2, ncols = 1, sharex = False, sharey = False
helper.axes_labels("", "Hz", "Gain (dB)", "dB", title = "Bode Plot for Open Loop Tr
ax1.semilogx(open_loop_bode["Frequency (Hz)"], open_loop_bode["Channel 2 Magnitude
ax1.semilogx(open_loop_bode["Frequency (Hz)"], to_decibels(Zc), linestyle = "dashed
ax1.semilogx(open_loop_bode["Frequency (Hz)"], to_decibels(G), linestyle = "dashed"
ax1.semilogx(open_loop_bode["Frequency (Hz)"], to_decibels(G * Zc), linestyle = "da
ax1.axhline(y = 0, color = "black", linestyle = "dashdot", linewidth = 0.5, alpha =
ax1.semilogx(open_loop_gc_freq, 0, marker = 's', markersize = 6, markerfacecolor =
ax1.semilogx([open_loop_pc_freq, open_loop_pc_freq], [0, open_loop_pc_gain_margin],
ax1.legend(loc = "lower left")

helper.axes_labels("Frequency (Hz)", "Hz", "Angle (degrees)", "°", title = "Bode Pl
ax2.semilogx(open_loop_bode["Frequency (Hz)"], np.unwrap(open_loop_bode["Channel 2
ax2.semilogx(open_loop_bode["Frequency (Hz)"], np.unwrap(np.angle(Zc, deg = True),
ax2.semilogx(open_loop_bode["Frequency (Hz)"], np.unwrap(np.angle(G, deg = True), p
ax2.semilogx(open_loop_bode["Frequency (Hz)"], np.unwrap(np.angle(G * Zc, deg = Tru
ax2.axhline(y = -180, color = "black", linestyle = "dashdot", linewidth = 0.5, alph
ax2.semilogx([open_loop_gc_freq, open_loop_gc_freq], [-180, -180 + open_loop_gc_pha
ax2.semilogx(open_loop_pc_freq, -180, marker = 's', markersize = 6, markerfacecolor
ax2.legend(loc = "lower left")
```

```
Open-loop gain cross-over frequency: 3.53 kHz
Open-loop gain cross-over phase margin: 42.26°
Open-loop phase cross-over frequency: 17.03 kHz
Open-loop phase cross-over gain margin: -17.87dB
```

Out[ ]: <matplotlib.legend.Legend at 0x255065bf280>



This bode plot was taken on the upstream (closer to Vout) side of the feedback loop *with reference to* the downstream (closer to VFB) side of the injected current. Thus, the *response* of the system in this configuration corresponds with the **open-loop** transfer function G_{OL}, since the feedback loop is **factored out**.

As expected, the open-loop transfer function is a product of the compensator gain Z_c and plant transfer function $G_p(s)$ obtained in Lab 9. It follows a trajectory towards infinite gain at DC from the compensator's integrating capacitor `C2`. The primary pole of the plant corresponds loosely with the plateau caused by the lead element so that the gain decreases roughly linearly (on the log scale) with frequency, at least up to around the switching frequency. The theoretical and measured curves match quite well for both the angle and the magnitude up to about 10kHz, after which they diverge significantly. This is not a problem, since we do not care about the transfer function very close to the switching frequency, and our controller operates in a region where the theoretical and experimental curves match.

The gain-crossover phase margin is 42°, which is large enough to be stable, but still well below the industry standard of 60°. The phase-crossover gain margin is 17dB, which is also

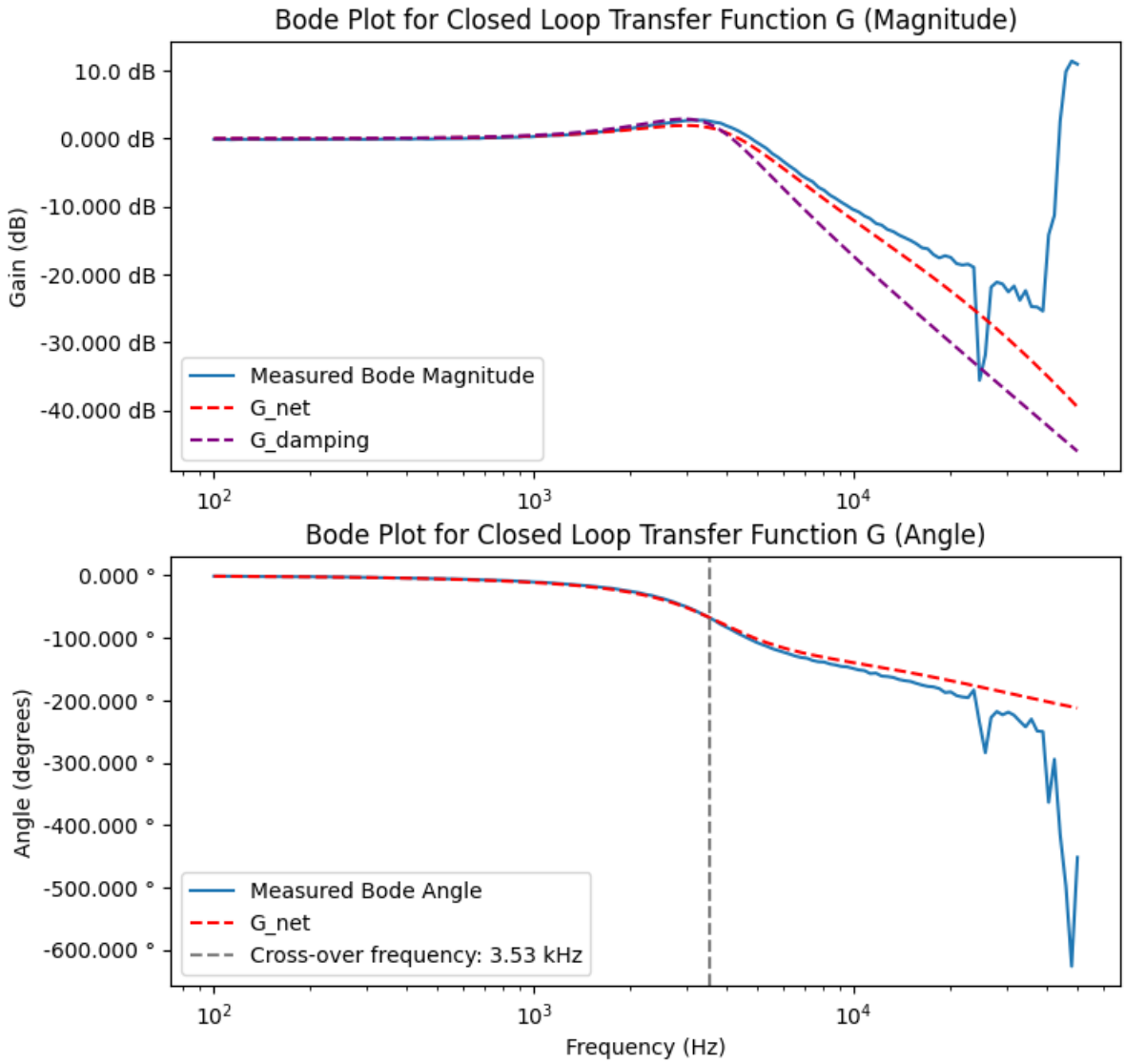
sufficiently large.

Closed-Loop Transfer Function (G)

In [ ]:
```
w = closed_loop_bode["Frequency (Hz)"]
G_new = G * Zc / (1 + G * Zc)

GOL_gc = np.exp(1j * np.deg2rad(-180 + open_loop_gc_phase_margin))
G_gc = GOL_gc / (1 + GOL_gc)
print(f"Gain at cross-over frequency: {np.round(np.abs(G_gc), 3)} * exp({np.round(n
print(f"That's {np.round(to_decibels(np.abs(G_gc)), 2)} dB and {np.round(np.angle(G

zeta = 0.39
w_gc = open_loop_gc_freq
G_damping = w_gc ** 2 / ((1j * w) ** 2 + 2 * zeta * w_gc * 1j * w + w_gc ** 2)
print(f"Damping ratio used: {zeta}")
print(f"Maximum gain of damped equation: {np.round(max(np.abs(G_damping)), 3)}")

fig, (ax1, ax2) = plt.subplots(nrows = 2, ncols = 1, sharex = False, sharey = False
helper.axes_labels("", "Hz", "Gain (dB)", "dB", title = "Bode Plot for Closed Loop
ax1.semilogx(closed_loop_bode["Frequency (Hz)"], closed_loop_bode["Channel 2 Magnit
ax1.semilogx(closed_loop_bode["Frequency (Hz)"], to_decibels(G_new), linestyle = "d
ax1.semilogx(w, to_decibels(G_damping), linestyle = "dashed", color = "purple", lab
# ax1.axhline(y = to_decibels(np.abs(G_gc)), linestyle = "dashed", color = "black",
# ax1.axvline(x = open_loop_gc_freq, linestyle = "dashed", color = "grey", label =
ax1.legend(loc = "lower left")

helper.axes_labels("Frequency (Hz)", "Hz", "Angle (degrees)", "°", title = "Bode Pl
ax2.semilogx(closed_loop_bode["Frequency (Hz)"], np.unwrap(closed_loop_bode["Channe
ax2.semilogx(closed_loop_bode["Frequency (Hz)"], np.unwrap(np.angle(G_new, deg = Tr
ax2.axvline(x = open_loop_gc_freq, linestyle = "dashed", color = "grey", label = f"
# ax2.axhline(y = -90, linestyle = "dashed", color = "black")
ax2.legend(loc = "lower left")
```

```
Gain at cross-over frequency: 1.387 * exp(-1.2j)
That's 2.84 dB and -68.87 degrees
Damping ratio used: 0.39
Maximum gain of damped equation: 1.392
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x25504316820>
```



This bode plot was taken on the upstream (closer to Vout) side of the feedback loop with reference to the *source* of the perturbation. The feedback loop is left intact and not accounted for separately, so this bode plot represents the response of the **closed-loop system** to a perterbation on the feedback line corresponding to a change in Vout.

According to Black's formula, the closed-loop (behavior after being feedback-connected) transfer function $G(s)$ is obtained from the open-loop (behavior while disconnected) $G_{OL}(s)$ by $G(s) = \frac{G_{OL}(s)}{1+G_{OL}(s)}$. The integrator component of G_{OL} creates a unity gain at DC, because $\lim_{s\to 0} \frac{G_{OL}(s)}{1+G_{OL}(s)} = \lim_{G_{OL}\to\infty} \frac{G_{OL}}{1+G_{OL}} = 1$.

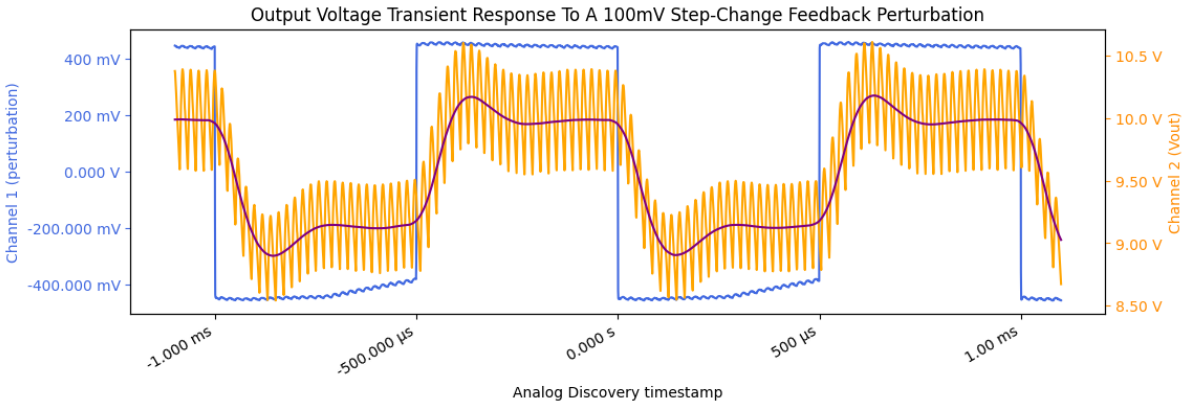
The closed-loop transfer function resembles a second-order damped system with a double pole at the gain cross-over frequency. The zeros are located at $G_{OL} = -1$, so the phase margin becomes important. If $\angle G_{OL}(\omega = \omega_{gc}) == -180^\circ$, then there would be infinite gain; if $\angle G_{OL}(\omega = \omega_{gc}) == 0^\circ$, then there would be a gain of $\frac{1}{2} = -6dB$. We have an angle of -138°, which is in between but closer to the pole, so we see a small hump of around +3dB at the cross-over frequency.

The transfer function for a system with a damped second-order pole is given by $G(s) = \frac{{\omega_{gc}}^2}{s^2 + 2\zeta\omega_{gc}s + {\omega_{gc}}^2}$, where $\omega_0 = \omega_{gc}$ is the location of the pole (assumed to be at the gain cross-over frequency). I tuned the parameter ζ until I reached a similar peak hump value. The ζ value that achieved the best fit was $\zeta = 0.39$.

In [ ]:
```
df_step_response = pd.read_csv("bode_plots/step-resp.csv")

# print(df_step_response.head(5))

def moving_average(x, w):
    w = int(w / 2) * 2
    convolution = np.convolve(x, np.ones(w), 'full') / w
    return convolution[int(w/2):-int(w/2)+1]

df_step_response["Channel 1 (V)"] = -df_step_response["Channel 1 (V)"]
df_step_response["Channel 2 (V)"] = -df_step_response["Channel 2 (V)"]
# df_step_response["Ch2_moving_average"] = moving_average(df_step_response["Channel

peak_idx = find_peaks(df_step_response["Channel 2 (V)"])[0]
trough_idx = find_peaks(-df_step_response["Channel 2 (V)"])[0]
df_peak_trace = df_step_response.copy().iloc[np.sort(np.concatenate((peak_idx, trou
df_peak_trace["peak_moving_average"] = np.convolve(df_peak_trace["Channel 2 (V)"],
df_step_response["Ch2_moving_average"] = moving_average(np.interp(df_step_response[

df_zoom = df_step_response[(df_step_response["Time (s)"] > -1.1e-3) & (df_step_resp
# df_peak_trace_zoom = df_peak_trace_2[(df_peak_trace_2["Time (s)"] > -1.1e-3) & (d

overshoot = np.max(df_zoom["Ch2_moving_average"]) # 10.6 # V
top_steady_state = 10 # V
bottom_steady_state = 9.13 # V
undershoot = np.min(df_zoom["Ch2_moving_average"]) # 8.5 # V
output_swing = top_steady_state - bottom_steady_state

overshoot_ratio = (overshoot - top_steady_state) / output_swing
zeta_from_overshoot = np.sqrt(np.log(overshoot_ratio) ** 2 / (np.pi ** 2 + np.log(o
print(f"Output swing: {si_format(output_swing, precision = 2)}V for a 100 mV pertur
print(f"That's a gain of {np.round(output_swing / 0.1, 2)}")
print(f"It overshoots by {si_format(overshoot - top_steady_state, precision = 2)}V
print(f"Zeta from overshoot: {np.round(zeta_from_overshoot, 2)}")
print(f"Zeta from earlier (for comparison): {np.round(zeta, 2)}")

fig, (ax1_left) = plt.subplots(nrows = 1, ncols = 1, sharex = False, sharey = True,
ax1_right = ax1_left.twinx()
fig.autofmt_xdate()
helper.axes_labels("Analog Discovery timestamp", "s", "Channel 1 (perturbation)", "
helper.axes_labels("Analog Discovery timestamp", "s", "Channel 2 (Vout)", "V", titl
ax1_left.plot(df_zoom["Time (s)"], df_zoom["Channel 1 (V)"], color = "royalblue", l
ax1_right.plot(df_zoom["Time (s)"], df_zoom["Channel 2 (V)"], color = "orange", lab
ax1_right.plot(df_zoom["Time (s)"], df_zoom["Ch2_moving_average"], color = "purple"
# ax1_right.axhline(y = overshoot, linestyle = "dashed", color = "black", label = "
# ax1_right.axhline(y = top_steady_state, linestyle = "dashed", color = "black", la
# ax1_right.axhline(y = bottom_steady_state, linestyle = "dashed", color = "black",
# ax1_right.axhline(y = undershoot, linestyle = "dashed", color = "black", label =
ax1_left.yaxis.label.set_color('royalblue')
```

```
ax1_left.tick_params(axis='y', colors='royalblue')
ax1_right.tick_params(axis='y', colors='darkorange')
ax1_right.yaxis.label.set_color('darkorange')
```

```
Output swing: 870.00 mV for a 100 mV perturbation
That's a gain of 8.7
It overshoots by 181.34 mV and undershoots by 232.71 mV
Zeta from overshoot: 0.45
Zeta from earlier (for comparison): 0.39
```



A 100mV square-wave perturbation was applied to the feedback loop. The output voltage had time to settle between each cycle, so the response can be analyzed like a step response. The output voltage ripple (orange) is inherent to the power converter, so what we care about is the DC output voltage response (purple). The output voltage is centered around 9.6V and swings by around 870mV in response to the 100mV feedback perturbation; this is in line with the gain of the plant analyzed in Lab 9, in which the output voltage had a gain of 8.49 relative to `V_COMP`.

According to this resource, the damping ratio ζ can be obtained from the overshoot ratio $O = \frac{\text{overshoot}}{\text{output swing}}$ by the equation $\zeta = \sqrt{(\frac{ln^2(O)}{\pi^2+ln^2(O)})}$. Plugging in the values for overshoot and output swing yields a damping ratio of $\zeta = 0.45$. This is very similar to the damping ratio of 0.39 found earlier.

Overall, the compensator transfer function stands to be improved by increasing the phase margin and optimizing the damping ratio. The industry standard for the phase margin is 60°, whereas we achieved 42°, and the optimal damping occurs at 0.707, whereas our damping ratio is around 0.45. These aspects are related, because a greater phase margin at the cross-over frequency would get us further away from the open-loop pole at $G_{OL} = -1$, which would reduce the ringing at the cross-over frequency (exhibit greater damping) and make the system more stable.