(Alex) Xi Zeng, 1007099373 no partner

In [1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

A Notch Digital Filter (6 marks, 3/2/1)

A notch filter is a filter that passes almost all frequencies with unit amplitude, except for a narrow range of frequencies centered on the rejection frequency f_0 (at which frequency nothing is passed).

A simple notch digital filter has a z-transform

$$W(z) = MF(z)F(z^*)^* = M\frac{z-q}{z-p}\frac{z-q^*}{z-p^*}$$

where M is a normalization factor, $q = e^{-i2\pi f_0/f_s}$, $p = (1+\epsilon)q$, f_s is the sampling rate ($f_s = 1/\Delta$) and ϵ is a small positive number ($0 < \epsilon \ll 1$).

1. What are the poles and zeros of this system? Where are they located with respect to the unit circle? Is this a stable system? Is this filter real?

The pole of the system are given by the roots of the denominator of the transfer function W(z), therefore, the poles are p1 = p and p2 = p*, where $p = (1+\epsilon)q$ and $q = e^{-i2\pi f_0/f_s}$

The zero of the system are given by the roots of the numerator of the transfer function W(z), therefore, the zeros are z1 = q and z2 = q*, where $q = e^{-i2\pi f_0/f_s}$

To find the locations of the poles and zeros:

$$|p| = |1+\epsilon||q|$$

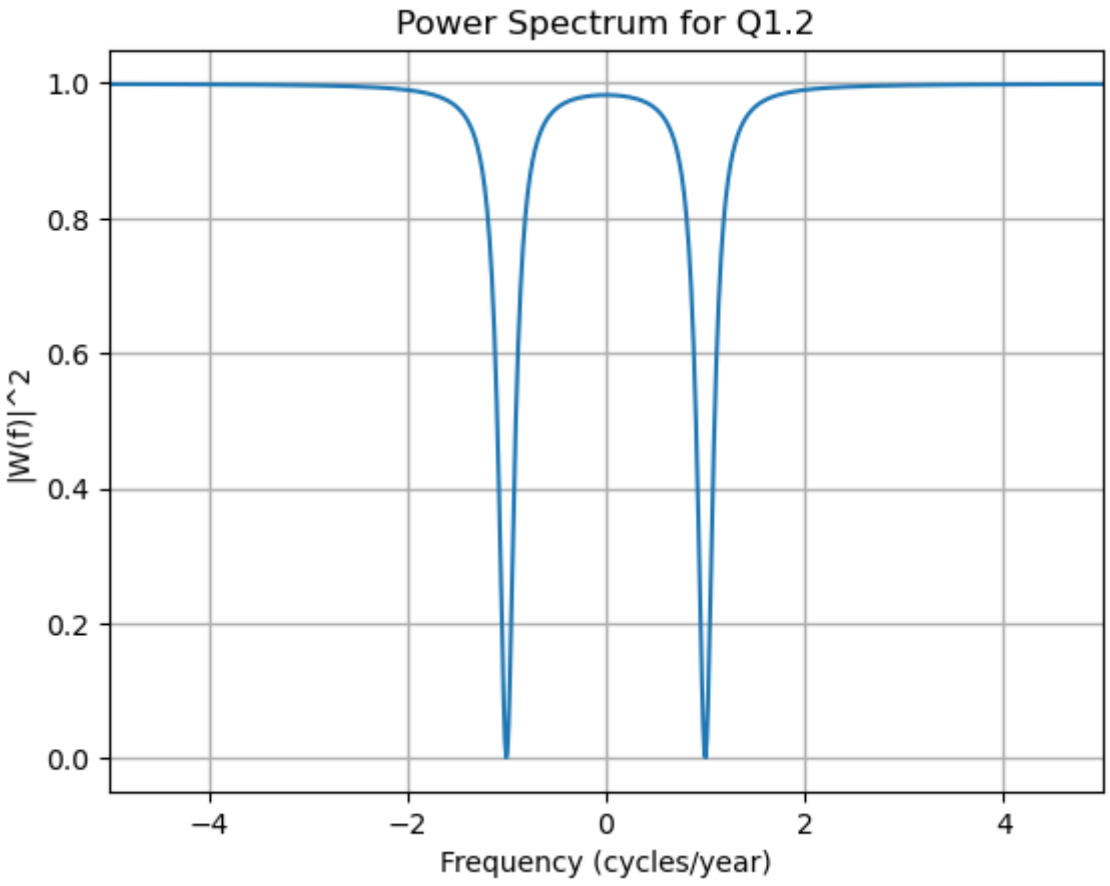
Given that 0 < ϵ << 1, we know that $|1+\epsilon| \approx 1$, thus $|p| \approx |q|$. Meaning that to find the magnitude of p, we only need to find the magnitude of q. $$|q| = |e^{-i2 \pi f_0 / f_s| = 1 $$ since the magnitude of p and q are 1, that means they are both on the unit circle.

Since all of the poles lie within the unit circle, therefore this system is stable. And since this system have real coefficients, this is a real filter.

1. Given $f_s = 12$ cycles/year, $f_0 = 1$ cycle/year, $M = 1.05$ and $\epsilon = 0.05$, plot the power spectrum $|W(f)|^2 = W(f)W(f)^*$ (i.e., square of amplitude spectrum). Sample densely in $[-f_s/2, \ldots, f_s/2]$ (e.g. 1000 points), where $f_s/2$ is the Nyquist frequency.

In [16]:
```
import numpy as np
import matplotlib.pyplot as plt
```

```
fs = 12  # cycles/year
f = np.linspace(-fs/2, fs/2, 1000)

f0 = 1
q = np.exp(-1j*2*np.pi*f0/fs)
p = (1 + 0.05)*q
M = 1.05

n = (np.exp(1j*2*np.pi*f/fs) - q) * (np.exp(1j*2*np.pi*f/fs) - np.conj(q))
d = (np.exp(1j*2*np.pi*f/fs) - p) * (np.exp(1j*2*np.pi*f/fs) - np.conj(p))
W = M*n/d
power_spectrum = np.abs(W)**2

plt.plot(f, power_spectrum)
plt.xlim(-5, 5)
plt.grid(True, linewidth=1)
plt.xlabel('Frequency (cycles/year)')
plt.ylabel('|W(f)|^2')
plt.title("Power Spectrum for Q1.2")
plt.show()
```

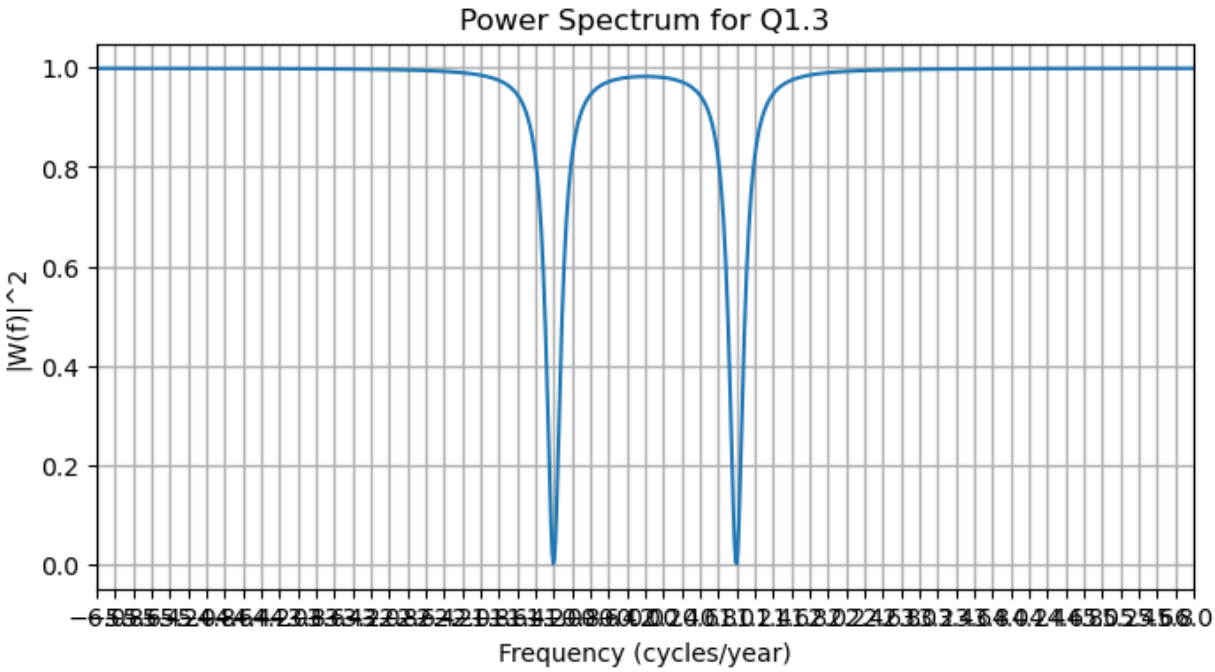


1. What is the full-width-half-max (FWHM) value f_{fwhm} of this notch filter (based on the plot)? Which parameter(s) should you change to make the notches sharper, i.e., f_{fwhm} smaller?

Hint: For Part 2, $W(f)$ is obtained by $W(z = e^{-i\omega\Delta})$. For Part 3, you don't have to compute the FWHM value analytically (although it can be done); an inspection of the discrete array of

$|W(f)|^2$ vector is sufficient. Note here f_{fwhm} is in terms of frequency (1/year), not angular frequency.

In [19]:
```
import numpy as np
import matplotlib.pyplot as plt

fs = 12  # cycles/year
f = np.linspace(-fs/2, fs/2, 1000)

f0 = 1
q = np.exp(-1j*2*np.pi*f0/fs)
p = (1 + 0.05)*q
M = 1.05

n = (np.exp(1j*2*np.pi*f/fs) - q) * (np.exp(1j*2*np.pi*f/fs) - np.conj(q))
d = (np.exp(1j*2*np.pi*f/fs) - p) * (np.exp(1j*2*np.pi*f/fs) - np.conj(p))
W = M*n/d
power_spectrum = np.abs(W)**2

plt.figure(figsize=(8,4))
plt.plot(f, power_spectrum)
plt.xlim(-5, 5)
plt.grid(True, linewidth=1)
plt.xticks(np.arange(min(f), max(f)+0.2, 0.2))
plt.xlabel('Frequency (cycles/year)')
plt.ylabel('|W(f)|^2')
plt.title("Power Spectrum for Q1.3")
plt.show()
```



By inspecting the diagram above, knowing that the distance between each verticle line is 0.2, we can estimate that around the 0.5 mark of the power spectrum, the FWHM value should be around 0.2 cycles/year. TO make the notches sharper, we need to change the ϵ value.

Implementing the Notch Filter (6 marks, 2/2/1/1)

Notch filter introduced in section 1 can be written out fully as

$$W(z) = \frac{N(z)}{D(z)} = \frac{a + bz + cz^2}{1 + Bz + Cz^2}$$

A filter of the form above can be implemented as two filters in succession: first the 'numerator filter' `[a b c]` as a 3-term direct convolution, then the 'denominator filter' as the inverse filter of the 3-term filter `[1 B C]` by feedback filtering.

1. What are the values of a; b; c; B; C for the notch filter defined in Question 1.2?

From Question 1, we know that

$$W(z) = M\frac{(z-q)(z-q*)}{(z-p)(z-p*)}$$

Expanding the top and bottom of the fraction, we get

$$W(z) = M\frac{z^2 - 2Re(zq^*) + |q|^2}{z^2 - 2Re(zp^*) + |p|^2}$$

Rearrange, we get

$$W(z) = M\frac{|q|^2 - 2Re(q^*)z + z^2}{|p|^2 - 2Re(p^*)z + z^2}$$

From Question 1, we know that $|p| = |q| \approx 1$, and $p = (1+\epsilon)q$ and $q = e^{-i2\pi f_0/f_s}$, thus

$$W(z) = M\frac{|q|^2 - 2Re(e^{i2\pi f_0/f_s})z + z^2}{1 - 2Re((1+\epsilon)e^{i2\pi f_0/f_s})z + z^2}$$

Now, we know that:

$$a = M|q|^2$$

$$b = -2MRe(e^{i2\pi f_0/f_s})$$

$$c = M$$

$$B = -2MRe((1+\epsilon)e^{i2\pi f_0/f_s})$$

$$C = M$$

Substituting with the values from Question 1.2, we get (WolframeAlpha):

$$a = 1.05 * |1|^2 \approx 1.05$$

$$b = -2MRe(e^{i2\pi f_0/f_s}) = -2 * 1.05 * Re(e^{i2\pi 1/12}) \approx -1.82$$

$$c = 1.05$$

$$B = -2MRe((1+\epsilon)e^{i2\pi f_0/f_s}) \approx -1.91$$

$$C = 1.05$$

In [26]:
```
a = 1.05
b = -1.82
c = 1.05
B = -1.91
C = 1.05
```

1. Write a function for a general rational digital filter with numerator and denominator coefficients N and D which produces the filtered time series y for a given input x, `y = ratFilter(N,D,x)`.

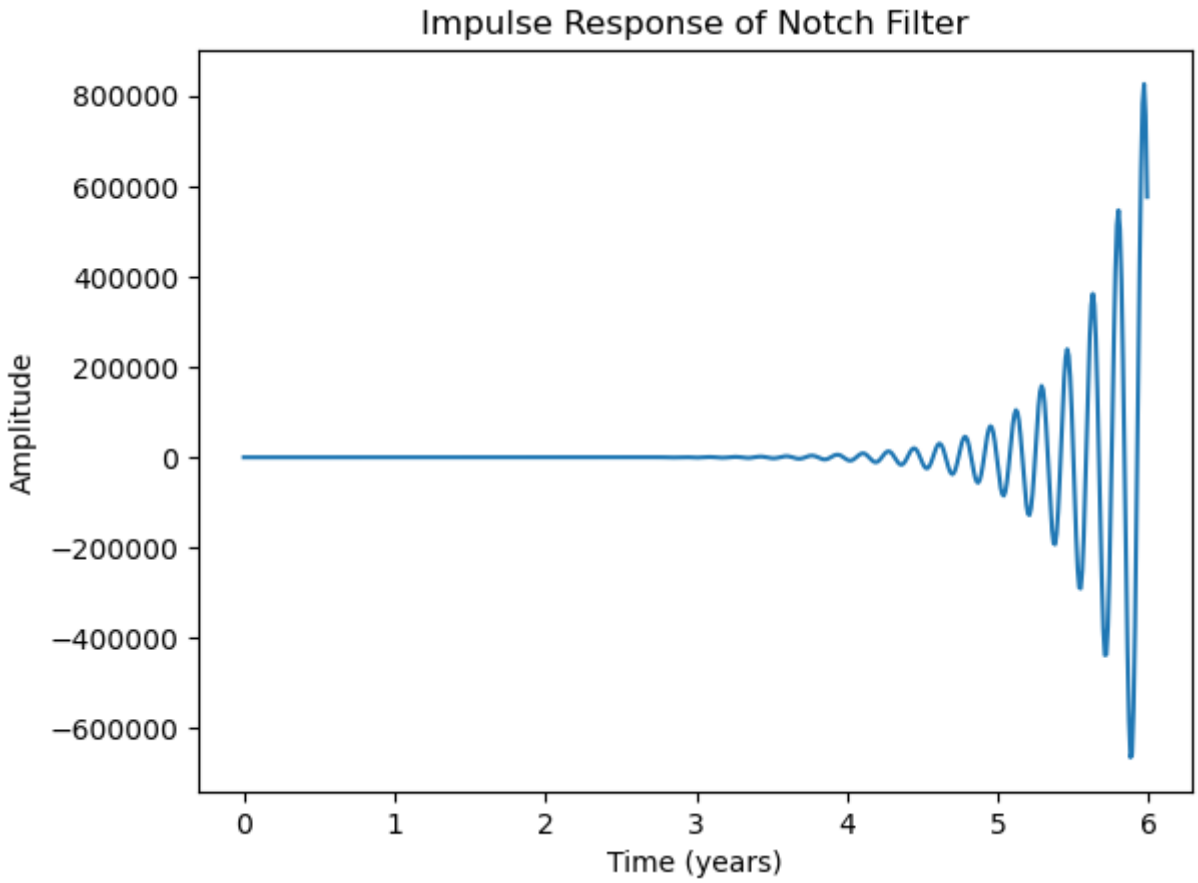
In [21]:
```
import numpy as np
import matplotlib.pyplot as plt

def ratFilter(N, D, x):
    y = np.zeros_like(x)

    # Compute scaling factors for numerator and denominator coefficients
    s_n = N[0] / D[0]
    s_d = np.asarray(D) / D[0]

    for i in range(len(x)):
        # Numerator contribution
        for j in range(len(N)):
            if i-j >= 0:
                y[i] += s_n * N[j] * x[i-j]
        # Denominator contribution
        for j in range(1, len(D)):
            if i-j >= 0:
                y[i] -= s_d[j] * y[i-j]
    return y
```

1. Use `ratFilter` function to determine the impulse response of this notch filter (i.e., the output of this filter when the input is a discrete delta function). Define the impulse using $dt = 1/f_s$ and $t = 0$ to $t_{max} = 100$ years (i.e. 1200 samples). Plot the impulse response from 0 to 6 years. Speculate on how the impulse response would change if we halve the f_{fwmh} value.

1. Fourier transform the impulse response to obtain the frequency response $|W(f)|$ of this notch filter. Plot it on top of the magnitude of the theoretical spectrum calculated based on the z-transform, with f ranging from 0 to 6 cycles per year.

In [41]:

```
import numpy as np
import matplotlib.pyplot as plt

def ratFilter(N, D, x):
    y = np.zeros_like(x)

    # Compute scaling factors for numerator and denominator coefficients
    s_n = N[0] / D[0]
    s_d = np.asarray(D) / D[0]

    for i in range(len(x)):
        # Numerator contribution
        for j in range(len(N)):
            if i-j >= 0:
                y[i] += s_n * N[j] * x[i-j]
        # Denominator contribution
        for j in range(1, len(D)):
            if i-j >= 0:
                y[i] -= s_d[j] * y[i-j]
    return y

# from before
a = 1.05
b = -1.82
c = 1.05
B = -1.91
C = 1.05

# impulse response of the notch filter
t = np.arange(0, 6, 0.01)
impulse = np.zeros_like(t)
impulse[0] = 1 # Set the first value to 1 to create the impulse
filtered_impulse = ratFilter([a, b, c], [1, B, C], impulse)

plt.plot(t, filtered_impulse)
plt.xlabel('Time (years)')
plt.ylabel('Amplitude')
plt.title('Impulse Response of Notch Filter')
plt.show()
```



We know that if we want to alters the FWMH value, then we will need to alter the ϵ value. From the math in Question 2.1, we can see that that will have an affect on the B coefficient.

The Mauna Loa CO_2 Data (6 marks, 1/1/1/1/2)

The provided file `co2data.py` contains carbon dioxide values (in parts per million) measured every month at the Mauna Loa Observatory in Hawaii, from January 1965 to December 2022. They show clearly a rising trend in atmospheric CO_2. The trend is overlaid with a strong annual oscillation. Your job is to remove the annual oscillation and display the trend more clearly. There are two possible approaches: (a) you could apply your notch filter to the series to remove this annual variation, or (b) you could Fourier transform it with `fft`, remove the annual variation by setting the spectrum at appropriate frequencies to zero, and transform back to the time domain with `ifft`.

Write code to accomplish the following:

1. Before applying the filters, it is helpful to remove the trend of the signal using numpy function `polyfit`. Fit a straight line to your data and then detrend your data by removing the straight line. Plot both the original data and the detrended data.
2. Apply your notch filter to the detrended data and add back the trend.
3. FT the detrended data into the frequency domain, and plot both its amplitude and phase spectrum. Make another plot that zooms in at $f = [0, 3.5]$ cycles per year. Now

set the Fourier spectrum corresponding to frequencies beyond 0.9 cycles per year to zero (keeping in mind symmetry requirements), which effectively removes the annual oscillation. Transform the spectrum back to time domain and add back the trend.

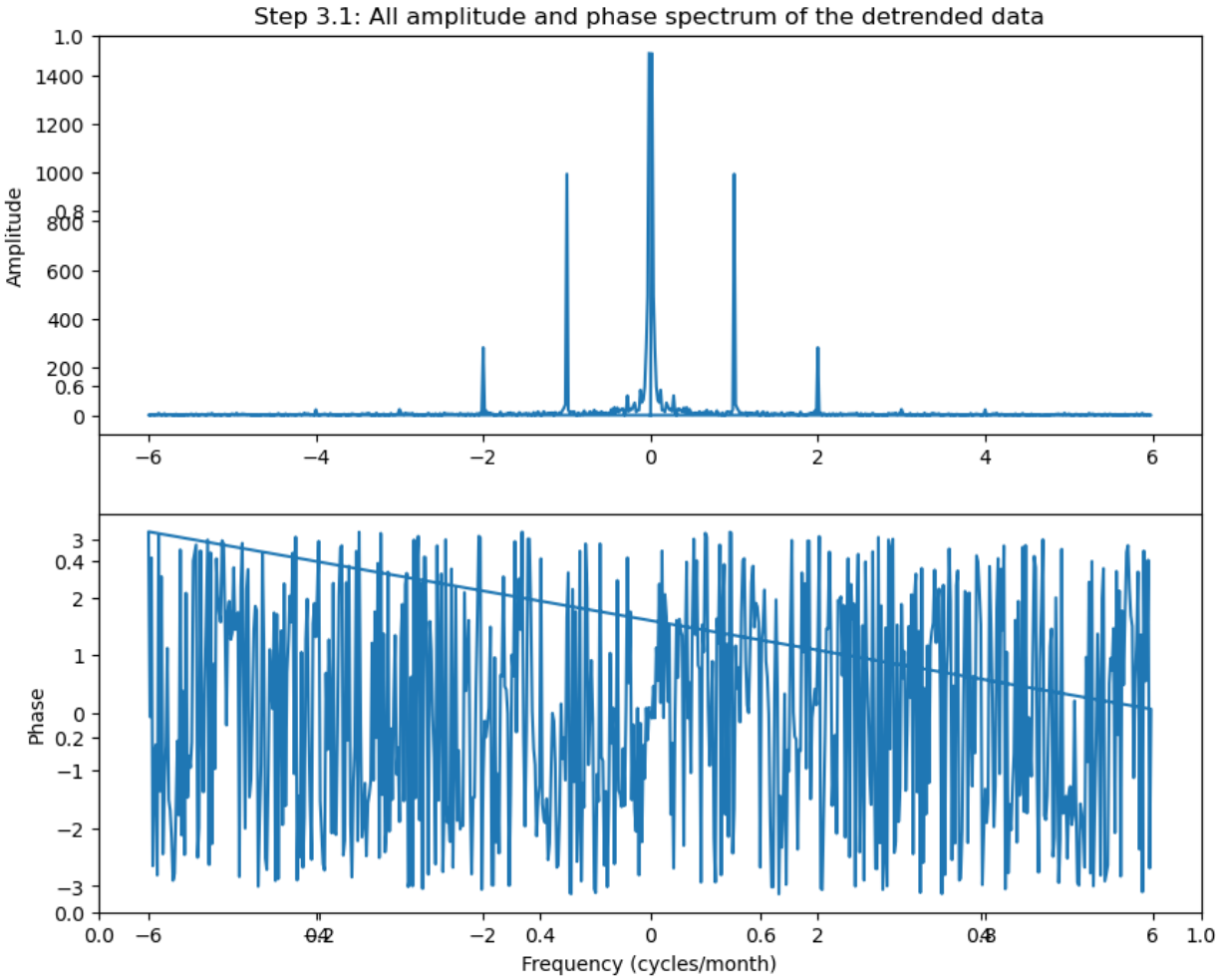
4. Now plot the original data, the notch-filtered data from Part 2 and f-domain filtered data from Part 3 on top of each other with different colors. Which method gives more satisfactory result? But can you think of any advantages/disadvantages in using either method?
5. Now try redo Parts 2, 3, and 4 with the original data, not the detrended data. Of course you don't need to add back the trend after filtering any more. Display your results and comment on the importance of detrending before applying the filters.

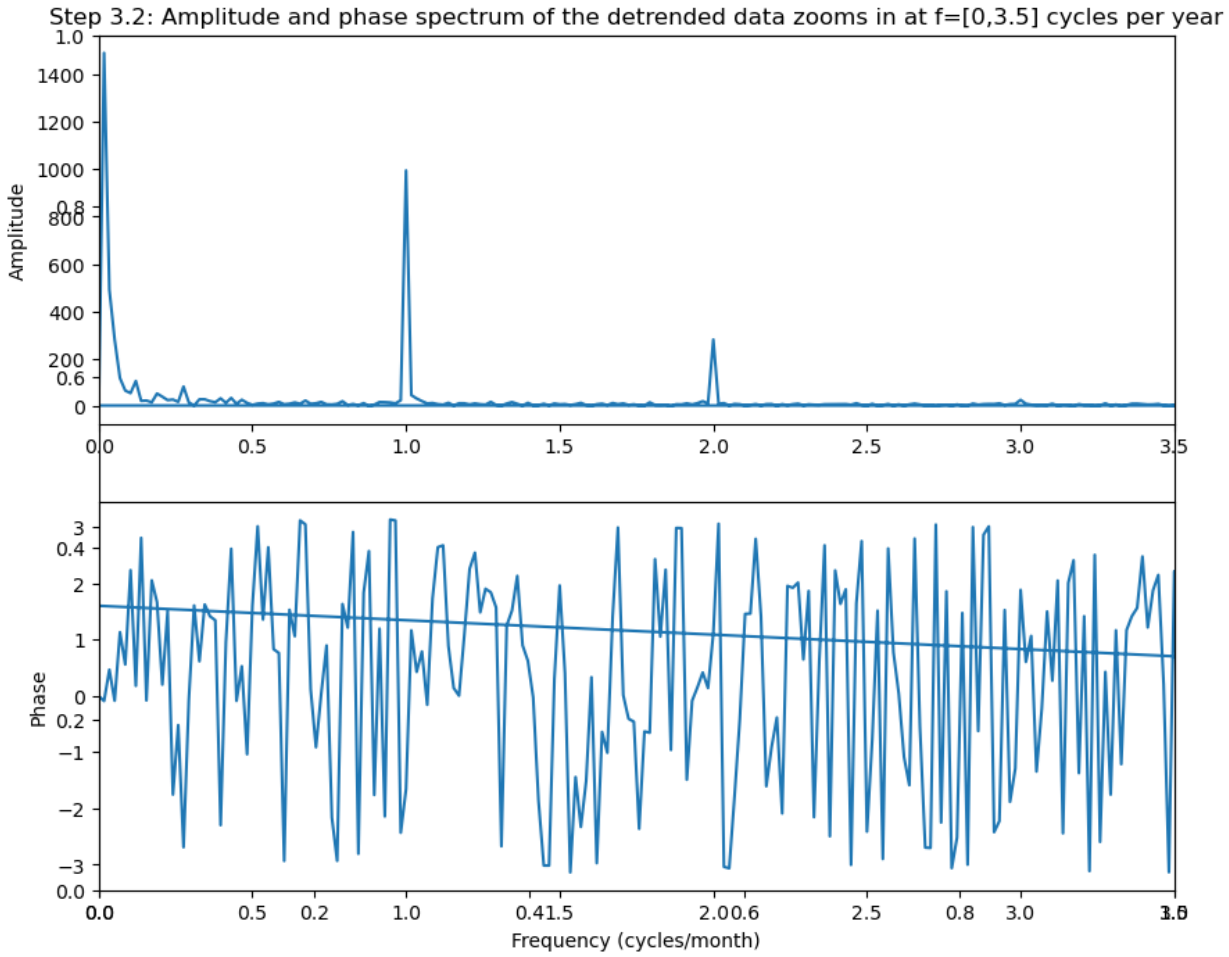
In [39]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal

co2Values = np.array([
    [319.44, 320.44, 320.89, 322.14, 322.17, 321.87, 321.21, 318.87, 317.81, 31
    [320.62, 321.60, 322.39, 323.70, 324.08, 323.75, 322.38, 320.36, 318.64, 31
    [322.33, 322.50, 323.04, 324.42, 325.00, 324.09, 322.54, 320.92, 319.25, 31
    [322.57, 323.15, 323.89, 325.02, 325.57, 325.36, 324.14, 322.11, 320.33, 32
    [324.00, 324.42, 325.63, 326.66, 327.38, 326.71, 325.88, 323.66, 322.38, 32
    [325.06, 325.98, 326.93, 328.13, 328.08, 327.67, 326.34, 324.69, 323.10, 32
    [326.17, 326.68, 327.17, 327.79, 328.93, 328.57, 327.36, 325.43, 323.36, 32
    [326.77, 327.63, 327.75, 329.72, 330.07, 329.09, 328.04, 326.32, 324.84, 32
    [328.55, 329.56, 330.30, 331.50, 332.48, 332.07, 330.87, 329.31, 327.51, 32
    [329.35, 330.71, 331.48, 332.65, 333.19, 332.20, 331.07, 329.15, 327.33, 32
    [330.73, 331.46, 331.94, 333.11, 333.95, 333.42, 331.97, 329.95, 328.50, 32
    [331.56, 332.74, 333.36, 334.74, 334.72, 333.98, 333.08, 330.68, 328.96, 32
    [332.68, 333.17, 334.96, 336.14, 336.93, 336.17, 334.88, 332.56, 331.29, 33
    [334.94, 335.26, 336.66, 337.69, 338.02, 338.01, 336.50, 334.42, 332.36, 33
    [336.14, 336.69, 338.27, 338.82, 339.24, 339.26, 337.54, 335.72, 333.97, 33
    [337.90, 338.34, 340.07, 340.93, 341.45, 341.36, 339.45, 337.67, 336.25, 33
    [339.29, 340.55, 341.63, 342.60, 343.04, 342.54, 340.82, 338.48, 336.95, 33
    [340.93, 341.76, 342.78, 343.96, 344.77, 343.88, 342.42, 340.24, 338.38, 33
    [341.57, 342.79, 343.37, 345.40, 346.14, 345.76, 344.32, 342.50, 340.46, 34
    [344.21, 344.92, 345.68, 347.14, 347.77, 347.16, 345.79, 343.74, 341.59, 34
    [345.48, 346.41, 347.91, 348.66, 349.28, 348.65, 346.90, 345.26, 343.47, 34
    [346.78, 347.48, 348.25, 349.86, 350.52, 349.98, 348.25, 346.17, 345.48, 34
    [348.73, 348.92, 349.81, 351.40, 352.15, 351.59, 350.21, 348.20, 346.66, 34
    [350.51, 351.70, 352.50, 353.67, 354.35, 353.88, 352.80, 350.49, 348.97, 34
    [353.07, 353.43, 354.08, 355.72, 355.95, 355.44, 354.05, 351.84, 350.09, 35
    [353.86, 355.10, 355.75, 356.38, 357.38, 356.39, 354.89, 353.07, 351.38, 35
    [354.93, 355.82, 357.33, 358.77, 359.23, 358.23, 356.30, 353.97, 352.34, 35
    [356.34, 357.21, 357.97, 359.22, 359.71, 359.43, 357.15, 354.98, 353.01, 35
    [357.10, 357.42, 358.59, 359.39, 360.30, 359.64, 357.46, 355.76, 354.14, 35
    [358.36, 359.04, 360.11, 361.36, 361.78, 360.94, 359.51, 357.59, 355.86, 35
    [360.04, 361.00, 361.98, 363.44, 363.83, 363.33, 361.78, 359.33, 358.32, 35
    [362.20, 363.36, 364.28, 364.69, 365.25, 365.06, 363.69, 361.55, 359.69, 35
    [363.24, 364.21, 364.65, 366.49, 366.77, 365.73, 364.46, 362.40, 360.44, 36
    [365.39, 366.10, 367.36, 368.79, 369.56, 369.13, 367.98, 366.10, 364.16, 36
    [368.35, 369.28, 369.84, 371.15, 371.12, 370.46, 369.61, 367.06, 364.95, 36
    [369.45, 369.71, 370.75, 371.98, 371.74, 371.87, 370.02, 368.27, 367.15, 36
    [370.76, 371.69, 372.63, 373.55, 374.03, 373.40, 371.68, 369.78, 368.34, 36
    [372.70, 373.37, 374.30, 375.19, 375.93, 375.69, 374.16, 372.03, 370.92, 37
    [375.07, 375.82, 376.64, 377.92, 378.78, 378.46, 376.88, 374.57, 373.34, 37
    [377.17, 378.05, 379.06, 380.54, 380.80, 379.87, 377.65, 376.17, 374.43, 37
```

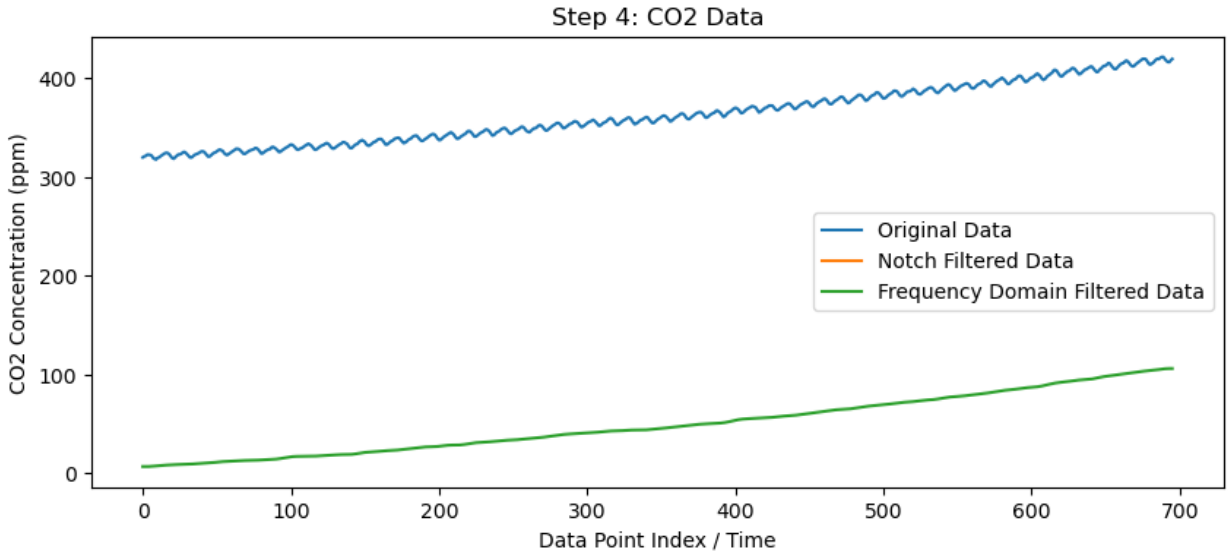
```
    [378.63, 379.91, 380.95, 382.48, 382.64, 382.40, 380.93, 378.93, 376.89, 37
    [381.58, 382.40, 382.86, 384.80, 385.22, 384.24, 382.65, 380.60, 379.04, 37
    [383.10, 384.12, 384.81, 386.73, 386.78, 386.33, 384.73, 382.24, 381.20, 38
    [385.78, 386.06, 386.28, 387.33, 388.78, 387.99, 386.61, 384.32, 383.41, 38
    [387.17, 387.70, 389.04, 389.76, 390.36, 389.70, 388.25, 386.29, 384.95, 38
    [388.91, 390.41, 391.37, 392.67, 393.21, 392.38, 390.41, 388.54, 387.03, 38
    [391.50, 392.05, 392.80, 393.44, 394.41, 393.95, 392.72, 390.33, 389.28, 38
    [393.31, 394.04, 394.59, 396.38, 396.93, 395.91, 394.56, 392.59, 391.32, 39
    [395.78, 397.03, 397.66, 398.64, 400.02, 398.81, 397.51, 395.39, 393.72, 39
    [398.04, 398.27, 399.91, 401.51, 401.96, 401.43, 399.38, 397.32, 395.64, 39
    [400.18, 400.55, 401.74, 403.35, 404.15, 402.97, 401.46, 399.11, 397.82, 39
    [402.73, 404.25, 405.06, 407.60, 407.90, 406.99, 404.59, 402.45, 401.23, 40
    [406.36, 406.66, 407.53, 409.22, 409.89, 409.08, 407.33, 405.32, 403.57, 40
    [408.15, 408.52, 409.59, 410.45, 411.44, 410.99, 408.90, 407.16, 405.71, 40
    [411.03, 411.96, 412.18, 413.54, 414.86, 414.16, 411.97, 410.18, 408.76, 40
    [413.61, 414.34, 414.74, 416.45, 417.31, 416.60, 414.62, 412.78, 411.52, 41
    [415.52, 416.75, 417.64, 419.05, 419.13, 418.94, 416.96, 414.47, 413.30, 41
    [418.19, 419.28, 418.81, 420.23, 420.99, 420.99, 418.90, 417.19, 415.95, 41
])

co2Data = co2Values.flatten()
co2TimeRange = [1965.0411, 2022.9583]

################ above are from co2data.py ################




y = np.arange(len(co2Data))
data = np.array(co2Data)
fs = 12

################ Step 1 ################
p = np.polyfit(y, data, 1)
detrended = data - (p[0] * y + p[1]) # degree 1


################ Step 2 ################
def notch_filter(data, f0, Q, fs):
    w0 = f0/(fs/2)  # notch frequency in normalized frequency (0, 1)
    b, a = [1, -2*np.cos(w0*np.pi), 1], \
        [1, -2*Q*np.cos(w0*np.pi), (2*Q)**2 - 2*np.cos(w0*np.pi) + 1]
    filtered = filtfilt(b, a, data)
    return filtered

filtered1 = notch_filter(detrended, 1/12, 30, fs)



################ Step 3 ################
spectrum = np.fft.fft(detrended)
freq = np.fft.fftfreq(len(detrended), d=1/fs)

# plot ALL amplitude and phase spectra
fig = plt.figure(figsize=(10, 8))
plt.title("Step 3.1: All amplitude and phase spectrum of the detrended data")
ax1 = fig.add_subplot(2, 1, 1)
ax1.plot(freq, np.abs(spectrum))
```

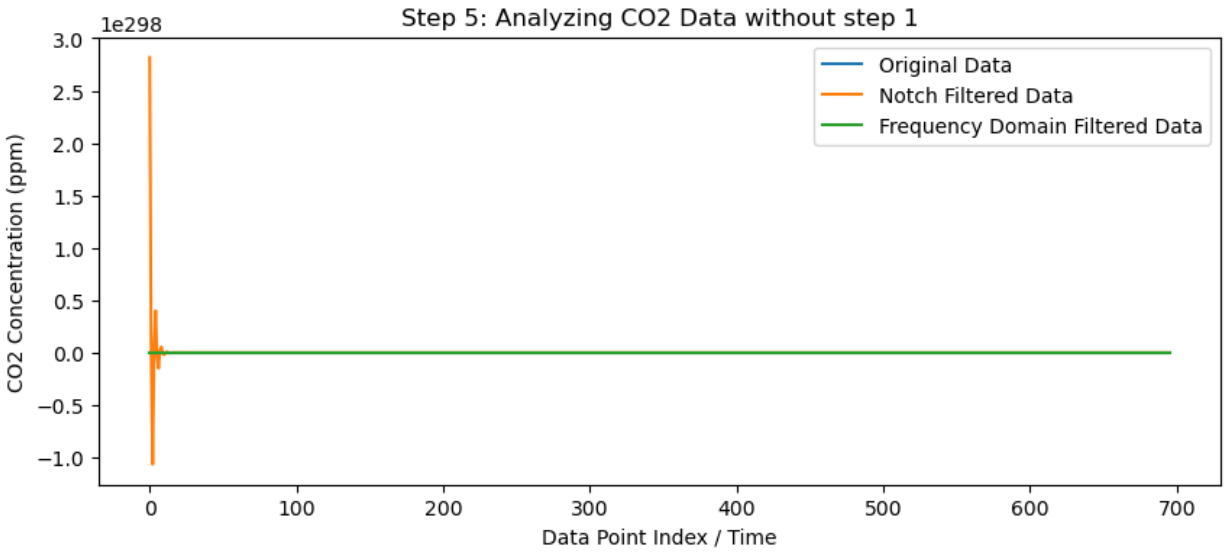
```
ax1.set_ylabel('Amplitude')
ax2 = fig.add_subplot(2, 1, 2)
ax2.plot(freq, np.angle(spectrum))
ax2.set_ylabel('Phase')
ax2.set_xlabel('Frequency (cycles/month)')
plt.show()

# zoom at f=[0,3.5]
fig = plt.figure(figsize=(10, 8))
plt.title("Step 3.2: Amplitude and phase spectrum of the detrended data zooms i
ax1 = fig.add_subplot(2, 1, 1)
ax1.plot(freq, np.abs(spectrum))
ax1.set_xlim([0, 3.5])
ax1.set_ylabel('Amplitude')
ax2 = fig.add_subplot(2, 1, 2)
ax2.plot(freq, np.angle(spectrum))
ax2.set_xlim([0, 3.5])
ax2.set_ylabel('Phase')
ax2.set_xlabel('Frequency (cycles/month)')
plt.show()

# set the Fourier spectrum corresponding to frequencies beyond 0.9 cycles per y
for i in range(len(spectrum)):
    if abs(freq[i]) > 0.9:
        spectrum[i] = 0

filtered2 = np.real(np.fft.ifft(spectrum)) # inverse FFT to obtain the filtered



################ Step 4 ################
plt.figure(figsize=(10, 4))
plt.plot(t, data, label='Original Data')
plt.plot(t, filtered1+p[0]*t, label='Notch Filtered Data')
plt.plot(t, filtered2+p[0]*t, label='Frequency Domain Filtered Data')
plt.xlabel('Data Point Index / Time')
plt.ylabel('CO2 Concentration (ppm)')
plt.title('Step 4: CO2 Data')
plt.legend()
plt.show()



################ Step 5 ################
filtered1 = notch_filter(data, fs, 1/12, 30)

# FFT of the original data
spectrum = np.fft.fft(data)
# set the Fourier spectrum corresponding to frequencies beyond 0.9 cycles per y
for i in range(len(spectrum)):
    if abs(freq[i]) > 0.9:
        spectrum[i] = 0


filtered2 = np.real(np.fft.ifft(spectrum))  # inverse FFT to obtain the filtere

# Plot results
plt.figure(figsize=(10, 4))
plt.plot(t, data, label='Original Data')
plt.plot(t, filtered1, label='Notch Filtered Data')
```

```
plt.plot(t, filtered2, label='Frequency Domain Filtered Data')
plt.xlabel('Data Point Index / Time')
plt.ylabel('CO2 Concentration (ppm)')
plt.title('Step 5: Analyzing CO2 Data without step 1')
plt.legend()
plt.show()
```




Step 3.2: Amplitude and phase spectrum of the detrended data zooms in at f=[0,3.5] cycles per year
Amplitude
Phase
Frequency (cycles/month)


Step 4: CO2 Data
CO2 Concentration (ppm)
Data Point Index / Time
Original Data
Notch Filtered Data
Frequency Domain Filtered Data



References:

- https://courses.egr.uh.edu/ECE/ECE5358/Homework/ECE%205358_6358_F17_HW%2012_
- http://www.isle.illinois.edu/speech_web_lg/coursematerials/ece401/fa2020/slides/lec15.pdf
- https://stackoverflow.com/questions/24777788/design-of-a-notch-filter-in-python
- http://www.neurotec.uni-bremen.de/drupal/node/61
- https://www.youtube.com/watch?v=BgoJ5qnLFBM&ab_channel=DavidDorran

In [ ]: