Dang_Kevin_Lab3

March 25, 2020

[1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
from numpy.fft import fft, fftfreq, ifft, fftshift, ifftshift
import warnings
warnings.filterwarnings('ignore')
```

1 1 A Notch Digital Filter (4 marks, 2/1/1)

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?
2. Given $f_s = 12$ cycles/year, $f_0 = 1$ cycle/year, $M = 1.04$ and $\epsilon = 0.04$, 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.
3. 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 question 1.1, a filter is "real" if a real input timeseries passed through the filter remains real (see question 2.2). Completing question 2.1 first may be helpful to prove this.

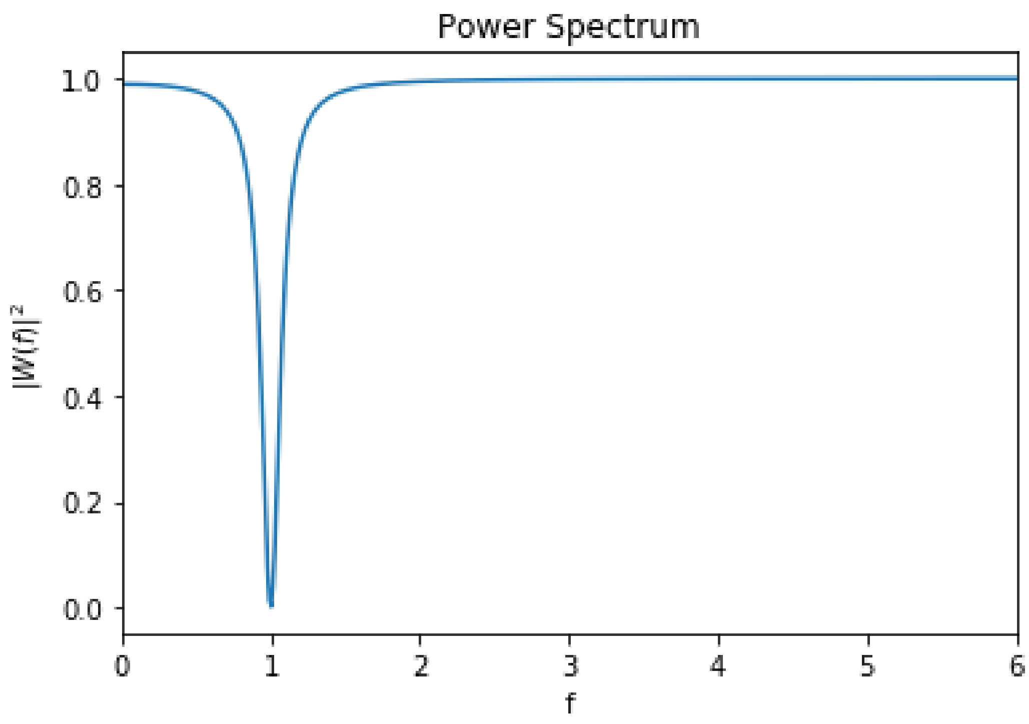
Hint: For question 1.2, $W(f)$ is obtained by $W(z = e^{-i\omega\Delta})$. For question 3, you don't have to compute the FWHM value analytically (although it can be done); an inspection of the discrete array of $|W(f)|^2$ values is sufficient. Note here f_{fwhm} is in terms of frequency (1/year), not angular frequency.

1.1 Part 1

The poles of this system are the zeros of the denominator, which are p and p*. They lie outside the unit circle since p = 1 + ϵ = 1.04. Since the poles lie outside the unit circle, the system is unstable. The filter is real.

1.2 Part 2

[2]:
```
fs = 12
f0 = 1
M = 1.04
eps = 0.04

q = np.exp(-1j*2*np.pi*f0/fs)
p = (1+eps)*q
f_axis = np.linspace(-fs/2,fs/2,1000)
omega = 2*np.pi*f_axis
delta = 1/(fs)
z = np.exp(-1j*omega*delta)
W = M*(z-q)*(z-np.conj(q))/((z-p)*(z-np.conj(p)))
power = np.abs(W)**2

plt.plot(f_axis,power)
plt.xlim(0,6)
plt.xlabel('f')
plt.ylabel('$|W(f)|^2$')
plt.title('Power Spectrum')
plt.show()
```



1.3 Part 3

[3]:
```
halfmax1 = np.where(power.round(7) == 0.5385623)
halfmax2 = np.where(power.round(8) == 0.50034259)
print(f_axis[576], f_axis[589])
fwhm = f_axis[589] - f_axis[576]
fwhm
```

```
0.9189189189189193 1.075075075075075
```

[3]:
```
0.15615615615615575
```

The half max occurs roughly where power is 0.5. So I chose the closest possible power values to 0.5, then looked up those indices on the x axis and took the difference to find the full-width-half-maximum. The value I obtained for the FWHM is $f_{\text{fwhm}} = 0.156$.

2 2 Implementing the Notch Filter (5 marks, 1/2/1/1)

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?
2. 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)`.
3. 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.
4. 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.

2.1 Part 1

$$\begin{aligned}
W(z) &= M\frac{z-q}{z-p}\frac{z-q^*}{z-p^*} \\
&= M\frac{z^2 - zq - zq^* + qq^*}{z^2 - zp - zp^* + pp^*} \\
&= M\frac{qq^* - (q+q^*)z + z^2}{|p|^2 - (p+p^*)z + z^2} \\
&= M\frac{1-(q+q^*)z+z^2}{|(1+\epsilon)q|^2 - ((1+\epsilon)q + (1+\epsilon)q^*)z + z^2} \\
&= M\frac{1-(e^{-i2\pi f_0/f_s} + e^{i2\pi f_0/f_s})z + z^2}{|1+\epsilon|^2 - (1+\epsilon)(e^{-i2\pi f_0/f_s} + e^{i2\pi f_0/f_s})z + z^2} \\
&= M\frac{1-(e^{-i\pi/6} + e^{i\pi/6})z + z^2}{M^2 - M(e^{-i\pi/6} + e^{i\pi/6}) + z^2} \\
&= M\frac{1-2\cos(\pi/6)z + z^2}{M^2 - M(2\cos(\pi/6))z + z^2} \\
&= M\frac{1-\sqrt{3}z + z^2}{M^2 - M\sqrt{3}z + z^2} \\
&= \frac{1-\sqrt{3}z + z^2}{M - \sqrt{3}z + z^2/M}\cdot\frac{1/M}{1/M} \\
&= \frac{(1-\sqrt{3}z + z^2)/M}{1-\sqrt{3}z/M + z^2/M^2}
\end{aligned}$$

Equating this with $W(z) = \frac{N(z)}{D(z)} = \frac{a+bz+cz^2}{1+Bz+Cz^2}$ we have:

$$N(z) = a + bz + cz^2 = (1-\sqrt{3}z + z^2)/M \Rightarrow a = 1/M; b = -\sqrt{3}/M; c = 1/M$$

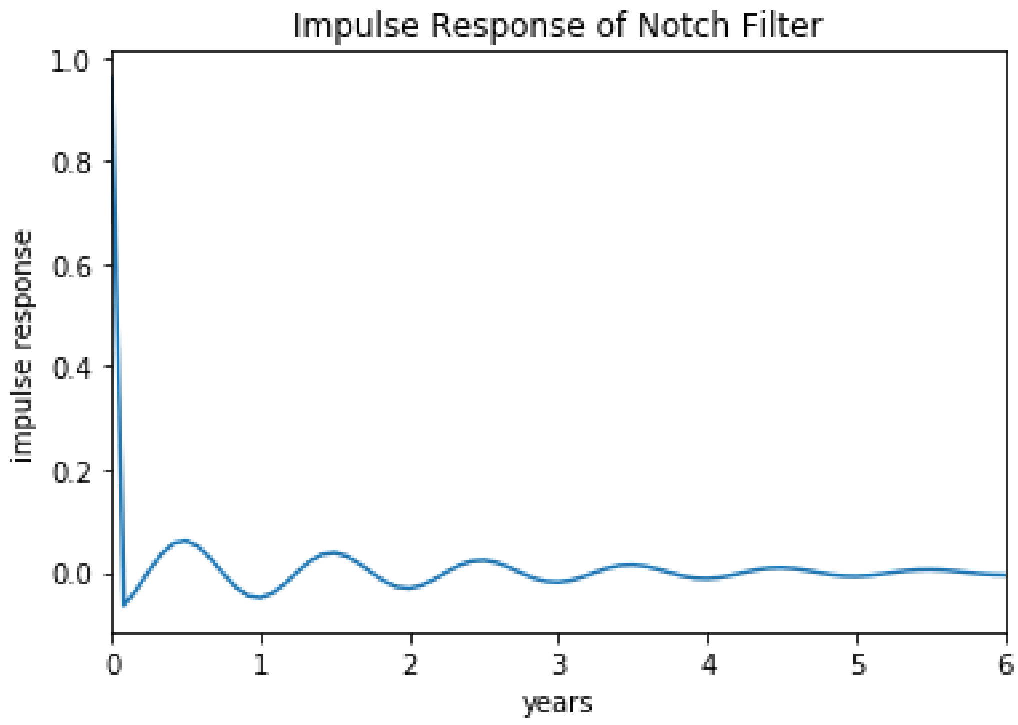
$$D(z) = 1 + Bz + Cz^2 = 1 - z/M^2 + z^2/M^2 \Rightarrow B = -\sqrt{3}/M; C = 1/M^2$$

2.2 Part 2

[4]:
```
def ratFilter(N,D,x):
    x1 = np.convolve(x,N)
    x2 = signal.deconvolve(x1,D)[0]
    return x2
```

2.3 Part 3

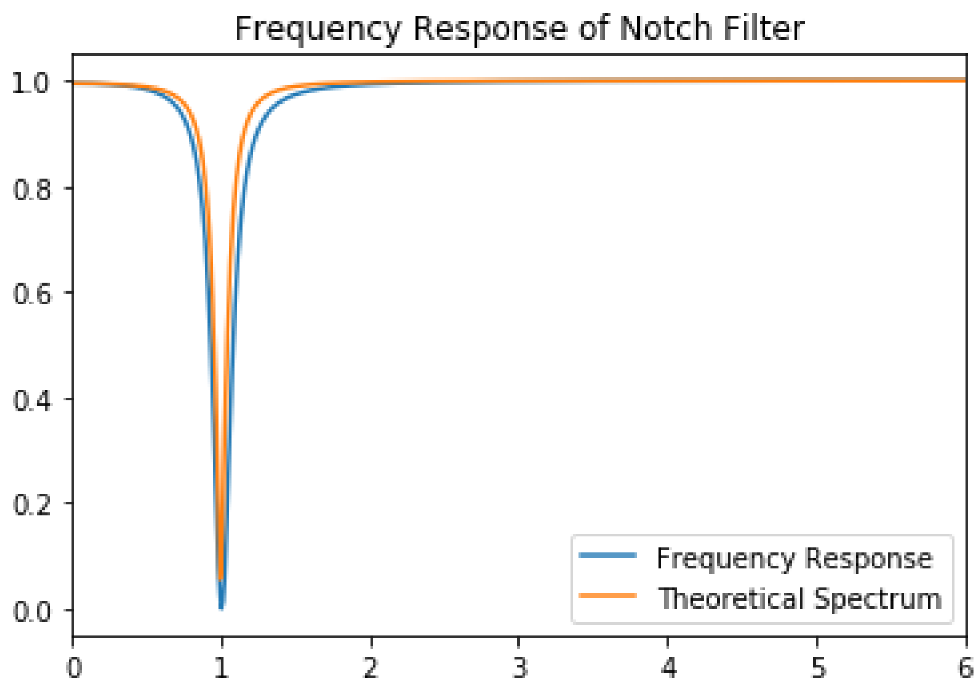
[5]:
```
N = [1/M, -np.sqrt(3)/M, 1/M]
D = [1, -np.sqrt(3)/M, 1/(M**2)]
dt = 1/fs
t = np.arange(0,100,dt)
x = signal.unit_impulse(len(t))
y = ratFilter(N,D,x)

plt.plot(t,y)
plt.xlim(0,6)
plt.xlabel('years')
plt.ylabel('impulse response')
plt.title('Impulse Response of Notch Filter')
plt.show()
```



If we halve the f_{fwmh} value, the imulse response would have a larger amplitude on the bounces, so the decay would take longer to become flat.

2.4 Part 4

[6]:
```
fs = 12
f0 = 1
M = 1.04
eps = 0.04

q = np.exp(-1j*2*np.pi*f0/fs)
p = (1+eps)*q
f_axis = np.linspace(-fs/2,fs/2,1200)
omega = 2*np.pi*f_axis
delta = 1/(fs)
z = np.exp(-1j*omega*delta)
W = M*(z-q)*(z-np.conj(q))/((z-p)*(z-np.conj(p)))

plt.plot(f_axis,fftshift(fft(y)),label='Frequency Response')
plt.plot(f_axis,np.abs(W),label='Theoretical Spectrum')
plt.xlim(0,6)
plt.legend()
plt.title('Frequency Response of Notch Filter')
plt.show()
```



3 3 The Mauna Loa $CO_2$ Data (6 marks, 1/1/1/1/2)

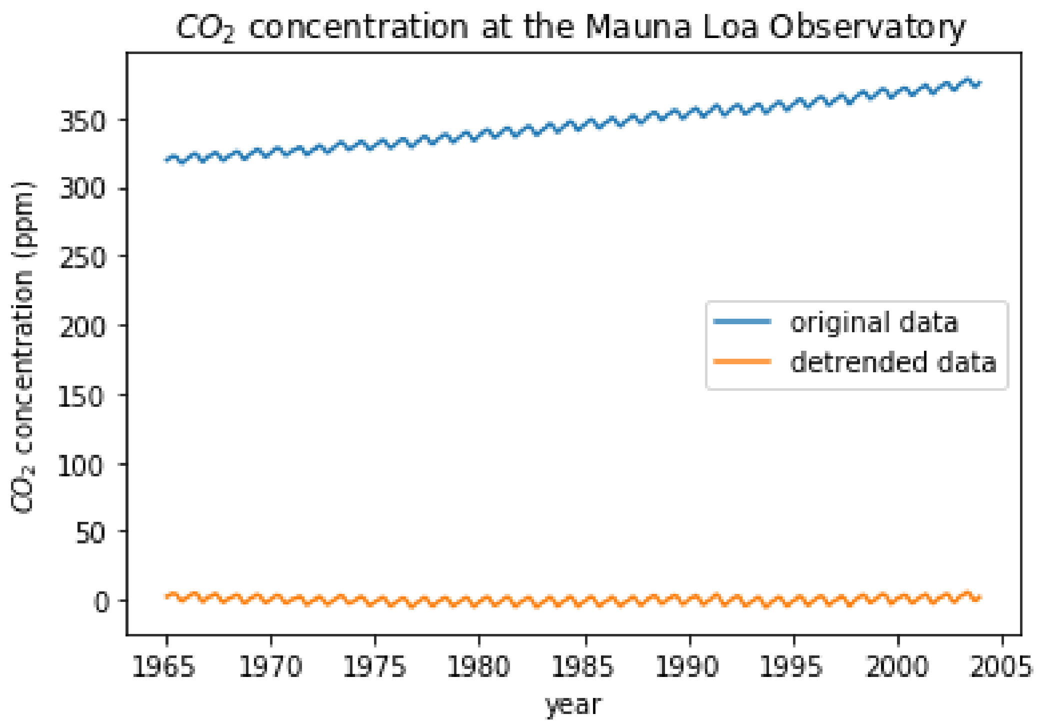
The provided file `co2data.py` contains carbon dioxide values (in parts per million) measured every month at the Mauna Loa Observatory in Hawaii, from January 1965 to December 2003. They show clearly a rising trend in atmospheric $CO_2$. The trend is overlaid with a strong annual oscillation. Your job 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 part 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.

3.1 Part 1

[7]:
```
%run co2data.py

# co2Data
# co2TimeRange
time = np.linspace(co2TimeRange[0],co2TimeRange[1],len(co2Data))
fit = np.polyfit(time,co2Data,1)
line = time*fit[0] + fit[1]
detrend = co2Data-line

plt.plot(time,co2Data,label='original data')
plt.plot(time,detrend,label='detrended data')
plt.legend()
plt.xlabel('year')
plt.ylabel('$CO_2$ concentration (ppm)')
plt.title('$CO_2$ concentration at the Mauna Loa Observatory')
plt.show()
```

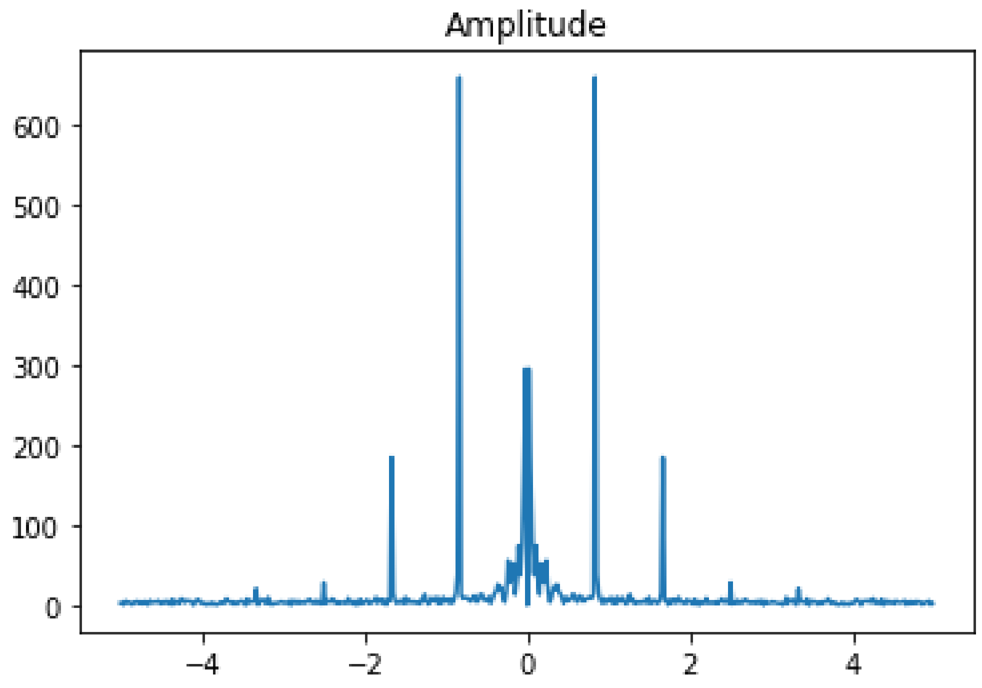


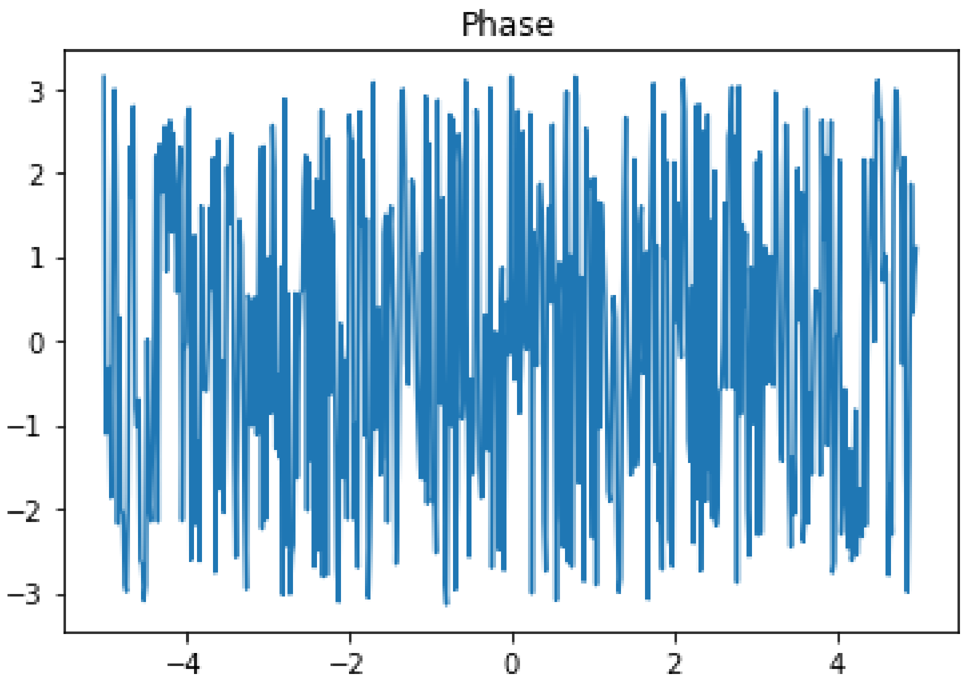
3.2 Part 2

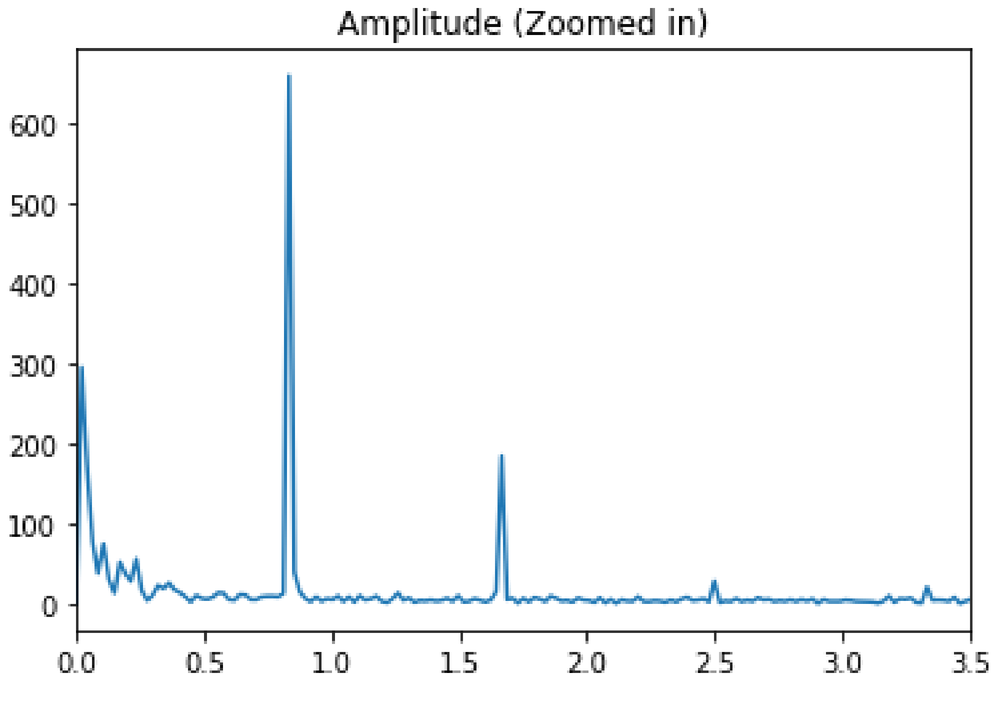
[8]:
```
co2filtered = ratFilter(N,D,detrend) + line
```

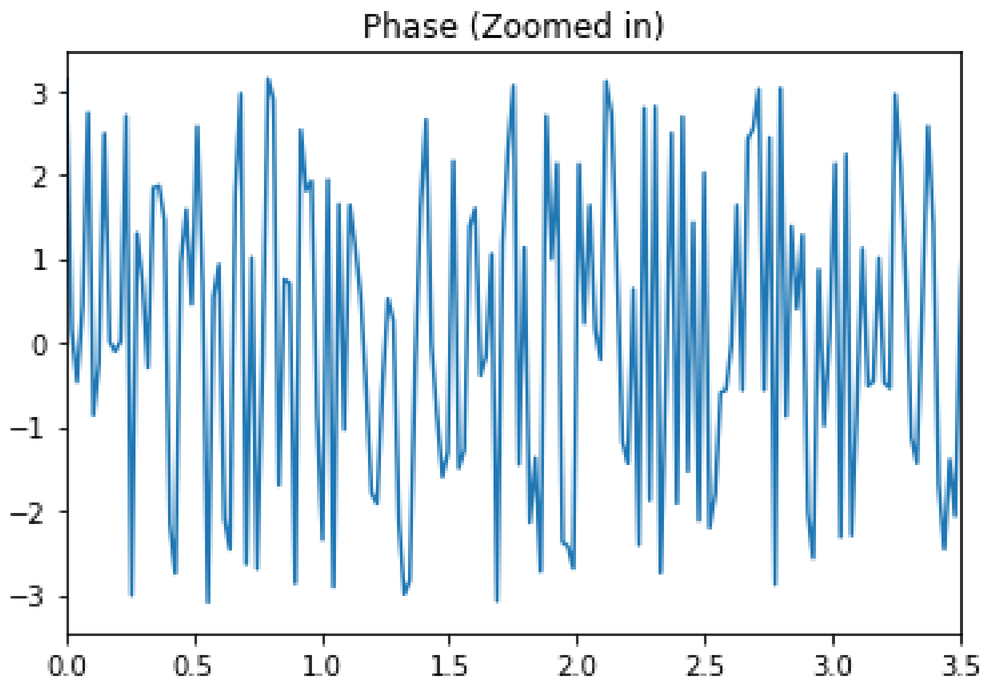
3.3 Part 3

[9]:
```
co2ft = fftshift(fft(detrend))
amp = np.abs(co2ft)
phase = np.angle(co2ft)
f_axis = fftshift(fftfreq(len(detrend),0.1))

plt.plot(f_axis,amp)
plt.title('Amplitude')
plt.show()
plt.plot(f_axis,phase)
plt.title('Phase')
plt.show()
plt.plot(f_axis,amp)
plt.xlim(0,3.5)
plt.title('Amplitude (Zoomed in)')
plt.show()
plt.plot(f_axis,phase)
plt.xlim(0,3.5)
plt.title('Phase (Zoomed in)')
plt.show()

for i in range(len(f_axis)):
    if f_axis[i] > 0.9 or f_axis[i] < -0.9:
        co2ft[i] = 0

co2fdomain = ifft(ifftshift(co2ft)) + line
```



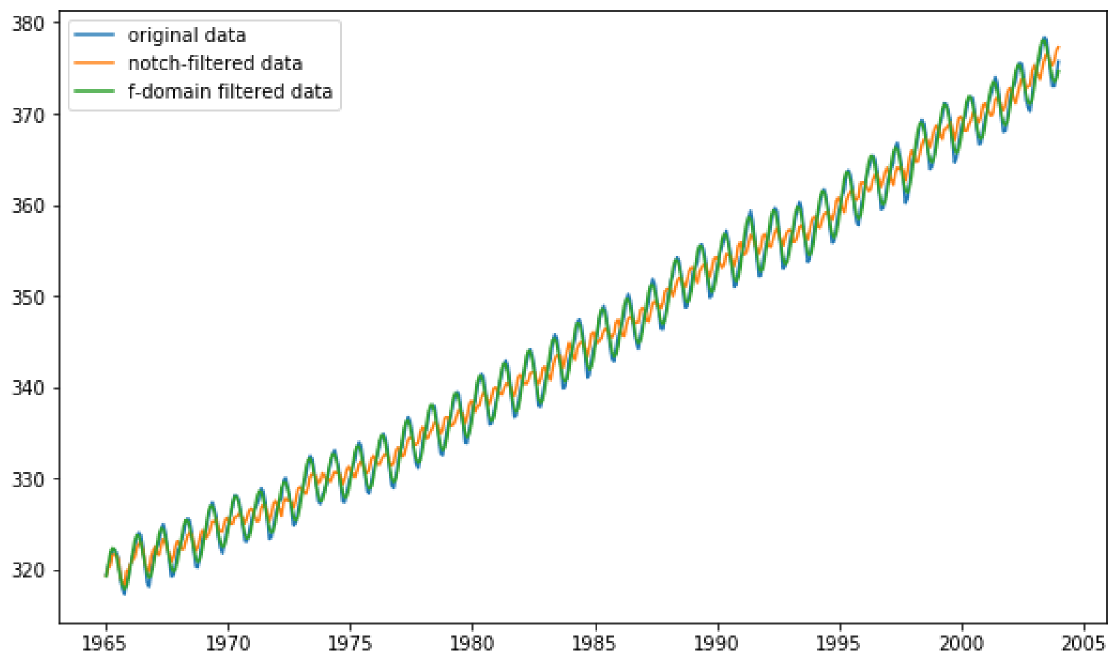

Phase
3
2
1
0
−1
−2
−3
−4
−2
0
2
4


Amplitude (Zoomed in)
600
500
400
300
200
100
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5



3.4 Part 4

[10]:
```
plt.figure(figsize=(10,6))
plt.plot(time,co2Data,label='original data')
plt.plot(time,co2filtered,label='notch-filtered data')
plt.plot(time,co2fdomain,label='f-domain filtered data')
plt.legend()
plt.show()
```

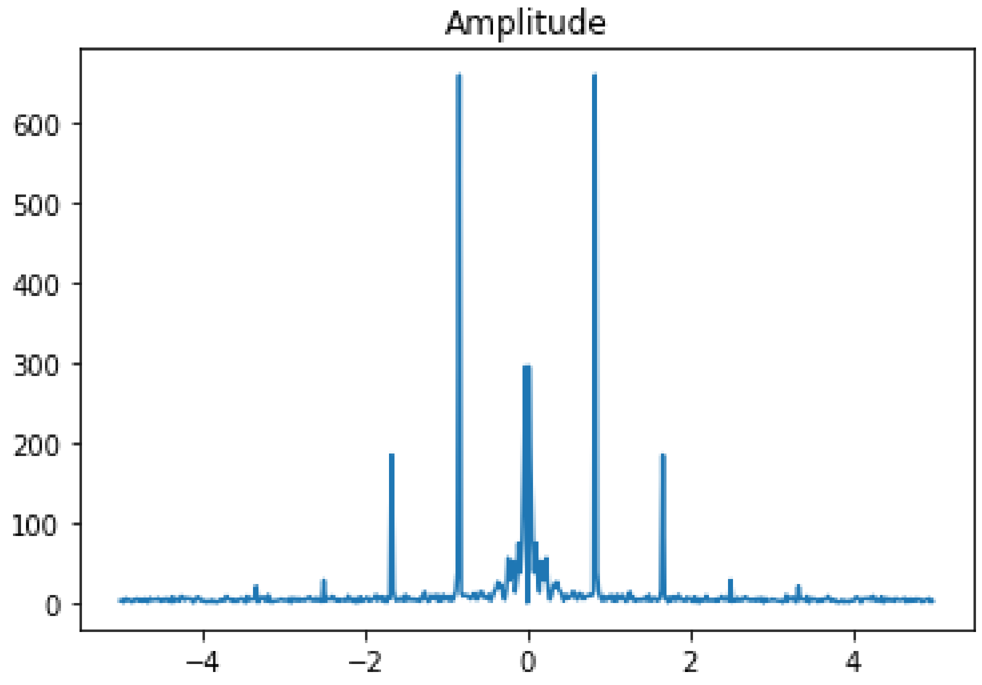


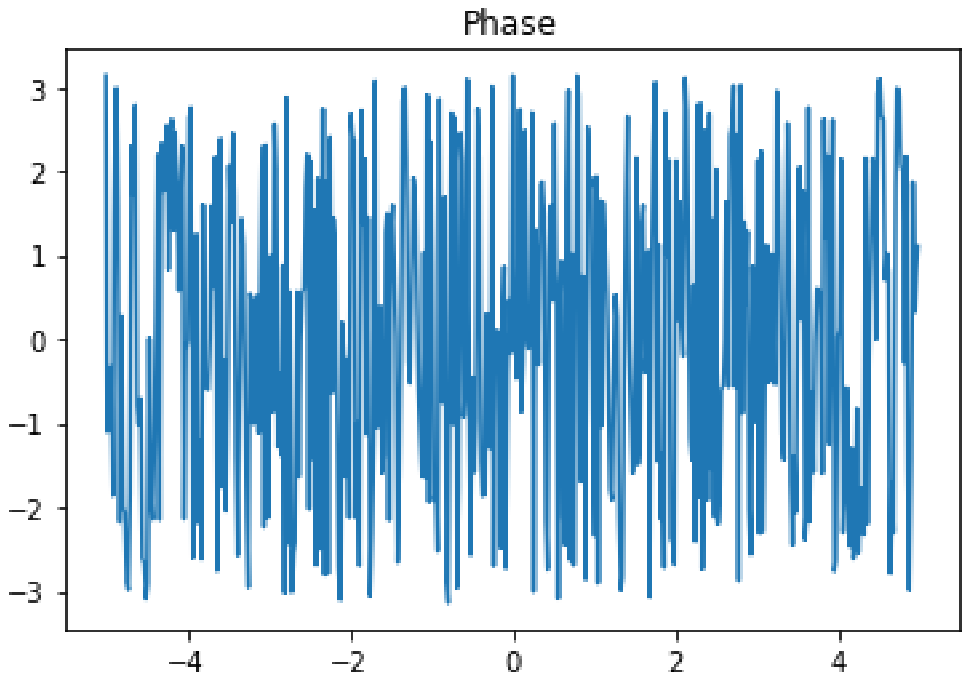
The notch-filter gives a better result than the f-domain filtered data as it removes more of the noise and focuses more on the trend. The f-domain filter looks similar to the original data, so it's not a very effective filter. The Notch-filter is better for overall trends, and the f-domain filter is better for maintaining the noise.

3.5 Part 5

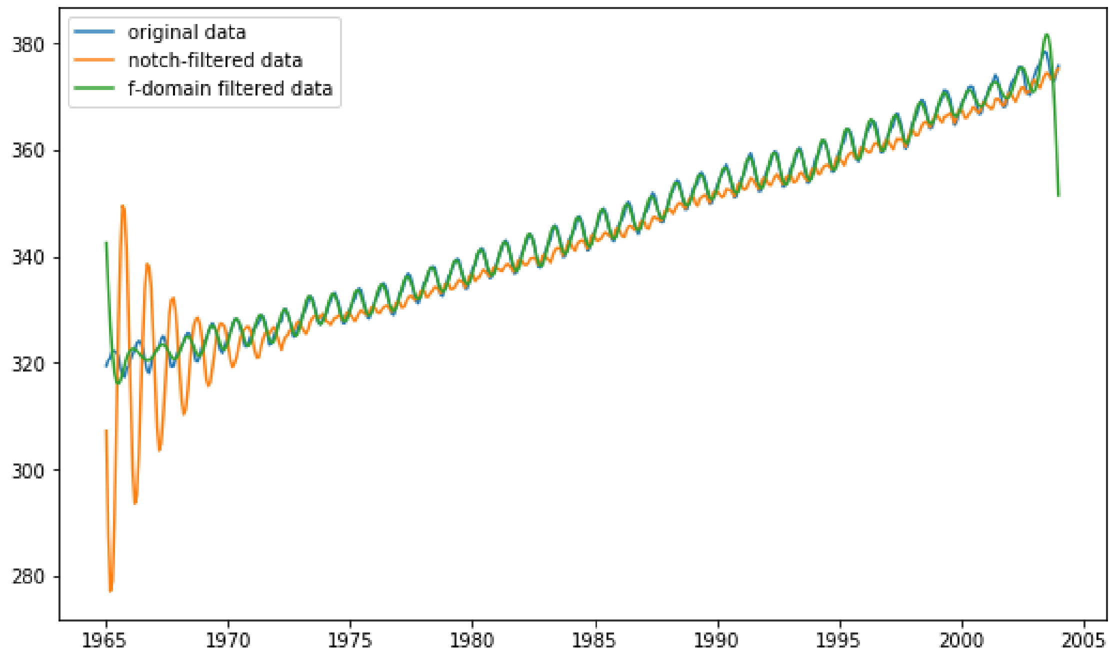
[11]:
```
# Part 2
co2filtered2 = ratFilter(N,D,co2Data)

# Part 3
co2ft2 = fftshift(fft(co2Data))
amp2 = np.abs(co2ft)
phase2 = np.angle(co2ft)
f_axis2 = fftshift(fftfreq(len(co2Data),0.1))

plt.plot(f_axis,amp)
plt.title('Amplitude')
plt.show()
plt.plot(f_axis,phase)
plt.title('Phase')
plt.show()

for i in range(len(f_axis2)):
    if f_axis2[i] > 0.9 or f_axis2[i] < -0.9:
        co2ft2[i] = 0
```

```
co2fdomain2 = ifft(ifftshift(co2ft2))
```





[12]:
```
# Part 4
plt.figure(figsize=(10,6))
plt.plot(time,co2Data,label='original data')
plt.plot(time,co2filtered2,label='notch-filtered data')
plt.plot(time,co2fdomain2,label='f-domain filtered data')
plt.legend()
plt.show()
```



We can see that without removing the trend before filtering, output is very messy and incorrect. We need to remove the trend so that it does not interfere with our filtering process.

4 4 Bonus question: Butterworth filter (2 marks)

Note: Please only do this part if you have extra time.
Low-pass Butterworth filter is very commonly used to filter out high-frequency noise in digital data. It has a power spectrum in the analytical form of

$$|H(\omega)|^2 = \frac{1}{1 + \left(\frac{\omega}{\omega_0}\right)^{2N}}$$

where ω_0 is the reference angular frequency below which input signal will be low-pass filtered through.

A common practice of converting the power spectrum response (in ω domain) into a rational filter (in z domain) is to use the bilinear transformation:

$$j\omega t = 2\frac{1-z}{1+z}$$

For simplicity, let's consider a second-order Butterworth low-pass filter, i.e. $N = 2$.

1. If we let $S(z) = H(z)H^*(1/z^*)$, prove that $|H(\omega)|^2 = S(z)\Big|_{z=e^{-i\omega\Delta t}}$.

   Hint: Note here $H^*(1 - z^*)$ is just an abstract way of saying its poles and zeros are conjugate reciprocal pairs of those of $H(z)$. Here we use $H(\omega)$ to denote the amplitude spectrum, i.e. $H(\omega) = H(z)\Big|_{e^{i\omega\Delta}}$.

2. Derive the expression for $S(z)$ through bilinear transformation of (3). Write it in terms of division of a numerator polynomial of z over a denominator polynomial of z.

3. Factor $H(z)$ out from $S(z)$ by picking the right poles and zeros. Write $H(z)$ in the form of

$$H(z) = \frac{a_0 + a_1 z + a_2 z^2}{1 + b_1 z + b_2 z^2}$$

   and give the expressions for a_0, a_1, a_2, b_1, b_2 in terms of ω_0 and Δt. Bear in mind that for $H(z)$ to be a stable filter, all of its poles should be outside the unit circle.

   Hint: To make your derivation easier, you may want to define some auxiliary variables along the way: $d = \frac{\omega_0 \Delta t}{2}$, $e_\pm = \frac{1 \pm i}{\sqrt{2}}$, and use the fact

$$A^4 + B^4 = (A + e_- B)(A - e_- B)(A + e_+ B)(A - e_+ B)$$

4. Set $f_0 = 1/20$ Hz ($\omega_0 = 2\pi f_0$), using `ratFilter()`, apply this low-pass Butterworth filter to the data $x(t)$ from Lab 1 (`RAYN.II.LHZ.sem`) and plot the filtered time series $y(t)$ on top of the original time series $x(t)$. Does this filter preserve the exact phase of the original time series?

5. Now apply the same filter to the *time-reversed* $y(t)$ and then *time-reverse* the filtered result (i.e. applying the Butterworth filter twice to $x(t)$, forward and then backward). Plot this result against the original time series $x(t)$. Is the phase preserved in this case? Why?