PHY408 Lab 3

Wednesday, March 20, 2024

Jace Alloway - 1006940802 - alloway1

This assignment was compiled in VS Code using the Python and LaTeX extensions. Matplotlib does not produce inline plots in terminal, so the `%matplotlib inline` command was commented out. **Collaborators for all questions: none.**

Problem 1

(**a**) The z-transform of a typical notch filter is

$$W(z) = M\frac{z-q}{z-p}\frac{z-\overline{q}}{z-\overline{p}} \tag{1.1}$$

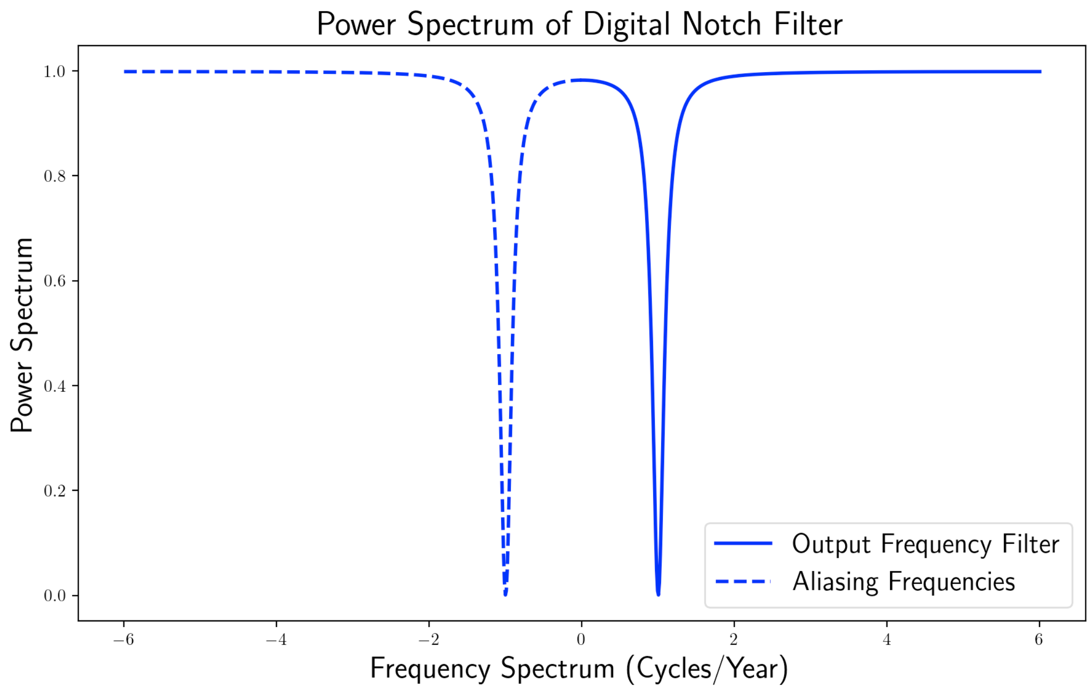
where M is the normalization factor, $q = e^{-2\pi i f_0/f_s}$ and $p = (1+\epsilon)q$. The sample rate is given as $f_s = \dfrac{1}{\Delta}$ and $0 < \epsilon \ll 1$ renders the width of the filter. We restrict the input of z to lie on the unit circle, $z(f) = e^{-2\pi i f/f_s}$ where f is the input frequency. The zeros of W are when $z = q$ and $z = \overline{q}$, which occur on the unit circle at an angle $2\pi f_0/f_s$. In a similar way, the poles of W are when $z = p$ and $z = \overline{p}$, which lie at the same angle $2\pi f_0/f_s$ but are just outside of the unit circle, at a radius $1+\epsilon$. Hence, reducing ϵ implies reducing the notch width, and increasing ϵ increases the width. Writing the denominators of (1.1) as power series expressions, we observe that

$$W(z) = \left(1-\frac{q}{z}\right)\left(1-\frac{\overline{q}}{z}\right)\left(\frac{1}{1-\frac{p}{z}}\right)\left(\frac{1}{1-\frac{\overline{p}}{z}}\right) \tag{1.2}$$

$$= \left(1-\frac{q}{z}\right)\left(1-\frac{\overline{q}}{z}\right)\left[\sum_{k=0}^{\infty}\left(\frac{p}{z}\right)^k\right]\left[\sum_{k=0}^{\infty}\left(\frac{\overline{p}}{z}\right)^k\right] \tag{1.3}$$

from which only converges if $|p| < 1$ and $|\overline{p}| < 1$, hence this filter is unstable since the poles lie outside of the unit circle (since $1+\epsilon > 1$). Since all the poles are outside of the unit circle, this further implies that $W(z)$ is realizable, and hence $W(z)$ only depends on previous and past inputs by convolution.

(**b**) We can plot the power spectrum by imposing an absolute square on the output array from (1.1) in the complex plane. Defining $z = e^{-2\pi i f/f_s}$, the frequency spectrum lies on the complex unit circle, where we define the spectrum to be `np.linspace(-f_s/2, f_s/2, 1000)` where $f_s/2$ is the Nyquist frequency.
Since the poles and zeros from (1.1) located on the complex unit circle occur in complex conjugate pairs, we observe output frequencies for f/f_s between 0 and π, and aliasing frequencies (plotted as a blue dashed line) for the equivalent f/f_s frequencies between π and 2π. In generality, we only take the $[0, \pi]$ interval as the frequency domain and scale it appropriately, since the inclusion of aliasing frequencies is redundant.



(c) The full-width at half-max of the notch can also be calculated, which is the distance between data points at the $y = 0.5$ level. This was calculated by programming a loop which takes in the closest samples of $|W(z)|^2$ within some interval range and consoling their maximum distance. The interval width was chosen to be 0.05, but is unstable for sharper / wider notches (since you're either including more or less samples within the interval). Within the for loop we check whether the power spectrum sample is within the interval `0.45 < power_spectrum[i] < 0.55` and append the array to include these values. Subtracting the maxima and minima from the array yield the FWHM, which was found to be approximately 0.19 years^{-1} for $\epsilon = 0.05$.
Since ϵ represents the distance between the zero and the pole in the complex plane, then increasing ϵ means widening the notch width (greater FWHM), and decreasing ϵ implies narrowing the notch, since we move the pole close to the zero.

Problem 2

(**a**) By expanding the numerator and denominator in (1.1), we find that

$$W(z) = M\frac{z^2+|q|^2-(q+\overline{q})z}{z^2+|p|^2-(p+\overline{p})z} \tag{2.1}$$

$$= M\frac{z^2+1-2\cos\left(2\pi\frac{f_0}{f_s}\right)z}{z^2+(1+\epsilon)^2-2(1+\epsilon)\cos\left(2\pi\frac{f_0}{f_s}\right)z} \tag{2.2}$$

$$= \frac{M}{(1+\epsilon)^2}\frac{1-2\cos\left(2\pi\frac{f_0}{f_s}\right)z+z^2}{1-\frac{2}{1+\epsilon}\cos\left(2\pi\frac{f_0}{f_s}\right)z+\frac{z^2}{(1+\epsilon)^2}} \tag{2.3}$$

which yields the numerator and denominator z-transform coefficients,

$$\text{a, b, c} = \left[\frac{M}{(1+\epsilon)^2},\ -\frac{2M}{(1+\epsilon)^2}\cos\left(2\pi f_0/f_s\right),\ \frac{M}{(1+\epsilon)^2}\right] \tag{2.4}$$

$$\text{1, B, C} = \left[1,\ -\frac{2}{1+\epsilon}\cos\left(2\pi f_0/f_s\right),\ \frac{1}{(1+\epsilon)^2}\right]. \tag{2.5}$$

(**b**) To write a real filter function given through (1.1), we must first understand the recursion between feedback filtering. Let us suppose we have a transfer function given as

$$H(z) = \frac{O(z)}{I(z)} \tag{2.5}$$

where $I(z)$ is the z-transform of the input and $O(z)$ the z-transfom of the output. The output is thus given as $O(z) = H(z)I(z)$, and $H(z)$ is written as a fraction composed of the z-transforms of the feedback and feedforward arrays:

$$O(z) = \sum_k o_k z^k = \frac{\sum_n f_n z^n}{\sum_m b_m z^m}\sum_\ell i_\ell z^\ell = H(z)I(z) \tag{2.6}$$

which implies that

$$\sum_m\sum_k o_k b_m z^{k+m} = \sum_n\sum_\ell i_\ell f_n z^{n+\ell}. \tag{2.7}$$

By re-indexing ($q = m+k$; $p = n+\ell$), we can write the z-transform as

$$\sum_q\sum_m o_{q-m} b_m z^m z^{q-m} = \sum_p\sum_n i_{p-n} b_n z^n z^{p-n} \tag{2.8}$$

$$\implies \sum_q\sum_m o_{q-m} b_m z^q = \sum_p\sum_n i_{p-n} b_n z^p \tag{2.9}$$

which allows us to undo the z-transform of either side to match array components to obtain

$$\sum_m b_m o_{q-m} = \sum_n f_n i_{q-n} \tag{2.10}$$

therefore

$$b_0 o_q + \sum_{m\geq 1} b_m o_{q-m} = \sum_{n\geq 0} f_n i_{q-n} \tag{2.11}$$

$$\implies o_q = \frac{1}{b_0}\left[\sum_{n\geq 0} f_n i_{q-n} - \sum_{m\geq 1} b_m o_{q-m}\right] \tag{2.12}$$

under the assumption that $b_0 \neq 0$ in the feedback array (the denominator), which we impose to be 1.

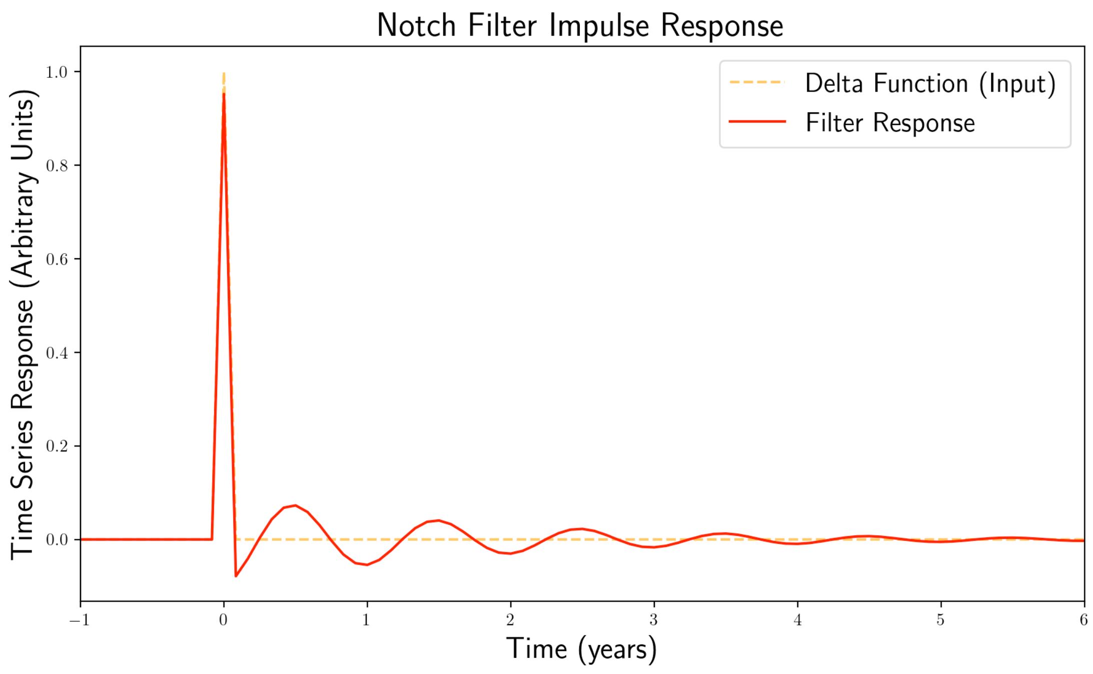
We note that (2.12) consists of two discrete convolution terms: one term convolving the input array with the feedforward array coefficients, and the other previous output values with the feedback coefficients given from the denominator fo the filter. The current output o_q is thus a recursion relation between inputs, coefficients, and previous output values.

We program (2.12) into Python by defining the length of the output as an array of zeros, then by convolving the input array with the feedforward coefficients, while simultaneously subtracting the convolution between the feedback and output arrays.

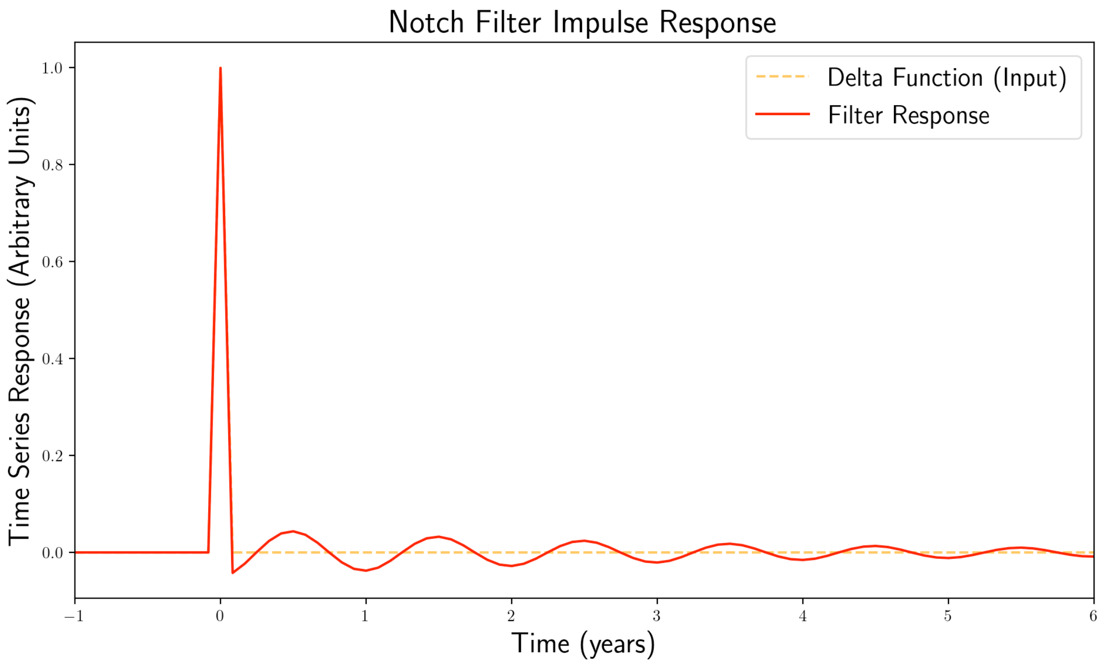
```
def ratFilter(N, D, x):
"""
N: Numerator Array
D: Denominator Array with D[0]=1
x: Input Time Series
"""
output=np.zeros(len(x))
for i in range(len(x)):
    output[i] = 1/D[0] * (np.convolve(N, x)[i] - np.convolve(D, output)[i])

return output
```

Observe that by defining the output arrays first by zeros, we have access to each element which is written over with a non-zero value for each convolution value. Otherwise the convolved terms are zero.

(c)

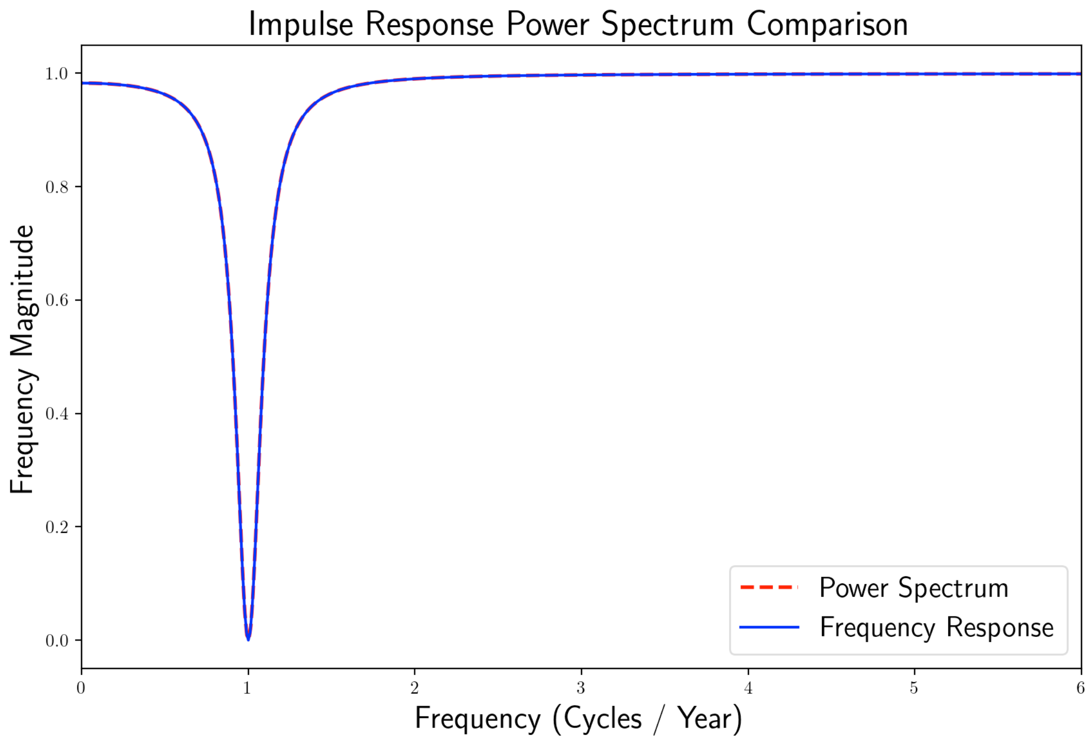


The impulse response of the notch filter was calculated by using the filter defined in (2.12) with an input spectrum delta function, `x = [1, 0, 0, 0, ...]`. Using the same numerator and denominator coefficients as in (2.4) and (2.5), we find that the impulse trails with a periodic wave of frequency f_0.
If we were to halve the FWHM, this means (approximately) halving ϵ (to make the peak more narrow), which means filtering a more precise frequency. If the filtered frequency is more precise, we then expect less uncertainty in the temporal response, so we would expect to see a lengthening in the trailing periodic wave after the delta response. This is what was observed when ϵ was scaled to $\epsilon = 0.025$:



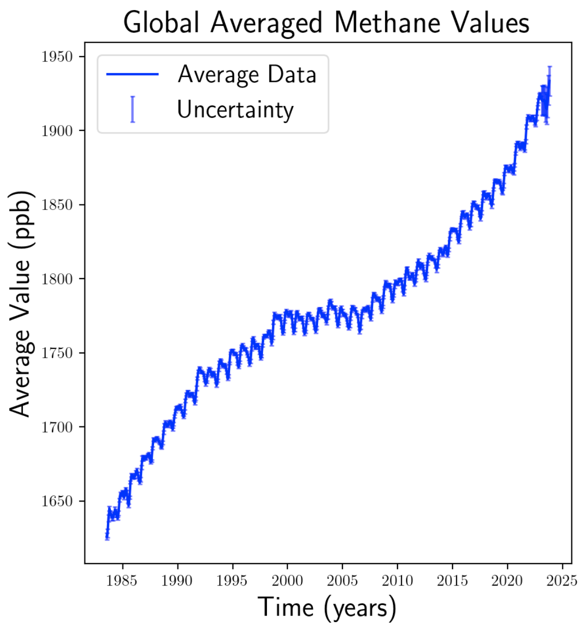
In a similar way, doubling the FWHM implies filtering a broader range of frequencies, which would increase the uncertainty in the filtered impulse, making the periodic tail shorter.
Note that I chose to plot $(-1, 6)$ years instead of $(0, 6)$ to show the absence of feedback prior to the input signal entering the filter.

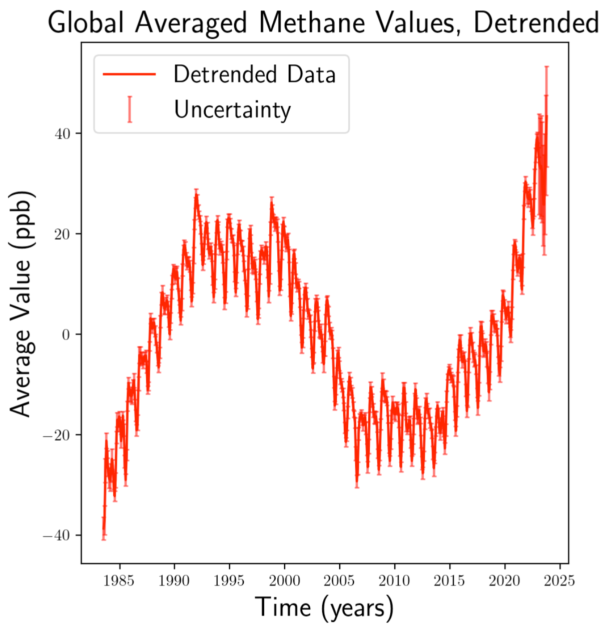
(d) Using `np.fft.fft()`, the impulse response from part (c) was fast Fourier transformed to obtain the frequency response. This response corresponds to the filter's response to a range of frequencies, which aligns with the square root of the power spectrum calculated in question 1. To show that the frequency response is the power spectrum, after shifting the FFT, the absolute value and square was taken out using `np.square(np.abs())`. We plot the spectrum in the absence of aliasing: from 0 to 6 cycles per year.


Impulse Response Power Spectrum Comparison
Frequency Magnitude
Frequency (Cycles / Year)
Power Spectrum
Frequency Response
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6

Problem 3

(**a**) The data was first imported using `pandas.read_csv()` and unpacking the data according to the column labels. The 'decimal' time array was plotted with the 'average' and the errorbars. Using `slope, y_int = np.polyfit(time, average, 1)` (first degree polynomial), a straight line was calculated to minimize the least squares for a first degree polynomial, and the trend was then removed.





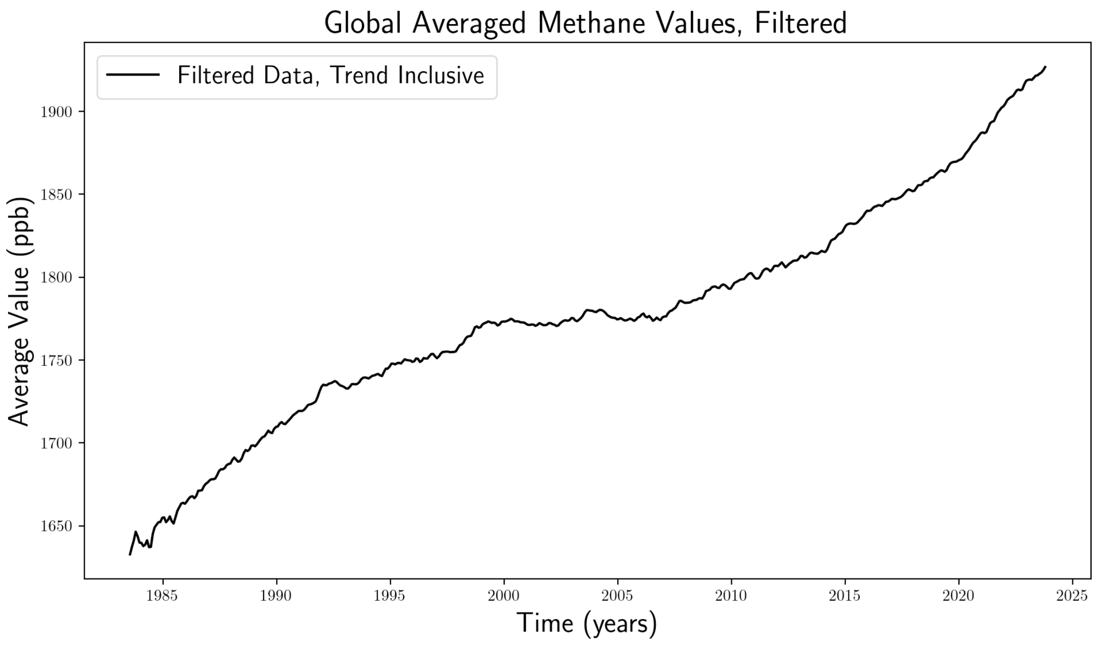
For the sake of plot clarity, I will not be plotted errorbars for the remainder of this part of the lab.

(**b**) We now aim to apply the notch filter to the detrended data to attempt to filter out the annual oscillations and to clarify the trend over the timeline. The frequencies of the data are close to zero (the large oscillatory sine wave), $1, 2, 3$ and 4 (I kind of cheated here, and FFT'd the data first to see the frequency distribution to see what frequencies I needed to filter out instead of just guessing). I applied the filters in series in a 'for' loop, each filtering out different frequencies with each pass. The notch width (the 'Q') could also be adjusted, but I found leaving it at $\epsilon = 0.05$ was sufficient.

```
f_0s = (1, 2, 3, 4) #filter freqs
eps = (0.05, 0.05, 0.05, 0.05)  #Q adjust
numerators = []
denominators = []
for i in range(len(f_0s)):
    num = [M/((1+eps[i])**2),
            -2*M / ((1+eps[i])**2) * np.cos(2*np.pi*f_0s[i]/f_s),
            M /((1+eps[i])**2)]
    den = [1,
            -2/(1+eps[i])*np.cos(2*np.pi*f_0s[i]/f_s),
            1/((1+eps[i])**2)]
    numerators.append(num)
    denominators.append(den)
```

```
filtering = detrended
for i in range(len(f_0s)):
    filtering = ratFilter(numerators[i], denominators[i], filtering)

retrended = filtering + line_trend
```

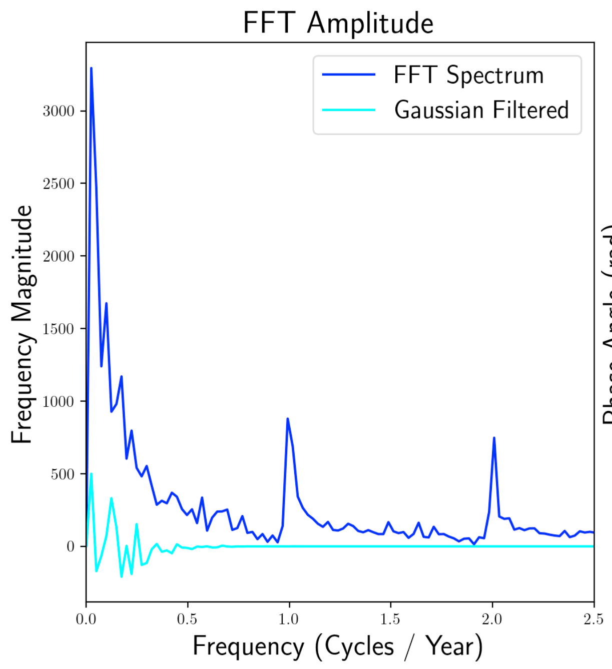
In the last line the trend was added back, and the filtering was plotted.

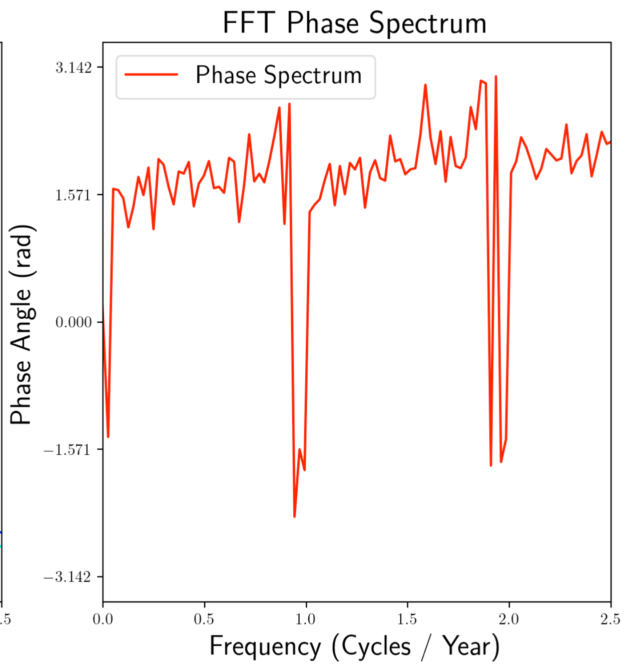


After applying the notch filter four times, the annual oscillation(s) were removed, but there was still a lot of noise in the data. To remove this, one would need to construct a lowpass filter (I tried this, but I found it hard to pinpoint the cutoff frequency else I would get a straight line back when I added back the trend).

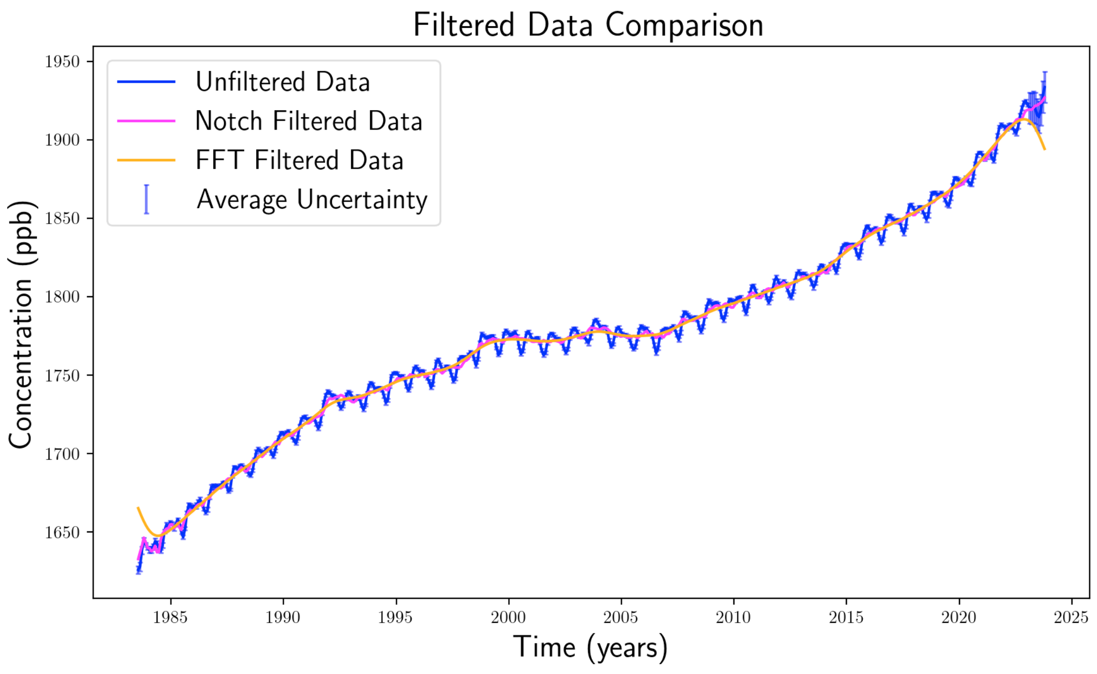
(c) To show a similar method of filtering, the data was FFT'd into the frequency spectrum and it's magnitude (`np.abs()`) and phase spectrum (`np.angle()`) was plotted for frequency cycles between 0 and 2.5 cycles/year.
Since the annual oscillations were made up of frequencies higher than 0.9 cycles/year, a Gaussian function centered at 0 was multiplied into the frequency domain to filter out higher frequencies, hence filtering out noise (I just found 0.4 to be the best width to filter the noise out).

```
    #filter the data with a ""gaussian""
gauss_filt = ft * (lambda x, t: np.exp(-x**2 / (t**2)))(f_axis, 0.4)
manual_filt = np.fft.ifft(np.fft.ifftshift(gauss_filt))
```



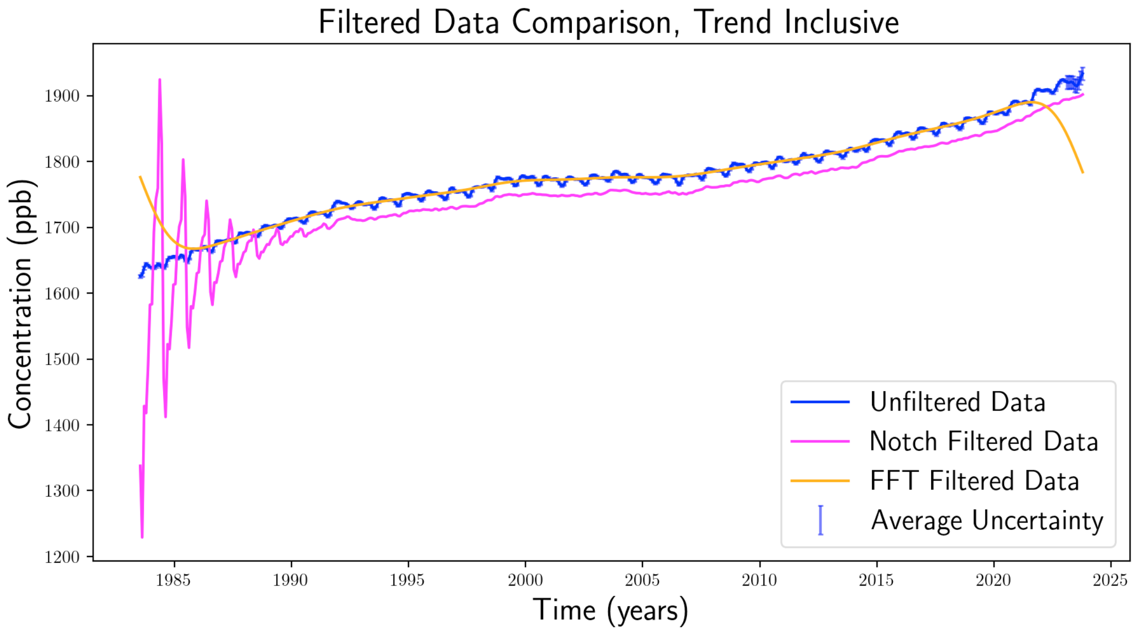


One can then inverse-FFT the manually filtered Gaussian data and add back the trend to obtain the set of filtered data, which can then be compared with the notch-filtered outputs and the initial trend given in the methane data:



Though both methods yield a result which aligns with the trend of the data, the notch-filtered data still includes a lot of noise which is not included in the FFT Gaussian-filtered data, so I would prefer to use a FFT Gaussian filter to remove unwants frequencies, although notch filters are good for removing specific frequencies. Another option would be to low-pass the notch-filtered data, but this wasn't investigated. At the same time, the FFT data is distorted at the data endpoints because of the effects of discontinuity on the input data, while the notch-filtered data is not. This can be fixed by slicing the distortions of out the data arrays.

(d) Now, returning to the initial data prior to removing the trend, we run the same code with the untrended data: filtering out frequencies in a loop, FFT'ing the data to Gaussian filter it, and again plotting the result:



We observe two phenomenon between the notch-filtered data and the FFT Gaussian-filtered data: the first being signal amplification at the beginning of the signal, and the second being an emphasis on endpoint discontinuity. Removing the linear trend prior to filtering is important because you must remove bias to achieve pure periodic behaviour in your data, which reflects the true trend. The presence of a bias in a time signal implies a misrepresentation of the frequencies which compose the input signal, leading to either a removal or an amplification of the wrong frequencies depending on filter values.