Phys 386 HW 4

(1) Propose two ideas for your final project

> **Idea 1:** Perform a simulation of the measurement of the muon revolution frequency distribution in the g-2 experiment by using the partial-time Fourier transform of the g-2 signal. This is interesting since a standard Fourier transform leads to the wrong estimation of the revolution frequency distribution.
>
> References:
>
> Y. Orlov, C.S. Ozben and Y.K. Semertzidis, "Muon revolution frequency distribution from a partial-time Fourier transform of the g-2 signal in the muon g-2 experiment", *Nucl. Instrum. Methods Phys. Res. Sect. A*, vol. 482, pp. 767-775, Jul. 2001.
>
> T. Albahri *et al.*, "Beam dynamics corrections to the Run-1 measurement of the muon anomalous magnetic moment at Fermilab", *Phys. Rev. Accel. Beams*, vol. 24, p. 044002, Apr. 2021.

> **Idea 2:** Something related to uncertainty quantification in particle accelerator modeling. Maybe running a simulation of an accelerator, constructing a model of the beam's phase-space with simulation data, and then provide the uncertainty of the reconstruction.
>
> Reference:
>
> A. A. Mishra, A. Edelen, A. Hanuka and C. Mayes, "Uncertainty quantification for deep learning in particle accelerator applications", *Phys. Rev. Accel. Beams*, vol. 24, p. 114601, Nov. 2021

(2) Generate the following data vectors that are 1024 samples long with each sample corresponding to the uniformly sampled time between zero and one second.

- Time: Generate a data vector giving the time of each sample.
- Low: S/N: Generate a simulated data vector with a sine wave that is 0.1 s in period, with an amplitude of 1 and a sine wave with amplitude 0.5 and period of 0.07s Add gaussian noise with an amplitude of 10.
- High: S/N: Generate a simulated data vector with a cosine wave that is 0.1 s in period, with an amplitude of 1. Add noise with an amplitude of 0.5

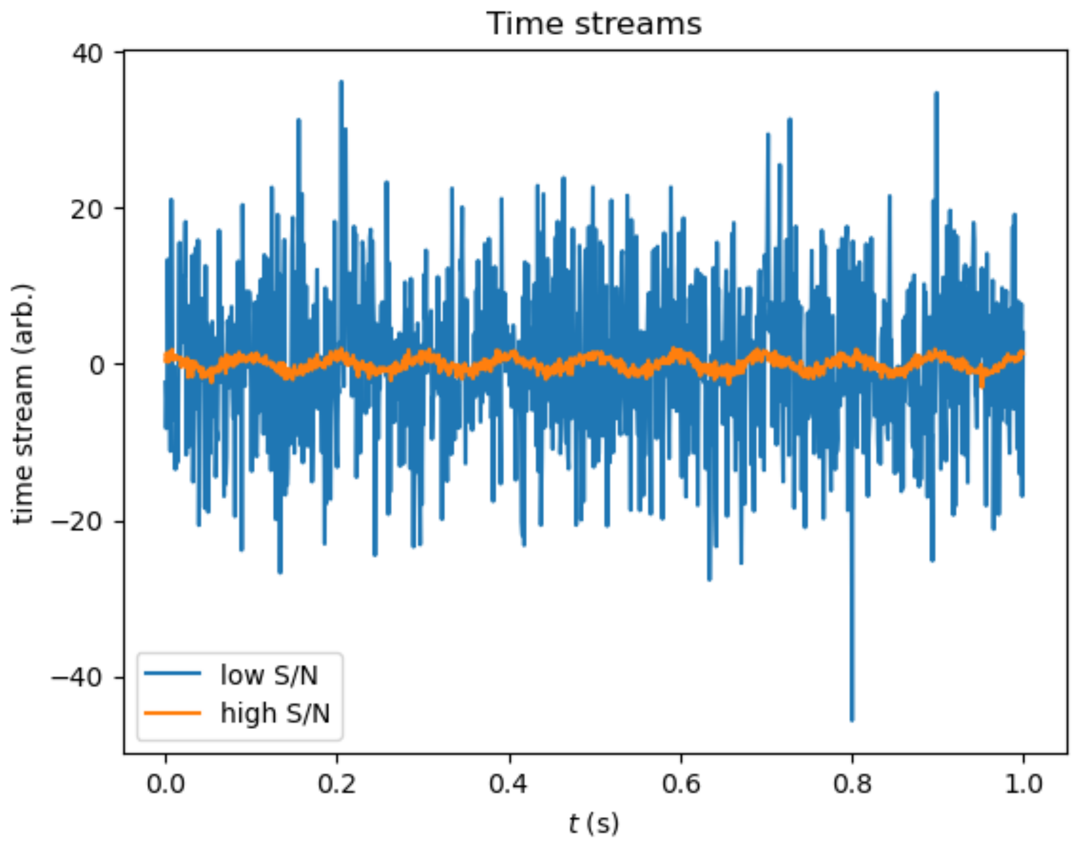
Present plots of these time streams, and then use the following methods to detect the signals in these.

The low pass, and high pass filters correspond to (1) fourier transforming your data, and multiplying by a mask (1 to let data through, zero to filter). Show the steps of applying these filters (data in fourier space, the filter in fourier space, and the output). Make sure you keep track of what the correspondence between bin and frequency is in fourier space. Also, if you get weird results the most likely thing you need to do is to apply the fourier shift command to properly align the arrays. Note– if you have done this right, the filtered time stream will have the signal line up with the input if you turn the S/N up.

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

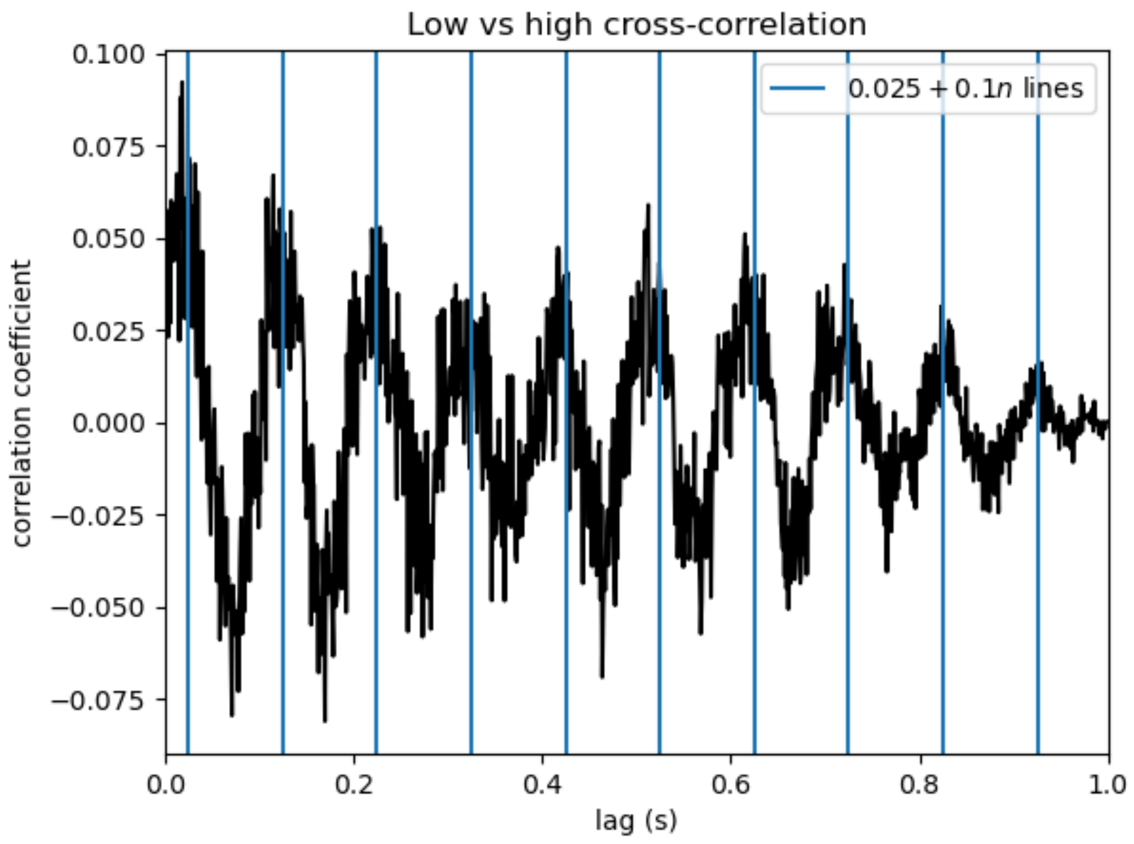
```
In [ ]: n_samples = 1024
        time = np.linspace(0, 1, n_samples)
        low_sn = (np.sin( time * 2*np.pi/0.1 )
                  + 0.5 * np.sin( time * 2*np.pi/0.07 )
                  + 10.0 * np.random.randn( n_samples ) )

        high_sn = (np.cos( time * 2*np.pi/0.1 )
                   + 0.5 * np.random.randn( n_samples ) )
```

```
In [ ]: plt.plot(time, low_sn, label='low S/N')
        plt.plot(time, high_sn, label='high S/N')
        plt.legend()
        plt.xlabel('$t$ (s)')
        plt.ylabel('time stream (arb.)')
        plt.title('Time streams')
        plt.show()
```



(a) Compute the correlation of the low and high signal to noise time streams. Estimate the signal to noise of the detection. What does the offset at which the correlation peaks tell you?

```
In [ ]: from scipy import signal
        corr1 = signal.correlate((low_sn-np.mean(low_sn))/np.std(low_sn),
                                 (high_sn-np.mean(high_sn))/np.std(high_sn),
                                 method='direct') / len(time)
        lags = signal.correlation_lags(len(low_sn), len(high_sn))/len(time)
        plt.plot(lags, corr1, 'k')
        plt.xlim(0,1)
        plt.axvline(0.025, label='$0.025 + 0.1n$ lines')
        plt.axvline(0.125)
        plt.axvline(0.225)
        plt.axvline(0.325)
        plt.axvline(0.425)
        plt.axvline(0.525)
        plt.axvline(0.625)
        plt.axvline(0.725)
        plt.axvline(0.825)
        plt.axvline(0.925)
        plt.ylabel('correlation coefficient')
        plt.xlabel('lag (s)')
        plt.title('Low vs high cross-correlation')
        plt.legend()
        plt.show()
```

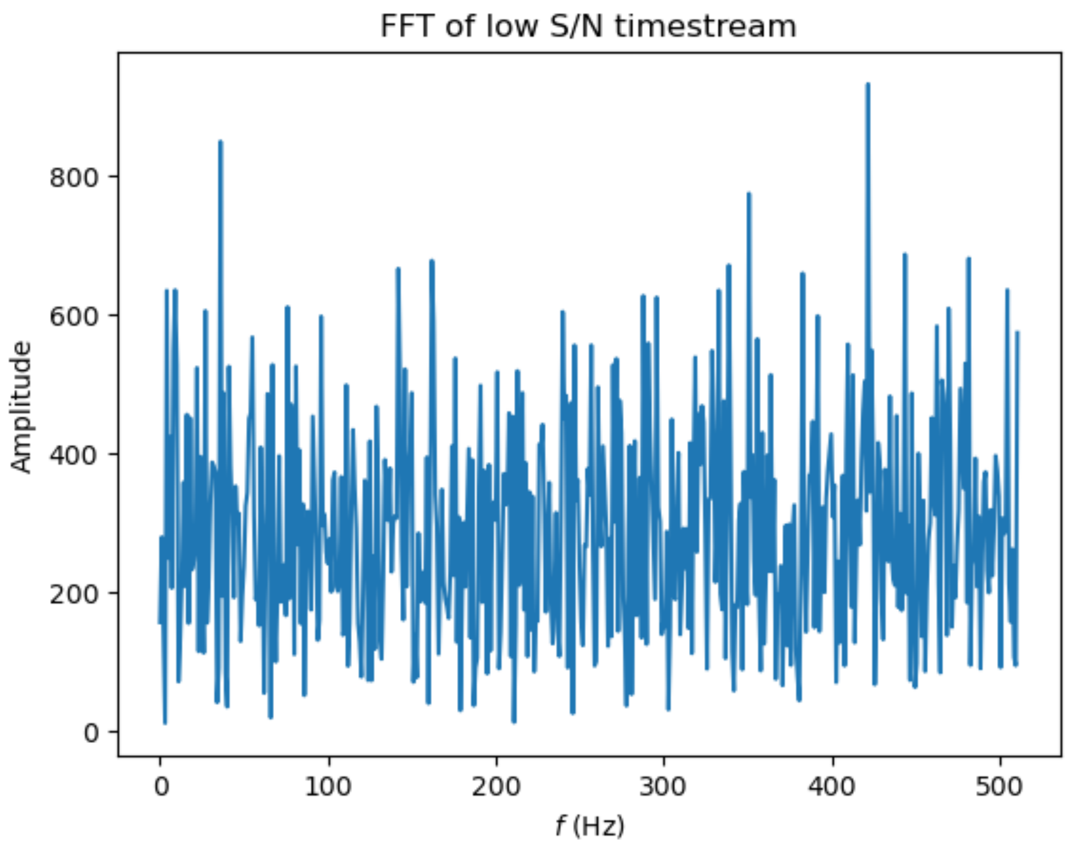


> The S/N of the detection is around 2.

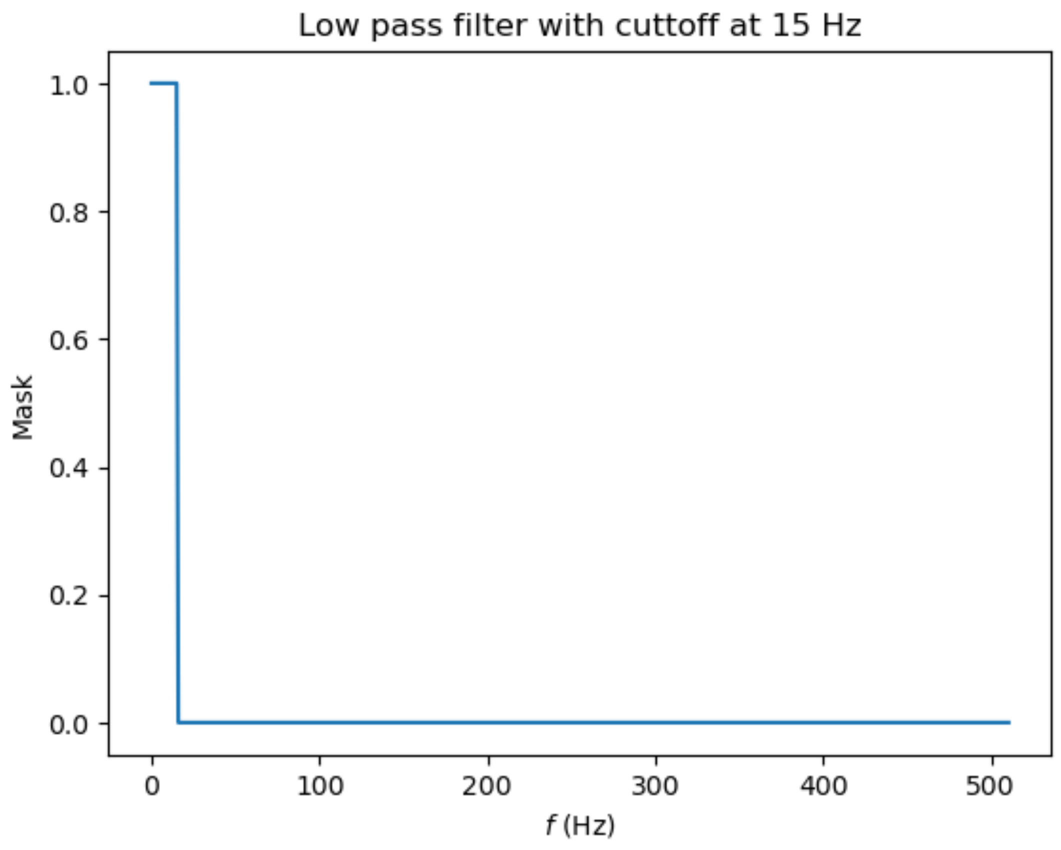
> Note that the cross-correlation seems to have periodic peaks with period 0.1 and with an offset of 0.025. This makes sense since sine and cosine are out of phase by quarter of a period, i.e., there is constructive interference at a lag of a fourth of a period.

(b) Use a low pass filter to see if you can isolate the low S/N signals

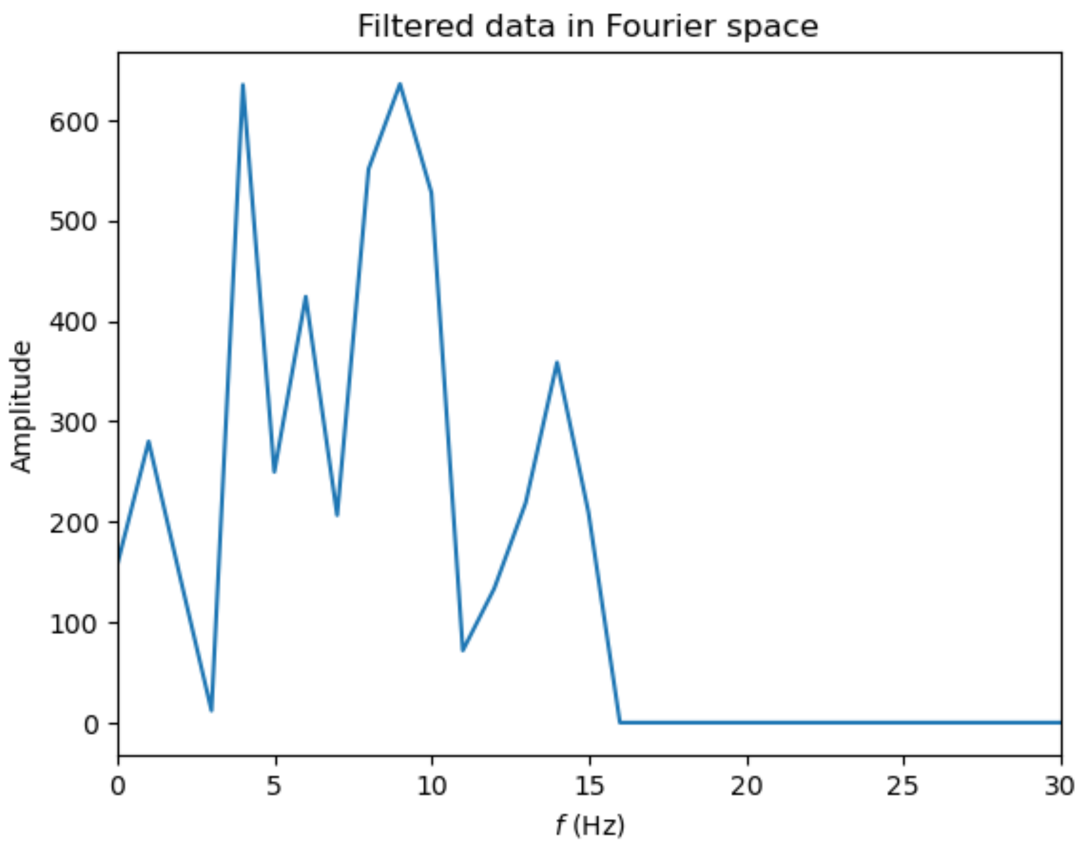
```
In [ ]: low_ft = np.fft.fft(low_sn)
        freq = np.fft.fftfreq(len(time), d=time[1]-time[0])
        plt.plot(freq[:int(n_samples/2)], np.abs(low_ft)[:int(n_samples/2)])
        plt.xlabel('$f$ (Hz)')
        plt.ylabel('Amplitude')
        plt.title('FFT of low S/N timestream')
        plt.show()
```



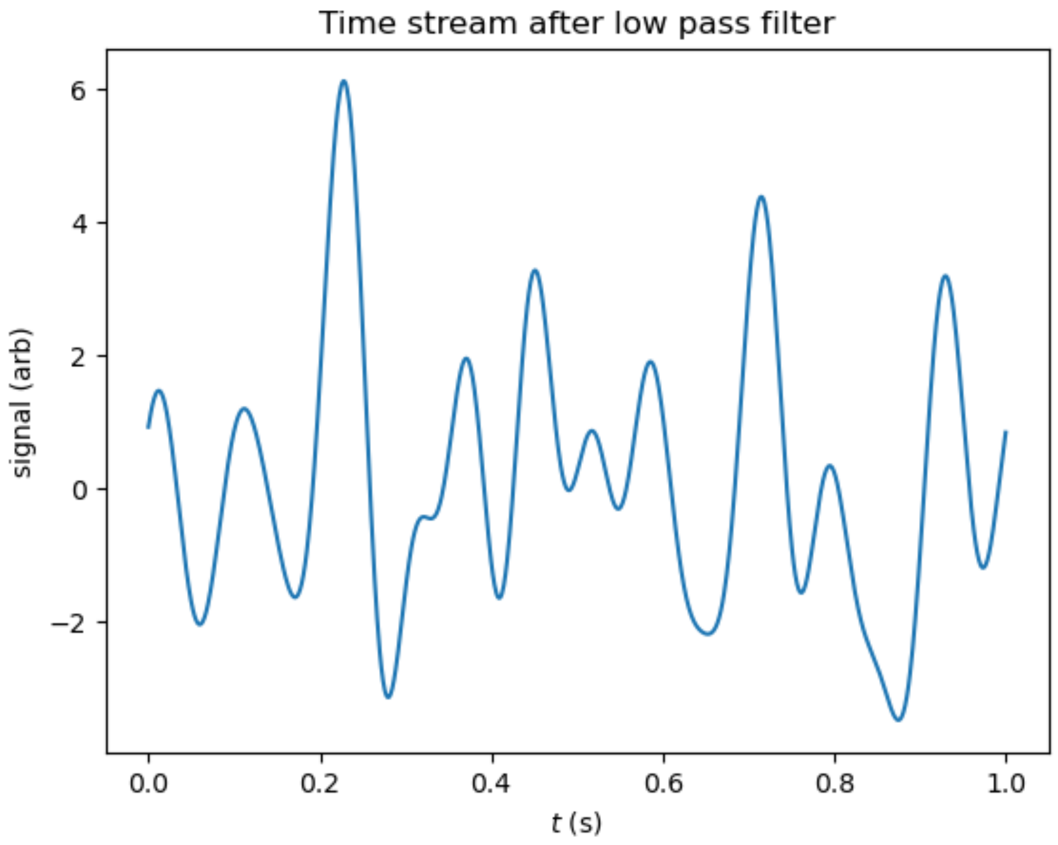
```
In [ ]: lowpass_cutoff = 15 #  Low pass filter cuttoff in Hz
        lowpass_filter = np.where(np.abs(freq) > lowpass_cutoff, 0, 1)
        plt.plot(freq[:int(n_samples/2)], lowpass_filter [:int(n_samples/2)])
        plt.xlabel('$f$ (Hz)')
        plt.ylabel('Mask')
        plt.title(f'Low pass filter with cuttoff at {lowpass_cutoff} Hz')
        plt.show()
```



```
In [ ]: lp_filtered_fft = low_ft * lowpass_filter
        plt.plot(freq[:int(n_samples/2)], np.abs(lp_filtered_fft)[:int(n_samples/2)])
        plt.xlabel('$f$ (Hz)')
        plt.ylabel('Amplitude')
        plt.title('Filtered data in Fourier space')
        plt.xlim(0, 30)
        plt.show()
```

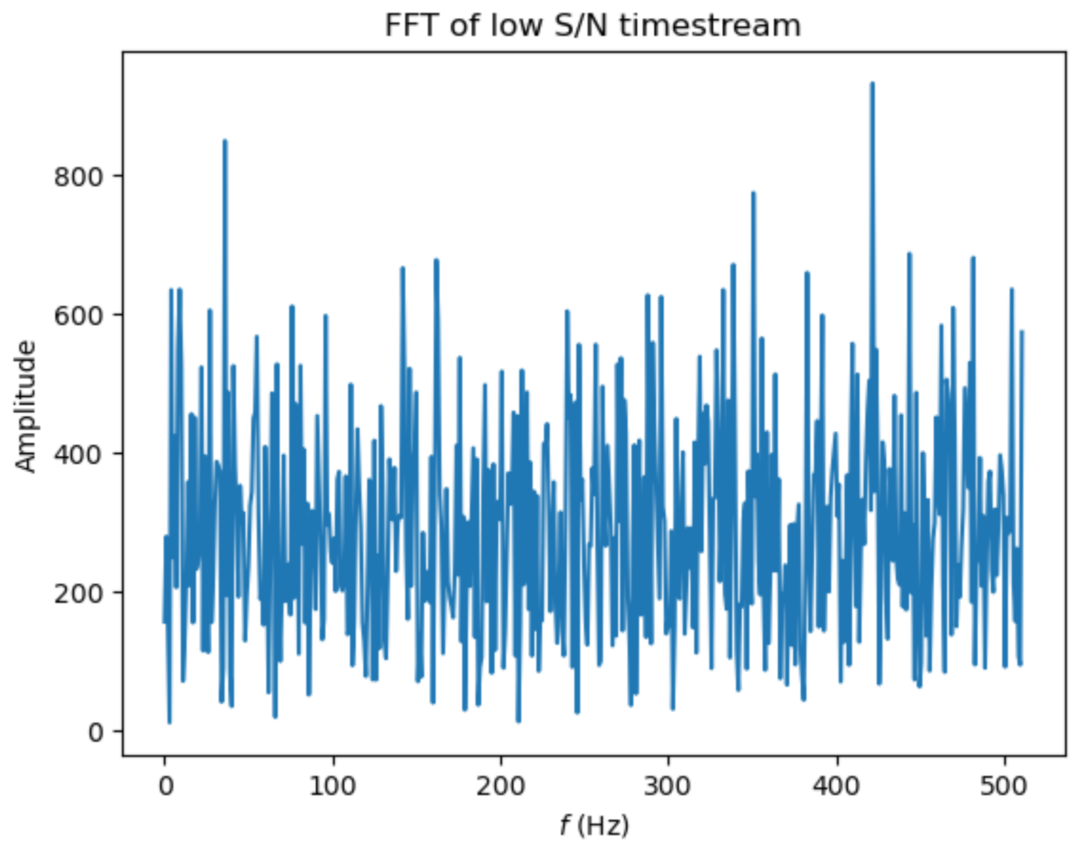


```
In [ ]: low_ft_lowpass = np.fft.ifft(lp_filtered_fft)
        plt.plot(time, low_ft_lowpass.real)
        plt.xlabel('$t$ (s)')
        plt.ylabel('signal (arb)')
        plt.title('Time stream after low pass filter')
        plt.show()
```

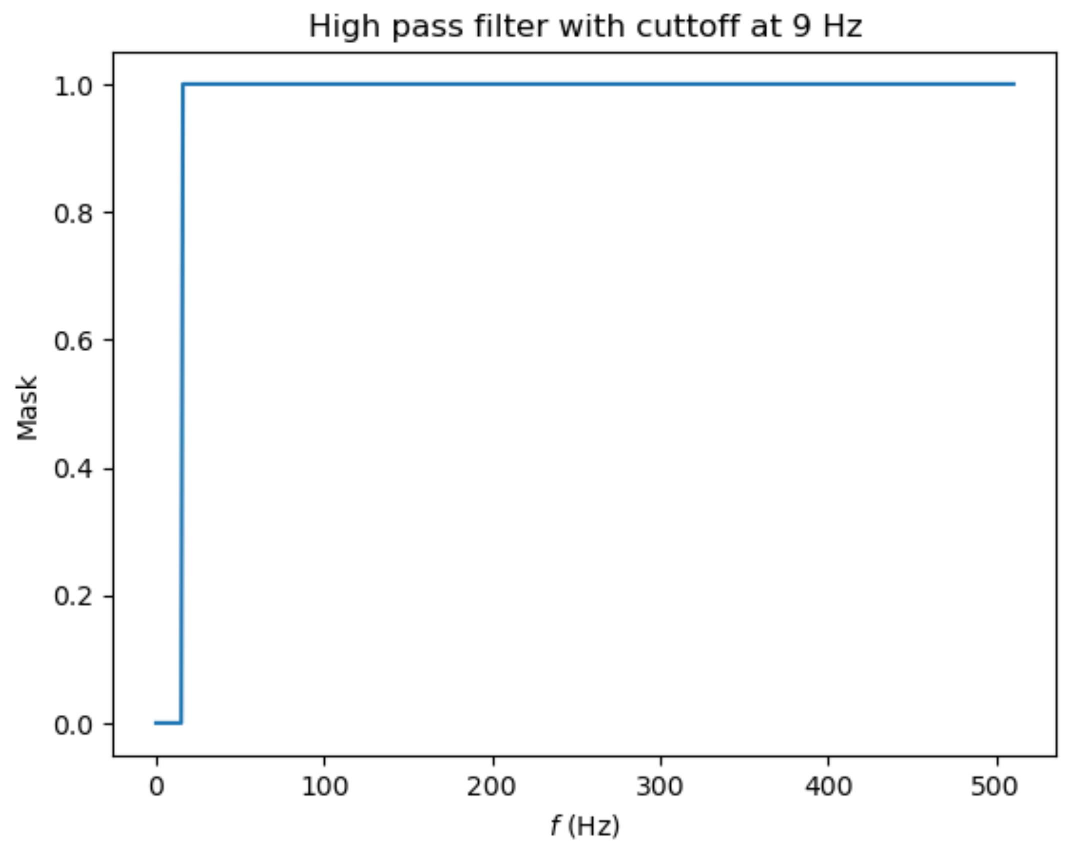


(c) Use a high pass filter to see if you can isolate the low S/N signals

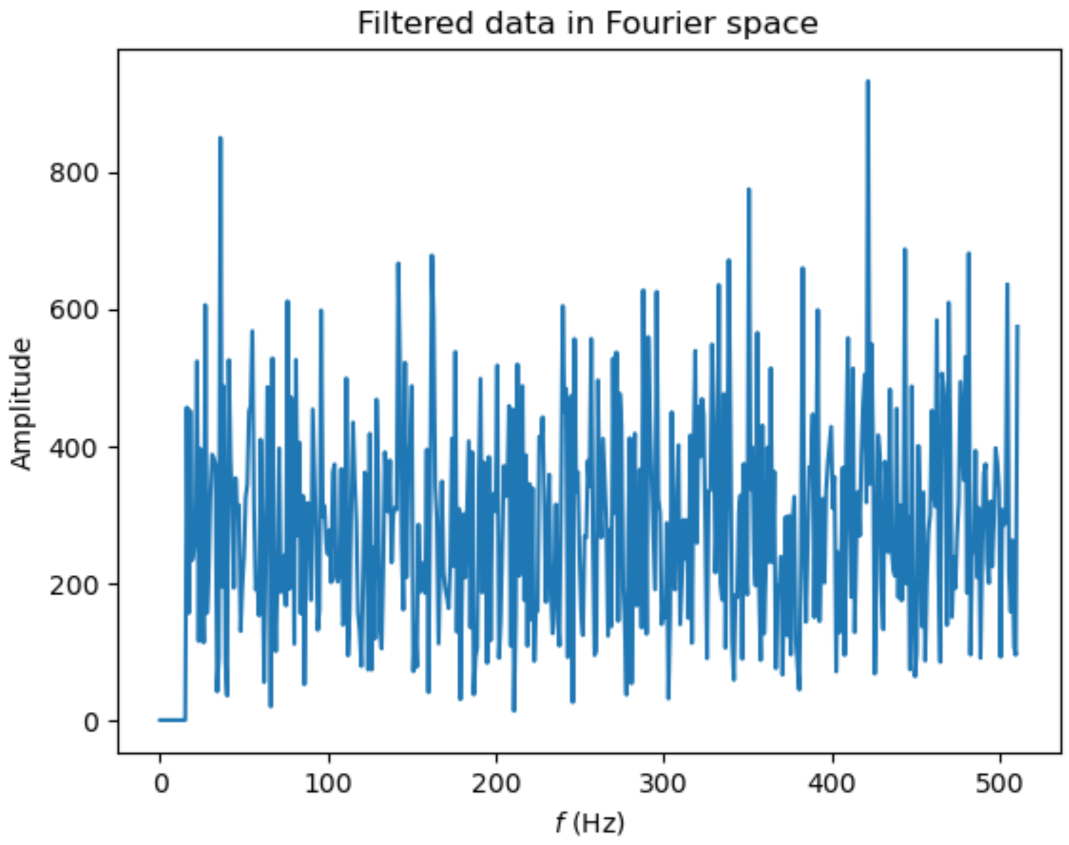
```
In [ ]: low_ft = np.fft.fft(low_sn)
        freq = np.fft.fftfreq(len(time), d=time[1]-time[0])
        plt.plot(freq[:int(n_samples/2)], np.abs(low_ft)[:int(n_samples/2)])
        plt.xlabel('$f$ (Hz)')
        plt.ylabel('Amplitude')
        plt.title('FFT of low S/N timestream')
        plt.show()
```



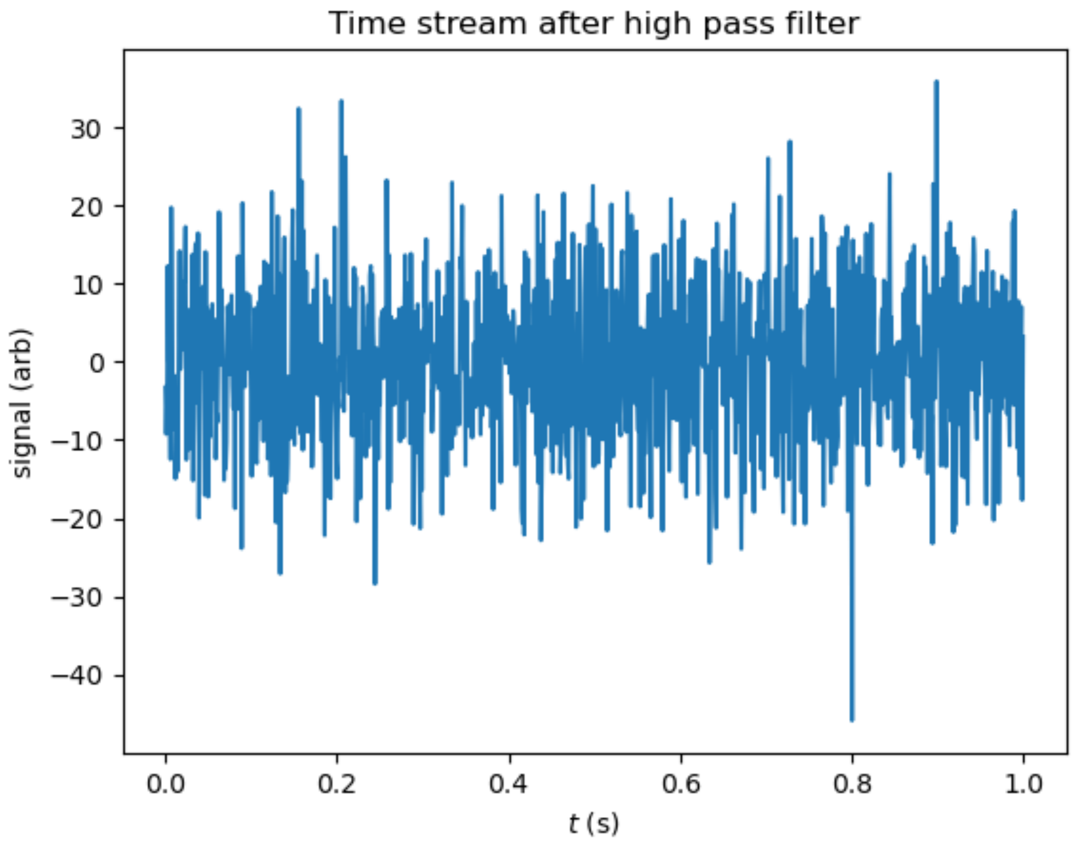
```
In [ ]: highpass_cutoff = 9 #  High pass filter cuttoff in Hz
        highpass_filter = np.where(np.abs(freq) < lowpass_cutoff, 0, 1)
        plt.plot(freq[:int(n_samples/2)], highpass_filter [:int(n_samples/2)])
        plt.xlabel('$f$ (Hz)')
        plt.ylabel('Mask')
        plt.title(f'High pass filter with cuttoff at {highpass_cutoff} Hz')
        plt.show()
```



In [ ]:
```
hp_filtered_fft = low_ft * highpass_filter
plt.plot(freq[:int(n_samples/2)], np.abs(hp_filtered_fft)[:int(n_samples/2)])
plt.xlabel('$f$ (Hz)')
plt.ylabel('Amplitude')
plt.title('Filtered data in Fourier space')
plt.show()
```

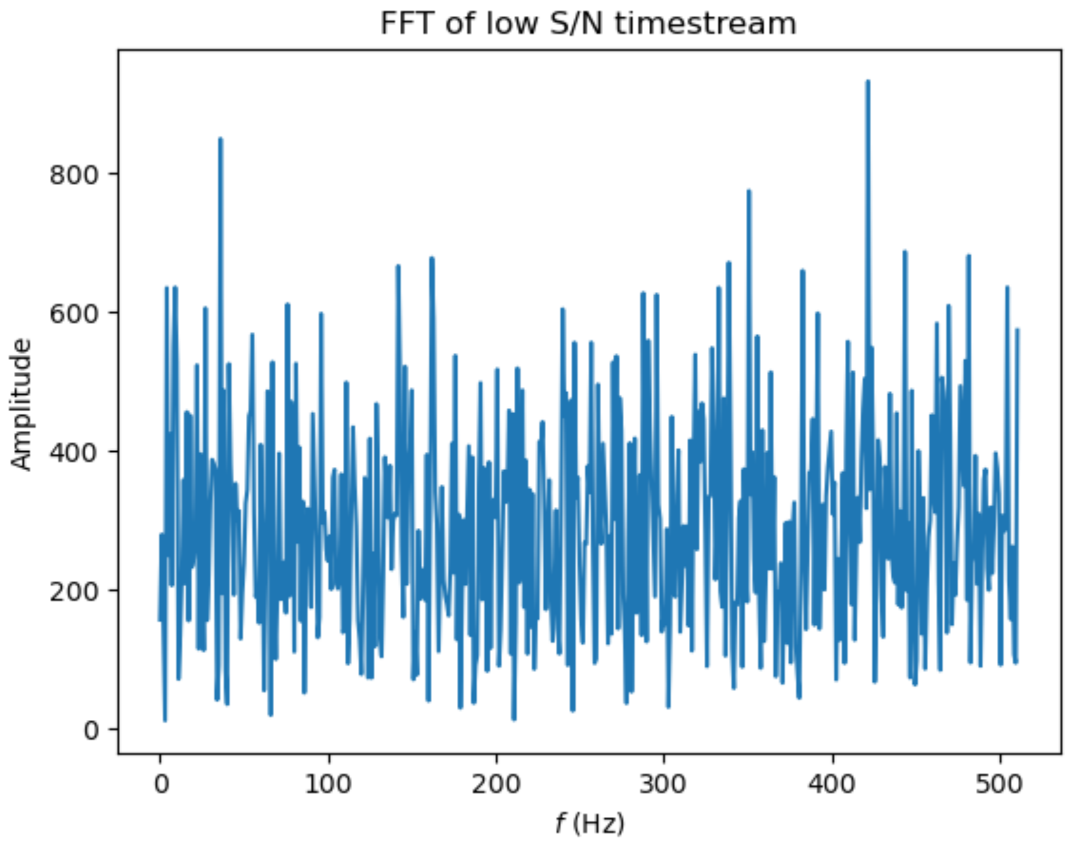


In [ ]:
```
low_ft_highpass = np.fft.ifft(hp_filtered_fft)
plt.plot(time, low_ft_highpass.real)
plt.xlabel('$t$ (s)')
plt.ylabel('signal (arb)')
plt.title('Time stream after high pass filter')
plt.show()
```

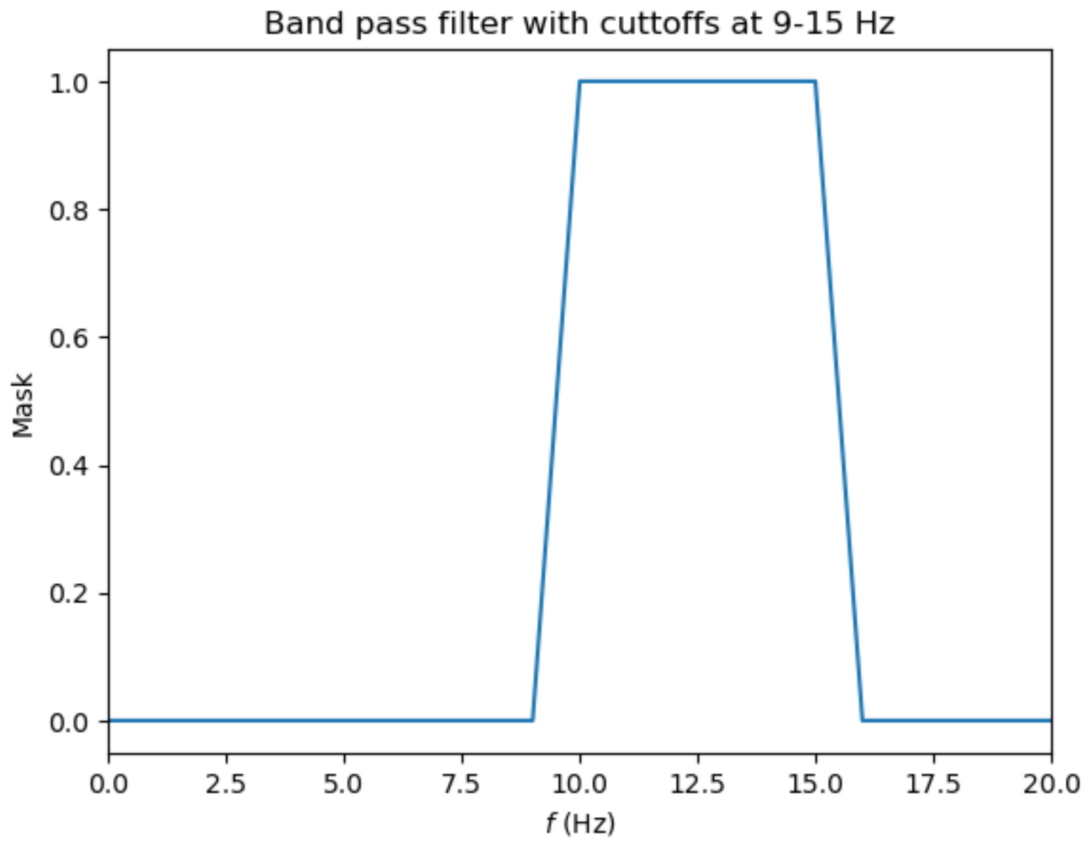


(d) Use a band pass filter to isolate the low S/N signals

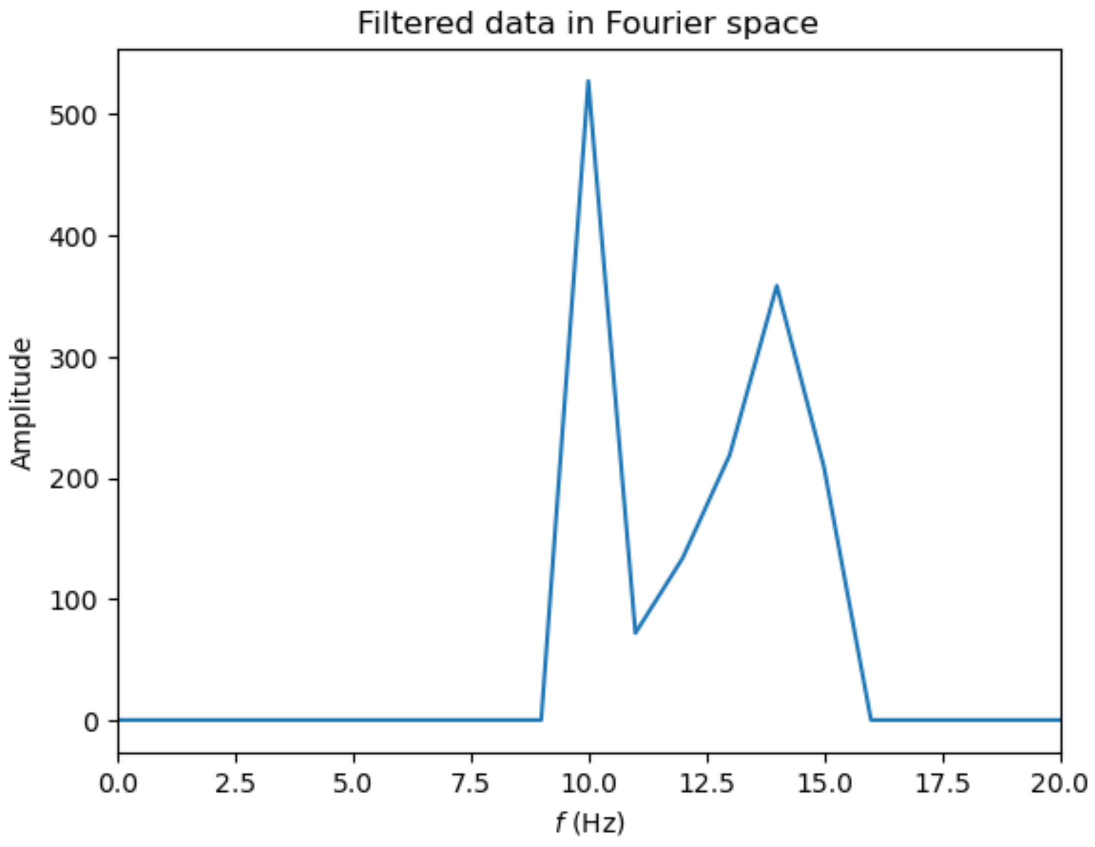
```
In [ ]: low_ft = np.fft.fft(low_sn)
        freq = np.fft.fftfreq(len(time), d=time[1]-time[0])
        plt.plot(freq[:int(n_samples/2)], np.abs(low_ft)[:int(n_samples/2)])
        plt.xlabel('$f$ (Hz)')
        plt.ylabel('Amplitude')
        plt.title('FFT of low S/N timestream')
        plt.show()
```



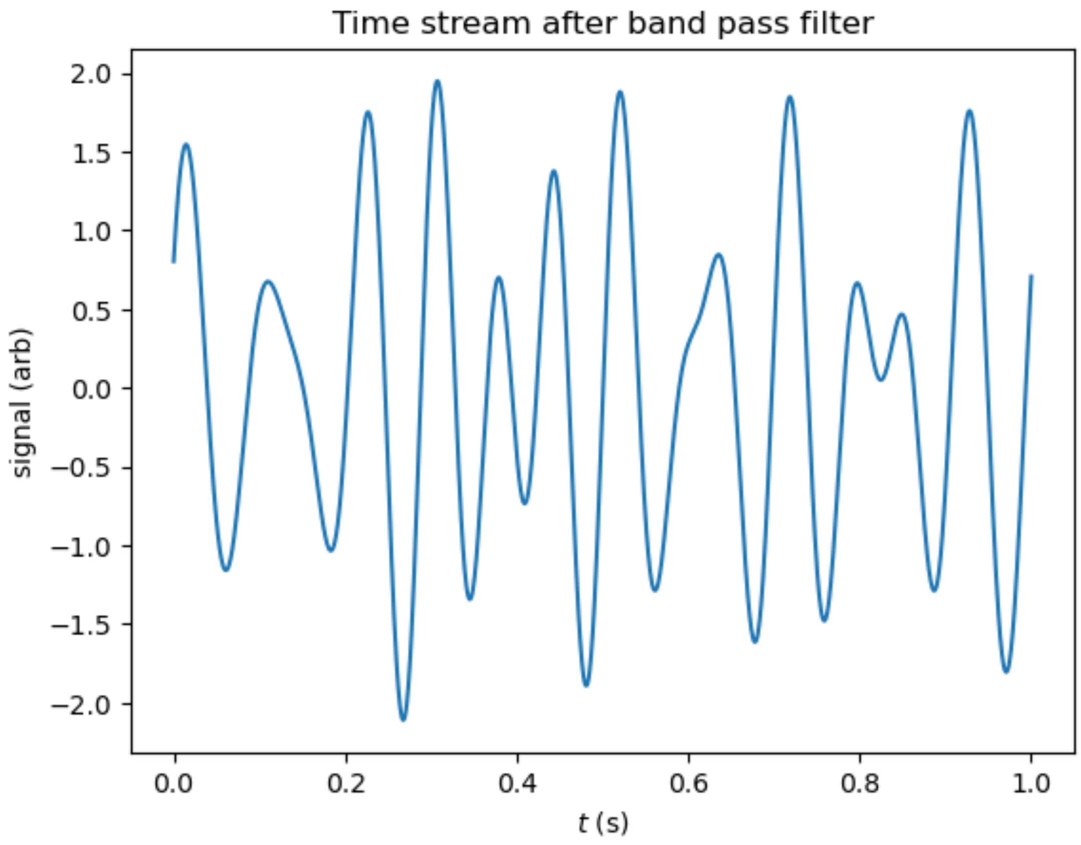
```
In [ ]: # I'll do a step with cuttoffs instead of a Gaussian since I
        # don't want the 0.07 period signal to disappear.
        band_cutoffs = [9, 15] #  Band pass filters cuttoffs in Hz
        bandpass_filter = np.where(np.abs(freq) < band_cutoffs[0], 0, 1)
        bandpass_filter = np.where(np.abs(freq) > band_cutoffs[1], 0, bandpass_filter)
        plt.plot(freq[:int(n_samples/2)], bandpass_filter [:int(n_samples/2)])
        plt.xlabel('$f$ (Hz)')
        plt.ylabel('Mask')
        plt.title(f'Band pass filter with cuttoffs at {band_cutoffs[0]}-{band_cutoffs[1]} Hz')
        plt.xlim(0,20)
        plt.show()
```



```
In [ ]: bp_filtered_fft = low_ft * bandpass_filter
        plt.plot(freq[:int(n_samples/2)], np.abs(bp_filtered_fft)[:int(n_samples/2)])
        plt.xlabel('$f$ (Hz)')
        plt.ylabel('Amplitude')
        plt.title('Filtered data in Fourier space')
        plt.xlim(0,20)
        plt.show()
```



In [ ]:
```
low_ft_bandpass = np.fft.ifft(bp_filtered_fft)
plt.plot(time, low_ft_bandpass.real)
plt.xlabel('$t$ (s)')
plt.ylabel('signal (arb)')
plt.title('Time stream after band pass filter')
plt.show()
```



> **Comment:** It is possible to detect the low S/N signal just because we correlated it with a high S/N signal to get an estimate of the leading frequency and filter it out using the band-pass filter.
>
> **Other comment:** I used a rectangular band pass filter to see if I could detect the 0.07 period signal without knowing it a priori. A narrower Gaussian filter would have eliminated that extra frequency.