Research Methods

Robert E Simpson

Singapore University of Technology & Design

robert_simpson@sutd.edu.sg

November 14, 2018

Objective

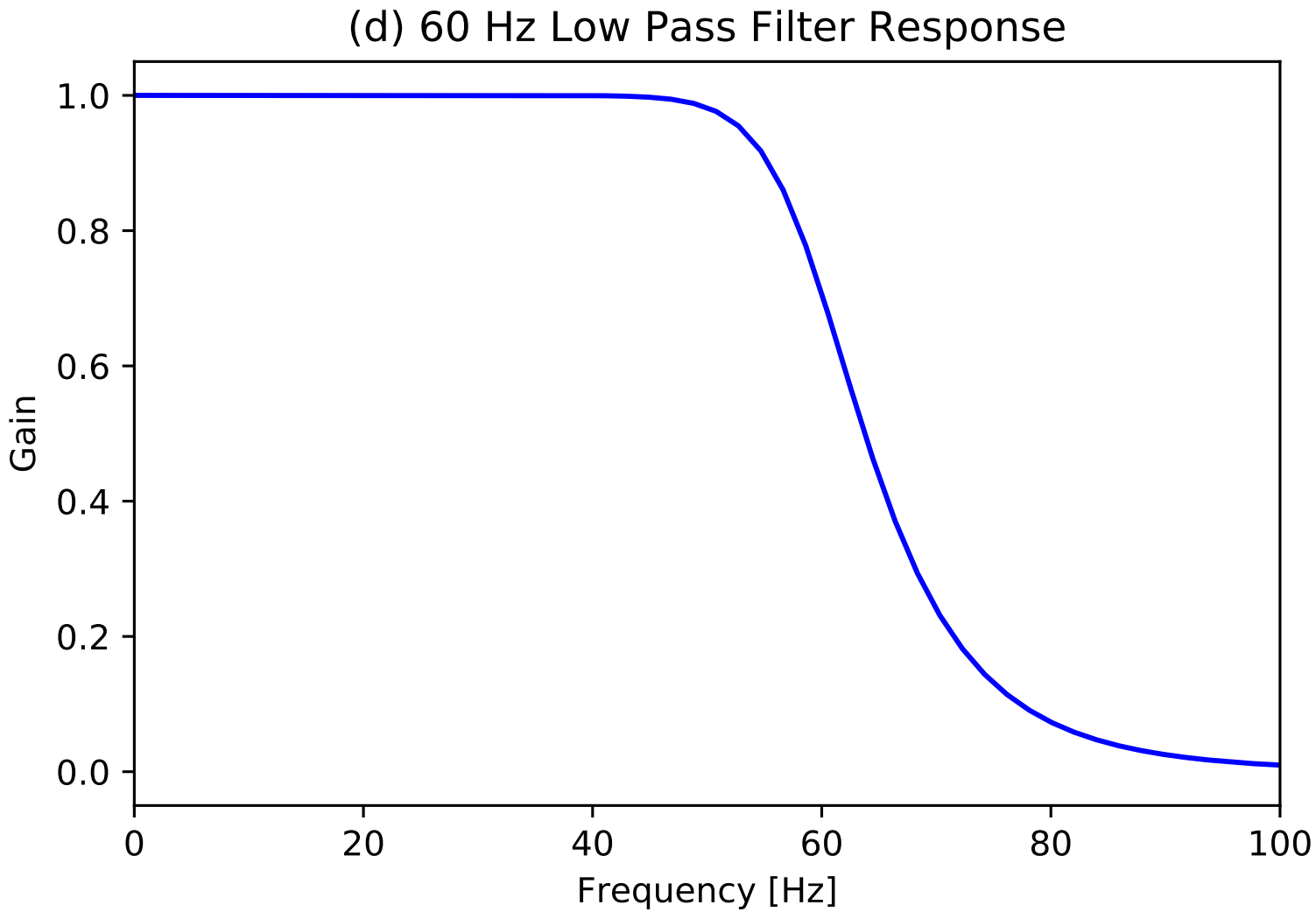
You should be able to

- Use convolutions to filter data.
- Use scipy.signals library to set-up high pass, low pass, and band pass filters
- Remove noise from discretised waveforms using Fourier filters
- Isolate and plot the amount of signal in specific frequency bands

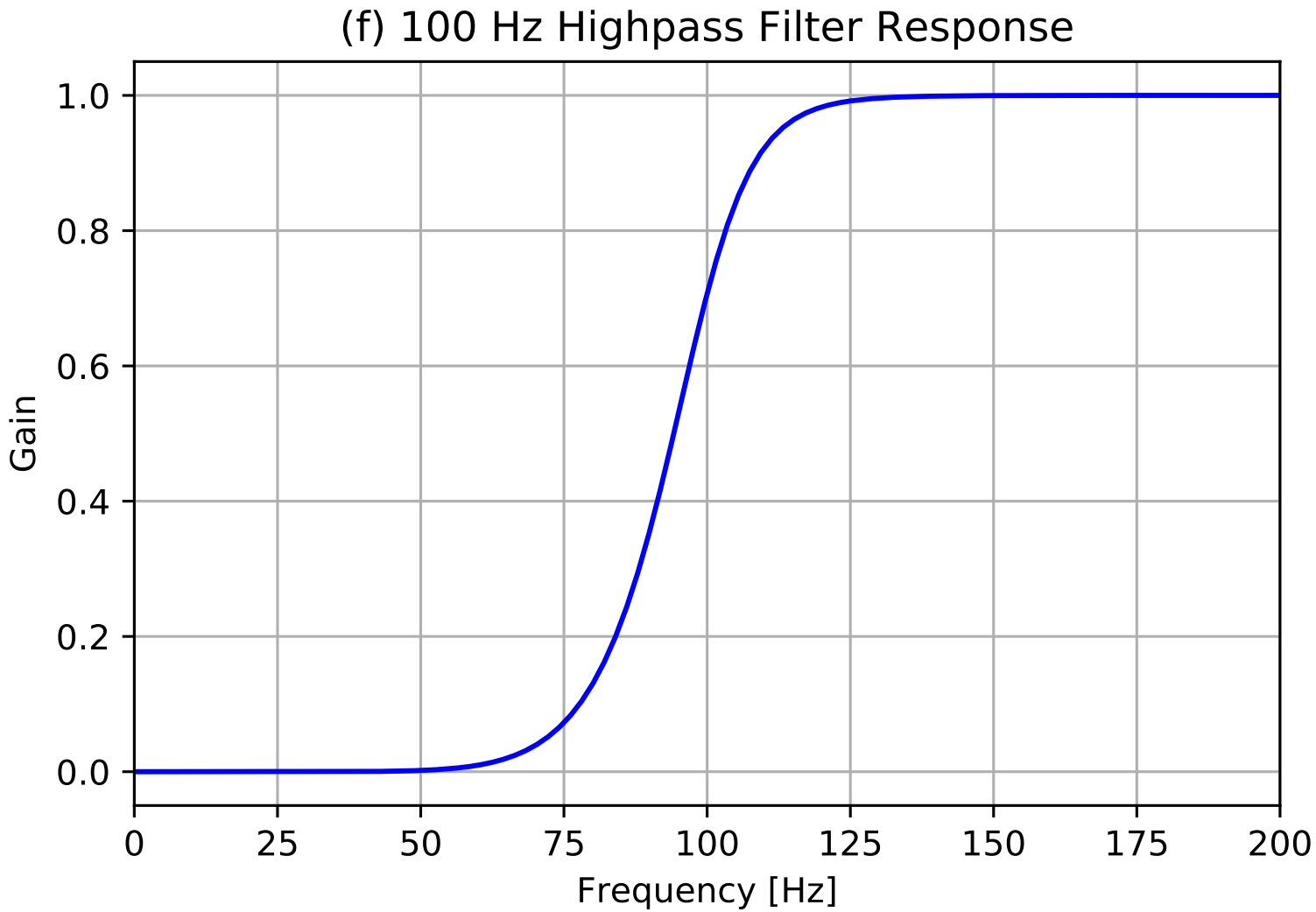
Filtering

Last week we learnt how to Fourier transform time series data to the frequency space, and then manually apply a crude 'sqaure' filter by deleting the unwanted frequencies from from the frequency space spectrum. After reverse Fourier transforming the data, the time series signal was reconstructed with the unwanted frequencies. This was a nice illustrative example of how Fourier filters work. However, the python signal library has an automated algorithm to do this and much more.

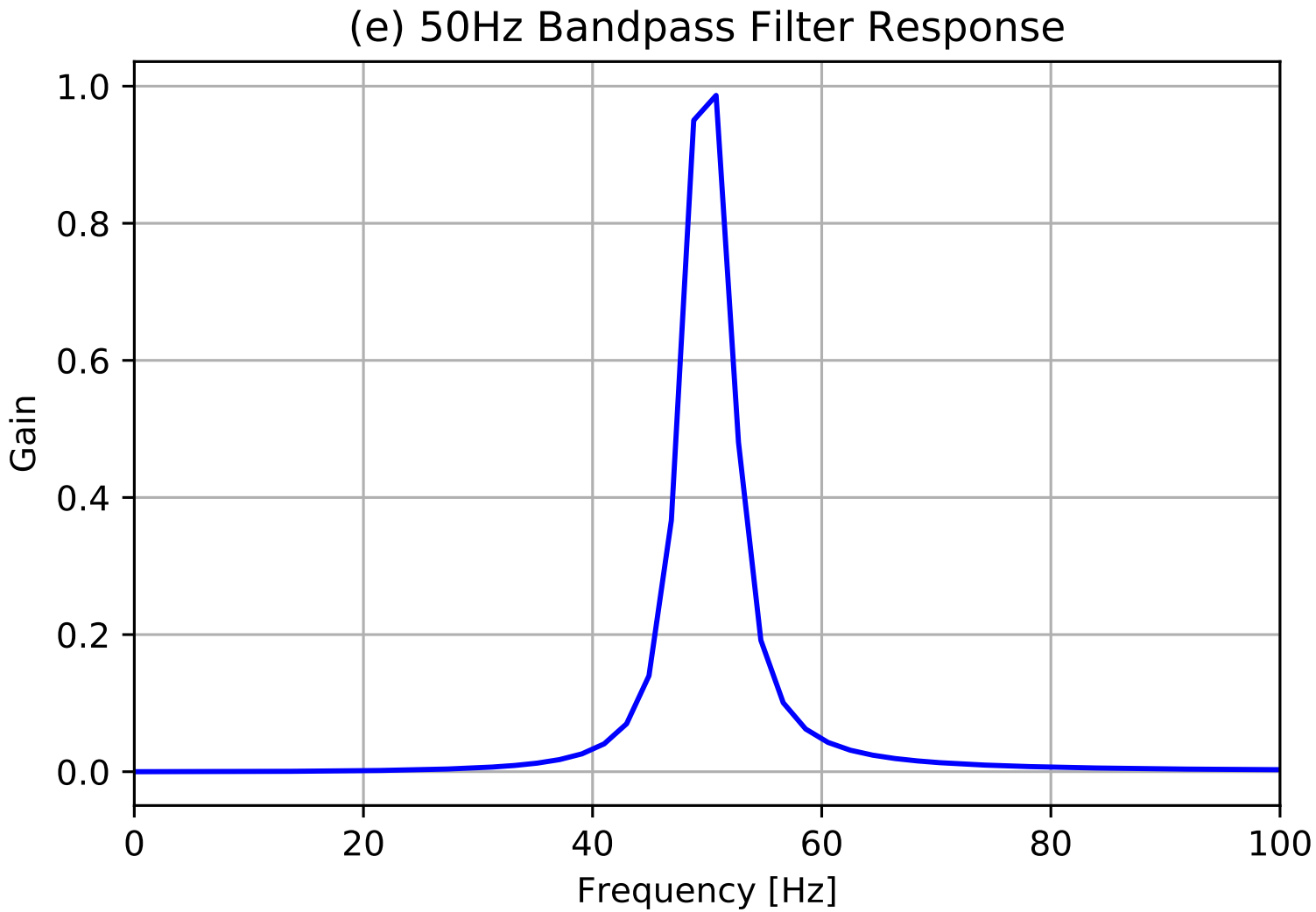
Low pass filter



High pass filter



Band pass filter



Filtering commands

w, h = signal.freqz(a) compute the frequency response of filter a. Return phase, a, and gain, h. Note– the filter is performed using angular frequencies (rad/s) and it uses the Nyquist frequency, which is half the sampling frequency.

20 * np.log10(abs(h)) convert the amplitude into dB

b, a = butter(order, normal_cutoff, btype='low', analog=False) Butterworth Low Pass Filter Window

lfilter(w,h,data) Filter the 'data' with the filter described by the plot b, a

Nyquist Frequency

Sampling theorem states that the sampling frequency must be at least twice the highest frequency in the signal in order to properly resolve the highest frequencies.

Thus, if we sample at $f_s = 1000$ Samples/sec, the highest frequency we can resolve is 500 Hz.

The Nyquist Frequency is the highest resolvable frequency for a given sampling rate, i.e. $f_s/2$.

scipy.signal.filtfilt

If you are more used to matlab, then the filtfilt function can also be used. It combines the forward and reverse fourier transforms into a single function. This function converts the signal into frequency space, applies a filter and then reverse Fourier filters the filtered signal.

`scipy.signal.filtfilt(b,a,signal)`

a & b are the filter window numerator and denominator coefficient vectors.

Filtering Process

```
import numpy as np
from scipy.signal import butter, lfilter, freqz
import matplotlib.pyplot as plt

###Define Low Pass Filter
def butter_lowpass(cutoff, fs, order=5):
    nyq = 0.5 * fs
    normal_cutoff = cutoff / nyq
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    return b, a

###Define Low Pass Filter
def butter_highpass(cutoff, fs, order=5):
    nyq = 0.5 * fs
    normal_cutoff = cutoff / nyq
    b, a = butter(order, normal_cutoff, btype='high', analog=False)
    return b, a

###Define Band Pass Filter
def butter_bandpass(start,stop, fs, order=5):
    nyq = 0.5 * fs
    startstop = [float(start) / nyq, float(stop)/nyq]
    b, a = butter(order, startstop, btype='band', analog=False)
    return b, a

# Plot the frequency response.
w, h = freqz(b, a)

#Filter the data
fdata=lfilter(b,a,data)
```

Note, that the `butter` filter uses the Nyquist frequency (`nyq`)

Convolutions

A convolution is an integral expressing the overlap between two functions as one function passes over the other.
GIF credit:
`http://mathworld.wolfram.com/Convolution.html`

Convolutions

$$f(t) * g(t) = \int_{-\infty}^{\infty} f(u)g(t-u)du \tag{1}$$

$$= \int_{-\infty}^{\infty} f(t-u)g(u)du \tag{2}$$

$$\tag{3}$$

For a finite series of data between 0 and u:

$$f(t) * g(t) = \int_{0}^{t} f(t-u)g(u)du \tag{4}$$

$$\tag{5}$$

Smoothing using convolutions

- Define a window function (kernal). A thin window will be sensitive to high frequencies. A wide window will be sensitive to low frequencies.
- Use `filt=np.convolve(y,kernal)` to pass the kernal across the function y.
- Divide the convoluted signal by the area of the kernal

Example smoothing using convolutions

Import data to be smoothed. Here we will smooth the daily temperature records since 2000.

```
import csv
import numpy as np
import matplotlib.pyplot as plt
nohead=[]
avgR=[]
avgT=[]
maxT=[]
minT=[]
avgW=[]
maxW=[]
with open('/Users/robert_simpson/Downloads/DailyWeather.csv', 'rb') as csvfile:
    myfile = csv.reader(csvfile, delimiter=',', quotechar='|')
    for row in myfile:
        nohead.append(row)
    del nohead[0]
    for row in nohead:
        avgR.append(float(row[4])) #rain
        avgT.append(float(row[8]))
        maxT.append(float(row[9])) #temp
        minT.append(float(row[10]))
```

Example smoothing using convolutions

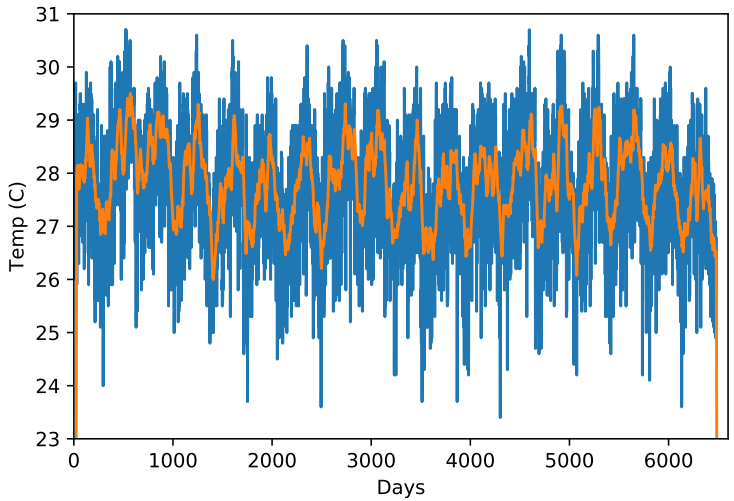
The kernal width is 30 with an amplitude of 1.

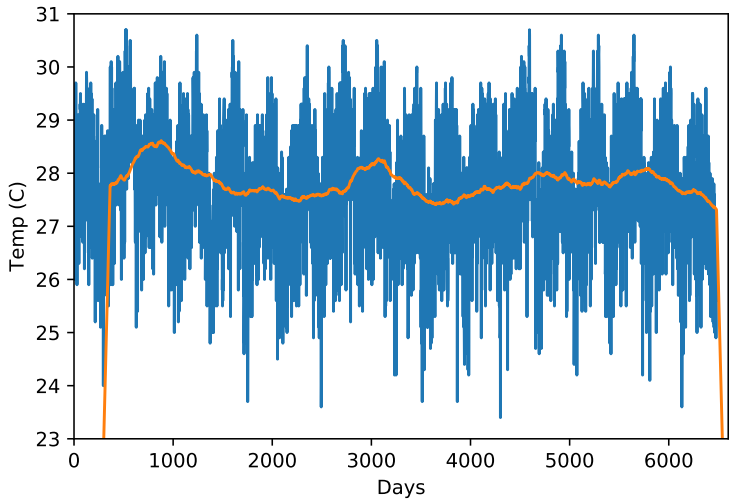
```
length=len(avgT)
windowlen=30

kernal=[0]*length
kernal[0:windowlen]=[1]*windowlen

filtT=np.convolve(avgT,kernal)/windowlen
plt.figure()
plt.plot(filtT)
plt.ylim(23,31)
plt.xlim(0,6000)
plt.savefig("AverageTemp.pdf")
```

Effect of kernel width





When the kernal width is 30 days [left], daily fluctuations are smoothed, when the kernel width is 365 days [right], yearly variations are smoothed.

Always consider the frequency that you want to resolve when choosing the kernel

Case Problem 10.2

(a) Create a 5 second long wave with a 10 Hz component of amplitude 10, a 50 Hz component of amplitude 1, and an offset of 1.

(b) Add Gaussian noise with $\sigma = 2$ to the wave

(c) Plot the frequency spectrum for the wave

(d) Plot the wave after removing the high frequency Gaussian noise

(e) Plot the wave after isolating the 50 Hz signal

(f) Plot the noise on the wave by using a high pass filter

CP 10.2 Solution

Define the lowpass, highpass, and bandpass filter functions

```
import numpy as np
from scipy.signal import butter, lfilter, freqz
import matplotlib.pyplot as plt

def butter_lowpass(cutoff, fs, order=5):
    nyq = 0.5 * fs
    normal_cutoff = cutoff / nyq
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    return b, a

def butter_highpass(cutoff, fs, order=5):
    nyq = 0.5 * fs
    normal_cutoff = cutoff / nyq
    b, a = butter(order, normal_cutoff, btype='high', analog=False)
    return b, a

def butter_bandpass(start,stop, fs, order=5):
    nyq = 0.5 * fs
    startstop = [float(start) / nyq, float(stop)/nyq]
    b, a = butter(order, startstop, btype='band', analog=False)
    return b, a
```

CP 10.2 Solution

Set the sample rate, signal duration, number of times the signal is sampled, and the list of time points when the signal was sampled.

```
fs =2000.0          # sample rate, Hz
T = 5.0            # duration of signal in seconds
n = int(T * fs) # total number of samples in signal
t = np.linspace(0, T, n, endpoint=False) #time axis
```

CP10.2(a) Create a 5 second long wave with a 10 Hz component of amplitude 10, a 50 Hz component of amplitude 1, and an offset of 1.

```
##### CP 10.2(a)
data = 10*np.sin(10*2*np.pi*t) + 5*np.sin(50*2*np.pi*t) + 1
```

CP10.2(b) Add Gaussian noise with $\sigma = 2$ to the wave

```
noisy=[]
for pnt in data:
    noisy.append(pnt+np.random.normal(0,2))
plt.figure()
plt.xlabel('time␣(s)')
plt.ylabel('Amp␣(A.U)')
plt.plot(t, noisy)
plt.title('Raw␣Signal')
plt.xlim(0,1)
```

CP 10.2(b) Solution

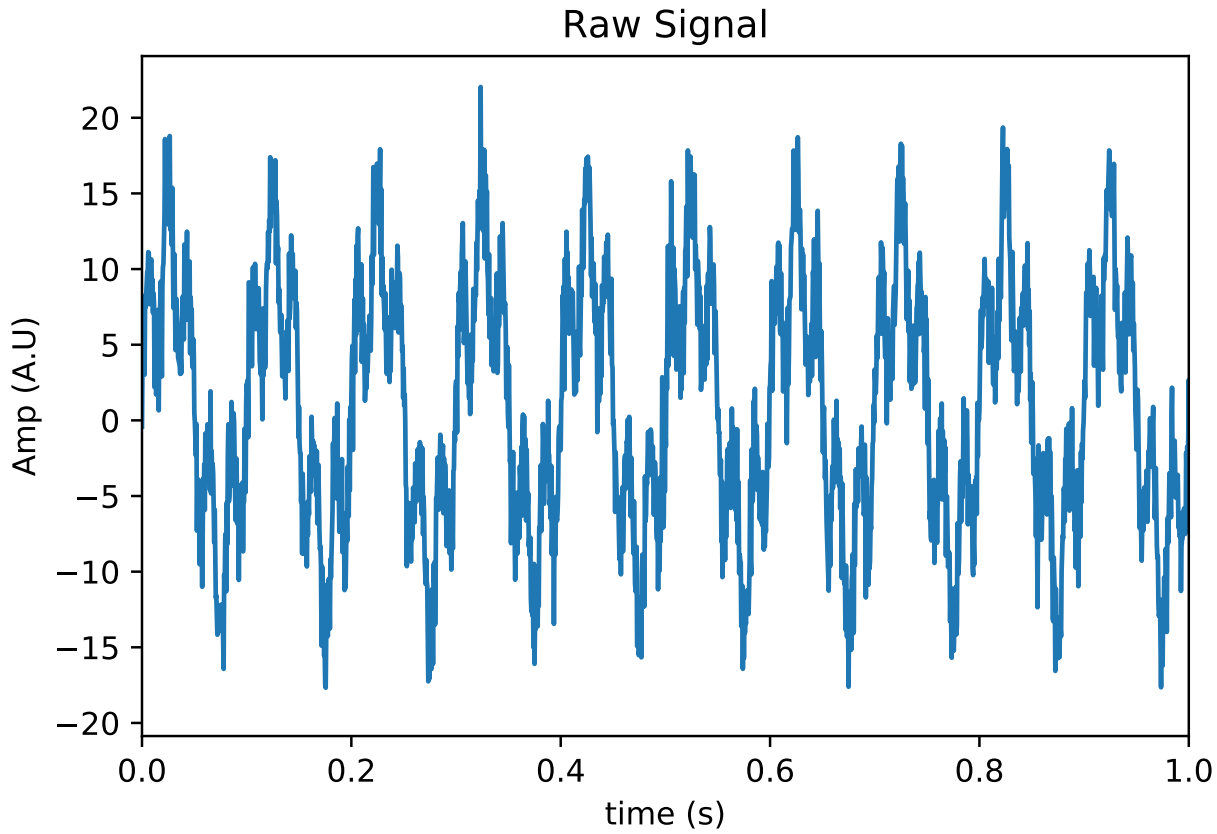


Figure: Raw signal with 10 Hz and 50 Hz sinusoidal components at Gaussian noise with $\sigma = 2$

CP 10.2(c) Solution

We need to take theFourier Transform to see the frequencies that are present in the signal

```
freqspec=np.fft.fft(noisy)
#sampling frequency is n Hz
Hz=np.linspace(0,fs,n)

plt.figure()
plt.title('(c) Spectrum of Siganal')
plt.plot(Hz, np.abs(freqspec))
plt.xlim(0,100)
plt.xlabel('Frequency [Hz]')
plt.ylabel('Magnitude(Fourier Component)')
```

CP 10.2(c) Solution

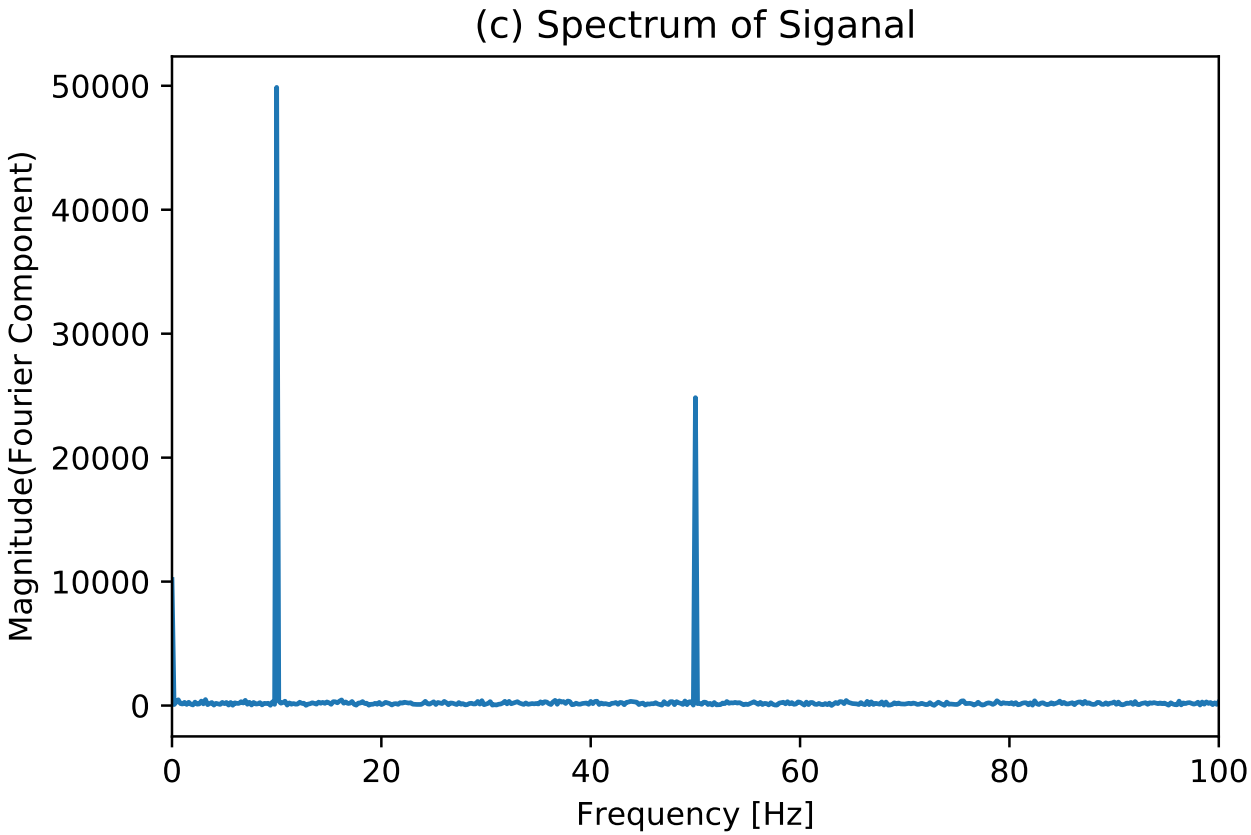


Figure: Fourier Component Magnitude Spectrum

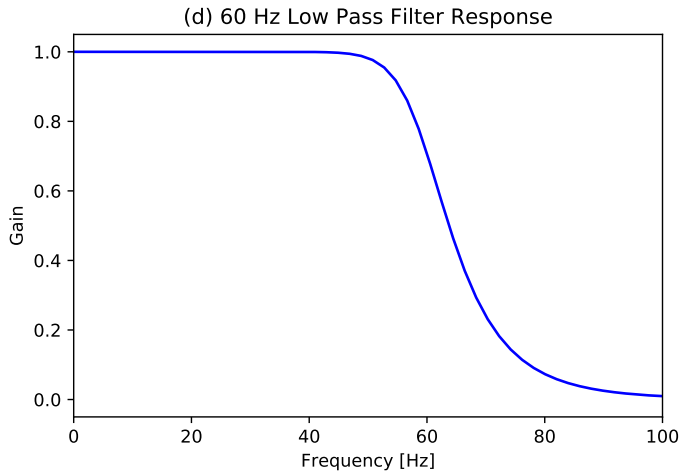
CP 10.2(d) Solution

Use a Low Pass filter at 60 Hz. Using a higher order (9) filter produces a sharper transition.

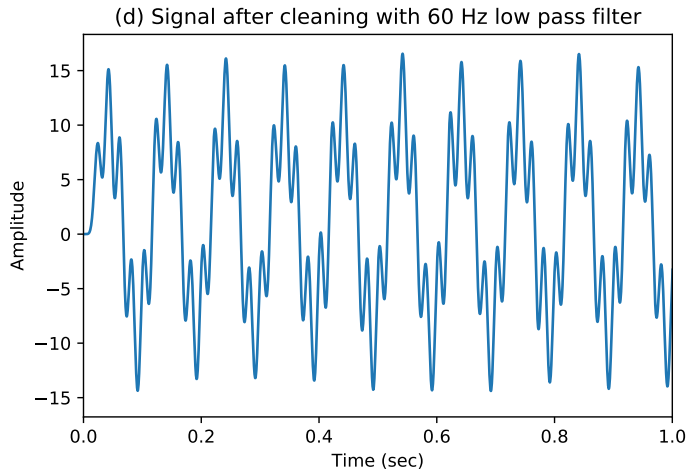
```
order=9 #order of the filter
b, a = butter_lowpass(60, fs, order)
w, h = freqz(b, a)
plt.figure()
plt.title('(d) 60 Hz Low Pass Filter Response')
plt.plot(fs*w/(2*np.pi), np.abs(h), 'b')
plt.xlim(0,100)
plt.xlabel('Frequency [Hz]')
plt.ylabel('Gain')
plt.savefig('10_2d_response.pdf')

nonoise=lfilter(b,a,noisy)
plt.figure()
plt.title('(d) Signal after cleaning with 60 Hz low pass filter')
plt.plot(t, nonoise)
plt.xlim(0,1)
plt.xlabel('Time (sec)')
plt.ylabel('Amplitude')
```

CP 10.2(d) Solution



(a) Low Pass Response



(b) Filtered Signal

Figure: Filtering signal with a 60 Hz low pass filter

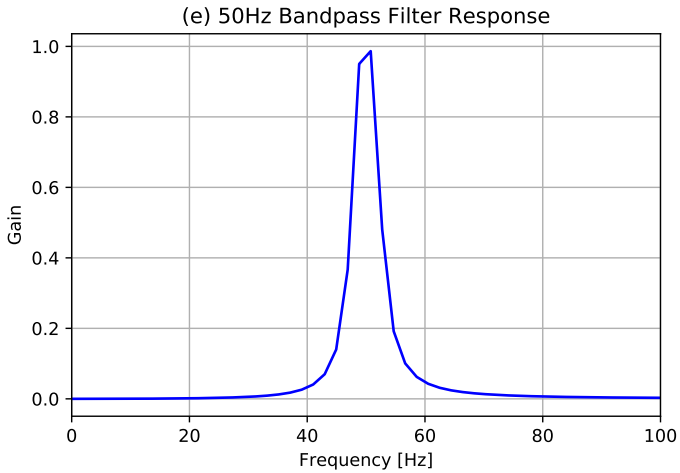
CP 10.2(e) Solution

Use a bandpass filter that starts at 48 Hz and ends at 52 Hz. A lower order (2) filter produces a best results.

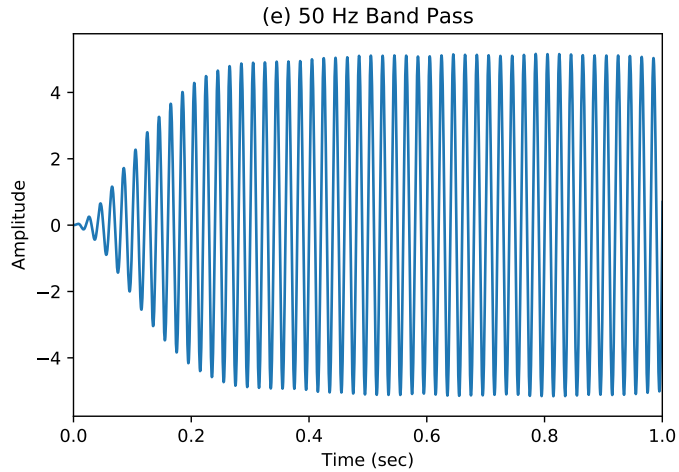
```
order = 2  #the order of the filter--- related to the sharpness of the filter
     window
b, a = butter_bandpass(48,52, fs, order) #create a butterworth bandpass filter
w, h = freqz(b, a) #calculate the window in frequency space
plt.figure()  #plot the bandpass window
plt.plot(fs*w/(2*np.pi), np.abs(h), 'b-')
plt.xlim(0, 100)
plt.title("(e) 50Hz Bandpass Filter Response")
plt.xlabel('Frequency [Hz]')
plt.ylabel('Gain')
plt.grid()

# Plot the signal after filtering
w, h = freqz(b, a)
fdata=lfilter(b,a,noisy)
plt.figure()
plt.title('(e) 50 Hz Band Pass')
plt.plot(t, fdata)
plt.xlim(0.,1)
plt.xlabel('Time (sec)')
plt.ylabel('Amplitude')
```

CP 10.2(e) Solution



(a) Band Pass Response



(b) Filtered Signal

Figure: Filtering signal with a 50 Hz band pass filter

Notice that the filtered signal builds up relatively slowly. This is due to original signal starting abruptly and we are then effectively convoluting it with a narrow and abrupt bandpass filter (This is similar to a square shaped kernel convoluting an abrupt signal).

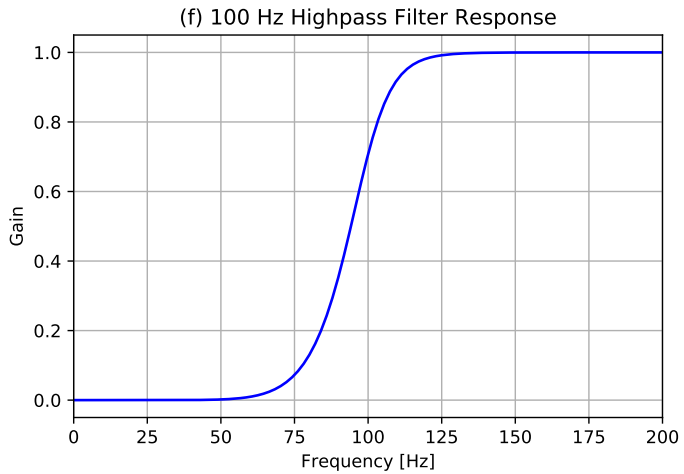
CP 10.2(f) Solution

Used a 100 Hz High Pass Filter with order=9

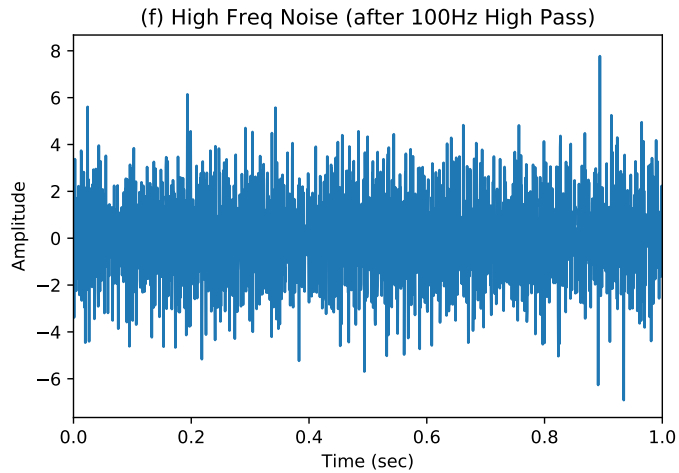
```
order=9
b, a = butter_highpass(100, fs, order)
w, h = freqz(b, a) #calculate the window in frequency space
plt.figure()  #plot the bandpass window
plt.plot(0.5*fs*w/np.pi, np.abs(h), 'b-')
#plt.plot(cutoff, 0.5*np.sqrt(2), 'ko')
#plt.axvline(cutoff, color='k')
plt.xlim(0, 200)
plt.title("(f) 100 Hz Highpass Filter Response")
plt.xlabel('Frequency [Hz]')
plt.ylabel('Gain')
plt.grid()

fdata=lfilter(b,a,noisy)
plt.figure()
plt.title('(f) High Freq Noise (after 100Hz High Pass)' )
plt.plot(t, fdata)
plt.xlabel('Time (sec)')
plt.ylabel('Amplitude')
plt.xlim(0.,1)
```

CP 10.2(f) Solution



(a) Band Pass Response



(b) Filtered Signal

Figure: Filtering signal with a 100 Hz high pass filter

We can see that most of the noise has an amplitude less than 2, which makes sense because the Gaussian Noise was set with a standard deviation $\sigma = 2$. Also notice that the offset of 1, and the 50 Hz and 10 Hz components have been removed because they are at a lower frequency than the 100 Hz cut-off.

Summary

We have learnt how to filter data in python using:

- Convolutions
- Filters within the scipy.signals library