Fourier Transform HW

November 26, 2021

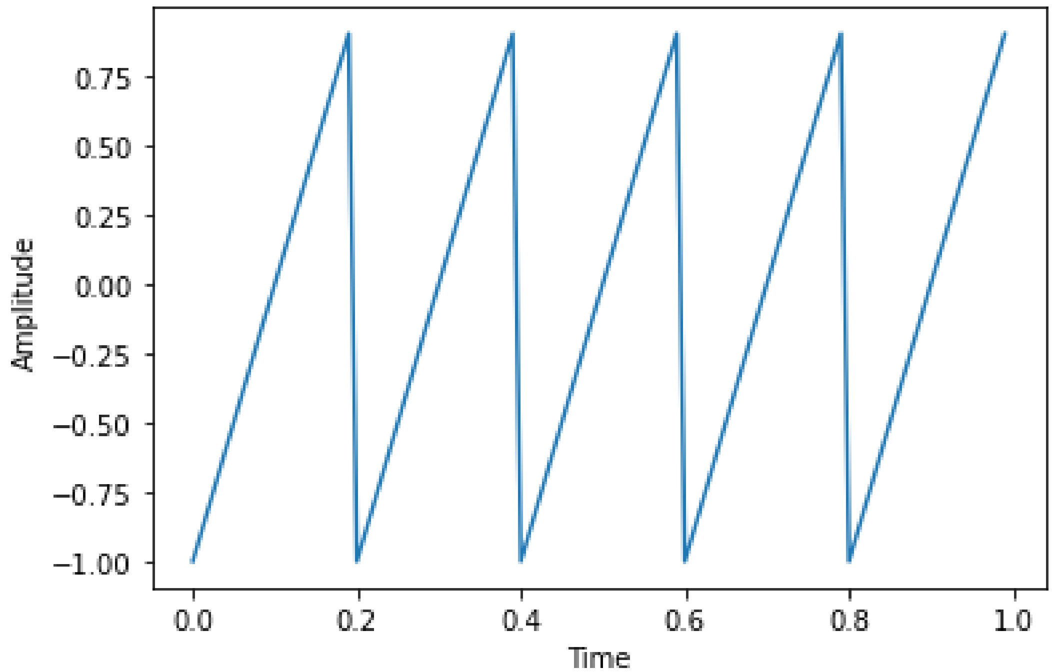
#Fourier Transform HW#

Name: Hila Man

ID: 315212092

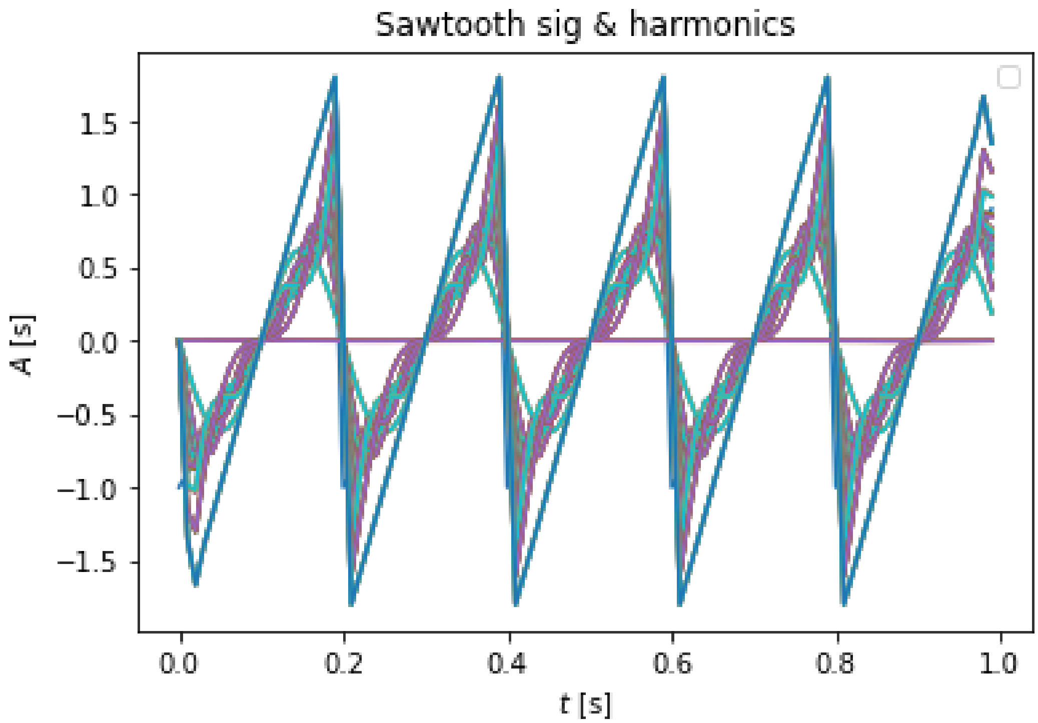
#Question 1#

[3]:
```
import matplotlib.pyplot as plt
from scipy import signal
import numpy as np
from scipy.integrate import simps

Fs = 100
amplitude = 1
duration = 1  # in seconds
N = Fs * duration
t = np.linspace(0, duration, num=N, endpoint=False)

# create the signal
frequency = 5
sig = amplitude * signal.sawtooth(2 * np.pi * frequency * t)
plt.plot(t, sig)
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.show()
```



[5]:
```python
def sawtooth_harmonics(number_of_harmonics):
    n = number_of_harmonics
    fig, ax = plt.subplots()
    ax.plot(t, sig)
    sum = 0

    # it's an odd function, fft has only sin
    def b(n):
        return (2.0 / duration) * simps(sig * np.sin(2.0 * np.pi * n * t /
 ↪duration), t)

    for i in range(1, n + 1):
        sum += b(i) * (np.sin(2.0 * np.pi * i * t / duration))
        ax.plot(t, sum)
        ax.set_xlabel('$t$ [s]')
        ax.set_ylabel('$A$ [s]')
    ax.legend()
    ax.set_title("Sawtooth sig & harmonics")
    plt.show()

sawtooth_harmonics(100)
```

```
No handles with labels found to put in legend.
```



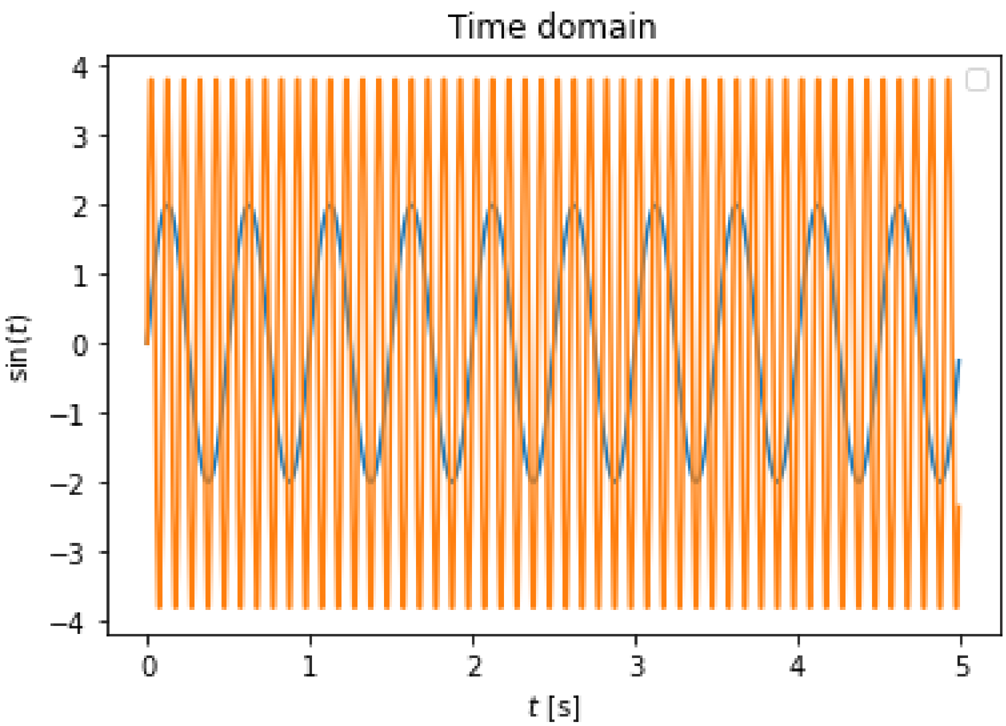
After a few trials, we can see that we need around 100 harmonics to represent this wave well.

#Question 2#

[12]:
```
fig, ax = plt.subplots()
Fs = 100
d = 5  # in seconds
N = Fs * d
t = np.linspace(0, d, num=N, endpoint=False)


f1 = 2
amp1 = 2
sig1 = amp1*np.sin(2 * np.pi * f1 * t)

f2 = 10
amp2 = 4
sig2 = amp2*np.sin(2 * np.pi * f2 * t)
ax.plot(t, sig1)
ax.plot(t, sig2)

ax.legend()
plt.xlabel('$t$ [s]')
plt.ylabel('$\sin (t)$')
```

```
ax.set_title("Time domain")
plt.show()
```

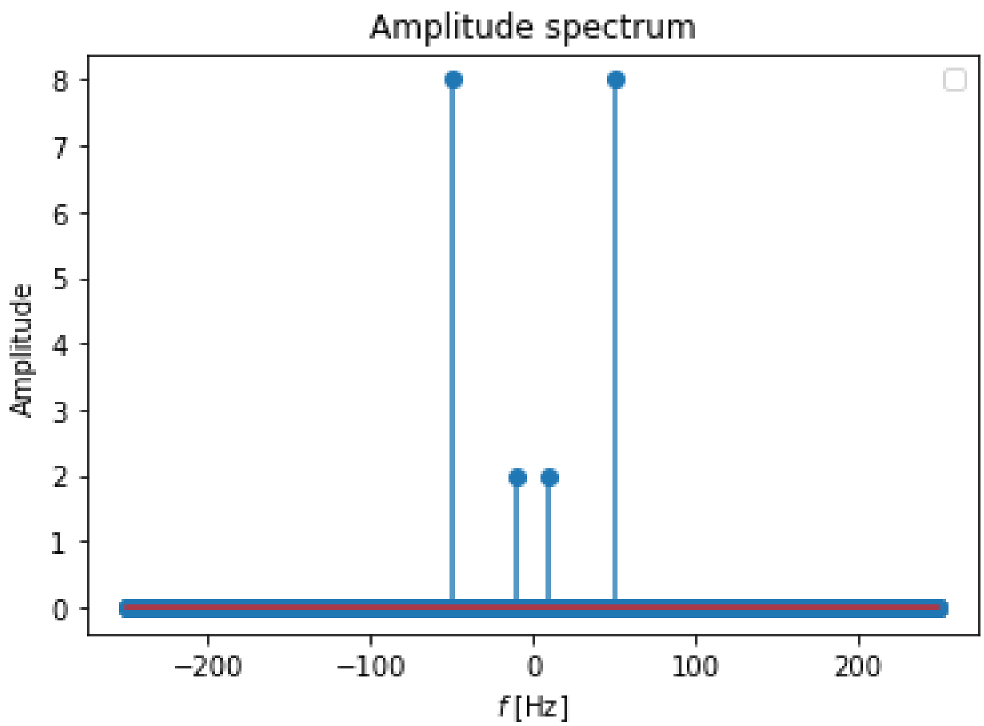
```
No handles with labels found to put in legend.
```




```
[13]: fig2, ax2 = plt.subplots()
fft1 = np.fft.fftshift(np.fft.fft(sig1))
fft2 = np.fft.fftshift(np.fft.fft(sig2))

power1 = np.square(np.abs(fft1)) / N
power2 = np.square(np.abs(fft2)) / N

amp_spec1 = (2 / N) * np.abs(power1)
amp_spec2 = (2 / N) * np.abs(power2)

ax2.stem(np.arange(-N/2, N/2), amp_spec1, use_line_collection=True)
ax2.stem(np.arange(-N/2, N/2),amp_spec2, use_line_collection=True)



ax2.legend()
ax2.set_title("Amplitude spectrum")
ax2.set_xlabel('$f$ [Hz]')
```

```
ax2.set_ylabel('Amplitude')
plt.show()
```

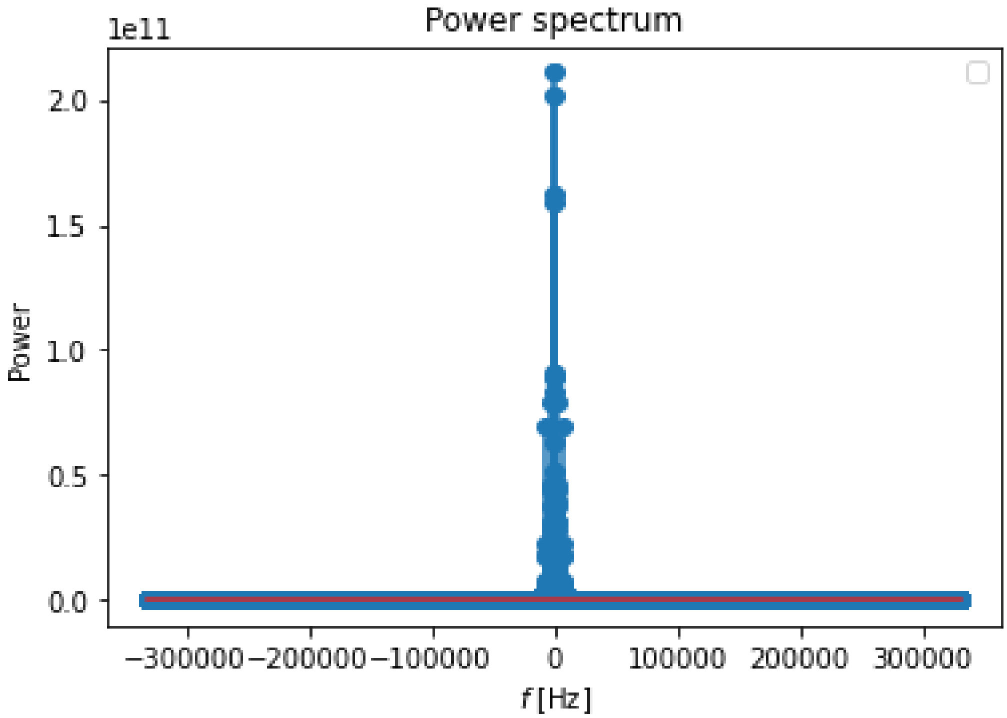
```
No handles with labels found to put in legend.
```



#Question 3#

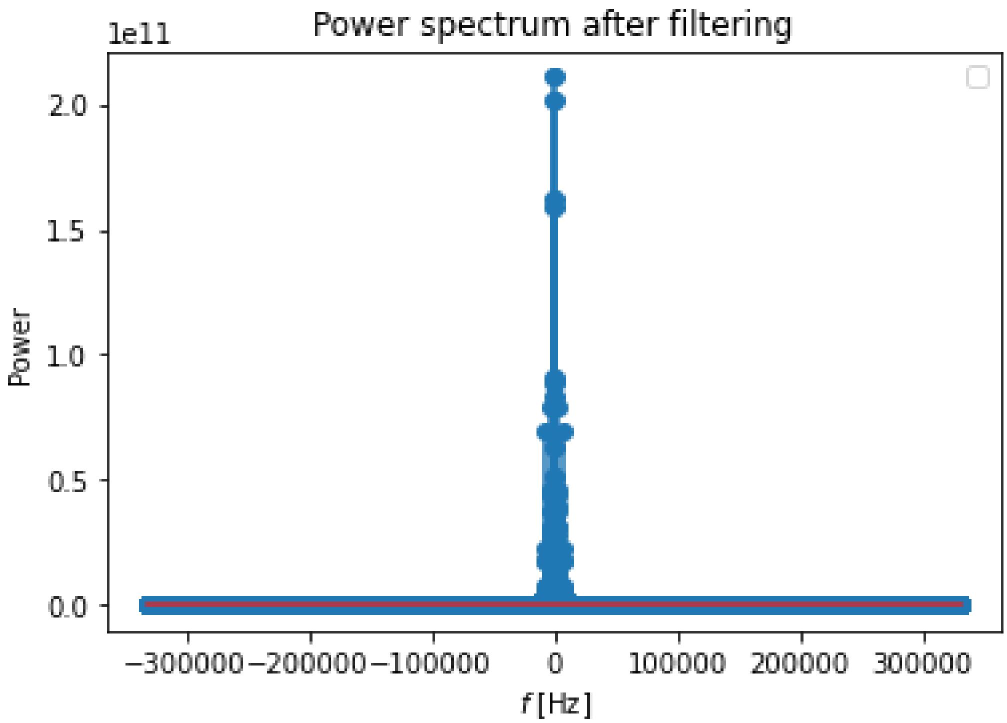
[24]:
```
from scipy.io import wavfile
Fs, sig = wavfile.read('guitartune.wav')
d = 15  # in seconds
N = Fs * d
t = np.linspace(0, d, num=N, endpoint=False)

fig, ax = plt.subplots()
fft = np.fft.fftshift(np.fft.fft(sig))
power_spectrum = np.square(np.abs(fft))/N
ax.stem(np.arange(-N/2,N/2), power_spectrum, use_line_collection=True)

ax.legend()
ax.set_title("Power spectrum")
ax.set_xlabel('$f$ [Hz]')
ax.set_ylabel('Power')
plt.show()
```

No handles with labels found to put in legend.



[25]:
```
fig2, ax2 = plt.subplots()
fft2 = fft.copy()
ten_present = int(np.shape(fft2)[0]*0.1)
fft2[(-1)*ten_present:] = 0
fft2[:ten_present] = 0
power_spectrum2 = np.square(np.abs(fft2))/N
ax2.stem(np.arange(-N/2,N/2), power_spectrum2, use_line_collection=True)

ax2.legend()
ax2.set_title("Power spectrum after filtering")
ax2.set_xlabel('$f$ [Hz]')
ax2.set_ylabel('Power')
plt.show()
```

No handles with labels found to put in legend.



We can see that the Power spectrum hasn't change. Therefore, we didn't need to erase these frequencies. Now, let's apply the inversed Fourier Transform on our file:

[26]:
```
ifft2 = np.fft.ifft(np.fft.ifft(sig))

wavfile.write('ifft guitar.wav', Fs, (ifft2*2**16).astype(np.int16))
```

```
C:\Users\hillu\AppData\Local\Temp/ipykernel_14024/1480286687.py:3:
ComplexWarning: Casting complex values to real discards the imaginary part
  wavfile.write('ifft guitar.wav', Fs, (ifft2*2**16).astype(np.int16))
```

Let's apply some filters: #Butterworth filter#

[27]:
```
b, a = signal.butter(N=5, Wn=200, fs=Fs)
output = signal.filtfilt(b, a, sig)
wavfile.write('Butterworth.wav', Fs, (output*2**16).astype(np.int16))
```

#Bessel filter#

[28]:
```
b2, a2 = signal.bessel(N=5, Wn=0.5, analog=True)
output = signal.filtfilt(b2, a2, sig)
wavfile.write('Bessel.wav', Fs, (output*2**16).astype(np.int16))
```

#Flattop filter#

[29]:
```
window = signal.flattop(20)
b3 = np.ones(window.__len__()) * (1/window.__len__())
output = signal.filtfilt(b3, a2, sig)
wavfile.write('Flattop.wav', Fs, (output*2**16).astype(np.int16))
```

In my opinion, the originals signal sounds the best. But from the results from the filters we applyed, 'Bessel.wav' has less noise, and sounds better.