Computational Physics Assignment 6

Dara Corr - 18483836 - 22/02/21

This Assignment introduces us to signal analysis with a particular emphasis on Fourier Transforms. Fourier Transforms allow us to transform a signal from a time domain ($y(t)$) to a frequency domain ($Y(f)$) so we can see the different frequencies that are present in the signal. In this assignment we will be investigating how to sample a signal, take its fourier transform, plot its power spectrum, identify different components of the signal and how to remove noise from a signal.

The Fourier Transform and its inverse are defined as follows:

$$Y(f) = \int_{-\infty}^{\infty} y(t) \exp^{2\pi i f t}\, dt$$

$$y(t) = \int_{-\infty}^{\infty} Y(f) \exp^{-2\pi i f t}\, df$$

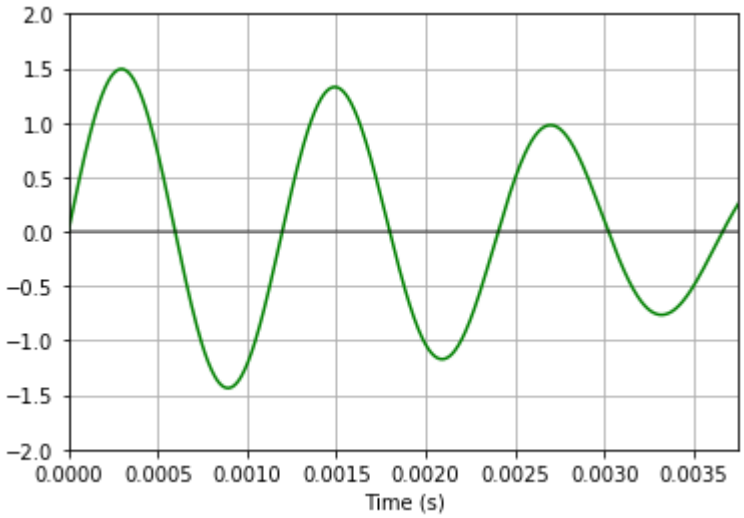
The first step in this assignment is to simulate a signal and then sample the signal. We begin with a simple sin wave $y(t) = sin(2\pi f t)$ with a frequency of 800 Hz. Then we sample this frequency at 5000 Hz with the initial number of samples being 32. Then we made a simple plot of the sampled signal.

The next step is where Fourier Transforms start to be used. Importing $numpy.fft$ allows us to use the function $fft()$ to quickly calcluate Fourier Transforms of Signals. The signal is transformed and then the upper half of the power spectrum is shifted down to give us our Power Spectrum plot. Each Frequency present in the Signal appears as a peak on this plot, so every frequency in the signal can be easily identified, analysed and modified and then transformed back to the time domain.

In this lab we analyse how the number of samples impacts the spectral resolution of the signal. We notice how a larger sample size makes the peaks in the Power Spectrum taller and thinner. The Nyquist rate is also investigated and we show how a sample rate below it and how it causes an aliasing effect where the reconstructed signal loses information from the original signal.

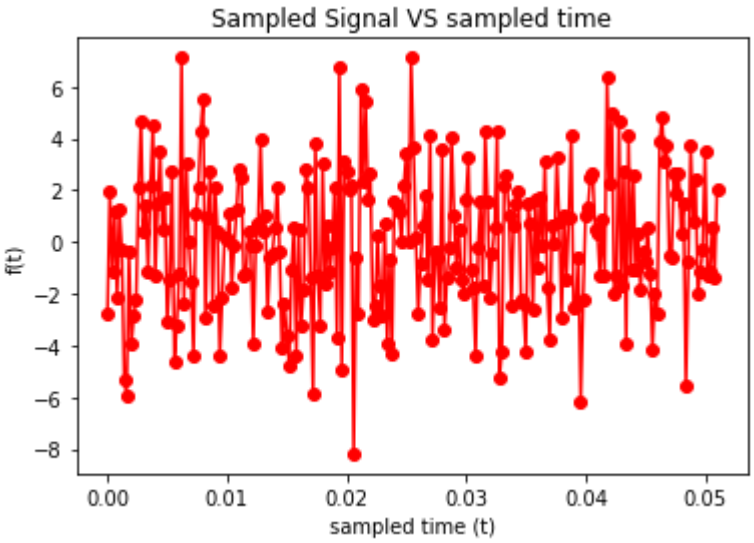
We look at how noise affects a signal and how it can be reduced usinig a noise filter. Finally we look at a mixture of sine waves as our input signal and we investigate the effect on the number of samples and the presence of noise in the signal on the ability to distinguish the frequencies from eachother on the Power Spectrum.

In [362]:
```
import numpy as np
import matplotlib.pyplot as plt

#task 1

f1 = 800 #Define a frequency for a sine wave; this is the SIGNAL.
T1 = 1/f1 #Tsig is the period of the SIGNAL

f2 = 910 #frequence of second signal
T1 = 1/f2 #period of second signal


t = np.linspace(0,Tsig*5,1000) #Set up array with 1000 t values over 5 wave cy
cles
ysig= np.sin(2*np.pi*f1*t) + 0.5*np.sin(2*np.pi*f2*t) #Calculate the SIGNAL va
lues at each time value


plt.plot(t,ysig, '-g') # Plot the SIGNAL against time using a blue line.
plt.plot(t,np.zeros(1000), 'k-', alpha = 0.5)
plt.ylim([-2, 2]) # Define the domain and range of the axes
plt.xlim([0, Tsig*3])
plt.xlabel('Time (s)')
plt.ylabel('')
plt.grid()
```



In [357]:
```
#task 2
fsamp = 5000 #Define a frequency to SAMPLE the signal at.
Tsamp = 1/fsamp #Calculate the time step between SAMPLES.
Nsamp = 256 #Define how many SAMPLE values to make (power of 2).
#Create array 'tsamp' that contains all the SAMPLE times


#create array 'tsamp' which contains all the sample times

tsamp = []


for i in range(0,Nsamp):
     tsamp.append(i*Tsamp)


#Create array 'ysamp' that contains all SIGNAL values at the SAMPLE times
ysamp = []
for i in range(0,Nsamp):
    ysamp.append((np.sin(2*np.pi*f1*tsamp[i]) + 0.5*np.sin(2*np.pi*f2*tsamp[i
])) + 0.01*np.random.normal(1,Nsamp))    #Calculate the SIGNAL value at the SA
MPLE times

plt.title("Sampled Signal VS sampled time")
plt.xlabel("sampled time (t)")
plt.ylabel("f(t)")
plt.plot(tsamp,ysamp,'or') #Plot the SAMPLE values using red circles
plt.plot(tsamp,ysamp,'r-') #and red lines

#calculate how many points you expect per wave cycle (6 or 7?)
#do calc and answer in markdown
```

Out[357]: [<matplotlib.lines.Line2D at 0xae27f0630>]

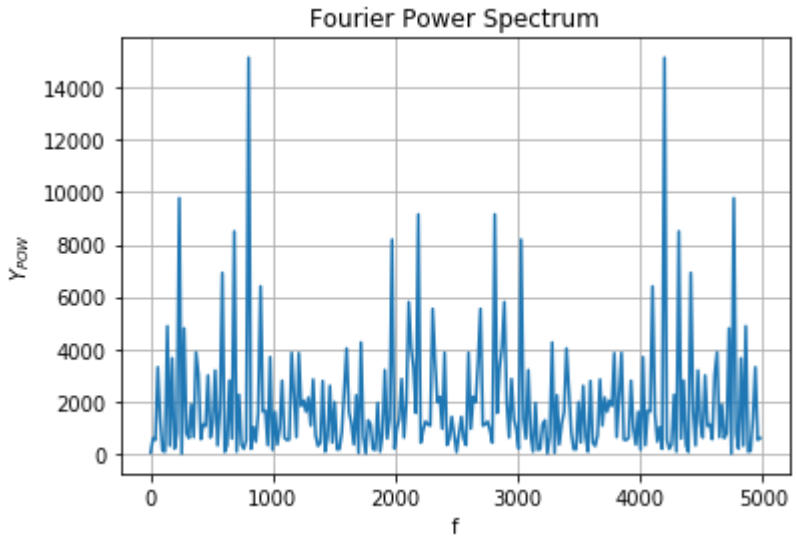


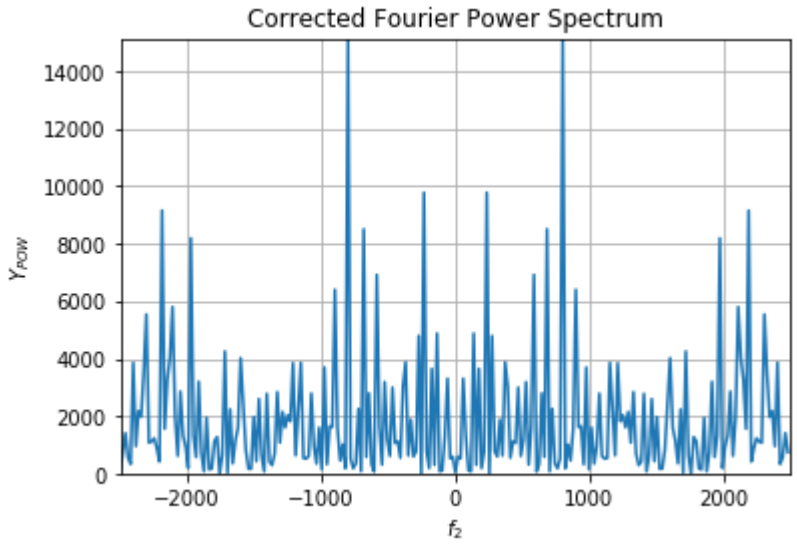
Task 2 question: about 6 or 7 points per wave cycle would be expected when Nsamp = 32

In [358]:
```
#task 3
from numpy.fft import fft #import fourier tranfsform function

Y_FT = fft(ysamp) # Calculate the Fourier Transform of the SAMPLED signal

if Y_FT.imag.any() != 0: # we see the fourier transform is complex as it conta
ins j components
    print("The Fourier Transform of the sampled signal is complex")

if len(Y_FT) == Nsamp: #showing Fourier Transform has Nsamp elements
    print("The Fourier Transform contains Nsamp elements")


#Convert the transfrom into a power spectrum by multiplying FT by its complex
 conjugate
Y_POW = Y_FT*np.conj(Y_FT)

if Y_POW.all() == np.conj(Y_POW).all(): #show the power spectrum is purely rea
l by checking if complex conjugation changes its values
    print("The Power Spectrum is real")


Y_POW = Y_POW.real #removes the 0j's from each value and makes Y_POW purely re
al (no longer complex type with trailing 0j components)


fstep = fsamp/Nsamp #Calculate the frequency step in the transform

#Create an array containing all the frequencies
f = []
for i in range(0,Nsamp):
     f.append(i*fstep)

#the transform has been calculated

#plot the Fourier power spectrum
plt.plot(f,Y_POW)
plt.xlabel('f')
plt.ylabel('$Y_{POW}$')
plt.title("Fourier Power Spectrum")
plt.grid()
plt.show()

#there is a second peak at about 4200 Hz which is actually a power spectrum of
negative frequencies
#need to slide upper half down using fftshift function and need to adjust freq
uency values to properly display power spectrum

Y_POW = np.fft.fftshift(Y_POW) #use fftshift fuction to properly display the s
ecind negative peak

f2 = [] #adjusting frequency values
for i in f:
     f2.append(i - (fsamp/2))
```

```
plt.plot(f2,Y_POW) #plotting the power spectrum again with the second peak cor
rected
plt.ylim([0, max(Y_POW)])
plt.xlim([-fsamp/2, +fsamp/2])
plt.xlabel('$f_2$')
plt.ylabel('$Y_{POW}$')
plt.title("Corrected Fourier Power Spectrum")
plt.grid()
plt.show()
```

```
The Fourier Transform of the sampled signal is complex
The Fourier Transform contains Nsamp elements
The Power Spectrum is real
```





Note that second peak occurs at minus the value of signal frequency as expected

In [360]:
```
#task 4

print("max Y_POW is", max(Y_POW)) #prints peak value from Power Spectrum Graph
Pow = Y_POW.tolist()

print("peak occurs at ", f2[Pow.index(max(Y_POW))], " Hz") #Finds Frequency th
at peak occurs at
```

```
max Y_POW is 15142.106066060922
peak occurs at  -800.78125  Hz
```

Task 4 comments

Nsamp	peak value	frequency
32	237.5	+/- 781.25 Hz
128	1816.1	+/- 781.25 Hz
256	16303.8	+/- 800 Hz

As Nsamp increases, I observed that the height of the peak along the y (Y_POW) axis increases and the Full width at half maximum of the Fourier Peak decreases.

The Fourier Power Spectrum becomes more concentrated when the number of sample values increases thus the size of the Fourier peak increases and the width of the peak decreases. So we see a greater number of samples increases precision and accuracy of the Fourier Spectrum when plotted.

Task 5

(i) Nyquist Rate is twice the highest frequency present in a signal so here it is 1600 hz

(ii):

Sampling Frequency (Hz)	Frequency Fourier Peak occurs (Hz)
2000	800
1800	800
1600	800
1400	600
1200	400
1000	200

(iii): Once the sampleing rate is reduced to the Nyquist rate, reducing the sampling rate further, reduces the Frequency where the Fourier Peak occurs. There is no change in the position of the peaks on the plot when changing the Sampling Frequency when it is above the Nyquist rate because the Nyquist frequency determines the maximum frequency representable in a sampled waveform.

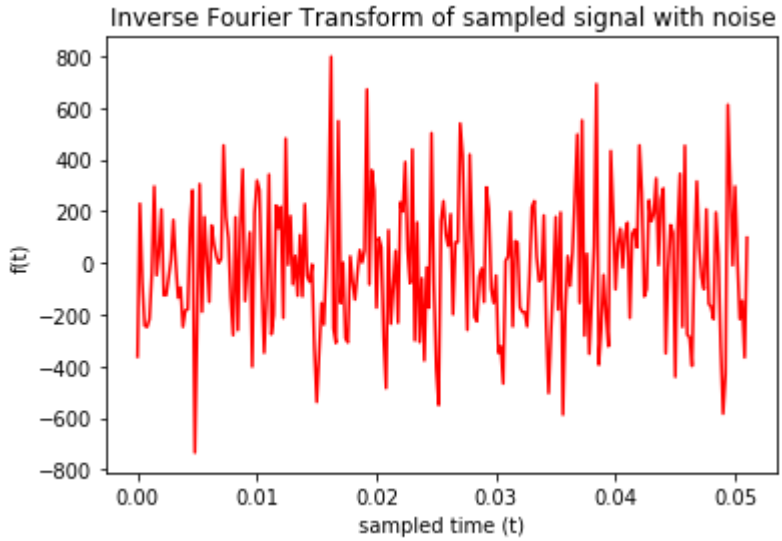
(iv)

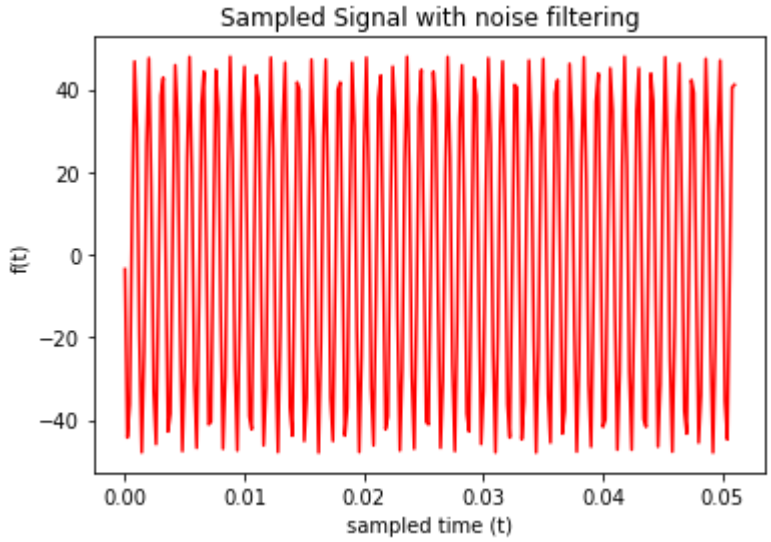
$$F - (F_N - F_S)$$

Where F is the Frequency of the SIgnal, F_N is the Nyquist Rate, F_S is the Sample Frequency

(v)Aliasing is when there is missing information in the reconstructed signal compared to the original signal. For example if we use a sampling rate of 1400, it will give us a reconstructed signal of 600 Hz when the original signal was 800 Hz.

In [195]:
```
#task 7
from numpy.fft import ifft #import inverse fourier transform function


Y_IFT = ifft(Y_FT) #get inverse transform of the fourier transform to give the
original sampled signal again

#plot inverse transform
plt.title("Inverse Fourier Transform of sampled signal with noise")
plt.xlabel("sampled time (t)")
plt.ylabel("f(t)")
plt.plot(tsamp,Y_IFT.real,'r-') #plot red line
plt.show()
#this is the same plot as the plot of the original sampled noisy signal in par
t 2

Y_FT1 = Y_FT


#set components of the Fourier Transform to zero if the Fourier power is less
 than some suitable threshold
for i in range(Nsamp):
    if Y_POW[i] < 0.9*max(Y_POW) : #set threshold as 90% of the peak of the po
wer spectrum
        Y_FT1[i] = 0 #peaks below this value are set to zero

Y_filtered = ifft(Y_FT1) #get inverse fourier transform to transform back to t
ime domain where we can plot the filtered sampled signal


plt.title("Sampled Signal with noise filtering")#plotting
plt.xlabel("sampled time (t)")
plt.ylabel("f(t)")
plt.plot(tsamp,Y_filtered.real,'r-') #plot red line
plt.show()
```





(iii): From the above graph we can see that almost all the noise was successfully removed but there is still some noise present as seen by the variance in peak sizes.

task 8:

(i) Two peaks in the power spectrum are visible at the correct frequencies

(ii) f2 is still distinct from f1 on the Power Spectrum when f1 = 800 and f2 = 880

(iii) I observed that a lower number of samples makes it more difficult to distinguish the frequencies from eachother

(iv) It becomes very difficult to distinguish between the two frequencies once there is noise present in the signal. My filtering method would not work since the f2 peak is significantly smaller than the f1 peak. I could not easily distinguish both frequencies when there was more than 1% of the noise that was present in task 6 and 7