FFT project

September 15, 2020

[ ]:

[1]:
```
#Lets insert the necessary libraries to work with
from scipy import signal
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import matplotlib as mpl

from scipy.stats import norm
import seaborn as sns
from scipy import stats
```

[2]:
```
file_1 = pd.read_table('fmserp1901.txt', sep="\t", header=None)
file_2 = pd.read_table('fmserp1902.txt', sep="\t", header=None)
file_3 = pd.read_table('fmserp1903.txt', sep="\t", header=None)
result =pd.concat([file_1, file_2, file_3], ignore_index=True)
# Replacing the comma values

vol = result.iloc[:,0] =result.iloc[:,0].str.replace(',', '.').astype(float)
vol=abs(vol)
pa = result.iloc[:,1] = result.iloc[:,1].str.replace(',', '.').astype(float)
pc = result.iloc[:,2] = result.iloc[:,2].str.replace(',', '.').astype(float)
i  = result.iloc[:,3] = result.iloc[:,3].str.replace(',', '.').astype(float)
```

[3]:
```
result.shape
```

[3]:
```
(3686400, 4)
```

[4]:
```
result.head()
```

[4]:

	0	1	2	3
0	-0.106095	0.064650	0.019172	-0.000478
1	-0.105342	0.064215	0.015870	-0.000455
2	-0.107630	0.061248	0.014717	-0.000380
3	-0.108123	0.061536	0.019126	-0.000384
4	-0.108258	0.062158	0.019610	-0.000360

[5]:
```
result.describe()
```

[5]:

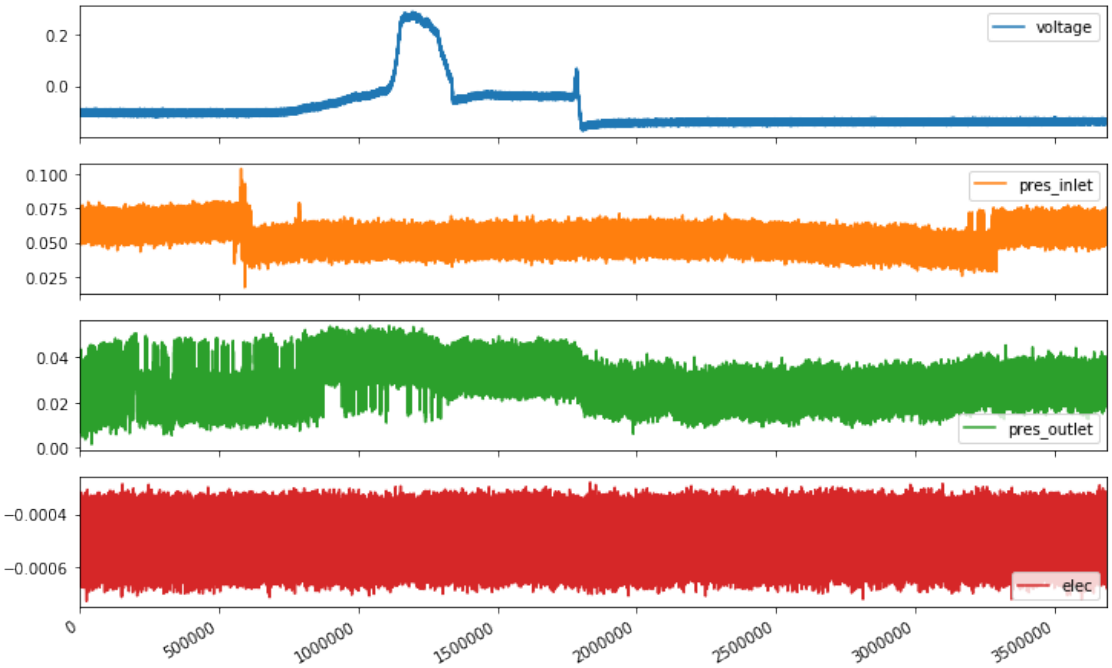
	0	1	2	3
count	3.686400e+06	3.686400e+06	3.686400e+06	3.686400e+06
mean	-9.500333e-02	5.386326e-02	2.838900e-02	-4.973958e-04
std	8.395515e-02	6.834604e-03	6.869441e-03	5.617059e-05
min	-1.752920e-01	1.721500e-02	1.452000e-03	-7.270000e-04
25%	-1.429140e-01	4.898700e-02	2.319000e-02	-5.390000e-04
50%	-1.347620e-01	5.277900e-02	2.713700e-02	-5.010000e-04
75%	-7.802700e-02	5.857100e-02	3.362125e-02	-4.570000e-04
max	2.888500e-01	1.041360e-01	5.407200e-02	-2.810000e-04

Considering data above, current value can be excluded from the calculations as it is way too less and it implies a reading / recording error.

[6]:
```
result.columns = ['voltage', 'pres_inlet', 'pres_outlet', 'elec']
```

Lets add column names and plot the data !!!!

[7]:
```
# result.plot(subplots = True )
result.plot(subplots = True , figsize=(12,8));
```



Colourful !! Signal processing is actually colourful !!! Decoding the information. Based on the plots , voltage is highly unevenly distributed which shows some issues with respect to the fluctuations. , pressure values again have fluctuations and current is dead .... Its good to keep it out. A very symmetrical distribution of current shows it has no effect by the system.

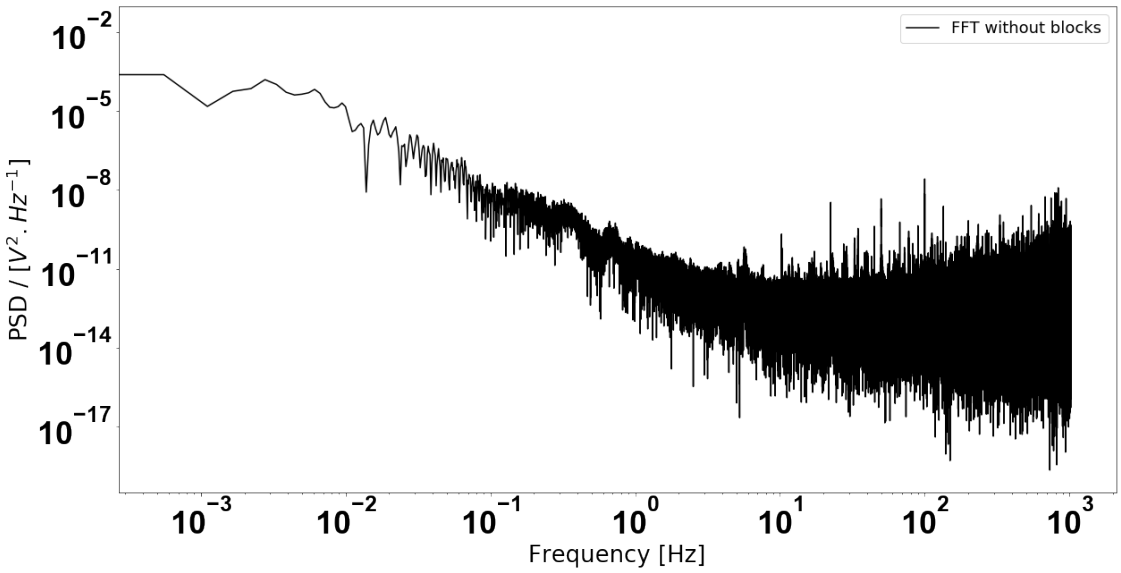
The data collection is in frequency (1s = 2048 Hz) of 2048 points. It means lets convert this to seconds and analyse further.

[8]:
```
t= len(result)
test_list = [0 + (x * 0.00048828125) for x in range(0, t)]
x = test_list[0:t]
```

Let's know check the Power spectral density of the signals. Power spectral density can tell the energy of the signal, can explain about the noise, if it is white noise or pink noise or fractural noise or it is the device noise. There are againmany methods to estimate the power spectral density (PSD), the most fast is the direct FFT analysis. Direct FFT analysis does not include the signal breakdown into parts, infact calculate the overall PSD at once. Lets do it !! Sampling rate is fixed to 2048 Hz

[9]:
```
import scipy
N = len(result)
srate = 2048
fft_raw = np.abs( scipy.fftpack.fft(vol)/N )**2
hz = np.linspace(0,srate/2,int(np.floor(N/2)+1))

figsizen = (20, 10)
fig = plt.figure(figsize =  figsizen)

plt.loglog(hz, fft_raw[0:len(hz)], color = 'black', label = 'FFT without blocks')
plt.legend(loc='upper right', fontsize =18)
plt.xlabel('Frequency [Hz]', fontsize = 26)
plt.ylabel('PSD / [$V^2.Hz^{-1}$]', fontsize = 26)
plt.xticks(fontsize = 36, fontname = 'Arial', weight = 'bold')
plt.yticks(fontsize = 36, fontname = 'Arial', weight = 'bold')
```

[9]:
```
(array([1.e-23, 1.e-20, 1.e-17, 1.e-14, 1.e-11, 1.e-08, 1.e-05, 1.e-02,
        1.e+01, 1.e+04]), <a list of 10 Text yticklabel objects>)
```



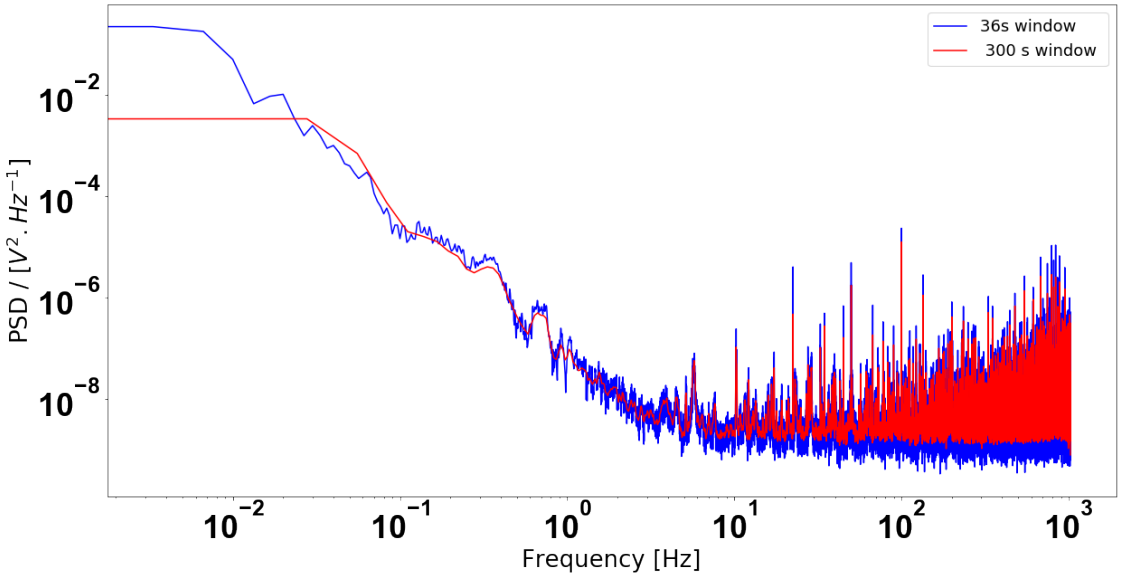
If we observe the psd data, it is noisy !!! Peaks cannot be evaluated clearly moreover the high frequency data (f>0.1Hz) cannot be identified for a certain peak. Also the PSD is plotted in log log scale. The reason for this scale is to plot values with huge difference in values. Analysing further shows on major peak and after PSD is inversely propotional to 1/f^2

At this point it is important to use othe PSD calculations tools , welch's scheme is mostly used , apart from them

[10]:
```
import math
winsize36 = int(36*srate)
hannw36  = .5 - (np.cos(2*math.pi*np.linspace(0,1,winsize36)))/2
f36, welchpow36 = scipy.signal.welch(vol,fs=srate,window=hannw36,␣
 ↪nperseg=winsize36,noverlap=None, nfft=None)

winsize3600 = int(300 *srate )
hannw3600 = .5 - (np.cos(2*math.pi*np.linspace(0,1,winsize3600)))/2
f3600, welchpow3600 = scipy.signal.welch(vol,fs=srate,window=hannw3600,␣
 ↪nperseg=winsize3600,noverlap=None, nfft=None)


figsizen = (20, 10)
fig = plt.figure(figsize =  figsizen)

plt.loglog(f3600, welchpow3600, color =   'b', label = '36s window')
plt.loglog(f36, welchpow36, color =   'r', label = ' 300 s window ')
plt.legend(loc='upper right', fontsize =18)
plt.xlabel('Frequency [Hz]', fontsize = 26)
plt.ylabel('PSD / [$V^2.Hz^{-1}$]', fontsize = 26)
plt.xticks(fontsize = 36, fontname = 'Arial', weight = 'bold')
```

```
plt.yticks(fontsize = 36, fontname = 'Arial', weight = 'bold')
```

[10]:
```
(array([1.e-12, 1.e-10, 1.e-08, 1.e-06, 1.e-04, 1.e-02, 1.e+00, 1.e+02]),
 <a list of 8 Text yticklabel objects>)
```



The only parameters which we have used in welch's scheme is window, Here we use Hann window. In literature there are many windows available - Hann/Hanning, Blackman, Hamming, Kaiser-Bessel. Every window has there own use. Why we require windows is to avoid the artifacts which can result by dividing the signals and applying individual PSD calculations on it. Also in the previous figure we have estimated PSD overall and there are few parameters which needs to be fixed mainly the number of blocks , the window used / not , overlapping done , sampling frequency can again play an important role in the analysis.

Compared to direct FFT static , the present PSD by welch's scheme is much more better with less noise , peaks can be identified.

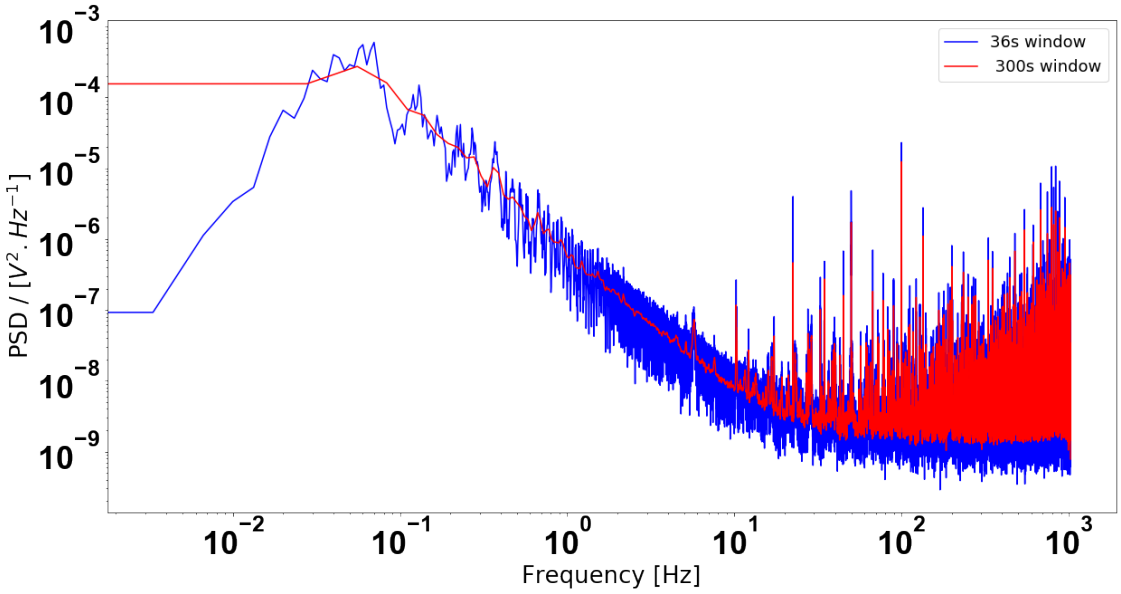
Let's now detrend the signal and check what can be the hidden information be ? Again as explained in my previous codes, a very large window will remove out smaller frequencies and a very samller window will eliminate the higher frequencies. Hence a window size of 5-20 seconds should be chosen.

[11]:
```
vol = signal.detrend(vol, axis = -1,type = 'linear', overwrite_data = False, bp␣
 ↪=  np.arange(0,len(vol),20*2048))
```

[12]:
```
import math
winsize36 = int(36*srate)
hannw36  = .5 - (np.cos(2*math.pi*np.linspace(0,1,winsize36)))/2
f36, welchpow36 = scipy.signal.welch(vol,fs=srate,window=hannw36,␣
 ↪nperseg=winsize36,noverlap=None, nfft=None)
```

```
winsize3600 = int(300 *srate )
hannw3600 = .5 - (np.cos(2*math.pi*np.linspace(0,1,winsize3600)))/2
f3600, welchpow3600 = scipy.signal.welch(vol,fs=srate,window=hannw3600,␣
 ↪nperseg=winsize3600,noverlap=None, nfft=None)


figsizen = (20, 10)
fig = plt.figure(figsize =  figsizen)

plt.loglog(f3600, welchpow3600, color =   'b', label = '36s window')
plt.loglog(f36, welchpow36, color =   'r', label = ' 300s window')
plt.legend(loc='upper right', fontsize =18)
plt.xlabel('Frequency [Hz]', fontsize = 26)
plt.ylabel('PSD / [$V^2.Hz^{-1}$]', fontsize = 26)
plt.xticks(fontsize = 36, fontname = 'Arial', weight = 'bold')
plt.yticks(fontsize = 36, fontname = 'Arial', weight = 'bold')
```

[12]:
```
(array([1.e-11, 1.e-10, 1.e-09, 1.e-08, 1.e-07, 1.e-06, 1.e-05, 1.e-04,
        1.e-03, 1.e-02, 1.e-01]), <a list of 11 Text yticklabel objects>)
```

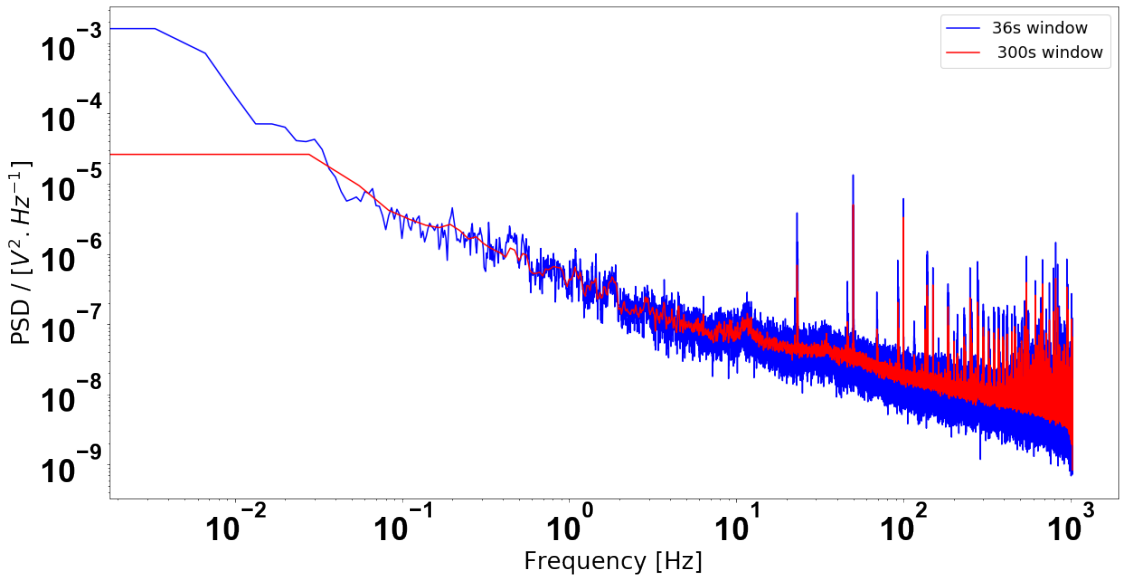


Lets check the same thing for the pressure analysis

[13]:
```
import math
winsize36 = int(36*srate)
hannw36  = .5 - (np.cos(2*math.pi*np.linspace(0,1,winsize36)))/2
f36, welchpow36 = scipy.signal.welch(pa,fs=srate,window=hannw36,␣
 ↪nperseg=winsize36,noverlap=None, nfft=None)
```

```
winsize3600 = int(300 *srate )
hannw3600 = .5 - (np.cos(2*math.pi*np.linspace(0,1,winsize3600)))/2
f3600, welchpow3600 = scipy.signal.welch(pa,fs=srate,window=hannw3600,␣
 ↪nperseg=winsize3600,noverlap=None, nfft=None)


figsizen = (20, 10)
fig = plt.figure(figsize =  figsizen)

plt.loglog(f3600, welchpow3600, color =   'b', label = '36s window')
plt.loglog(f36, welchpow36, color =   'r', label = ' 300s window')
plt.legend(loc='upper right', fontsize =18)
plt.xlabel('Frequency [Hz]', fontsize = 26)
plt.ylabel('PSD / [$V^2.Hz^{-1}$]', fontsize = 26)
plt.xticks(fontsize = 36, fontname = 'Arial', weight = 'bold')
plt.yticks(fontsize = 36, fontname = 'Arial', weight = 'bold')
```

[13]: (array([1.e-11, 1.e-10, 1.e-09, 1.e-08, 1.e-07, 1.e-06, 1.e-05, 1.e-04,
1.e-03, 1.e-02, 1.e-01]), <a list of 11 Text yticklabel objects>)



We are at the end of this analysis. We have done maximum treatment possible. This can further be updated if we can use filters in order to ward off some specific frequency in the domain.

Based on the effect of detrending , windowing and window size it can be observed that the only noticable slope is the slope in low frequency region (f<10Hz) and that is -1.6, this can further checked or calculated by using regression analysis. Apart from that at higher frequency is the noise (f>10 Hz). There are noticable peaks in the region at f = 8Hz, f = 9.1 Hz but they cannot be inferred to mix up with the noise at f>10Hz. They can have important information considering

the water hydrodynamics but the peaks of noise is higher at higher frequency …..

Two things are very important when analysing the signals in frequency doamin - 1) It should be understood that while capturing the signal information , the PSD should be more then the bench /setup noise . There is always a noise of the setup, if the PSD of the fuel cell voltage of pressure is more then the setup noise , then only analysis can be carreid out.

2) It should be properly verified that the signal is due to the interation with gas / fluid or other electrochemical operation and not the sensor noise in the analysis.

[ ]: