Whitepaper – August 17, 2020

**By Neha Jirafe – Data Scientist at Chord X**

# Using FFT to analyse and cleanse time series data

## Digitalize Product and unexpected noise

## 1. The Challenge

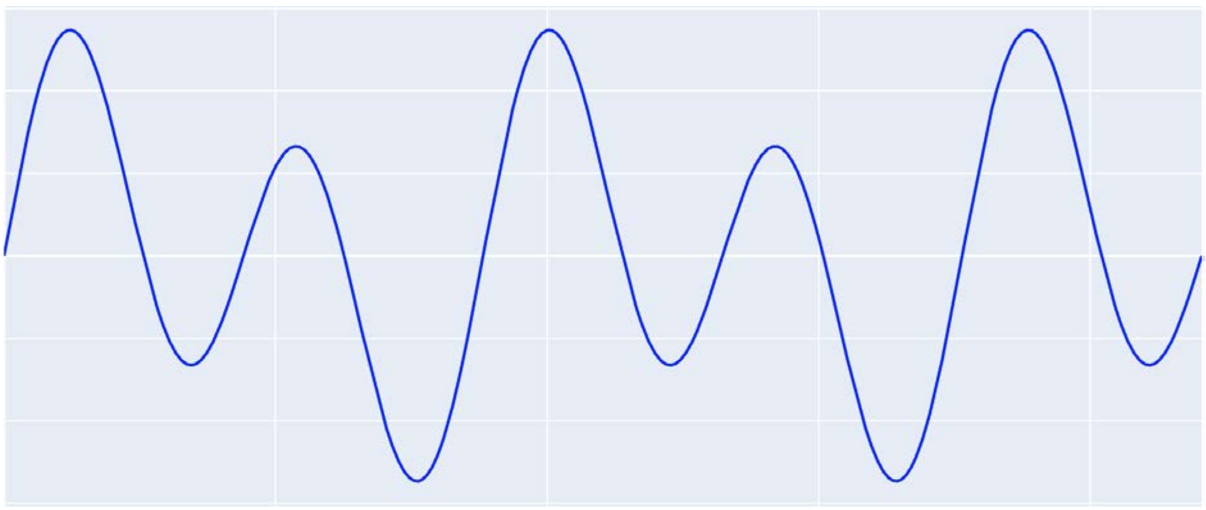
Often when dealing with IOT data, the biggest challenge is encountering unexpected noise. This noise can be very troublesome when you want to derive values from the signal. The noise can easily mislead your models with inaccurate values

The challenge lies in how we can clean the noise from the data when (often) we don't know the frequency of which. To make matters worse, these frequencies may varying constantly due to the operating conditions of sensors.

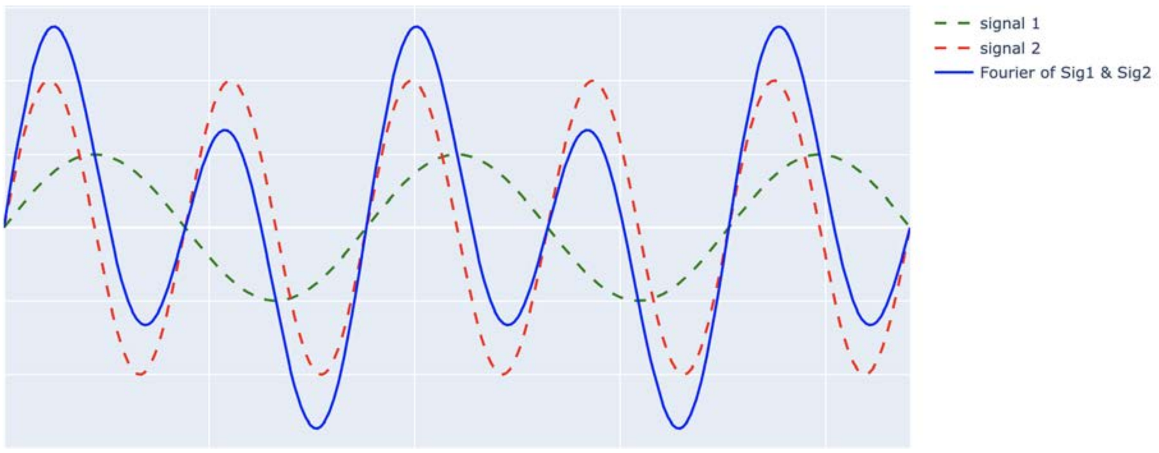
A possible solution is to decompose the signal. Once decomposed, it will be easier to filter out the nose.

## 2. The Fourier Series (FFT)

One such solution is the Fourier Series, where Baron Jean Baptiste Joseph Fourier introduced the idea that "Any reasonable continuous periodic function can be represented by a series of sines".



To illustrate the theory, the above signal can be decomposed into two discrete signals as shown in the picture below.



It will be easier to either filter recover or filter the noise, if we can find the frequency and amplitude components of these individual noise signals. This can be easily achieved by applying FFT on the original signal.

That is exactly what a Fourier transformation does. It takes the original signal and decomposes it to the frequencies that made it up.

## 3. Signal Generation

Let's start by creating a signal!

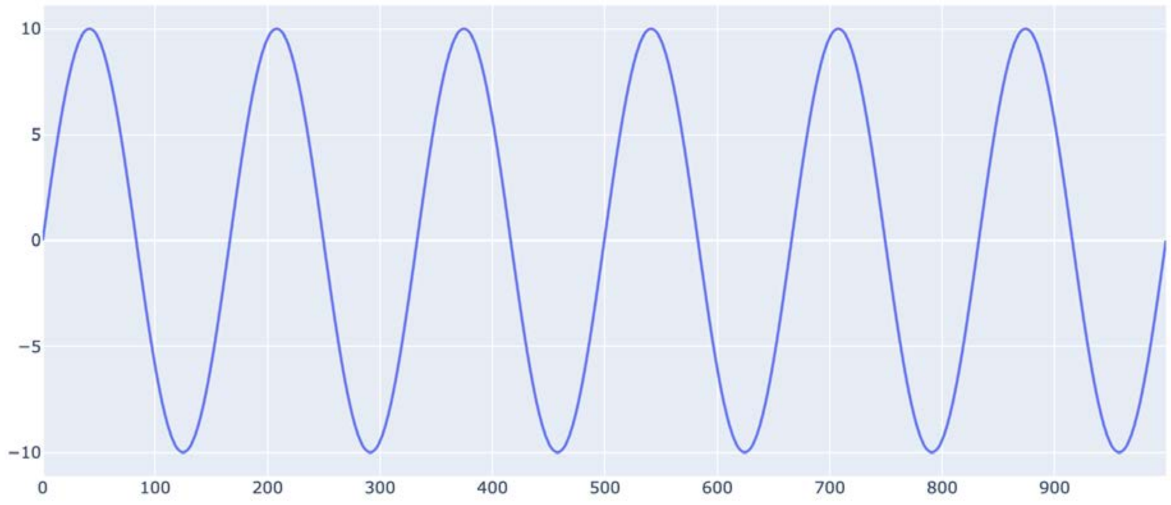
```
import numpy as np#Seconds to generate data for
```

```
sec = 3# time range with total samples of 1000 from 0 to 3 with time interval
equals 3/1000
time = np.linspace(0, sec, 1000, endpoint=True)
```

---

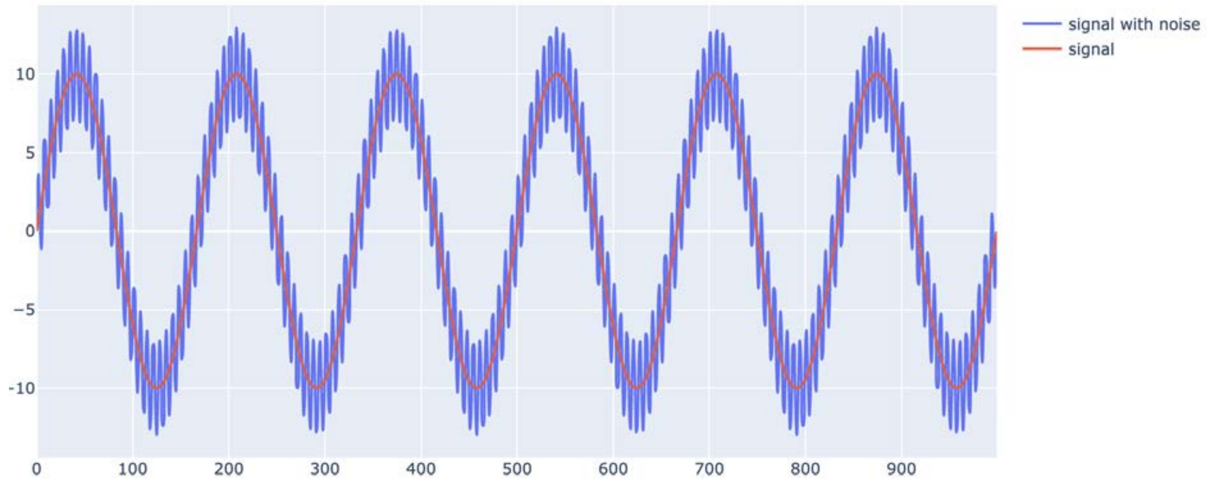
```
# Generate Signal
signal_freq = 2 # Signal Frequency
signal_amplitude = 10 # Signal Amplitude#Sine wave Signal
signal = signal_amplitude*np.sin(2*np.pi*signal_freq*time)
```

The code will generate a 2hz signal with amplitude of 10



Now let's add some 50hz noise with amplitude of 3 to the above signal and create a new signal

```
# Lets add some noise to the Signal
noise_freq = 50 # Noise Frequency
noise_amplitude = 3 # Noise Amplitude#Sine wave Noise
noise = noise_amplitude*np.sin(2*np.pi*noise_freq*time) wave# Generated
Signal with Noise
signal_noise = signal + noise
```

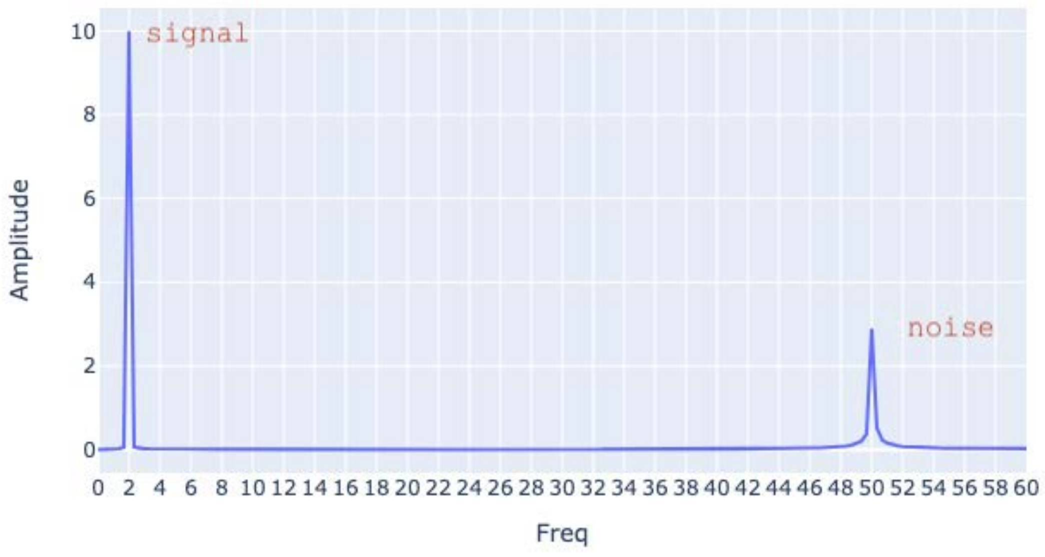


## 4. FFT to decompose Signal

We will use the python scipy library to calculate FFT and then extract the frequency and amplitude from the FFT,

```
from scipy import fftpacksig_noise_fft = scipy.fftpack.fft(signal_noise)
sig_noise_amp = 2 / time.size * np.abs(sig_noise_fft)
sig_noise_freq = np.abs(scipy.fftpack.fftfreq(time.size, 3/1000))
```

The following plot can be generated by plotting *"sig_noise_freq"* vs *"sig_noise_amp"*



## 5. Calculating the Amplitude

All the amplitudes are stored in *"sig_noise_amp",* we just need to fetch the top 2 amplitudes from the list.

## 6. Signal Amplitude

```
Signal_amplitude =
pd.Series(sig_noise_amp).nlargest(2).round(0).astype(int).tolist()print(signal
_amplitude)
```

**Output: [10, 3]**

## 7. Calculating the Frequency

```
#Calculate Frequency Magnitude
magnitudes = abs(sig_noise_fft[np.where(sig_noise_freq >= 0)])#Get index of
```

```
top 2 frequencies
peak_frequency = np.sort((np.argpartition(magnitudes, -2)[-
2:])/sec)print(peak_frequency)
```

```
Output: [ 2 ,50]
```

## 8. Filtering the Noise

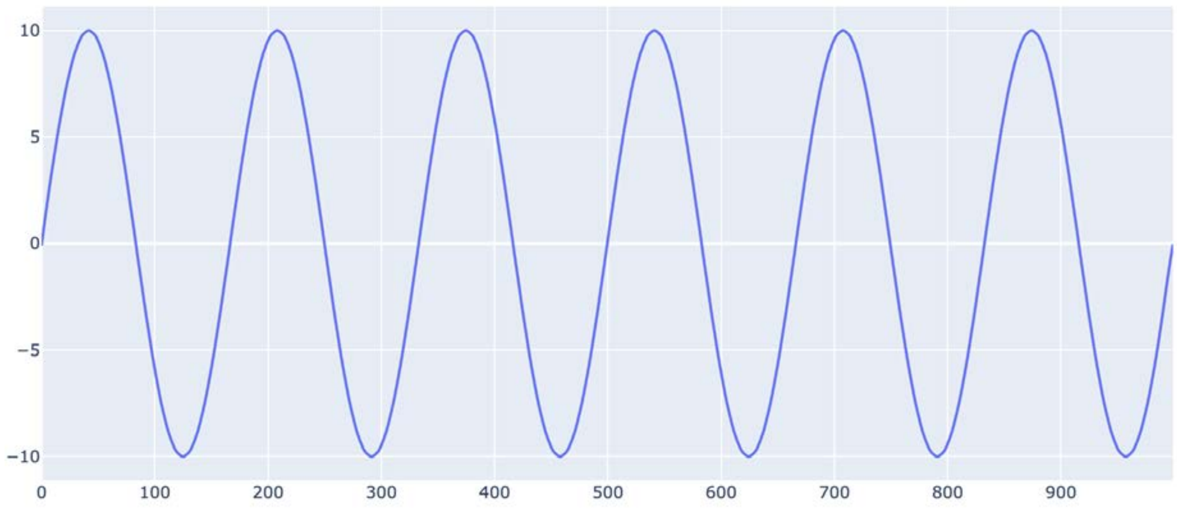
Noise can be classified as a signal with high frequency and low amplitude. In our case the noise has a frequency of 50 Hz and amplitude of 3.

Since we have already calculated the frequency and amplitude components of the signal, we can now filter out the signal with high frequency and low amplitude

We will use Butterworth Low Pass Filter , details can be found here , the cutoff frequency will be the ***peak_frequency[0]***

```
from scipy.signal import butter,filtfilt# Filter requirements.
fs = 50.0       # sample rate, Hz
cutoff = peak_frequency[0]     # desired cutoff frequency of the filter, Hz ,
slightly higher than actual 2 Hz
order = 2       # sin wave can be 5pprox. represented as quadratic
def butter_lowpass_filter(data, cutoff, fs, order):
    print("Cutoff freq " + str(cutoff))
    nyq = 0.5 * fs # Nyquist Frequency
    normal_cutoff = cutoff / nyq
    # Get the filter coefficients
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    y = filtfilt(b, a,data)
    return y
# Filter the data, and plot filtered signals.
y = butter_lowpass_filter(signal_noise, cutoff, fs, order)
```

Now let's plot the filtered Signal "Y"



Voila! we can see that we have recovered the original signal after filtering the Noise.