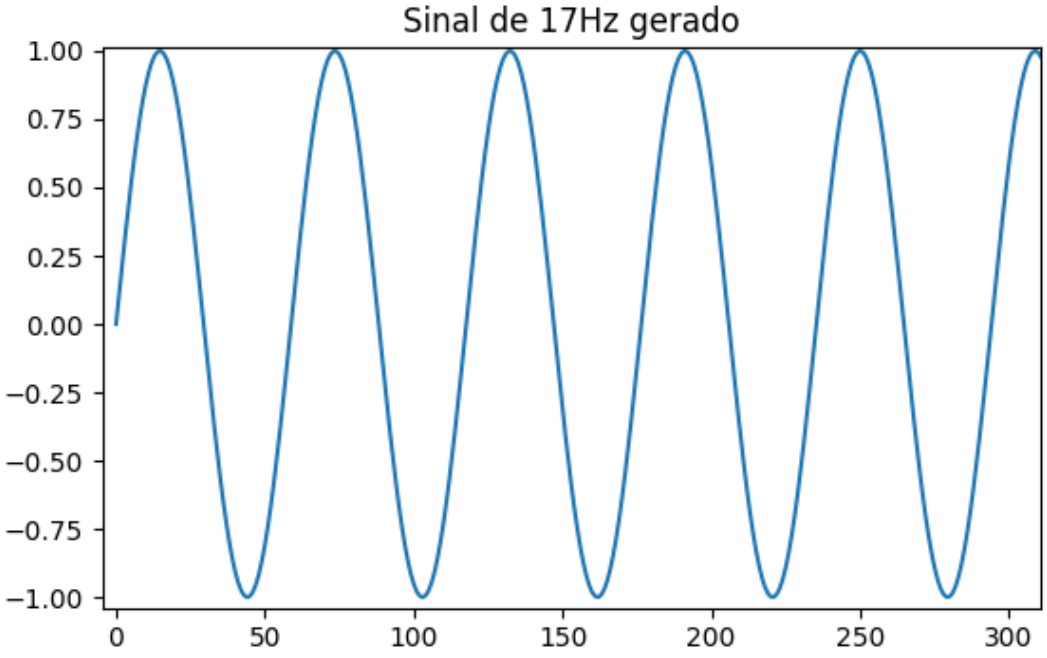
Bruno Samuel Luiz de Oliveira

https://github.com/brunosamuell **DATA: 13/12/2017**

Teste da biblioteca de filtragem (Butter_Bandpass_filter)

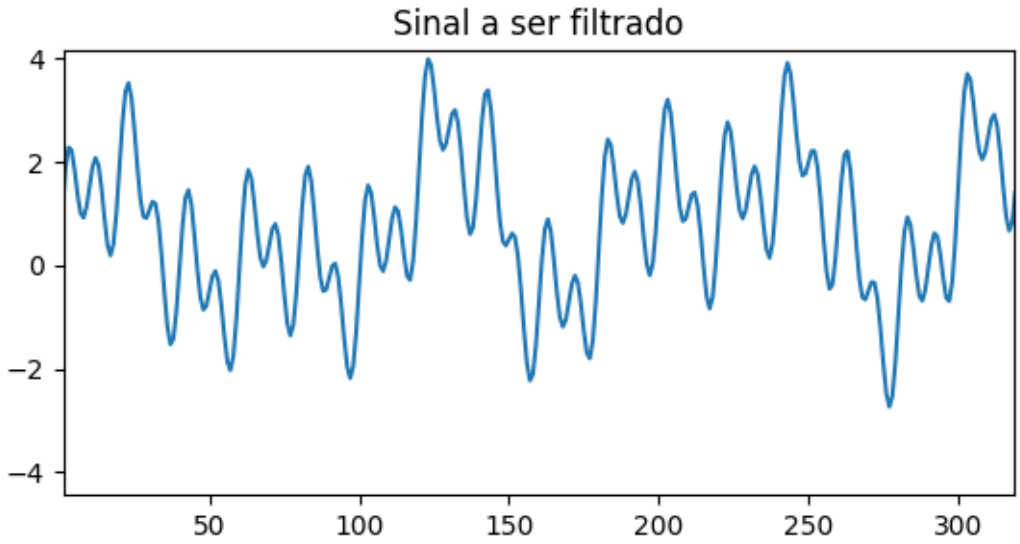
Este algoritmo tem por objetivo gerar cinco ondas senoidais, nos valores de 100Hz, 50Hz, 17Hz, 10Hz e 1Hz, conforme figura 1 demonstra e as somar. Posteriormente filtrar o sinal em uma frequência desejada, que no caso do projeto esta centrado em 17Hz.

Figura 1



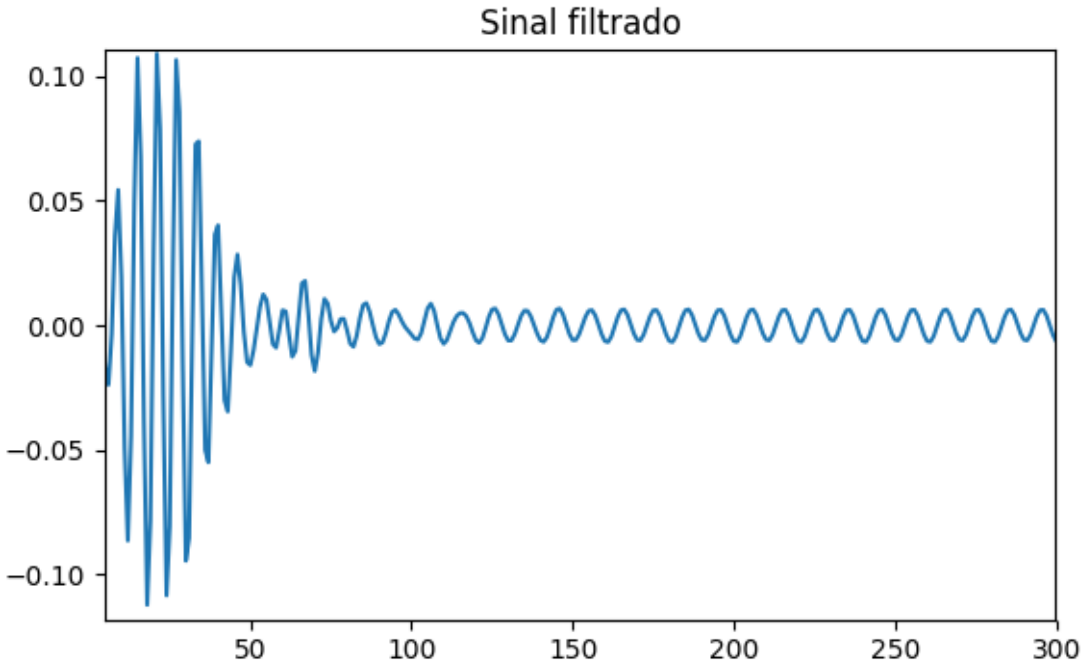
A soma das cinco ondas senoidais geradas esta representada na figura 2.

Figura 2



Logo em seguida com o auxilio das bibliotecas de filtragem para linguagem Python, explicitadas no código fonte, onde a partir destas se programou um filtro passa faixa, de terceira ordem, com frequências de corte inicial de 15Hz e final de 18Hz, onde tal filtro teria por objetivo selecionar apenas a frequência de 17Hz, os dados de resposta filtrados estão demonstrados na figura 3

Figura 3



Código fonte

```
# Autor: Bruno Samuel Luiz de Oliveira
# Ultima alteração: 12 /12 /2017

import numpy as np
import pandas as pd
from scipy import signal
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
import matplotlib.animation as animation

from numpy import cos, sin, pi, absolute, arange, array, append, abs,
angle,\
                fft, linspace, zeros, log10, unwrap
from scipy.signal  import kaiserord, lfilter, firwin, freqz, butter, lfilter
from pylab import figure, clf, plot, xlabel, ylabel, xlim, ylim, title, grid,
\
                axes, show, subplot, axis, plot
```

```
#funçao para geração da senoidal
def sine_generator(fs, sinefreq, duration):
    T = duration
    nsamples = fs * T*10
    w = 2. * np.pi * sinefreq
    t_sine = np.linspace(0, T, nsamples, endpoint=False)
    y_sine = np.sin(w * t_sine)
    result = pd.DataFrame({
        'data' : y_sine} ,index=t_sine)
    return result

#funçao butter_bandpass
def butter_bandpass(lowcut, highcut, fs, order=3):
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    b, a = butter(order, [low, high], btype='band')
    return b, a

#funçao butter_bandpass_filter
def butter_bandpass_filter(data, lowcut, highcut, fs, order=3):
    b, a = butter_bandpass(lowcut, highcut, fs, order=order)
    y = lfilter(b, a, data)

    return y
#freqeuncia da amostra
fps = 100

#Gerando sinal de 100Hz
sine_fq = 100 #Hz
duration = 100 #seconds
sine_100Hz = sine_generator(fps,sine_fq,duration)

#Gerando sinal de 50Hz
sine_fq = 50 #Hz
duration = 100 #seconds
sine_50Hz = sine_generator(fps,sine_fq,duration)

#Gerando sinal de 17Hz
sine_fq = 17 #Hz
duration = 100 #seconds
sine_17Hz = sine_generator(fps,sine_fq,duration)
```

```
#Gerando sinal de 10Hz
sine_fq = 10 #Hz
duration = 100 #seconds
sine_10Hz = sine_generator(fps,sine_fq,duration)

#Gerando sinal de 1Hz
sine_fq = 1 #Hz
duration = 100 #seconds
sine_1Hz = sine_generator(fps,sine_fq,duration)

#Soma as ondas geradas
sine = sine_100Hz + sine_50Hz + sine_17Hz + sine_10Hz + sine_1Hz

#Seleciona as frequenicas de corte baixa e alta
lowcut = 15
highcut = 18

#Chama a funão butter_bandpass_filter e filtra o sinal gerado
y = butter_bandpass_filter(sine.data, lowcut, highcut, fps, order=3)

#plota o sinal gerado
plt.figure(1)
plt.clf()
plt.plot(range(len(sine_17Hz)),sine_17Hz)
plt.title('Sinal de 17Hz gerado')

#plota o sinal desejado
plt.figure(2)
plt.clf()
plt.plot(range(len(sine)),sine)
plt.title('Sinal a ser filtrado')

#plota o sinal filtrado
plt.figure(3)
plt.clf()
plt.plot(range(len(y)),y)
plt.title('Sinal filtrado')
plt.show()
```

Obs.: Caso estas linhas contribuam com seu projeto, para fins de reconhecimento citar o autor Bruno Samuel Luiz de Oliveira, como também a fonte do material disponibilizado no GitHub.