Alteração de Taxas e Estruturas Polifásicas

May 14, 2021

Aluno: Leonardo José Held, 17203984.

Nesse trabalho, vamos implementar dois tipos de filtragem: filtragem usual - e lenta -, e por estruturas polifásicas - que pode ser mais rápida -. O sinal de voz utilizado foi gerado por computador, via o plano de estudante da https://www.naturalreaders.com. O texto falado é uma parte da música "Han-tyumi, the Confused Cyborg".

O código é em linguagem Python e a licença permissiva e aberta para todo o documento pode ser encontrada no final deste arquivo. Além disso, os códigos (interativos :-)!) e arquivos fonte podem ser encontrados em https://github.com/leonheld/explorations-into-dsp/tree/master/third-assignment.

A primeira parte desse trabalho (filtragem via método mais usual) é basicamente igual à primeira parte do segundo trabalho.

Incluindo algumas bibliotecas, comentários ao lado sobre o que cada uma faz

[1]:
```
import wave # para lidar com arquivos WAVE
import numpy as np # arrays, só que mais rápidos e melhores
import scipy.signal as sps # para usar função correlação
import scipy.fftpack as fftp # para gerar um gráfico de espectro de frequência
import matplotlib.pyplot as plt # para gerar plots
import seaborn as sns# plots mais belos, afinal, estética é sempre bom
from IPython.display import Audio # para ter um player de aúdio inline, sem␣
 ↪necessidade de um externo
from math import pi, cos

import warnings
warnings.filterwarnings('ignore') # Ignorar os warnings faz o programador ser␣
 ↪mais feliz

fs=48000

sns.set_theme()
```

Essa função separa os canais de aúdio. Copyright Andriy Makukha sob a Creative Commons License

[2]:
```
def save_wav_channel(fn, wav, channel):
    # Read data
```

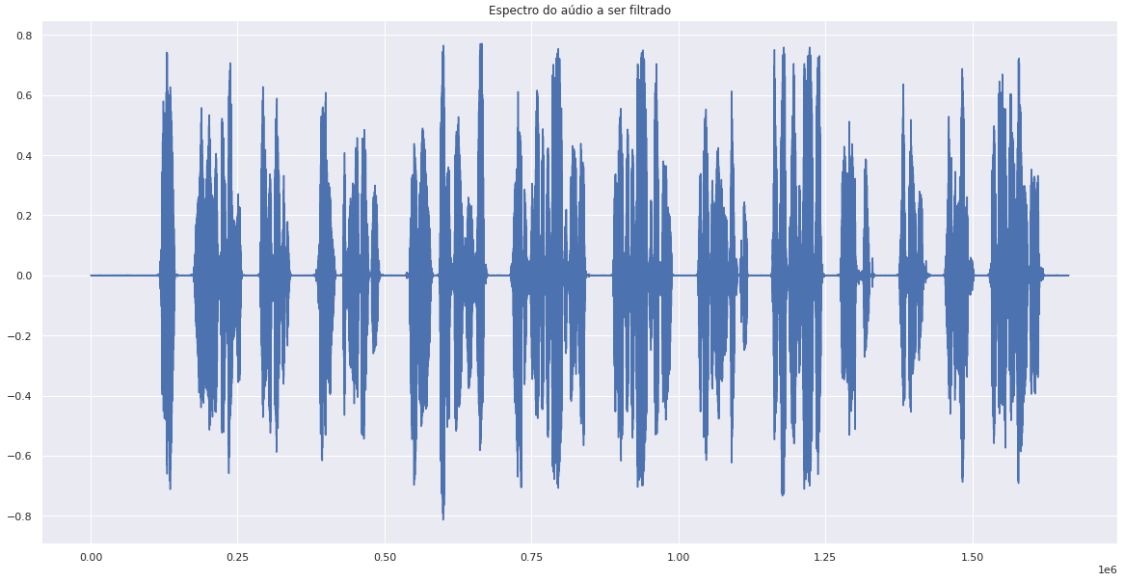
```
    nch   = wav.getnchannels()
    depth = wav.getsampwidth()
    wav.setpos(0)
    sdata = wav.readframes(wav.getnframes())

    # Extract channel data (24-bit data not supported)
    typ = { 1: np.uint8, 2: np.uint16, 4: np.uint32 }.get(depth)
    if not typ:
        raise ValueError("sample width {} not supported".format(depth))
    if channel >= nch:
        raise ValueError("cannot extract channel {} out of {}".
↪format(channel+1, nch))
    print ("Extracting channel {} out of {} channels, {}-bit depth".
↪format(channel+1, nch, depth*8))
    data = np.fromstring(sdata, dtype=typ)
    ch_data = data[channel::nch]

    # Save channel to a separate file
    outwav = wave.open(fn, 'w')
    outwav.setparams(wav.getparams())
    outwav.setnchannels(1)
    outwav.writeframes(ch_data.tostring())
    outwav.close()
```

Trecho de código simples que abre o arquivo .wav e gera um numpy array normalizado Copyright Matthew Walker sob a Creative Commons License, com modificações

[3]:
```
# Read file to get buffer
↪
voiceSampleTwoChannels = wave.open("audioSample.wav")

# Extracting the channels using save_wav_channel() defined above
save_wav_channel('ch1.wav', voiceSampleTwoChannels, 0)
save_wav_channel('ch2.wav', voiceSampleTwoChannels, 1)

voiceSampleFirstChannel = wave.open("ch1.wav")
samples = voiceSampleFirstChannel.getnframes()
audio = voiceSampleFirstChannel.readframes(samples)

# Convert buffer to float32 using NumPy
↪
audio_as_np_int16 = np.frombuffer(audio, dtype=np.int16)
audio_as_np_float32 = audio_as_np_int16.astype(np.float32)

# Normalise float32 array so that values are between -1.0 and +1.0
↪
max_int16 = 2**15
```

```
audio_normalised = audio_as_np_float32 / max_int16

fig, ax = plt.subplots(1, 1, figsize=(20, 10))
ax.plot(audio_normalised)
plt.title("Espectro do aúdio a ser filtrado")
plt.show()

Audio(audio_normalised, rate=fs)
```

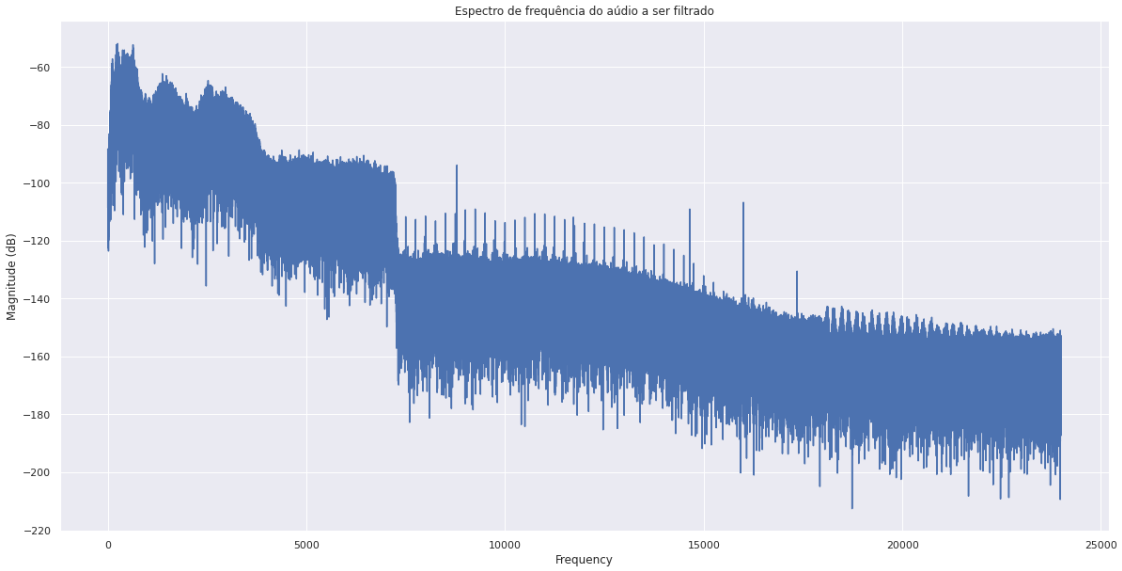
```
Extracting channel 1 out of 2 channels, 16-bit depth
Extracting channel 2 out of 2 channels, 16-bit depth
```



[3]:
```
<IPython.lib.display.Audio object>
```

E esse trecho de código será usado pra plots de espectro de frequência.

[4]:
```
fig, ax = plt.subplots(1, 1, figsize=(20, 10))
ax.magnitude_spectrum(audio_normalised, scale='dB', Fs=fs)
plt.title("Espectro de frequência do aúdio a ser filtrado")
plt.show()
```



0.1 Roteiro de implementação

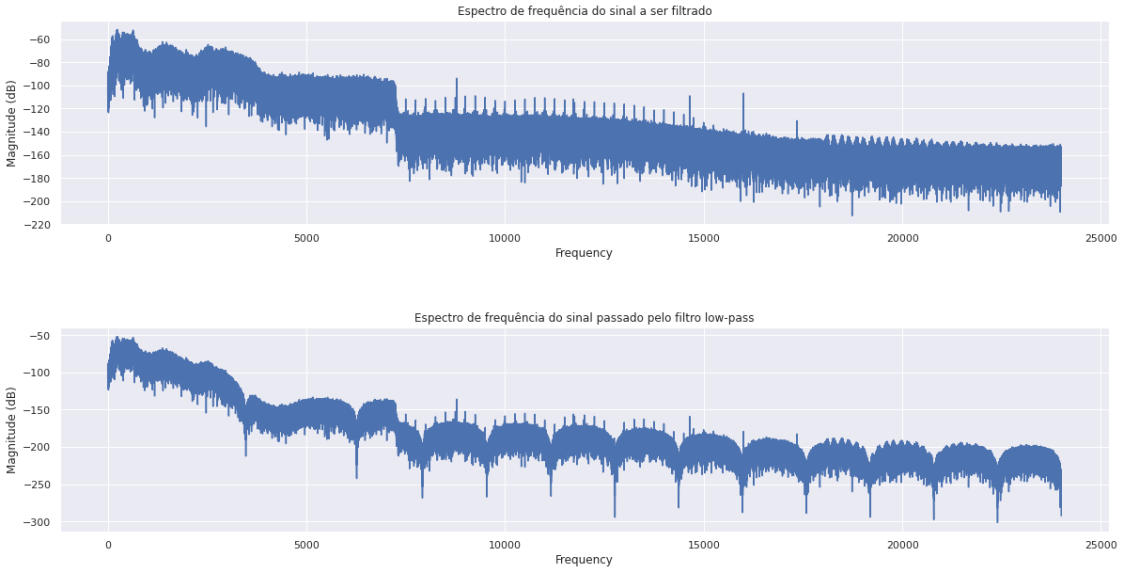
O procedimento a seguir segue o seguinte dataflow:

Filtragem low-pass inicial -> Redução de Taxa -> Aumento de Taxa -> Filtragem low-lass final

0.2 Filtragem low-pass inicial

[5]:
```
# Algumas definições úteis para o cálculo dos coeficientes do FIR
cutoff = pi / 4
nyquist = fs / 2
norm_cutoff = cutoff / nyquist

# Aqui os coeficientes do FIR com janela hamming são calculados...
low_pass_FIR = sps.firwin(30, norm_cutoff, window = "hamming")
# E aqui o audio é filtrado via implementação do tipo forma direta II
low_passed = sps.lfilter(low_pass_FIR, 1, audio_normalised)

# Por fim, é realizado o plot do espectro do espectro de frequência do aúdio␣
 ↪original e do aúdio filtrado
fig, (ax1, ax2) = plt.subplots(2, figsize=(20, 10))

ax1.magnitude_spectrum(audio_normalised, scale='dB', Fs=fs)
ax1.set_title("Espectro de frequência do sinal a ser filtrado")

ax2.magnitude_spectrum(low_passed, scale='dB', Fs=fs)
ax2.set_title("Espectro de frequência do sinal passado pelo filtro low-pass")

fig.subplots_adjust(hspace=0.5)
```

```
plt.show()

Audio(low_passed, rate=fs)
```



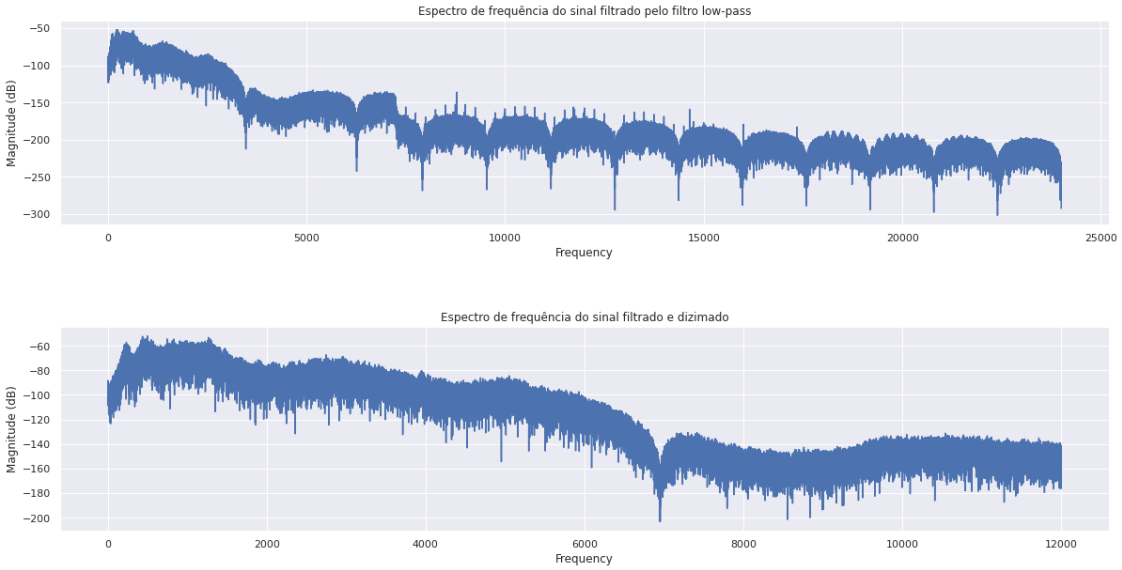
[5]: <IPython.lib.display.Audio object>

Percebe-se uma atenuação bem maior nas frequências mais altas. 20kHz por exemplo sofreu uma atenuação de aprox. 60dB. Enquanto frequências próximas a 0Hz sofreram atenuação máxima aproximada de 10dB.

0.3 Redução de Taxa

[6]:
```
# Usando uma notação de slicing (como a do Matlab) para decimação manual, a␣
 ↪diferença é que usando numpy arrays
# a notação fica arr[start:end:step], por alguma razão misteriosa de design...
decimated_audio = low_passed[0:low_passed.size:4]

fig, (ax1, ax2) = plt.subplots(2, figsize=(20, 10))

ax1.magnitude_spectrum(low_passed, scale='dB', Fs=fs)
ax1.set_title("Espectro de frequência do sinal filtrado pelo filtro low-pass")

ax2.magnitude_spectrum(decimated_audio, scale='dB', Fs=fs/2)
ax2.set_title("Espectro de frequência do sinal filtrado e dizimado")

fig.subplots_adjust(hspace=0.5)
plt.show()
```

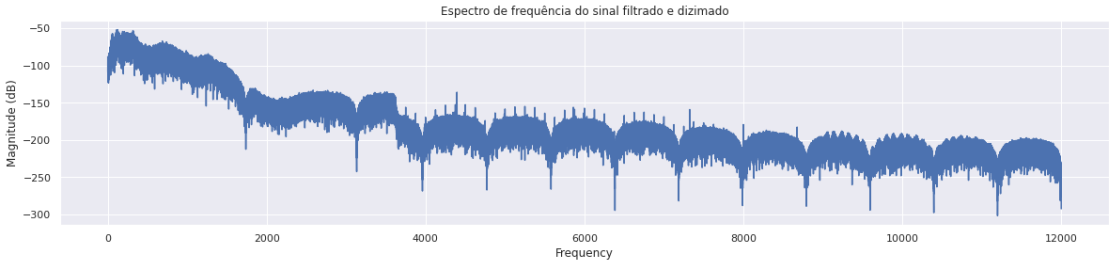
```
Audio(decimated_audio, rate=fs/2)
```

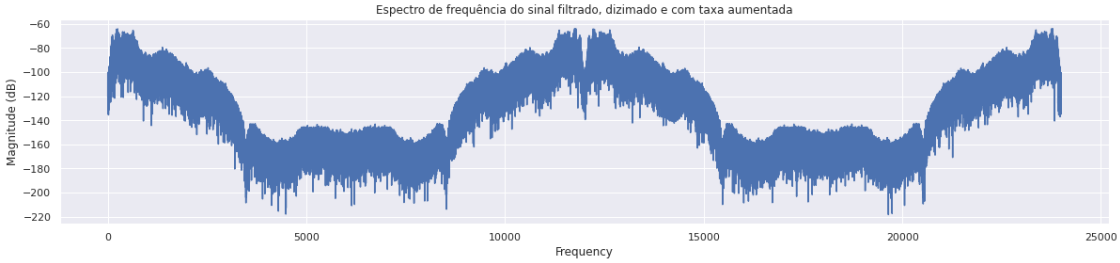


[6]: <IPython.lib.display.Audio object>

0.4 Aumento de Taxa

[7]:
```
upsampled_audio = [0]*(4*(decimated_audio.size)-1)
upsampled_audio[::4] = decimated_audio

fig, (ax1, ax2) = plt.subplots(2, figsize=(20, 10))

ax1.magnitude_spectrum(low_passed, scale='dB', Fs=fs/2)
ax1.set_title("Espectro de frequência do sinal filtrado e dizimado")

ax2.magnitude_spectrum(upsampled_audio, scale='dB', Fs=fs)
ax2.set_title("Espectro de frequência do sinal filtrado, dizimado e com taxa␣
 ↪aumentada")

fig.subplots_adjust(hspace=0.5)
plt.show()

Audio(upsampled_audio, rate=fs)
```

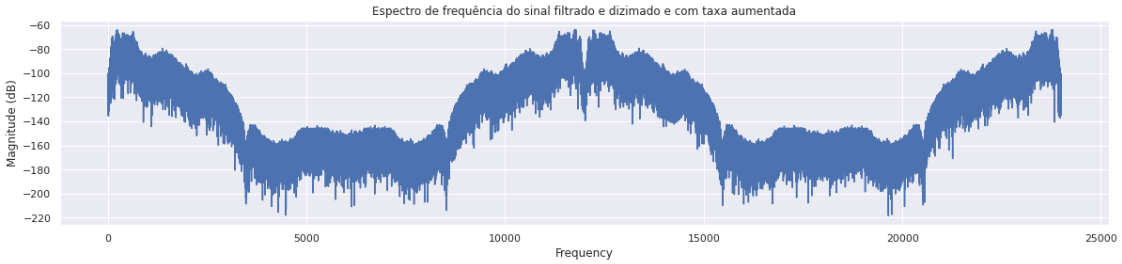


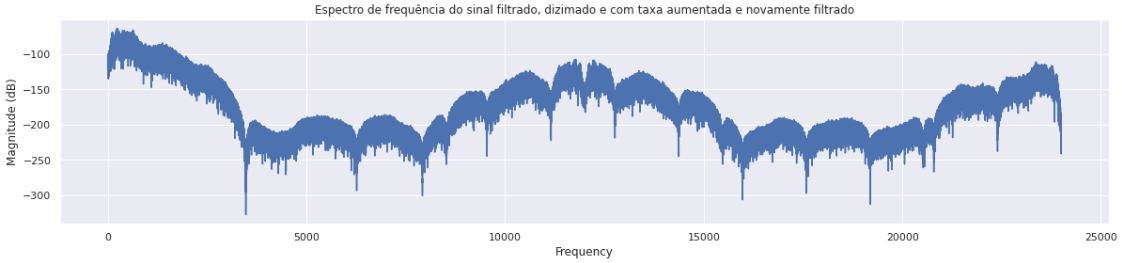


[7]: <IPython.lib.display.Audio object>

0.5 Filtro low-pass final

[8]:
```
low_pass_final = sps.lfilter(low_pass_FIR, 1, upsampled_audio)

# Por fim, é realizado o plot do espectro do espectro de frequência do áudio␣
 ↪filtrado
fig, (ax1, ax2) = plt.subplots(2, figsize=(20, 10))

ax1.magnitude_spectrum(upsampled_audio, scale='dB', Fs=fs)
ax1.set_title("Espectro de frequência do sinal filtrado e dizimado e com taxa␣
 ↪aumentada")

ax2.magnitude_spectrum(low_pass_final, scale='dB', Fs=fs)
ax2.set_title("Espectro de frequência do sinal filtrado, dizimado e com taxa␣
 ↪aumentada e novamente filtrado")

fig.subplots_adjust(hspace=0.5)
plt.show()

Audio(low_passed, rate=fs)
```





[8]:
```
<IPython.lib.display.Audio object>
```

Novamente é possível ver o comportamento, ainda sim que simples e esperado, do filtro passa-baixas, atenuando as frequências de mais alta frequência até -100dB.

1 Filtragem por estrutura polifásica

[9]:
```
# Nessa estrutura, ocorre primeiro uma dizimação e depois a filtragem␣
 ↪passa-baixas.
# Possivelmente implementável em threads, ou seja, concorrente e mais rápida.

# Filter bank
p0 = low_pass_FIR[0::2]
p1 = low_pass_FIR[1::2]

# Polyphase signals

x0 = audio_normalised[0::2]
x1 = audio_normalised[1::2]

poly_filtered = sps.lfilter(p0, 1, x0) + sps.lfilter(p1, 1, x1)
```

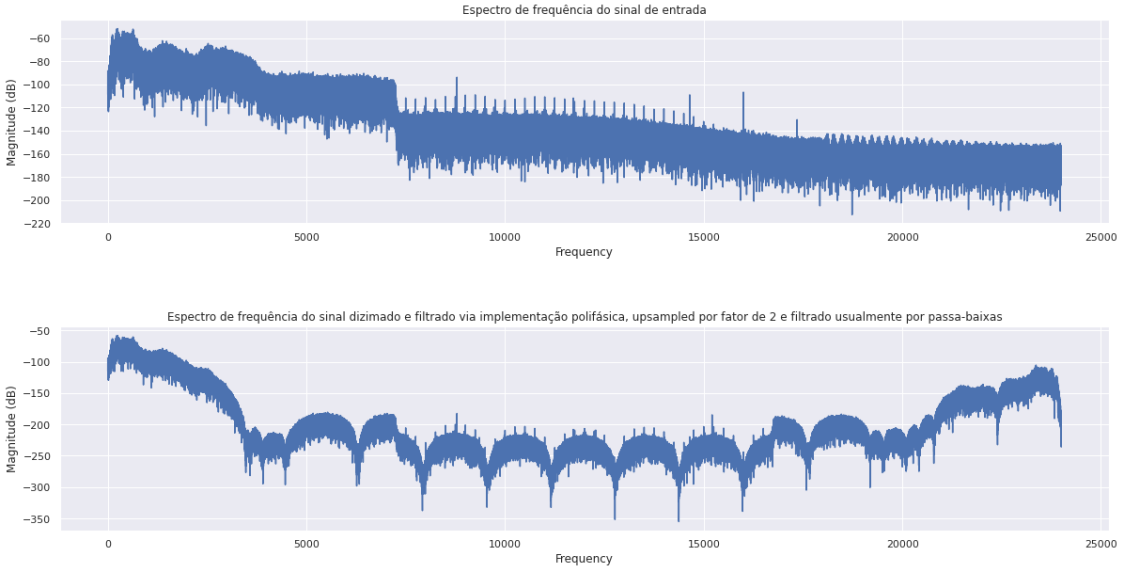
```
15
832000
```

2 Aumento de taxa usual

[16]:
```
upsampled_audio = [0]*(2*(poly_filtered.size)-1)
upsampled_audio[::2] = poly_filtered
```

3 Filtragem passa-baixas usual

[20]:
```
low_pass_final = sps.lfilter(low_pass_FIR, 1, upsampled_audio)

# Por fim, é realizado o plot do espectro do espectro de frequência do aúdio␣
 ↪filtrado
fig, (ax1, ax2) = plt.subplots(2, figsize=(20, 10))

ax1.magnitude_spectrum(audio_normalised, scale='dB', Fs=fs)
ax1.set_title("Espectro de frequência do sinal de entrada")

ax2.magnitude_spectrum(low_pass_final, scale='dB', Fs=fs)
ax2.set_title("Espectro de frequência do sinal dizimado e filtrado via␣
 ↪implementação polifásica, \
upsampled por fator de 2 e filtrado usualmente por passa-baixas")

fig.subplots_adjust(hspace=0.5)
plt.show()

Audio(low_pass_final, rate=fs)
```



[20]: <IPython.lib.display.Audio object>