UNIVERSIDADE FEDERAL DO SUL E SUDESTE DO PARÁ
FACULDADE DE COMPUTAÇÃO E ENGENHARIA ELÉTRICA
ENGENHARIA DA COMPUTAÇÃO

TAREFA DE RECUPERAÇÃO DE PROCESSAMENTO DIGITAL DE SINAIS
CRIAÇÃO DE FILTROS FIR E IIR EM PYTHON

Luana Batista Araújo

Marabá
2022

UNIVERSIDADE FEDERAL DO SUL E SUDESTE DO PARÁ
INSTITUTO DE GEOCIÊNCIAS E ENGENHARIAS
FACULDADE DE COMPUTAÇÃO E ENGENHARIA ELÉTRICA

LUANA BATISTA ARAÚJO

TAREFA DE RECUPERAÇÃO DE PROCESSAMENTO DIGITAL DE SINAIS
CRIAÇÃO DE FILTROS FIR E IIR EM PYTHON

Relatório referente a TAREFA DE RECUPERAÇÃO "RELATÓRIO ACERCA DO DESENVOLVIMENTO DE FILTROS FIR E IIR EM PYTHON" como critério de avaliação da disciplina PROCESSAMENTO DIGITAIS DE SINAIS da Prof. Dra. Leslye Eras.

Marabá
2022

1. INTRODUÇÃO

É fato que os sinais, por mais que imperceptíveis na maioria das vezes, estão presentes no cotidiano de todos os seres humanos em todos os seus formatos e dimensões. Dado este fato, viu-se a necessidade de aprofundar pela prática os conteúdos estudados em sala. Desta forma, este relatório trata de registros da execução de algumas atividades relacionadas a filtros e outros tipos de tratamento de sinais apresentados pela professora em laboratório.

2. DESENVOLVIMENTO

2.1. Comparativo entre filtros FIR e IIR

Neste relatório foram utilizados dois tipos de filtros. Os filtros FIR e IIR, entretanto antes que uma comparação entre ambos possa ser feita, faz-se necessário uma contextualização dos mesmos. O filtro de resposta finita ao impulso ou, simplesmente, **FIR** (**F**inite **I**mpulse **R**esponse) é um tipo de filtro digital caracterizado por uma resposta ao impulso que se torna nula após um tempo finito. O filtro de resposta infinita ao impulso ou filtro **IIR** (**I**nfinite **I**mpulse **R**esponse) é um filtro digital com resposta ao impulso de duração infinita.

A principal diferença entre esses filtros começa pelo seu modo de operação. Os filtros FIR operam por convolução da resposta a impulso (kernel) com o sinal, já os filtros IIR operam de forma recursiva, calculando a saída baseado não apenas nos sinais de entrada, mas também com base nas saídas anteriores. Tem-se como outro ponto de divergência a parte de desempenho e facilidade. Os filtros FIR permitem que todos os filtros lineares possíveis possam ser implementados dessa maneira e possuem desempenho impressionante, podendo, entretanto, serem lentos dependendo do comprimento de seu kernel. Os filtros IIR têm um bom desempenho e, em relação ao seu comprimento, são mais rápidos que os filtros FIR, porém podem se tornar instáveis.

A seguir serão apresentados algumas aplicações FIR e IIR desenvolvidas utilizando a linguagem de programação Python e baseadas nas atividades feitas utilizando MATLAB/OCTAVE em sala de aula. As principais bibliotecas utilizadas foram Numpy, Matplotlib e Scipy. Após uma breve explicação acerca do seu funcionamento, será exibido um comparativo entre os resultados obtidos na aplicação Python e na aplicação MATLAB.

2.2. Análise de filtros em função da frequência

2.2.1. Aplicação: Diferença entre resolução espectral e densidade espectral

Para ilustrar a diferença entre resolução espectral e densidade foi considerado o sinal x[n]=cos(0.48πn)+cos(0.52πn). Após isso foi calculada a DTF do sinal considerando os seguintes casos:

- Um número de amostras N=10 preenchendo as outras 90 amostras com 0 chegando a N=100.
- Um número de amostras N=100 sem preenchimento com 0

A Imagem 1, abaixo, exibe o código desenvolvido para a aplicação em Python, a Imagem 2 o resultado obtido na aplicação feita e a Imagem 3 o resultado obtido na atividade feita em sala de aula utilizando o MATLAB.

Imagem 1 - Código da aplicação em Python

DFT_DENSIDADE_RESOLUTION.PY > ...

```python
# diferença entre resolução espectral e densidade espectral
import numpy as np
import matplotlib.pyplot as plt
from numpy import fft
n = np.arange(0, 100)
Fs = 100
x = np.cos(0.48*np.pi*n) + np.cos(0.52*np.pi*n)
## calcula DFT com N=10 e plota
n1 = np.arange(0, 10)
sinal1 = x[0:10]
Y1 = fft.fft(sinal1, 10)
magY1 = abs(Y1[0:6])
N1 = 5
m1= np.arange(0, 6)
f1= m1*Fs/N1
plt.subplot(2,3,1); plt.stem(sinal1); plt.xlabel('n'); plt.title('N=10')
plt.subplot(2,3,4); plt.stem(f1, magY1); plt.xlabel('freq(Hz)');
## calc DFT com N=10+90 zeros e subplota (melhor resol espectro)
aux = np.zeros((1,90))
sinal2 = np.concatenate((x[0:10], aux[0]))   #preencheu sinal que essencialmente eh o mesmo
Y2 = fft.fft(sinal2, 100)
magY2 = abs(Y2[0:50])
N2 = 50
m2= np.arange(1, 51)
f2= m2*Fs/N2
plt.subplot(2,3,2); plt.stem(sinal2); plt.xlabel('n'); plt.title('N=10+90 zeros com boa resol. espec.')
plt.subplot(2,3,5); plt.stem(f2, magY2); plt.xlabel('freq(Hz)');
## calc DFT com N=100
sinal3 = x[0:100]
Y3 = fft.fft(sinal3, 100)
magY3 = abs(Y3[0:50])
N3 = 50;
m3= np.arange(1, 51)
f3= m3*Fs/N3;
plt.subplot(2,3,3); plt.stem(sinal3); plt.xlabel('n'); plt.title('N=100 com boa densid. espec.')
plt.subplot(2,3,6); plt.stem(f3, magY3); plt.xlabel('freq(Hz)');
plt.show()
```

Imagem 2 - Resultado obtido da aplicação em Python

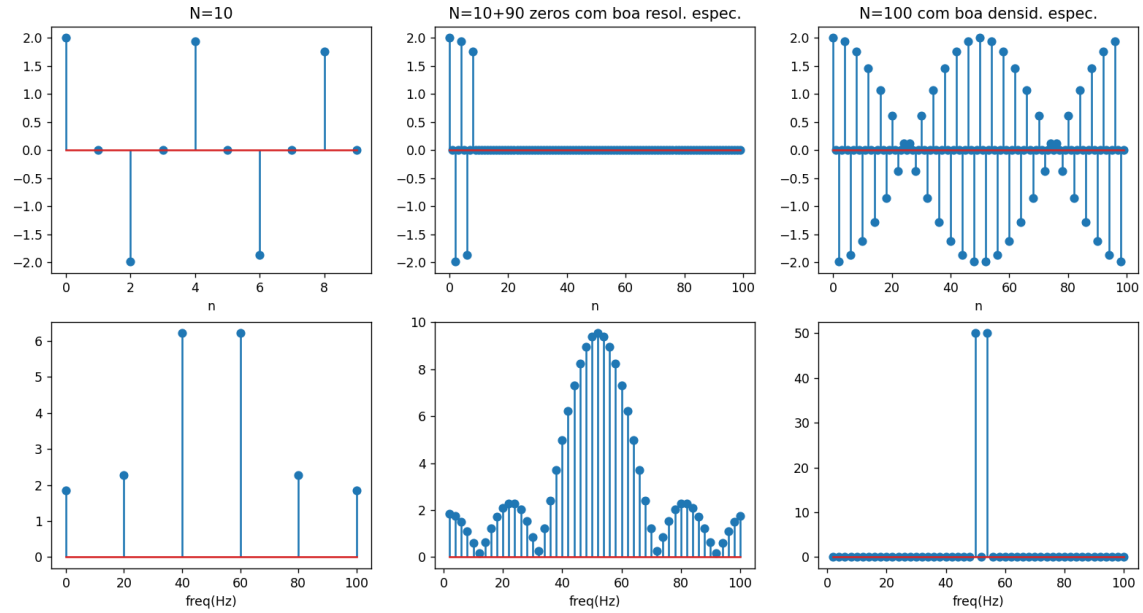
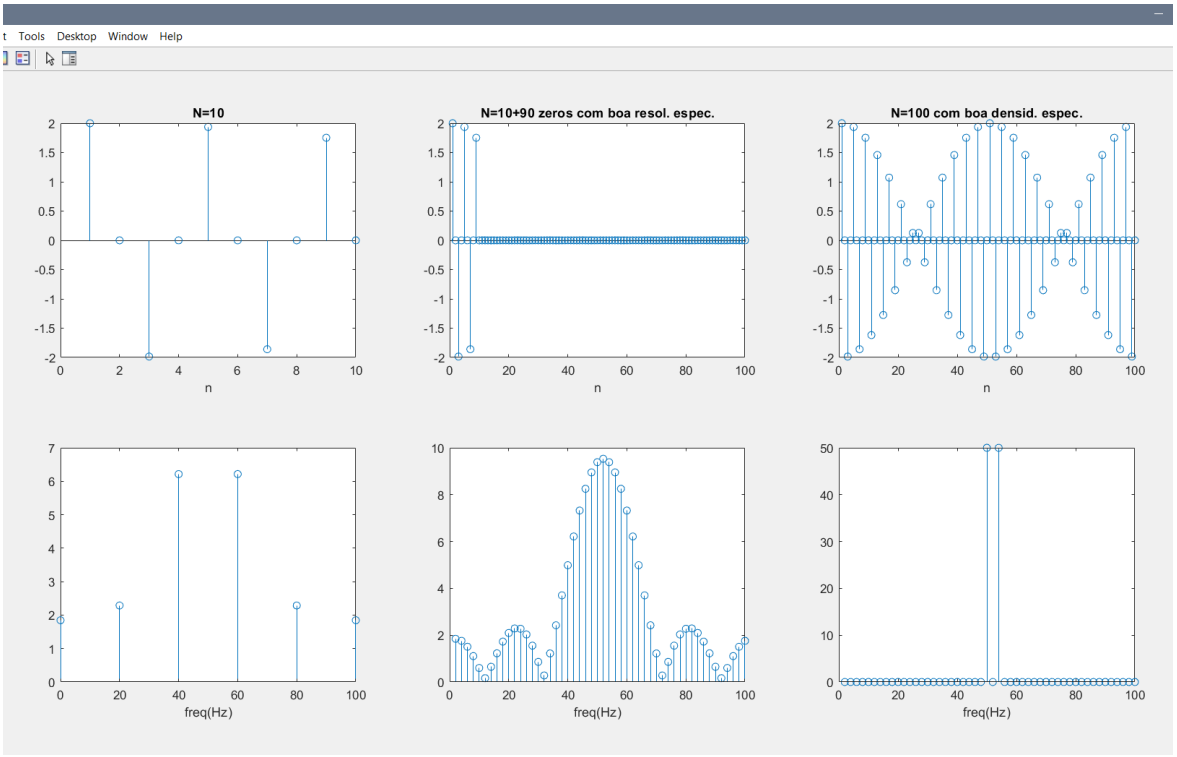


Imagem 3 - Resultado obtido utilizando o MATLAB



2.2.2. Aplicação: FFT

Apesar de a DTF poder ser calculada manualmente, não é computacionalmente eficiente pois não faz uso das propriedades de simetria de cálculo da mesma. A FFT é uma função mais eficiente para esse cálculo. Nela foi testado com e sem adição de ruído ao sinal para ver o efeito do ruído no espectro. A componente espectral de 400Hz se mostrou mais perceptível que o ruído devido ao fato que este tem pouca correlação com funções de seno e cosseno. É possível perceber também que a função logarítmica amplifica pequenas componentes e atenua grandes.

A Imagem 4, abaixo, exibe o código desenvolvido para a aplicação em Python, a Imagem 5 o resultado obtido na aplicação feita e a Imagem 6 o resultado obtido na atividade feita em sala de aula utilizando o MATLAB.

Imagem 4 - Código da aplicação em Python

DFT_USODEFFT.py > ...

```python
import numpy as np
import matplotlib.pyplot as plt
from numpy import fft

## gera um sinal "sintetico" com 3 componentes de frequencia
Fs = 1000; #taxa de aquisicao do sinal
ts = 1/Fs;
N = 1000; #numero amostras de amostras analisadas
t = (np.arange(0, N))*ts;
sinal = 0.6*np.sin(2*np.pi*50*t) + 2*np.sin(2*np.pi*120*t) + 0.3*np.sin(2*np.pi*400*t);
ruido = 0.4*np.random.randn(np.size(t));
## calcula a DFT usando a funcao fft
y = fft.fft(sinal+ruido);
y = y/N; #se desejar pode-se dividir por N para "acomodar" os calculos
## calcular o eixo das frequencias
m = np.arange(0, N)
f = m*Fs/N;
y = y[0:int(N/2)] #pegando so a primeira metade ja que a outra é copia
f = f[0:int(N/2)] #pegando so a primeira metade ja que a outra é copia
## calcular a potencia do espectro em db
magnitude = abs(y);
fase = np.angle(y);
f_ref = max(magnitude); #escolhe o maior valor para ser a referencia para normalizacao
y_db = 20*np.log10(magnitude/f_ref); #lembre que, por exemplo 0db = 1 unidade
## plotar
plt.subplot (3,1,1); plt.plot(f ,magnitude); plt.title('magnitude espectro'); plt.xlabel('freq(Hz)');
plt.ylabel('Amplitude');
plt.subplot (3,1,2); plt.plot(f ,y_db); plt.title('potencia espectro');
plt.subplot (3,1,3); plt.plot(f ,fase); plt.title('fase');
plt.show()
```

Imagem 5 - Resultado obtido da aplicação em Python

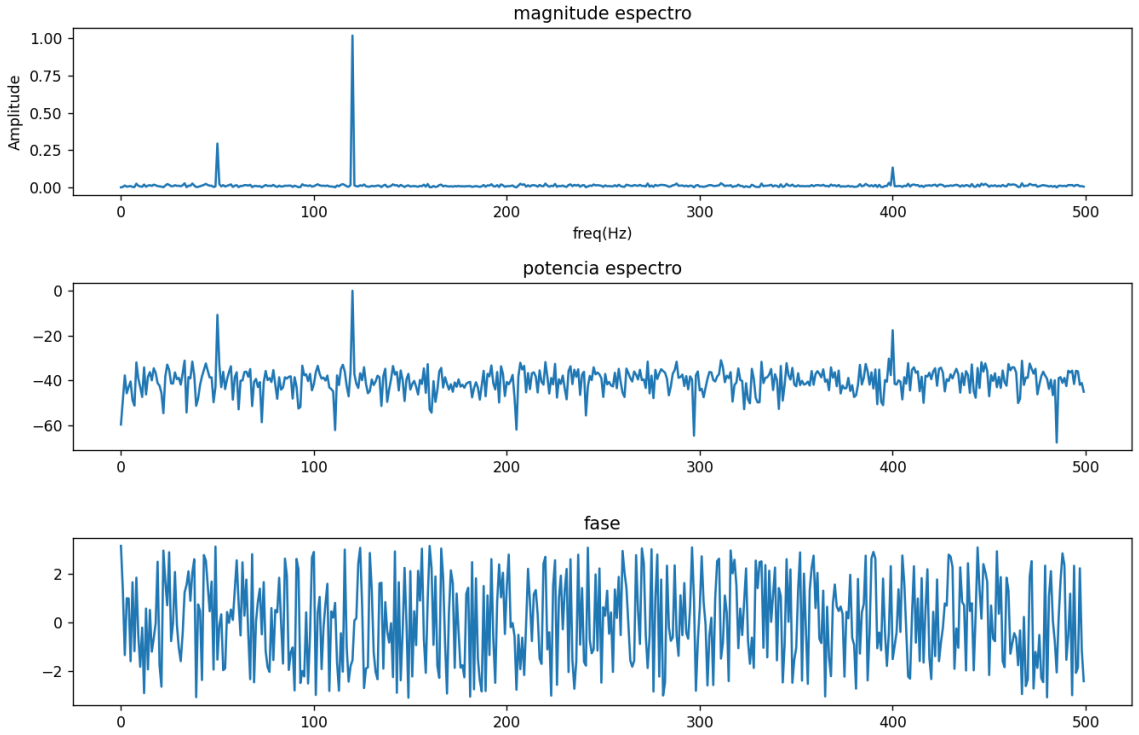
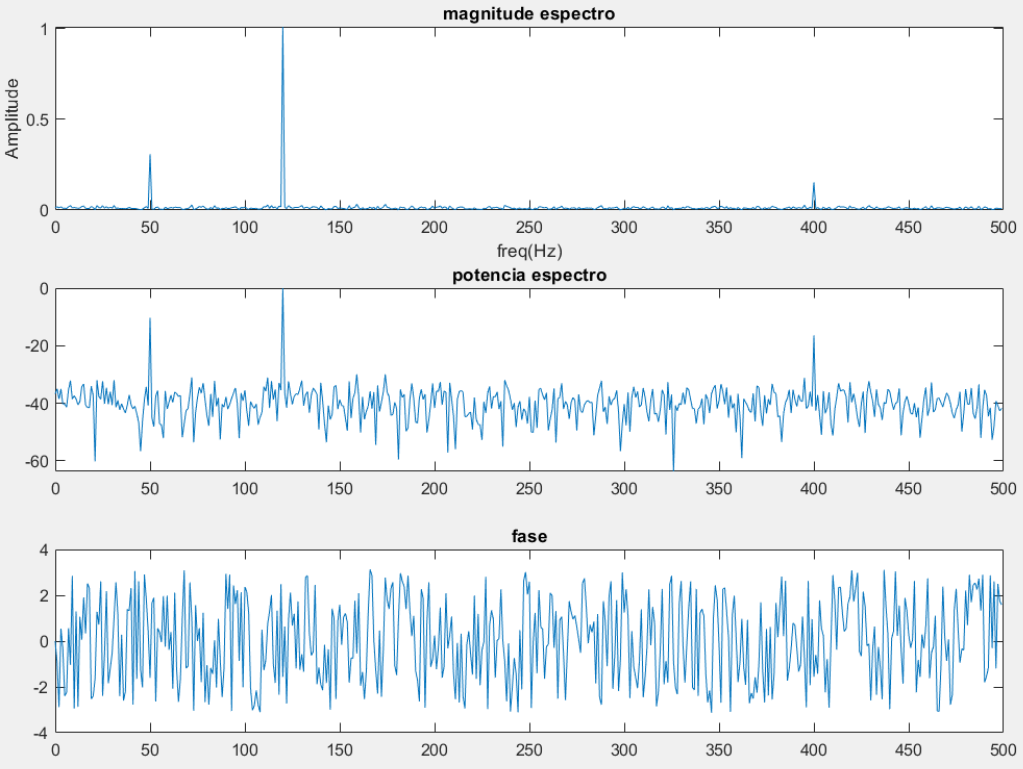


Imagem 6 - Resultado obtido utilizando o MATLAB



2.2.3. Aplicação: Implementação de filtros FIR - Projeção de filtro pelo método de janelas

Para implementar um filtro FIR é necessário ter conhecimento dos coeficientes do filtro. Nesta etapa foram utilizados os coeficientes já dados e foi considerado que o projeto já estava pronto e o único objetivo era implementar o filtro. A execução é feita por uma convolução da entrada com a resposta impulsiva do filtro.

Para este método foi primeiramente especificado as características do filtro. Depois foi especificado o espectro ideal para o filtro e, a partir disso, foi encontrado quais devem ser o coeficientes ideias do filtro para a implementação desse espectro idealizado. Foi, então, gerado um gráfico de coeficientes simétricos em relação ao eixo vertical e a metade que não pode ser estimada foi preenchida com espelhamento. Após isso, existe a opção de usar uma janela aos coeficientes para aumentar o ripple da banda passante. Por final, foi gerado um sinal sintético com 4 componentes de frequência Para testar o filtro e foram calculados os espectros do sinal sintético de entrada, o espectro da resposta impulsiva do filtro e espectro do sinal de saída.

Nas Imagens 7, 8 e 9 a seguir é mostrado o código feito para a aplicação em Python, já nas Imagens 10 e 11 pode-se observar os resultados obtidos e nas Imagens 12 e 13 os resultados da aplicação feita no MATLAB.

Imagem 7 - Código da aplicação em Python (Parte 1)

```python
#Projeto de um filtro FIR passa-baixas, passa-altas e faixas
import numpy as np
import matplotlib.pyplot as plt
from numpy import fft

def ctranspose(arr: np.ndarray) -> np.ndarray:
    tmp = arr.transpose()
    return tmp - 2j*tmp.imag

## PARTE 1: dados do filtro
Fs = 20000; # taxa de amostragem
fc = 2000; # frequencia de corte do filtro
N = 64; # Quantidade total de coeficientes filtro
Num_Coef_Filtro = 64; # Qtos coeficientes vou usar do total
tipo_filtro = input('\nTipo filtro(1=passa-baixas; 2=passa-altas; 3=passa-faixa) = ');

## PARTE 2: gera o espectro ideal do filtro
aux = np.zeros((1,N))
H = aux[0]; # resposta em freq ideal do filtro
f_resol = Fs/N; # calcula resolução frequencia
m_corte = round(fc/f_resol)+1; # estima indice "m" de fc

H = np.concatenate((np.ones((1, m_corte + 1))[0], np.zeros((1, int(N/2-m_corte)))[0],
np.zeros((1, int(N/2-m_corte)))[0], np.ones((1, m_corte- 1))[0])); # gera espectro idea
m = np.arange(0, N)
f = m*f_resol; # define o eixo de frequencias do grafico 1

## PARTE 3: gera todos coef. ideais do filtro
h_ideal = fft.ifft(H);
if (tipo_filtro==2): # se passa-alta, multiplicar coef por [1,-1,1,-1,...]
    n = np.arange(0, N)
    deslocamento_f = np.cos(2*np.pi*n*(Fs/2)/Fs);
    h_ideal = h_ideal*deslocamento_f;
```

Imagem 8 - Código da aplicação em Python (Parte 2)

```
if (tipo_filtro==3): # se passa-faixa, multiplicar coef por modulacao
    n = np.arange(0, N)
    fcc = input('\nDigite em Hz frequência central =');
    deslocamento_f = np.cos(2*np.pi*fcc*n/Fs);
    h_ideal = h_ideal*deslocamento_f;

## PARTE 4: GERA SIMETRIA PAR NA FUNCAO SYNC DOS COEFICIENTES
h_ideal[int((N/2)-1):N] = h_ideal[0:int((N/2)+1)];
for i in range(2,int(N/2+1)):
    h_ideal[i-1]= h_ideal[N-i+1]

inicio = int(N/2 - Num_Coef_Filtro/2 + 1);
fim = int(N/2 + Num_Coef_Filtro/2)

h = (h_ideal[inicio-1:fim]).real; # pega so parte dos coeficientes ideias do filtr

## PARTE 5 (opcional): aplica janela
resposta = input('\nDeseja aplicar janela aos coef (1=sim; 2=nao)? = ')
if (resposta == 1):
    #jan = window(@blackman,(fim-inicio)+1)
    jan = np.blackman((fim-inicio)+1)
    h = ctranspose(h*jan)

## PARTE 6: testa a implementação filtro com sinal sintetico
N_sinal_sintetico = 400; # quantidade de amostras do sinal sintetico
n1 = np.arange(0, N_sinal_sintetico)
entrada_discretizada = np.sin(2*np.pi*500.*n1/Fs) + np.sin(2*np.pi*2500.*n1/Fs) +
np.sin(2*np.pi*5000.*n1/Fs) + np.sin(2*np.pi*8000.*n1/Fs);
saida = np.convolve(entrada_discretizada, h);
```

Imagem 9 - Código da aplicação em Python (Parte 3)

```
N_sinal_saida = np.size(saida); #N_sinal_saida = size(saida,2);
N_resp_impulsitva = Num_Coef_Filtro;

## PARTE 7: calcula espectros sinal entrada, saida e h(n)
fft_sinal_entrada = fft.fft(entrada_discretizada)/N_sinal_sintetico;
fft_sinal_saida = fft.fft(saida)/N_sinal_saida;
fft_resp_filtro = fft.fft(h);
f_entrada = n1*(Fs/N_sinal_sintetico);
n3 = np.arange(0, np.size(fft_sinal_saida))
f_saida = n3*(Fs/N_sinal_saida)
n2 = np.arange(0, np.size(fft_resp_filtro))
f_h = n2*(Fs/N_resp_impulsitva);

## PARTE 8: plota
plt.subplot(2,2,1); plt.stem(f,H, label='Todos'); plt.title('H(f) idealizado'); plt.xlabel('f(Hz)')
plt.subplot(2,2,2); plt.stem((h_ideal).real); plt.xlabel('n');
plt.title('Coeficientes h(n) do filtro'); plt.ylabel('Amplitude'); plt.xlabel('n')
plt.legend()

plt.subplot(2,2,3); plt.plot(f_entrada[0:int(N_sinal_sintetico/2)],
abs(fft_sinal_entrada[0:int(N_sinal_sintetico/2)]), label='entrada');
plt.plot(f_saida[0:int(N_sinal_saida/2)],abs(fft_sinal_saida[0:int(N_sinal_saida/2)]),'r', label='saída');
plt.plot(f_h[0:int(N_resp_impulsitva/2)],abs(fft_resp_filtro[0:int(N_resp_impulsitva/2)]),'g',
label='resp. impulsiva');
plt.legend();

plt.xlabel('Freq (Hz)');
plt.title('Espectros dos sinais e filtro')
plt.subplot(2,2,4); plt.plot(n1, entrada_discretizada, label='entrada');
plt.plot(n3, saida,'r', label='saída');
plt.legend();
plt.xlabel('n')
plt.title('Sinais discretos do filtro');
plt.show()
```

Imagem 10 - Resultado obtido da aplicação em Python (Parte 1)

```
Tipo filtro(1=passa-baixas; 2=passa-altas; 3=passa-faixa) = 1

Deseja aplicar janela aos coef (1=sim; 2=nao)? = 2
```

Imagem 11 - Resultado obtido da aplicação em Python (Parte 2)

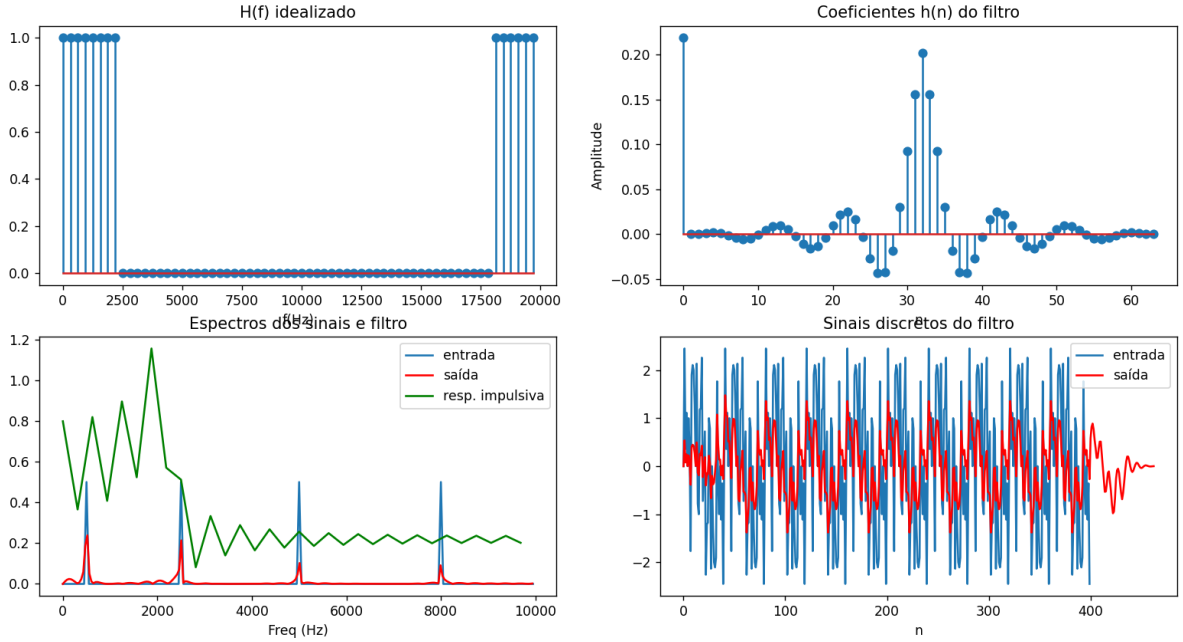
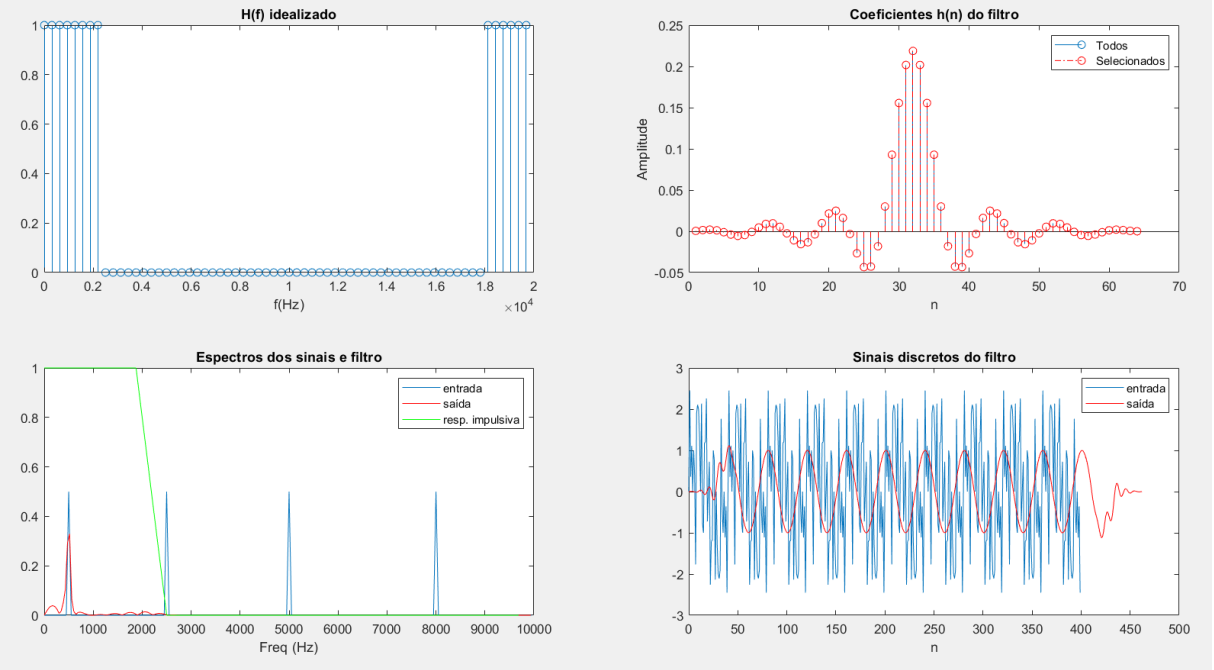


Imagem 12 - Resultado obtido utilizando o MATLAB (Parte 1)

```
Tipo filtro(1=passa-baixas; 2=passa-altas; 3=passa-faixa) =1

Deseja aplicar janela aos coef (1=sim; 2=nao)? = 2
```

Imagem 13 - Resultado obtido utilizando o MATLAB (Parte 2)



2.2.4. Aplicação: Implementação de filtros FIR - Projeção de filtro pelo método de Parks-McClellan

Esta é uma técnica muito otimizada para se encontrar coeficientes, porém seu algoritmo é muito complexo. Por esse motivo foram considerados apenas os principais parâmetros de um filtro, especificando-os para determinar os coeficientes.

A Imagem 14, abaixo, exibe o código desenvolvido para a aplicação em Python, a Imagem 15 o resultado obtido na aplicação feita e a Imagem 16 o resultado obtido na atividade feita em sala de aula utilizando o MATLAB.

Imagem 14 - Código da aplicação em Python

FIR_PARKS.py > ...

```python
import numpy as np
import matplotlib.pyplot as plt
from numpy import fft
from scipy import signal as sigs

# Ilustração do uso da função firpm que executa o algoritmo
# coeficientes para filtro passa-faixa:
#  f = [0 0.4 0.44 0.56 0.6 1];
#  a = [0 0 1 1 0 0];
# coeficientes para filtro passa-baixas:
f = np.array([0, 0.4, 0.5, 1]);
m = np.array([1, 0]);
coef = sigs.remez(33,bands=f,desired=m, fs=2);
w, h = sigs.freqz(coef);
angles = np.unwrap(np.angle(h))

plt.subplot(2,1,1); plt.plot(w, 20 * np.log10(abs(h)), 'b')
plt.subplot(2,1,2); plt.plot(w, angles*20, 'g')
plt.show()
```

Imagem 15 - Resultado obtido da aplicação em Python

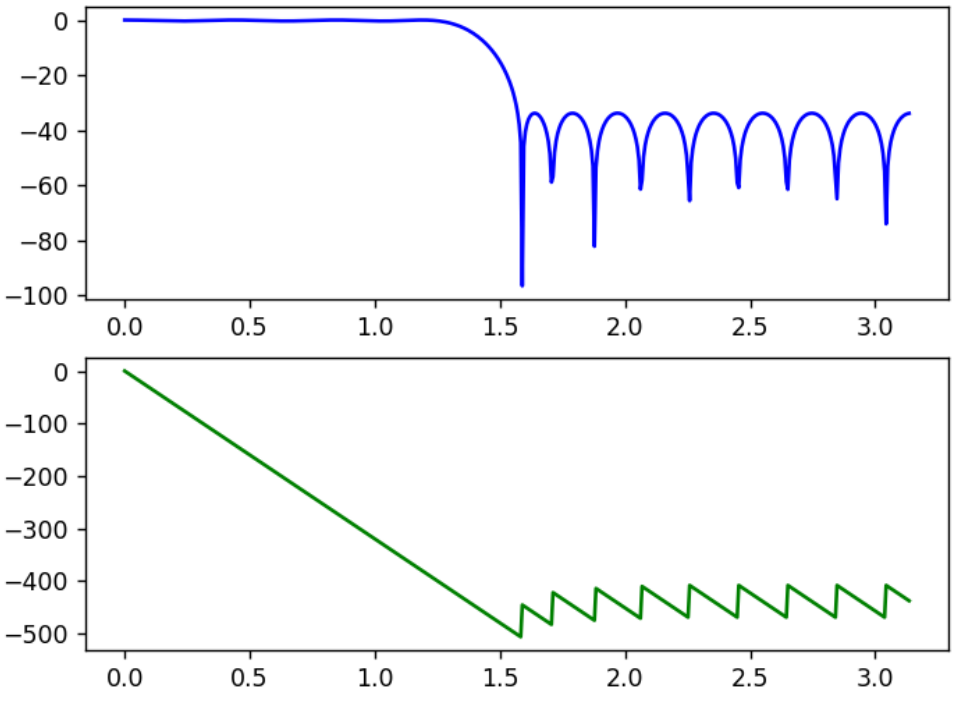
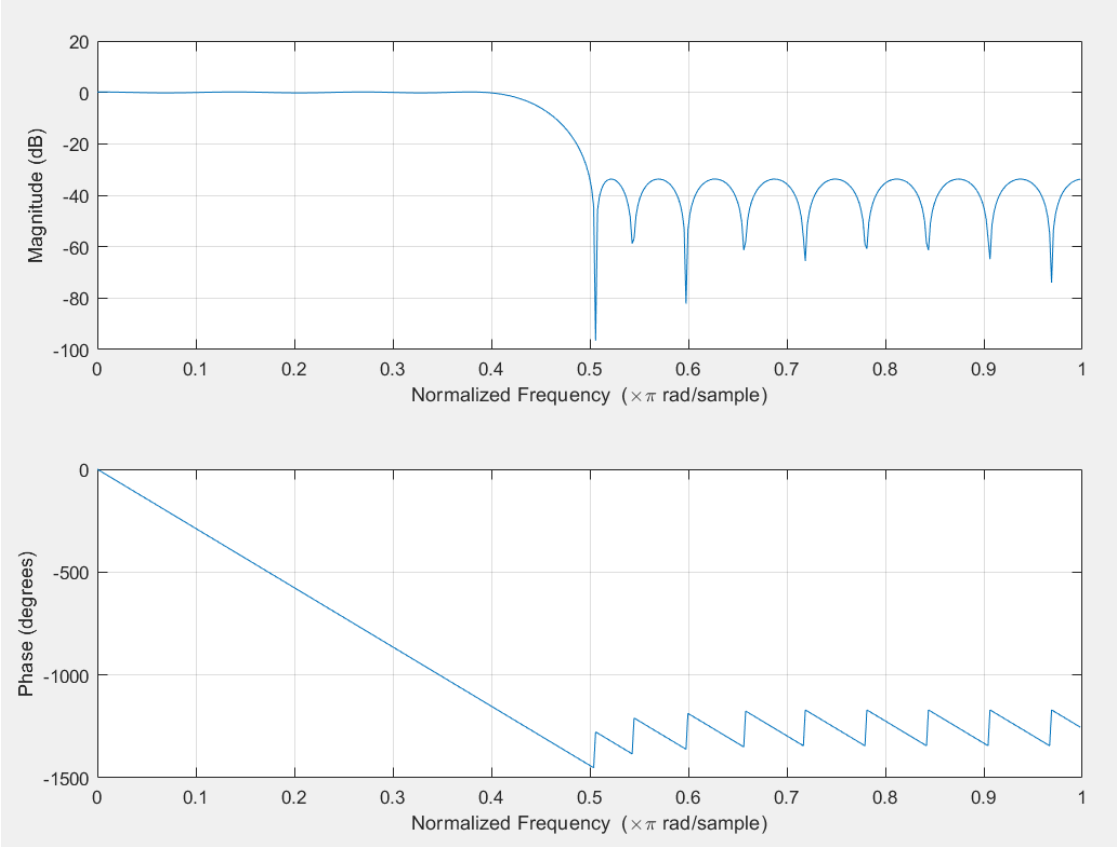


Imagem 16 - Resultado obtido utilizando o MATLAB



2.2.5. Aplicação: Implementação de filtros IIR - Filtro passa-faixas

Existem diversas formas de se criar um filtro as principais sendo especificando a frequência de corte e ordem do filtro ou especificando a banda de transição, ganho na banda de passagem e atenuação na banda de rejeição. No primeiro caso se projeta um filtro com frequência corte 0.5 (equivale a Nyquest) de ordem 3, a utilização da função também é feita e há a presença do termo ‘low’ indicando ser um filtro passa-baixas e o termo "s" indica que o filtro é feito no domínio do analógico de Laplace.

Levando em conta essas informações a aplicação foi desenvolvida, na Imagem 17 a seguir é possível observar o código feito em Python, na Imagem 18 o resultado obtido ao executar a aplicação e na Imagem 19 o resultado da aplicação no MATLAB.

Imagem 17 - Código da aplicação em Python

IIR_filtrobutter1.py > ...

```python
import numpy as np
import matplotlib.pyplot as plt
from numpy import fft
from scipy import signal as sigs

fs=8000;
wp1= 500;
ws1= 750;

#normaliza em funcao da freq Nysquest = Fs/2.
wp1= wp1/(fs/2);
ws1= ws1/(fs/2);
atenuacao_filtro=20;

[ord, wn]=sigs.buttord(wp1,ws1, 1, atenuacao_filtro);
[num, dem]= sigs.butter(ord,wn,btype='low');
[freq3, H] = sigs.freqz(num,dem,worN=512, fs=fs);

print(H)
plt.plot(freq3,abs(H))
plt.title('Magnitude filtro');
plt.xlabel('Hz');
plt.ylabel('graus');
plt.show()
```

Imagem 18 - Resultado obtido da aplicação em Python

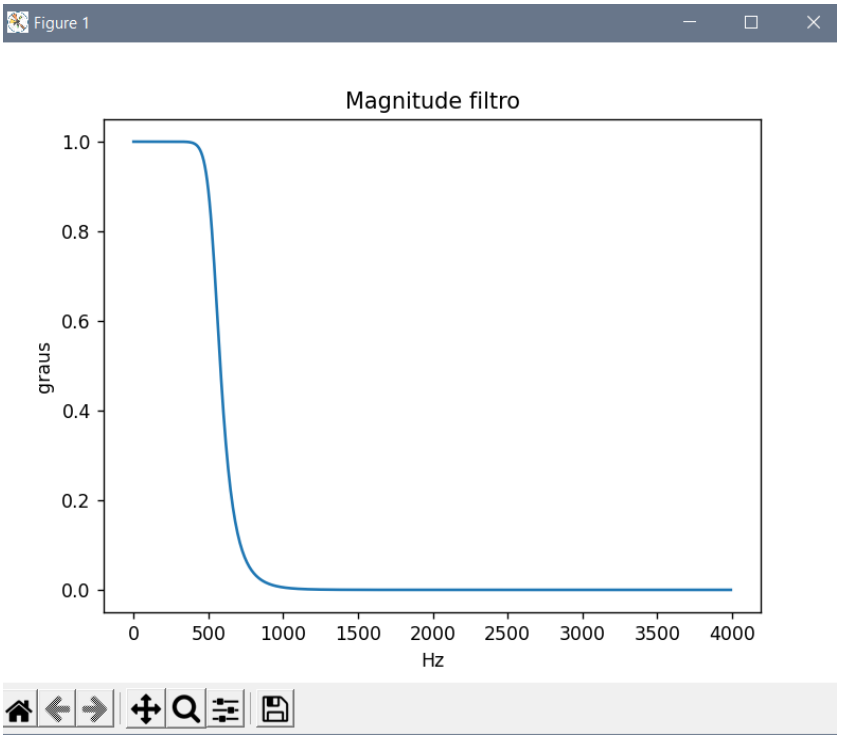
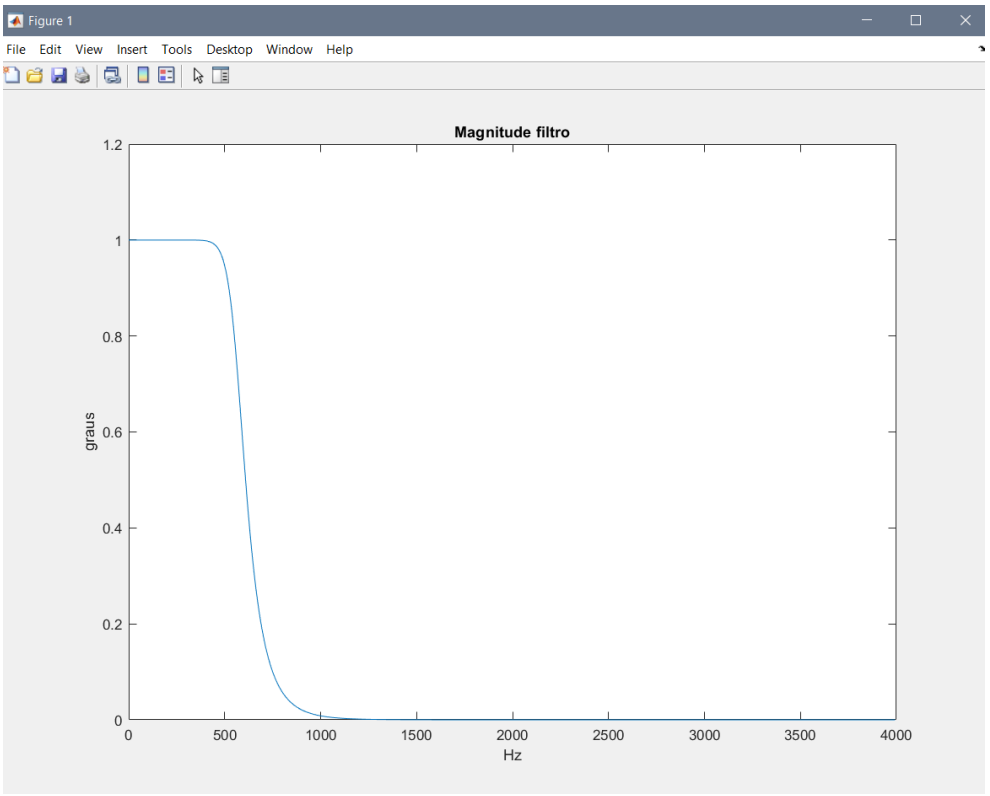


Imagem 19 - Resultado obtido utilizando o MATLAB



3. CONCLUSÃO

Através dos resultados obtidos conclui-se que foi possível realizar a criação de aplicações de filtros de forma eficiente utilizando a linguagem Python e suas bibliotecas. Além disso, a atividade avaliativa desenvolvida foi de grande valia para aquisição de conhecimentos práticos com base na parte tanto teórica quanto prática aplicada pelo docente da disciplina em sala de aula. Ademais, ao trabalhar com bibliotecas diferentes, o trabalho também permitiu aprofundar os conhecimentos em Python, ampliando a experiência na linguagem que é de fundamental importância na área da computação.

REFERÊNCIAS

1. SciPy. SciPy User Guide. Disponível em: <https://docs.scipy.org/doc/scipy/tutorial/index.html>. Acesso em: 22 de jun. de 2022.
2. MathWorks. Documentation Home . Disponível em: <https://www.mathworks.com/help/index.html?s_tid=CRUX_lftnav>. Acesso em: 22 de jun. de 2022.
3. MATHESAURUS. NumPy for MATLAB users. Disponível em: <http://mathesaurus.sourceforge.net/matlab-numpy.html>. Acesso em: 22 de jun. de 2022.
4. YNOGUTI, Carlos. INATEL - Instituto Nacional de Telecomunicações. Introdução aos Filtros Digitais. Disponível em: <https://www.inatel.br/docentes/ynoguti/graduacao-sp-2113502489/45-introducao-aos-filtros-digitais>. Acesso em: 22 de jun. de 2022.