Processamento Digital de Sinais
Tutorial 13 - Filtros de Resposta ao Impulso Infinita (IIR)

https://github.com/fchirono/AulasDSP

Objetivos de Aprendizagem

Ao final desta sessão, você será capaz de:

- Projetar e aplicar filtros IIR simples;
- Projetar um filtro baseado em um diagrama de blocos.

Sumário

1 Removendo uma Perturbação de um Sinal de Áudio

Objetivo: Remover um ruído tonal irritante em 4000 Hz de um sinal de música

1. Repita a filtragem da Tarefa 1 do Tutorial 6 usando filtros IIR (Butterworth e Chebyshev). Compare e discuta o desempenho destes com o dos filtros FIR. Compare as faixas de passagem e de rejeição, nível de atenuação, e complexidade computacional.

2 Projeto de Filtro Baseado em Diagrama de Blocos

A partir do diagrama de blocos mostrado na Figura 1:

1. Derive a equação de diferenças;
2. Calcule as primeiras 3 amostras da Resposta ao Impulso;
3. Calcule a Função de Transferência do sistema no domínio Z;

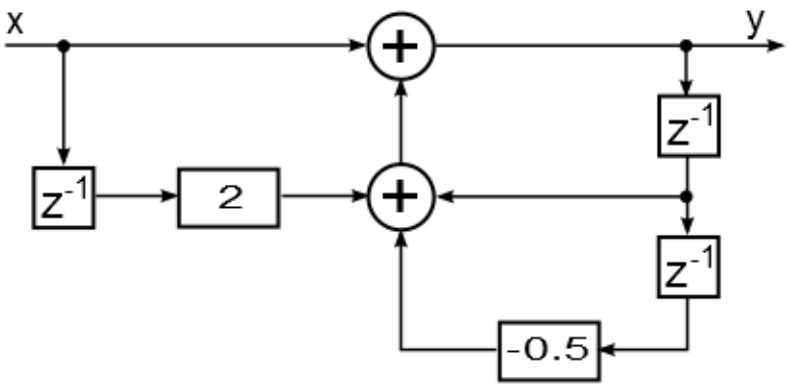

Figura 1: Diagrama de Blocos do Sistema.

4. Calcule a Resposta em Frequência;
5. Compare seus resultados calculados a partir de princípios básicos com a Resposta ao Impulso calculada pela função Python `scipy.signal.dimpulse` e a Resposta em Frequência de `scipy.signal.freqz`;
6. Plote o diagrama polo-zero (você pode usar a função `scipy.signal.tf2zpk`) e considere como a posição dos polos e zeros se reflete na resposta em frequência;
7. Aplique este filtro à amostra de música `BetterDaysAhead.wav` e ouça o resultado.

3 Apêndice: Projeto de Filtros IIR em Python

Para o projeto de filtros IIR, usaremos a biblioteca `scipy.signal`. Um filtro pode ser representado em SciPy/Python de algumas formas diferentes: usando seus coeficientes de função de transferência ("`ba`"), usando sua representação zero-polo-ganho ("`zpk`"), ou usando seções de segunda ordem em cascata ("`sos`"). Algumas formas são mais apropriadas para algumas aplicações do que outras, mas discutiremos como usar apenas a representação "`ba`" para este Tutorial.

3.1 Coeficientes da Função de Transferência do Filtro

Os coeficientes da função de transferência `(b, a)` de um filtro passa-baixas podem ser obtidos usando o seguinte código:

```
from scipy import signal

# Filtro Butterworth
b_butter, a_butter = signal.butter(N_filter, wn, btype='lowpass')

# Filtro Chebyshev Tipo 1
b_cheby1, a_cheby1 = signal.cheby1(N_filter, r1, wn, btype='lowpass')

# Filtro Chebyshev Tipo 2
b_cheby2, a_cheby2 = signal.cheby2(N_filter, r2, wn, btype='lowpass')
```

onde

- `(b, a)` são os vetores contendo os coeficientes do numerador e denominador da função de transferência do filtro, escritos em potências crescentes de z^{-1} (i.e. o numerador é escrito como $b[0] + b[1] \cdot z^{-1} + b[2] \cdot z^{-2} + \ldots$). No SciPy é necessário que os vetores `b` e `a` tenham o

mesmo comprimento para que os resultados sejam consistentes, então considere adicionar zeros a qualquer um deles se necessário;

- `N_filter` é a ordem do filtro;
- `wn` é a frequência de corte do filtro normalizada para *metade* da frequência de amostragem: para o filtro Butterworth, este é o ponto onde a resposta do filtro cai para -3 dB; para os filtros Chebyshev, este é o ponto onde a resposta atinge pela primeira vez `-r` dB;
- `r` é a oscilação máxima (em decibéis) permitida nos filtros Chebyshev: para filtros Tipo 1, `r1` define a oscilação máxima abaixo do ganho unitário na faixa de passagem, e para filtros Tipo 2, `r2` define a atenuação mínima na faixa de rejeição;
- e `btype` define o tipo da resposta desejada do filtro: a função aceita as strings `'lowpass'` para obter um filtro passa-baixas, e `'highpass'` para obter um filtro passa-altas.

O tipo de representação do filtro é controlado pelo argumento "`output`" na chamada da função; este é definido como "`ba`" por padrão, então não precisa ser explicitamente definido no nosso caso.

3.2 Diagrama Polo-Zero

Os zeros do filtro podem ser obtidos das raízes dos coeficientes do numerador `b`, enquanto os polos do filtro podem ser obtidos das raízes dos coeficientes do denominador `a`. A função `numpy.roots` pode ser usada para encontrar as raízes dos polinômios a partir de um vetor contendo seus coeficientes. O diagrama polo-zero é então construído plotando a parte real versus a parte imaginária dos polos e zeros; podemos plotar os zeros como círculos e os polos como "x" em tais diagramas.

Para tornar o diagrama mais visualmente atraente, pode-se também adicionar um círculo de raio unitário usando "artists" do Matplotlib[1] e adicionar linhas verticais e horizontais marcando $Re(x) = 0$ e $Im(x) = 0$:

```
import matplotlib.pyplot as plt

# cria uma figura
fig_pz = plt.figure()
ax_pz = fig_pz.add_subplot(111)

# cria um 'plt.Circle' artist para representar um circulo de raio unitario
origin = (0, 0)
radius = 1
unit_circle = plt.Circle(origin, radius, facecolor="none", edgecolor="k",
                         linestyle="dashdot")

# adiciona o artist a figura
ax_pz.add_artist(unit_circle)

# adiciona linha horizontal do eixo (y=0)
y_hline = 0
x_min = -1.5
x_max = 1.5
ax_pz.hlines(y_hline, x_min, x_max, linestyle="dashdot")

# adiciona linha vertical do eixo (x=0)
```

[1] "Artists" representam objetos gráficos padrão que queremos plotar; pense neles como funções de alto nível para desenhar formas geométricas simples. Veja `http://matplotlib.org/users/artists.html` para mais informações sobre Matplotlib Artists.

```
x_vline = 0
y_min = -1.5
y_max = 1.5
ax_pz.vlines(x_vline, y_min, y_max, linestyle="dashdot")

# faz os eixos x e y terem razao de aspecto igual
plt.axis("equal")
```

Note que a chamada da função artist `plt.Circle` requer a letra "`C`" maiúscula. Note também que é possível subdividir uma lista de argumentos para uma função em múltiplas linhas, se os argumentos estiverem envolvidos em parênteses.

3.3 Resposta em Frequência do Filtro

Uma vez que temos os coeficientes do filtro, podemos obter a resposta em frequência do filtro em `N_freq` pontos sobre o vetor de frequência normalizada `w` $\in [0, \pi)$ usando a função `scipy.signal.freqz`:

```
# calcular a resposta em freq dos filtros em 'N_freq' pontos sobre as
   frequencias 'w'
w, H_butter = signal.freqz(b_butter, a_butter, N_freq)
```

3.4 Resposta ao Impulso do Filtro

Também podemos obter as primeiras `N_IR` amostras da resposta ao impulso do filtro usando a função `scipy.signal.dimpulse`:

```
# calcular a RI do filtro em 'N_IR' pontos com resolucao temporal 'dt'
butter_tupla = (b_butter, a_butter, dt)
t_IR, IR_tupla = signal.dimpulse(butter_tupla, n=N_IR)

# obter o 1o elemento da tupla de RI
IR_butter = IR_tupla[0]
```

Esta função retorna uma tupla `IR_tuple` contendo as respostas ao impulso correspondentes a cada uma das entradas do filtro. Como lidaremos apenas com filtros de entrada única para este curso, precisamos usar apenas o primeiro elemento da tupla.

3.5 Aplicando um Sinal a um Filtro

Finalmente, um sinal `x` pode ser processado com um filtro definido por seus coeficientes de função de transferência (`b`, `a`) usando a função `scipy.signal.lfilter`:

```
# filtrar sinal 'x' com coeficientes de filtro 'b' e 'a'
x_filtered = signal.lfilter(b_butter, a_butter, x)
```

3.6 Observações Finais

As funcionalidades descritas neste Apêndice serão suficientes para completar o Tutorial; entretanto, essas funções são muito flexíveis e têm muitos mais parâmetros opcionais que não foram cobertos aqui. Recomenda-se ler a documentação para as funções mencionadas acima no Guia de Referência de Processamento de Sinais do SciPy[2] e se familiarizar com suas implementações.

[2] http://docs.scipy.org/doc/scipy/reference/signal.html

Todos os filtros têm limitações inerentes em termos de desempenho alcançável para uma dada ordem, e é frequentemente possível encontrar problemas ao usar ordens altas ou ao adotar frequências de corte muito próximas de 0 ou $f_s/2$; portanto, é uma boa prática verificar os resultados do projeto do seu filtro[3]. Você pode fazer isso analisando a função de transferência, a resposta ao impulso e o diagrama polo-zero do seu filtro recém-projetado, e confirmando que eles são razoáveis: projetos de filtros práticos devem ser estáveis, o que pode ser verificado através do diagrama polo-zero e através do gráfico da resposta ao impulso, e sua resposta em frequência deve corresponder à resposta desejada.

Se um comportamento de filtro desejado não for estável, pode-se reduzir a ordem do filtro, mas ao custo de desempenho um pouco pior. Se for imperativo usar filtros de ordem mais alta, implementações de seções de segunda ordem em cascata ('`sos`', também conhecidas como "*filtros biquad digitais*") são numericamente mais estáveis e podem ser uma boa alternativa.

[3]Na verdade, é uma boa prática sempre verificar *todos* os resultados!