

Digital Filtering

TD 7: Parametric Equalizers

Antonin NOVAK

Laboratoire d'Acoustique de l'Université du Maine, Le Mans, France



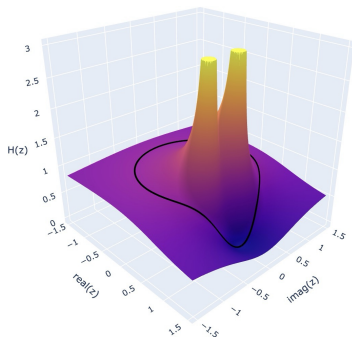
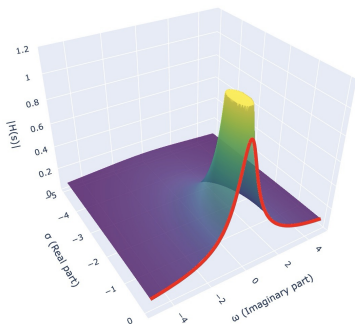


1 Bilinear transform

Audio Filters - Design

Z- and Laplace- transforms reminder

	Continuous-time	Discrete-time
Frequency Domain Transform	$H(\omega) = \int_{-\infty}^{\infty} h(t) e^{-j\omega t} dt$	$H_k = \sum_{n=0}^{N-1} h[n] e^{-j\frac{2\pi k}{N}n}$
Complex Domain Transform	$H(s) = \int_{-\infty}^{\infty} h(t) e^{-st} dt$	$H(z) = \sum_{n=0}^{N-1} h[n] z^{-n}$
	$s \rightarrow j\omega t$	$z \rightarrow e^{j\frac{2\pi k}{N}}$



Bilinear transform

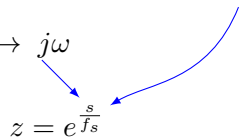
$$\text{z-domain: } z \rightarrow e^{j\frac{2\pi k}{N}} = e^{j\frac{2\pi f}{f_s}} = e^{\frac{j\omega}{f_s}}$$

$$\text{s-domain: } s \rightarrow j\omega$$

Bilinear transform

$$\text{z-domain: } z \rightarrow e^{j\frac{2\pi k}{N}} = e^{j\frac{2\pi f}{f_s}} = e^{\frac{j\omega}{f_s}}$$

$$\text{s-domain: } s \rightarrow j\omega$$

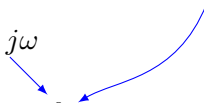
$$z = e^{\frac{s}{f_s}}$$


$$s = f_s \ln(z)$$

Bilinear transform

$$\text{z-domain: } z \rightarrow e^{j\frac{2\pi k}{N}} = e^{j\frac{2\pi f}{f_s}} = e^{\frac{j\omega}{f_s}}$$

$$\text{s-domain: } s \rightarrow j\omega$$

$$z = e^{\frac{s}{f_s}}$$


$$s = f_s \ln(z)$$

using 1st order approximation of $\ln$

$$s = 2f_s \frac{z - 1}{z + 1}$$

Bilinear transform

2nd order analog Lowpass filter

$$H_a(s) = \frac{\omega_c^2}{s^2 + \frac{\omega_c}{Q}s + \omega_c^2}$$

Bilinear Transform

$$s = 2f_s \frac{z - 1}{z + 1}$$

2nd order digital Lowpass filter

$$H_d(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}}$$

Bilinear transform

2nd order analog Lowpass filter

$$H_a(s) = \frac{\omega_c^2}{s^2 + \frac{\omega_c}{Q}s + \omega_c^2}$$

Bilinear Transform

$$s = 2f_s \frac{z - 1}{z + 1}$$

2nd order digital Lowpass filter

$$H_d(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}}$$

```
import scipy.signal as signal
import numpy as np
import matplotlib.pyplot as plt

fs = 48e3 # sampling frequency

# Analog Filter
Q = 3
fc = 1e3
wc = 2*np.pi*fc

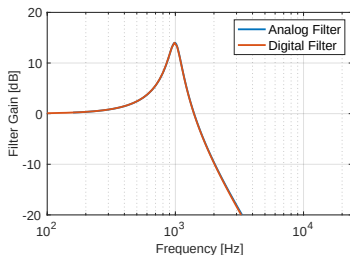
B = [0, 0, wc**2]
A = [1, wc/Q, wc**2]

# digital filter
[b, a] = signal.bilinear(B, A, fs)

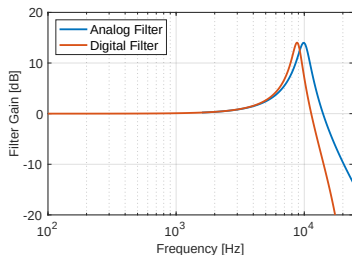
# Frequency Response Functions
Wd, Hd = signal.freqz(b, a, worN=4096)
F = Wd/(2*np.pi)*fs
_, Ha = signal.freqs(B, A, worN=2*np.pi*F)

fig, ax = plt.subplots()
ax.semilogx(F, 20*np.log10(np.abs(Ha)))
ax.semilogx(F, 20*np.log10(np.abs(Hd)), '--')
ax.set(xlim=[100, fs/2], ylim=[-20, 20])
ax.grid()
plt.show()
```

$$f_c = 1 \text{ kHz}$$



$$f_c = 10 \text{ kHz}$$



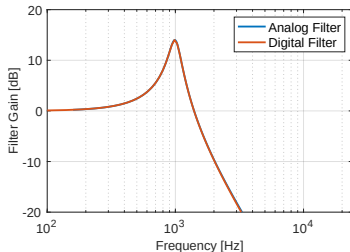
at high frequencies the filters do not match
it can be corrected with *frequency warping*

Frequency warping

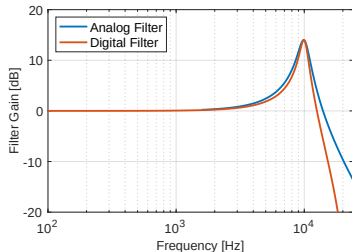
$$\omega_d = 2f_s \tan\left(\frac{\omega_c}{2f_s}\right)$$

```
# with frequency warping  
→ wd = 2*fs*np.tan(wc/(2*fs));  
B_wrap = [0, 0, wd**2];  
A_wrap = [1, wd/Q, wd**2];  
b, a = signal.bilinear(B_wrap,A_wrap,fs);
```

$f_c = 1$ kHz



$f_c = 10$ kHz



Bilinear Transform (2nd order)

$$H_a(s) = \frac{B_0 s^2 + B_1 s + B_2}{A_0 s^2 + A_1 s + A_2}$$

$$H_d(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}}$$

$$s = 2f_s \frac{z - 1}{z + 1}$$

$$b_0 = \frac{B_0 K^2 + B_1 K + B_2}{A_0 K^2 + A_1 K + A_2}$$

$$a_0 = 1$$

$$b_1 = \frac{2B_2 - 2B_0 K^2}{A_0 K^2 + A_1 K + A_2}$$

$$a_1 = \frac{2A_2 - 2A_0 K^2}{A_0 K^2 + A_1 K + A_2}$$

$$b_2 = \frac{B_0 K^2 - B_1 K + B_2}{A_0 K^2 + A_1 K + A_2}$$

$$a_2 = \frac{A_0 K^2 - A_1 K + A_2}{A_0 K^2 + A_1 K + A_2}$$

$$K = 2f_s$$

$$K = \frac{\omega_c}{\tan\left(\frac{\omega_c}{2f_s}\right)}$$



Bilinear transform

2

Audio Filters - Design

Audio IIR Filers

most often used filters in audio

- Low Pass / High Pass / Band Pass
- Low / High Shelf
- Peaking Filters
- Notch Filters
- All-pass Filters

they are very often designed from 2nd order **analog prototypes**

Low Pass Filter

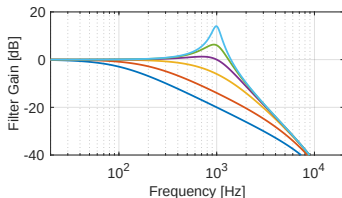
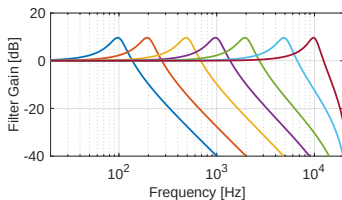
$$H_{LP}(s) = \frac{\omega_c^2}{s^2 + \frac{\omega_c}{Q}s + \omega_c^2}$$

Python code for a 2nd order low pass filter

```
wc = 2*np.pi*fc/fs
alpha = np.sin(wc)/(2*Q)

b0 = (1 - np.cos(wc))/2
b1 = 1 - np.cos(wc)
b2 = (1 - np.cos(wc))/2
a0 = 1 + alpha
a1 = -2*np.cos(wc)
a2 = 1 - alpha

b = [b0/a0, b1/a0, b2/a0]
a = [1, a1/a0, a2/a0]
```



High Pass Filter

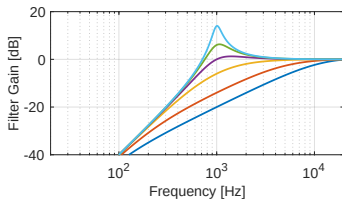
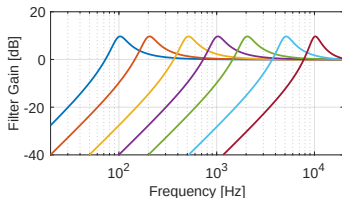
$$H_{HP}(s) = \frac{s^2}{s^2 + \frac{\omega_c}{Q}s + \omega_c^2}$$

Python code for a 2nd order high pass filter

```
wc = 2*np.pi*Fc/Fs
alpha = np.sin(wc)/(2*Q)

b0 = (1 + np.cos(wc))/2
b1 = -(1 + np.cos(wc))
b2 = (1 + np.cos(wc))/2
a0 = 1 + alpha
a1 = -2*np.cos(wc)
a2 = 1 - alpha

b = [b0/a0, b1/a0, b2/a0]
a = [1, a1/a0, a2/a0]
```



Band Pass Filter

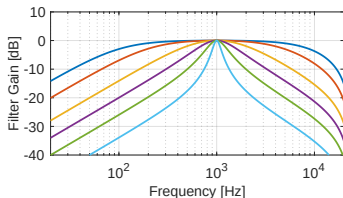
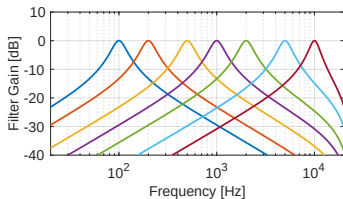
$$H_{BP}(s) = \frac{\frac{\omega_c}{Q}s}{s^2 + \frac{\omega_c}{Q}s + \omega_c^2}$$

Python code for a 2nd order band pass filter

```
wc = 2*np.pi*Fc/Fs
alpha = np.sin(wc)/(2*Q)

b0 = alpha
b1 = 0
b2 = -alpha
a0 = 1 + alpha
a1 = -2*np.cos(wc)
a2 = 1 - alpha

b = [b0/a0, b1/a0, b2/a0]
a = [1, a1/a0, a2/a0]
```



Peaking Filter

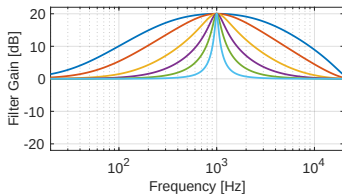
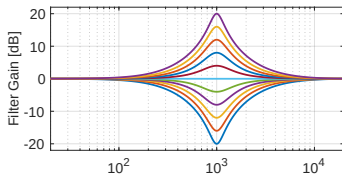
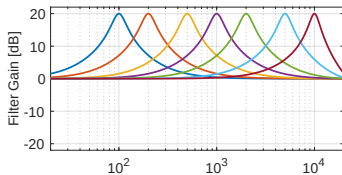
$$H_{PK}(s) = \frac{s^2 + \frac{\omega_c A}{Q} s + \omega_c^2}{s^2 + \frac{\omega_c}{AQ} s + \omega_c^2}$$

Python code for a 2nd order peaking filter

```
A = 10**(dBgain/40)
wc = 2*np.pi*Fc/Fs
alpha = np.sin(wc)/(2*Q)

b0 = 1 + alpha*A
b1 = -2*np.cos(wc)
b2 = 1 - alpha*A
a0 = 1 + alpha/A
a1 = -2*np.cos(wc)
a2 = 1 - alpha/A

b = [b0/a0, b1/a0, b2/a0]
a = [1, a1/a0, a2/a0]
```



Low-Shelf Filter

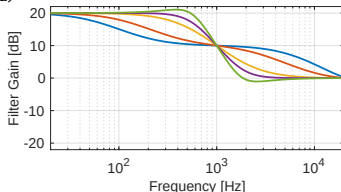
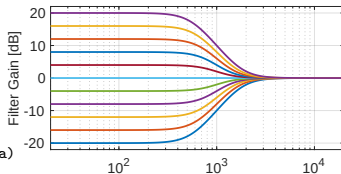
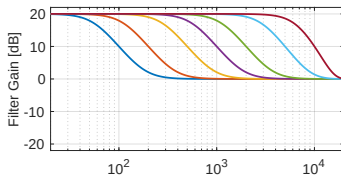
$$H_{LS}(s) = A \frac{s^2 + \frac{\omega_c \sqrt{A}}{Q}s + A\omega_c^2}{As^2 + \frac{\omega_c \sqrt{A}}{Q}s + \omega_c^2}$$

Python code for a 2nd order low-shelf filter

```
A = 10**(dBgain/40)
wc = 2*np.pi*Fc/Fs
alpha = np.sin(wc)/(2*Q)

b0 = A*((A+1) - (A-1)*np.cos(wc) + 2*np.sqrt(A)*alpha)
b1 = 2*A*((A-1) - (A+1)*np.cos(wc))
b2 = A*((A+1) - (A-1)*np.cos(wc) - 2*np.sqrt(A)*alpha)
a0 = (A+1) + (A-1)*np.cos(wc) + 2*np.sqrt(A)*alpha
a1 = -2*((A-1) + (A+1)*np.cos(wc))
a2 = (A+1) + (A-1)*np.cos(wc) - 2*np.sqrt(A)*alpha

b = [b0/a0, b1/a0, b2/a0]
a = [1, a1/a0, a2/a0]
```



High-Shelf Filter

$$H_{HS}(s) = A \frac{As^2 + \frac{\omega_c \sqrt{A}}{Q}s + \omega_c^2}{s^2 + \frac{\omega_c \sqrt{A}}{Q}s + A\omega_c^2}$$

```
# Python code for a 2nd order low-shelf filter
```

```
A = 10**(dBgain/40)
```

```
wc = 2*np.pi*Fc/Fs
```

```
alpha = np.sin(wc)/(2*Q)
```

```
b0 = A*((A+1) + (A-1)*np.cos(wc) + 2*np.sqrt(A)*alpha)
```

```
b1 = -2*A*((A-1) + (A+1)*np.cos(wc))
```

```
b2 = A*((A+1) + (A-1)*np.cos(wc) - 2*np.sqrt(A)*alpha)
```

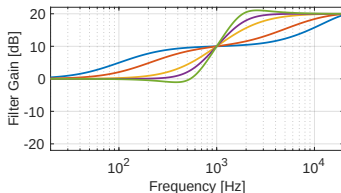
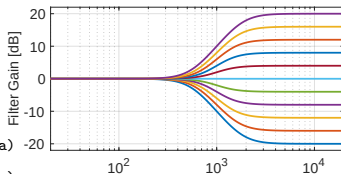
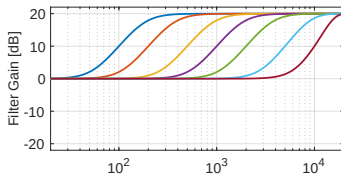
```
a0 = (A+1) - (A-1)*np.cos(wc) + 2*np.sqrt(A)*alpha
```

```
a1 = 2*((A-1) - (A+1)*np.cos(wc))
```

```
a2 = (A+1) - (A-1)*np.cos(wc) - 2*np.sqrt(A)*alpha
```

```
b = [b0/a0, b1/a0, b2/a0]
```

```
a = [1, a1/a0, a2/a0]
```



Notch Filter

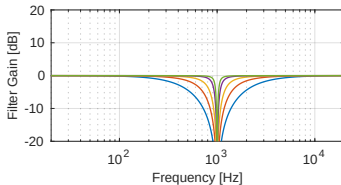
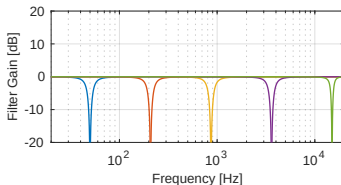
$$H_{notch}(s) = \frac{s^2 + \omega_c^2}{s^2 + \frac{\omega_c}{Q}s + \omega_c^2}$$

Python code for a 2nd order notch filter

```
wc = 2*np.pi*Fc/Fs
alpha = np.sin(wc)/(2*Q)

b0 = 1
b1 = -2*np.cos(wc)
b2 = 1
a0 = 1 + alpha
a1 = -2*np.cos(wc)
a2 = 1 - alpha

b = [b0/a0, b1/a0, b2/a0]
a = [1, a1/a0, a2/a0]
```



All-pass Filter (constant gain)

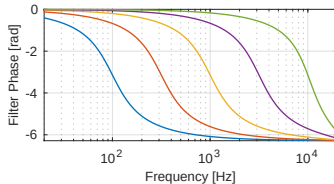
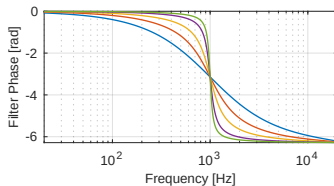
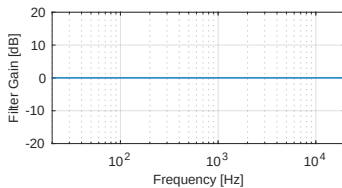
$$H_{AP}(s) = \frac{s^2 - \frac{\omega_c}{Q}s + \omega_c^2}{s^2 + \frac{\omega_c}{Q}s + \omega_c^2}$$

Python code for a 2nd order all-pass filter

```
wc = 2*np.pi*Fc/Fs
alpha = np.sin(wc)/(2*Q)

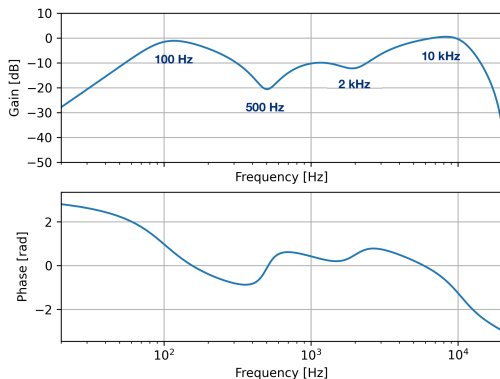
b0 = 1 - alpha
b1 = -2*np.cos(wc)
b2 = 1 + alpha
a0 = 1 + alpha
a1 = -2*np.cos(wc)
a2 = 1 - alpha

b = [b0/a0, b1/a0, b2/a0]
a = [1, a1/a0, a2/a0]
```



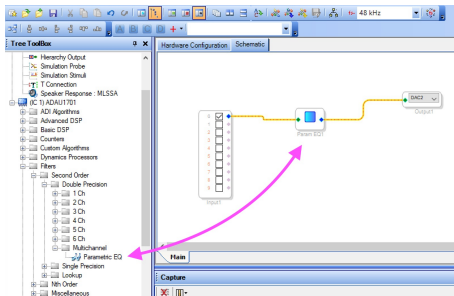
Work 1: Equalizer filters

Download TD07a.zip and extract it. It contains `IMDEA_equalizer.py`, a module with audio filters class and `equalizers.py` that you can use. Try to prepare an equalizer similar to the following one.



Work 2: Equalizer in Sigma Studio

- Implement the equalizer from the previous step in Sigma Studio using two methods:
 - ▶ Export the filter coefficients from Python and apply the Biquad filters in SigmaStudio.
 - ▶ Use the Parametric Equalizer block in SigmaStudio.



- Compare both methods.

Project

Continue working on the project.
The project assignment is available on
<https://imdea.ant-novak.com/DF>.