

Digital Filtering

TD 5: Z-plane, poles & zeros and their use in DSP for loudspeakers

Antonin NOVAK

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



Goals:

understand the Z-plane

design a filter from poles and zeros to artificially increase the stiffness K_{ms}



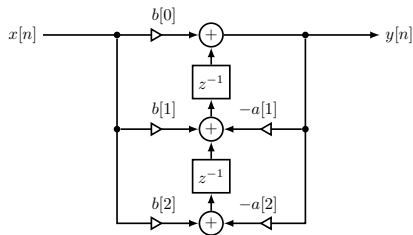
1 Poles and Zeros

Bass-boost for loudspeaker

Poles and Zeros

A 2nd-order IIR filter, also called biquad

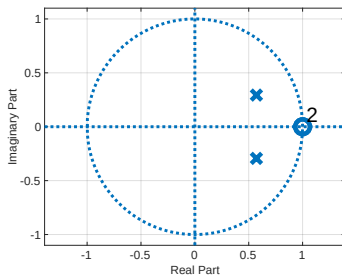
$$H(z) = \frac{b[0] + b[1]z^{-1} + b[2]z^{-2}}{1 + a[1]z^{-1} + a[2]z^{-2}}$$



Poles and Zeros

$$H(z) = \frac{b[0] + b[1]z^{-1} + b[2]z^{-2}}{1 + a[1]z^{-1} + a[2]z^{-2}}$$

$$H(z) = b[0] \cdot \frac{1 - q_0 z^{-1}}{1 - r_0 z^{-1}} \cdot \frac{1 - q_1 z^{-1}}{1 - r_1 z^{-1}}$$



Poles and Zeros

Python (scipy, or numpy) does not have a function to plot poles and zeros on the z-plane.

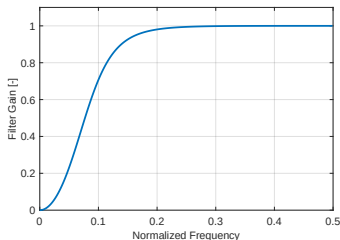
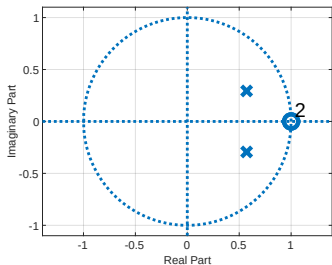
You can download [plot_zplane.py](#) from umtice.

```
## PYTHON code

import scipy.signal as signal
from plot_zplane import zplane

# butterworth high-pass filter
b, a = signal.butter(2, 0.2, 'high')

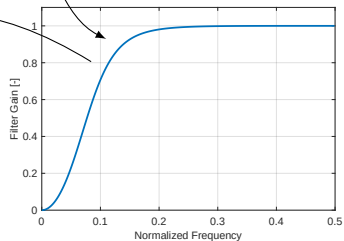
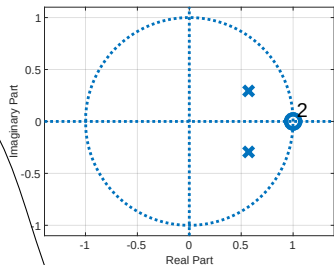
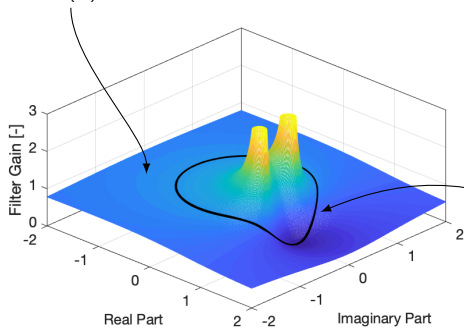
# plot the zeros and poles
zplane(b,a)
```



Poles and Zeros

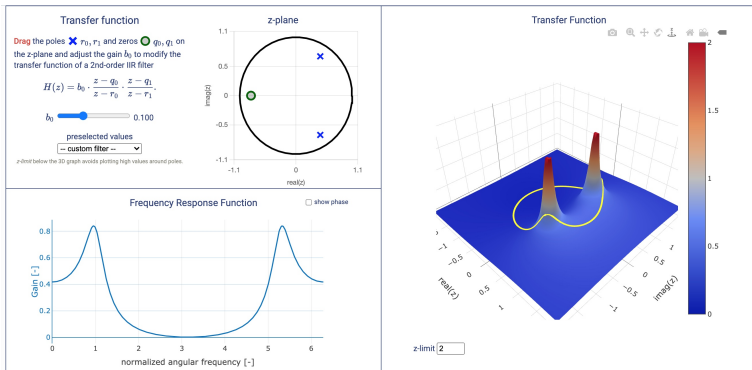
$H\left(e^{j2\pi \frac{f}{f_s}}\right)$... Frequency Response Function

$H(z)$... Transfer Function in the Z-domain



Poles and Zeros: a web app

<https://imdea.ant-novak.com/DF/Ztransform/>



Z-transform vs DFT

$$\text{z-transform: } H(z) = \sum_{n=0}^{N-1} h[n] z^{-n}$$

$$\text{DFT: } H_k = \sum_{n=0}^{N-1} h[n] e^{-j \frac{2\pi k}{N} n}$$

$$z = e^{j \frac{2\pi k}{N}} = e^{j \frac{2\pi f}{f_s}} = e^{j\Omega}$$

$\Omega \in (0, 2\pi)$
normalized angular frequency

To find the FRF (in other words the DFT of the impulse response),
we can put $z = e^{j\Omega} = \text{"a unit circle"}$.

Work 1: placing poles and zeros

Use the [web-app](#) or the Python code [z_transform_3D.py](#) and try, just by placing poles and zeros, to design:

1. a low-pass filter
2. a high-pass filter
3. a band-pass filter



Poles and Zeros

2

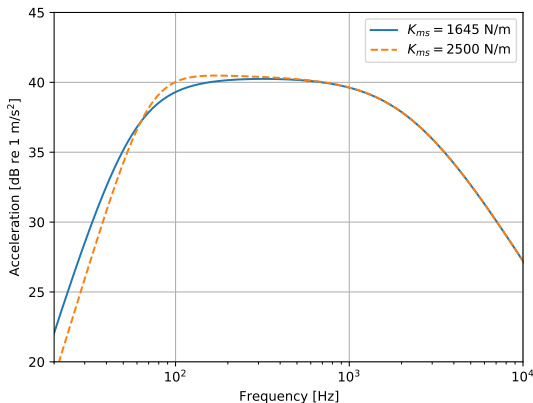
Bass-boost for loudspeaker

Work 2: Loudspeaker

You have a speaker $B_l=4.42$, $R_e=3.2$, $L_e=0.22\text{e-}3$, $M_{ms}=0.0136$, $R_{ms}=1.4$, $K_{ms}=1645$

Create a digital filter (on DSP) that artificially increases the stiffness of the speaker to behave like with $K_{ms} = 2500$

Follow the steps from next slides.



Step 1: Loudspeaker equations

Write the loudspeaker equations in the Laplace domain and extract the Laplace domain coefficients.

$$\mathbf{U}(s) = R_e \cdot \mathbf{I}(s) + L_e \cdot s \cdot \mathbf{I}(s) + Bl \cdot s^{-1} \cdot \mathbf{A}(s)$$

$$Bl \cdot \mathbf{I}(s) = M_{ms} \cdot \mathbf{A}(s) + R_{ms} \cdot s^{-1} \cdot \mathbf{A}(s) + K_{ms} \cdot s^{-2} \cdot \mathbf{A}(s)$$

$$\frac{\mathbf{A}(s)}{\mathbf{U}(s)} = \frac{s^2}{\frac{L_e M_{ms}}{Bl} s^3 + \frac{R_e M_{ms} + L_e R_{ms}}{Bl} s^2 + \frac{R_e R_{ms} + L_e K_{ms} + Bl^2}{Bl} s + \frac{R_e K_{ms}}{Bl}}$$

```
# Laplace doimain coefficients
```

```
A = np.zeros(4)
```

```
A[0] = Le* Mms /Bl
```

```
A[1] = Re* Mms /Bl + Le* Rms /Bl
```

```
A[2] = Re* Rms /Bl + Le* Kms /Bl + Bl
```

```
A[3] = Re* Kms /Bl
```

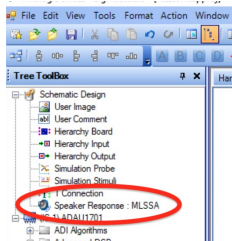
```
B = [1, 0, 0]
```

Step 2: Export measured / simulated FRF to Sigma Studio

Export the simulated FRF $A(s)/U(s)$ to a specific file format for Sigma Studio

1. Export the $A(s)/U(s)$ in a .txt file

- $B1 = 4.42$, $Re = 3.2$, $Le = 0.22e-3$, $Mms = 0.0136$, $Rms = 1.4$, $Kms = 1645$
- use MLSSA format compatible with Sigma Studio (see [prepare_MLSSA.py](#))
- In Sigma Studio, use block "Speaker Response: MLSSA" and load the .txt file
- use **Stumulus** and **Probe** to test it



```
f = np.logspace(np.log10(1), np.log10(40e3), 1000)
s = (1j*2*np.pi*f)
FRF = s ** 2 / (A[0] * s ** 3 + A[1] * s ** 2 + A[2] * s + A[3])

# save file for Sigma Studio
np.savetxt('loudspeakers_TD03.txt',
           np.array([f, 20*np.log10(np.abs(FRF)), 180/np.pi*np.angle(FRF)]).T,
           delimiter=', ',
           header='IMDEA Simulated Loudspeaker"\n "Hz" "Mag (dB)" "deg"',
           comments='',
           fmt='%0.5f')
```

Step 3: Bass-boost filter design for loudspeaker

Design a 2nd-order IIR filter in Python that artificially increases stiffness K_{ms}

For the following steps, **neglect L_e !!!**, see the next page for equations.

1. Design a 2nd-order **analog** filter such that:
 - cancels poles of the loudspeaker (denominator of loudspeaker's equation)
 - add new poles (new denominator) corresponding to new $K'_{ms} = 2500 \text{ N/m}$
2. Convert the analog filter to a digital one using bilinear transformation
 - use `b, a = scipy.signal.bilinear(B, A, fs)`
3. Implement it in Sigma Studio
 - add the 2nd-order IIR in front of the MLSSA block
 - compare the results with and without the IIR filter (stimuli and probe)

Loudspeaker equations for $Le=0$

$$\frac{A(s)}{U(s)} = \frac{s^2}{\frac{R_e M_{ms}}{Bl} s^2 + \frac{R_e R_{ms} + Bl^2}{Bl} s + \frac{R_e K_{ms}}{Bl}}$$

$$\frac{A(s)}{U(s)} = \frac{\frac{Bl}{R_e M_{ms}} s^2}{s^2 + \frac{R_{ms} R_e + Bl^2}{M_{ms} R_e} s + \frac{K_{ms}}{M_{ms}}}$$

Project

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