Filters

ADC X Dynamic Cast 2022
Practical DSP

Harriet Drury (@druryharriet)

Audio Filters - Frequency Response

The frequency response of a filter is the given magnitude and phase of the output as a function of input frequency. There are 4 common types of filter:

- High Pass
- Low Pass
- Band Pass
- All Pass

Example: Low Pass Frequency Response

Given the following digital system, determine the frequency response:

$$y(n)=0.5x(n)+0.5x(n-1)$$

Example: Low Pass Frequency Response

Given the following digital system, determine the frequency response:

$$y(n)=0.5x(n)+0.5x(n-1)$$

Let’s break this down:

y(n) - Output Signal

x(n) - Input Signal

x(n-1) - Previous input signal

Solution

y(n)=0.5x(n)+0.5x(n-1)

1. Transform both sides into the Z domain

$$Y_{(z)} = 0.5X(z) + 0.5z^{-1}X(z)$$

2. The transfer function can be found with:

$$H(z) = \frac{Y(z)}{X(z)}$$

Therefore:

$$H(z) = \frac{Y(z)}{X(z)} = 0.5 + 0.5z^{-1}$$

(we've moved the X(z))

Solution

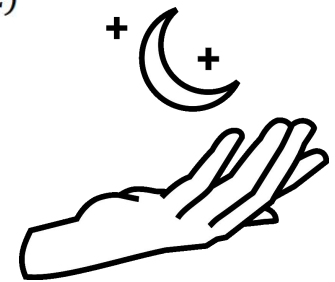
$$H(z) = \frac{Y(z)}{X(z)} = 0.5 + 0.5z^{-1}$$

3. Substitute Euler's identity ($e^{j\Omega}$)

$$H\left(e^{j\Omega}\right) = 0.5 + 0.5e^{-j\Omega}$$
$$= 0.5 + 0.5\cos(\Omega) - j0.5\sin(\Omega)$$

4. Therefore, magnitude and phase response are given by:

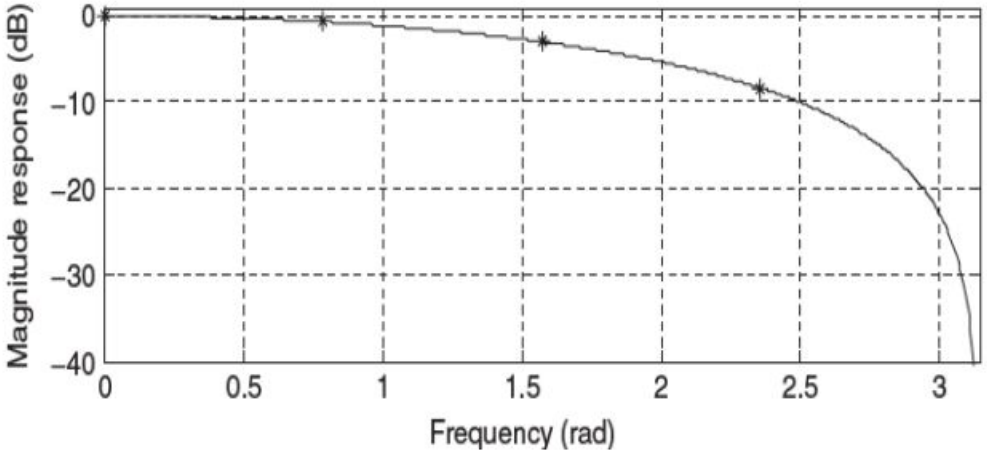
$$\left|H\left(e^{j\Omega}\right)\right| = \sqrt{(0.5 + 0.5\cos(\Omega))^2 + (0.5\sin(\Omega))^2} \quad \text{and} \quad \angle H\left(e^{j\Omega}\right) = \tan^{-1}\left(\frac{-0.5\sin(\Omega)}{0.5 + 0.5\cos(\Omega)}\right)$$



Solution

$$\left|H\left(e^{j\Omega}\right)\right| = \sqrt{(0.5 + 0.5\cos(\Omega))^2 + (0.5\sin(\Omega))^2} \quad \text{and} \quad \angle \mathrm{H}\left(e^{j\Omega}\right) = \tan^{-1}\left(\frac{-0.5\sin(\Omega)}{0.5 + 0.5\cos(\Omega)}\right)$$

It is observed that when the frequency increases, the magnitude response decreases. The DSP system acts like a digital low pass filter, and its phase response is linear.



This is a Low Pass FIR Filter

What does that mean?

Finite Impulse Response filters (FIR) outputs are reliant on the current and previous input samples

$$y(n)=0.5x(n)+0.5x(n-1)$$

- x(n): Current
- x(n-1): Previous

We consider this a 1st order FIR filter x(n-1).

FIR Online Visualiser

https://fiiir.com/

You can describe the input/output relation of an FIR filter with:

$$y[n] = \sum_{k=0}^{N} b[k]x\,[n-k]$$

Where N is the order of the FIR filter (its length-1) and those coefficients b[k] of length N+1 are the FIR filters impulse response, or the filter coefficients in practice.

FIR Filters: Pros and Cons

Pros

- Easy
- Stable
- Robust

Cons

- Large Storage Requirement
- Computationally Expensive
- Cannot simulate prototype analog filters

Infinite Impulse Response (IIR) Filters

For an infinite impulse response filter (IIR), the output of the filter is based not only on current and previous input samples, but also previous output samples. Given a difference equation:

$$y(n) = 0.5x(n) + 0.5x(n-1) + 0.2y(n-1)$$

We see that y(n-1) is added from the previous FIR filter difference equation.

Example

This is a second order IIR filter

$$y(n) + b_1 y(n-1) + b_2 y(n-2) = a_0 x(n) + a_1 x(n-1) + a_2 x(n-2)$$

(we've replaced the arbitrary numbers with a and b)

Changes from the FIR filter is the inclusion of previous output y(n) signals. Cool!

How do we find the impulse response?

Solution

1. Transform into the Z domain

$$y(z) + b_1 y(z) z^{-1} + b_2 y(z) z^{-2} = a_0 x(z) + a_1 x(z) z^{-1} + a_2 x(z) z^{-2}$$

2. The transfer function can be found with:

$$H(z) = \frac{Y(z)}{X(z)}$$

Therefore:

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

IIR Transfer Function

So far:

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

The transfer function is quadratic in the numerator, and the denominator. This is in fact called a “biquadratic filter” or a “biquad” for short.

Higher order filters (like EQs) are made by chaining multiple biquads together. Biquads are a pretty important staple of DSP.

Poles and Zeros

The zeros of the quadratic equation in the numerator are also the ZEROS of the filter.

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

The zeros of the quadratic equation in the denominator are also the POLES of the filter.

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

Pole Zero Plot

Unlike FIRs which are relatively smooth and can only make certain frequencies lower, IIRs can cause frequency amplitudes to get much higher and even shoot off to infinity. That makes for some interesting sounds by adding distortion to certain frequency notes of an instrument but not others.

By solving the quadratic equations, we can find the poles and zeros of the IIR filter

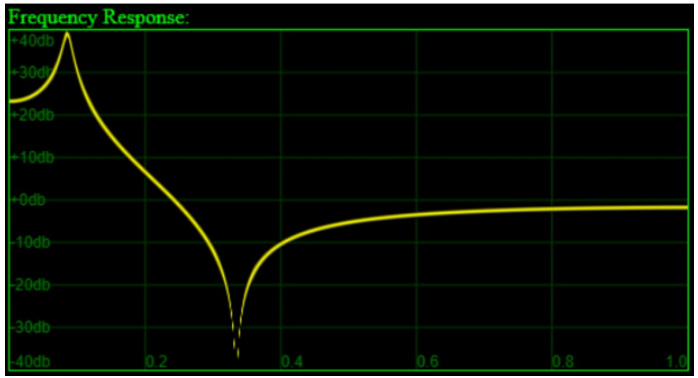
Pole Zero Plot

Zeros:

$$z = \frac{-\alpha_1}{2} + \frac{\sqrt{{\alpha_1}^2 - 4\alpha_2}}{2}$$

Poles:

$$z = \frac{-b_1}{2} + \frac{\sqrt{{b_1}^2 - 4b_2}}{2}$$



IIR Filters: Pros and Cons

Pros

- Implementation of IIR filter involves fewer parameters, less memory requirement, and lower computational complexity.
- Easy to Design
- Easy to Implement

Cons

- IIR Filters become unstable
- IIR filter has a feedback loop so they will accumulate rounding and noise error.

MATLAB

Let’s implement an IIR Filter…..

MATLAB Code:

```
% Various Plots for IIR Filtering
% h[n] = y[n]/x[n]
clc;
clear all;
num = [1 0];    % a -> coefficients of y[n] Changeable to see what happens
den = [1 -1];   % b -> coefficients of x[n] Changeable to see what happens
figure(1)
subplot(1,3,1)
zplane(num,den)          % Z plane plot of h(n)
title('Pole Zero Plot')
w = 0:pi/32:pi;          % Omega Frequency (Changeable)
[h,w]=freqz(num,den,w); % Freqz = frequency response
mag=abs(h);              % Absolute Value for magnitude
phase=angle(h);
subplot(1,3,2)
plot(w/pi,mag)
title('Magnitude Plot for IIR LPF')
subplot(1,3,3)
plot(w/pi,phase)
title('Phase Plot for IIR LPF')
```

MATLAB Code:

```
% Basic butterworth filter example
% Uses a built in IIR filter

[dataIn, Fs] = audioread('Lo-Fi-Demo.mp3');     % Read an audio file
% Filter the signal
fc = 800; % Make higher to hear higher frequencies.
% Design a Butterworth filter.
[b, a] = butter(6,fc/(Fs/2));
freqz(b,a)
% Apply the Butterworth filter.
filteredSignal = filter(b, a, dataIn);
% Play the sound.
player = audioplayer(filteredSignal, Fs);
play(player);
```

Thanks!