Second-Order Bandpass/Bandreject Filter Design

Yangang Cao

February 15, 2019

The signal can be seen as a set of partials having different frenquencies and amplitudes. The filter can modify the amplitude of partials according to their frenquency.

1 Definition of Bandpass/Bandreject Filters

The two types of filters can be defined according to the following classification:

- **Bandpass (BP)** filters select frenquencies between a lower cut-off frenquency f_{cl} and a higher cut-off frenquency f_{ch}. Frenquencies below f_{cl} and frenquencies higher than f_{ch} are attenuated.
- **Bandreject (BR)** filters attenuate frenquencies between a lower cut-off frenquency f_{cl} and a higher cut-off frenquency f_{ch}. Frenquencies below f_{cl} and frenquencies higher than f_{ch} are passed.

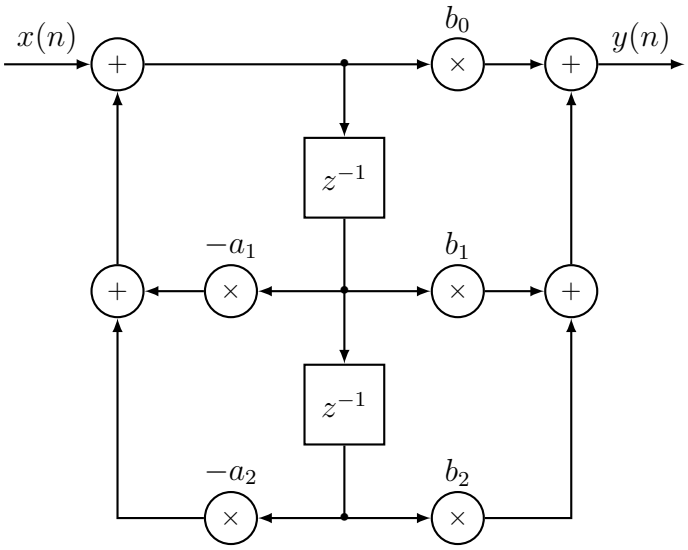
The bandpass can produce effects such as the imitation of a telephone line or of a mute on an acoustical instrument; the bandreject can divide the audible spectrum into two bands that seem to be uncorrelated.

2 Canonical Form

There are various ways to implement a filter, the simplest being the canonical filter, as shown in following figure for a second-order filter, which can be implemented by the defference equations

$$x_h(n) = x(n) - a_1 x_h(n-1) - a_2 x_h(n-2)$$

$$y(n) = b_0 x_h(n) + b_1 x_h(n-1) + b_2 x_h(n-2),$$



and leads to the transfer function by setting $a_2 = b_2 = 0$, this reduces to a first-order filter which, can be used to implement an allpass, lowpass or highpass with the coefficients of following table

	b_0	b_1	a_1
Lowpass	K/(K+1)	K/(K+1)	(K-1)/(K+1)
Highpass	1/(K+1)	-1/(K+1)	(K-1)/(K+1)
Allpass	(K-1)/(K+1)	1	(K-1)/(K+1)

where K depends on the cut-off frequency f_c by

$$K = \tan(\pi f_c / f_S).$$

3 Second-Order Allpass-Based Filter

The implementation of tunable bandpass and bandreject filters can be achieved with a second-order allpass filter. The transfer function of a second-order allpass filter is given by

$$A(z) = \frac{-c + d(1-c)z^{-1} + z^{-2}}{1 + d(1-c)z^{-1} - cz^{-2}}$$

$$c = \frac{\tan(\pi f_b / f_S) - 1}{\tan(\pi f_b / f_S) + 1}$$

$$d = -\cos(2\pi f_c / f_S),$$

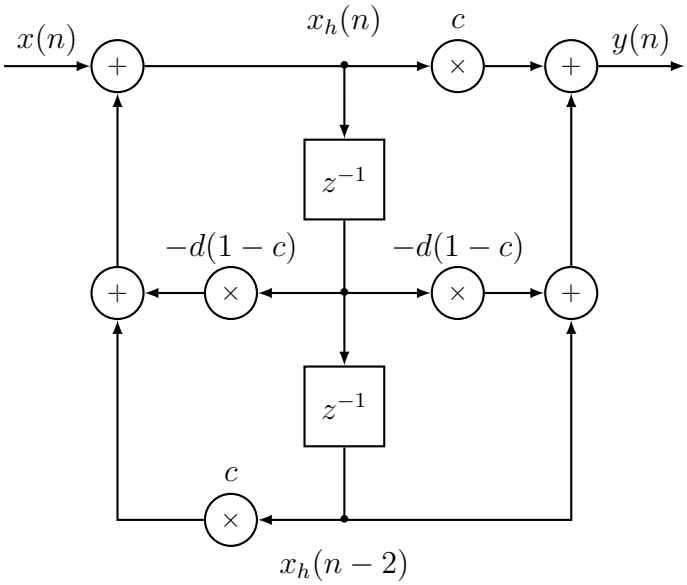
with the corresponding difference equatons

$$x_h(n) = x(n) - d(1-c)x_h(n-1) + cx_h(n-2)$$

$$y(n) = -cx_h(n) + d(1-c)x_h(n-1) + x_h(n-2).$$

The parameter d adjusts the center frequency and the parameter c the bandwidth. The magnitude response is again equal to one and the phase response

approaches $-360°$ for high frequencies. The cut-off frequency f_c determines the point on the phase curve where the phase response passes $-180°$. The width or slope of the phase transition around the cut-off frequency is controlled by the bandwidth parameter f_b. The following block diagram shows the second-order allpass filter,



and corresponding state and output equations are:

$$\begin{bmatrix} x_h(n) \\ x_h(n-1) \end{bmatrix} = \begin{bmatrix} -d(1-c) & c \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_h(n-1) \\ x_h(n-2) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} x(n)$$

$$y(n) = \begin{bmatrix} (1-c^2)d & 1-c^2 \end{bmatrix} \begin{bmatrix} x_h(n-1) \\ x_h(n-2) \end{bmatrix} + (-c)x(n).$$

Actually, the state and output equations represent the same meaning as the difference equations, but difference in format. The difference equations are usually easy to comprehend, howere, state and output equations are more widely used in modern control theory, so we implement all kinds of filters by this format. The second-order allpass filter implementation can be obtained by the following **Matlab** code.

```
function y = secondallpass(audio, para)
% y = secondallpass(audio, para)
% Author: Yangang Cao
% Applies a allpass filter to the input signal.
% para(1) is the normalized center frequency in (0,1), i.e. 2*fc/fS.
% para(2) is the normalized bandwidth in (0,1) i.e. 2*fb/fS.
c = (tan(pi*para(2)/2)-1) / (tan(pi*para(2)/2)+1);
d = -cos(pi*para(1));
x = [0; 0];
x_1 = 0;
A = [-d*(1-c), c; 1, 0];
B = [1; 0];
C = [d*(1-c^2), 1-c^2];
```

```
D = -c;
for n=1:length(audio)
    x_1 = A * x + B * audio(n);
    y(n) = C * x + D * audio(n);
    x = x_1;
end
```

4 Second-Order Bandpass/Bandreject Filter

Second-order bandpass and bandreject filters can be described by the following transfer function

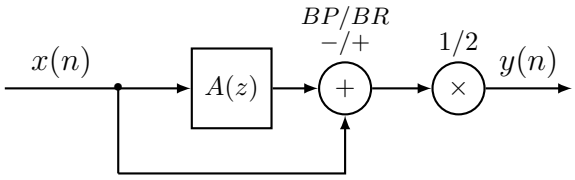
$$H(z) = \frac{1}{2}[1 \mp A(z)] \quad (BP/BR - /+)$$

$$A(z) = \frac{-c + d(1-c)z^{-1} + z^{-2}}{1 + d(1-c)z^{-1} - cz^{-2}}$$

$$c = \frac{\tan(\pi f_b/f_S) - 1}{\tan(\pi f_b/f_S) + 1}$$

$$d = -\cos(2\pi f_c/f_S),$$

where a tunable second-order allpass $A(z)$ with tuning parameters c and d is used. The minus sign (-) denotes the bandpass operation and the plus sign (+) the bandreject operation. The block diagram in following figure represents the operations involved in performing the bandpass/bandreject filtering.



The difference equations of second-order bandpass filter are:

$$x_h(n) = x(n) - d(1-c)x_h(n-1) + cx_h(n-2)$$

$$y(n) = \frac{1+c}{2}x_h(n) - \frac{1+c}{2}x_h(n-2),$$

and corresponding state and output equations are:

$$\begin{bmatrix} x_h(n) \\ x_h(n-1) \end{bmatrix} = \begin{bmatrix} -d(1-c) & c \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_h(n-1) \\ x_h(n-2) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} x(n)$$

$$y(n) = \begin{bmatrix} \frac{d(c^2-1)}{2} & \frac{c^2-1}{2} \end{bmatrix} \begin{bmatrix} x_h(n-1) \\ x_h(n-2) \end{bmatrix} + \frac{1+c}{2}x(n)$$

A second-order bandpass filter implementation can be obtained by the following **Matlab** code.

```
function y = apbandpass(audio, para)
% y = apbandpass(audio, para)
% Author: Yangang Cao
% Applies a bandpass filter to the input signal.
% para(1) is the normalized center frequency in (0,1), i.e. 2*fc/fS.
% para(2) is the normalized bandwidth in (0,1) i.e. 2*fb/fS.
c = (tan(pi*para(2)/2)-1) / (tan(pi*para(2)/2)+1);
d = -cos(pi*para(1));
x = [0; 0];
x_1 = 0;
A = [-d*(1-c), c; 1, 0];
B = [1; 0];
C = [d*(c^2-1)/2, (c^2-1)/2];
D = (1+c)/2;
for n=1:length(audio)
    x_1 = A * x + B * audio(n);
    y(n) = C * x + D * audio(n);
    x = x_1;
end
```

The difference equations of second-order bandreject filter are:

$$x_h(n) = x(n) - d(1-c)x_h(n-1) + cx_h(n-2)$$

$$y(n) = \frac{1-c}{2}x_h(n) + d(1-c)x_h(n-1) + \frac{1-c}{2}x_h(n-2),$$

and corresponding state and output equations are:

$$\begin{bmatrix} x_h(n) \\ x_h(n-1) \end{bmatrix} = \begin{bmatrix} -d(1-c) & c \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_h(n-1) \\ x_h(n-2) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} x(n)$$

$$y(n) = \begin{bmatrix} \frac{d(1-c^2)}{2} & \frac{1-c^2}{2} \end{bmatrix} \begin{bmatrix} x_h(n-1) \\ x_h(n-2) \end{bmatrix} + \frac{1-c}{2}x(n)$$

A second-order bandreject filter implementation can be obtained by the following **Matlab** code.

```
function y = apbandreject(audio, para)
% y = apbandreject(audio, para)
% Author: Yangang Cao
% Applies a bandreject filter to the input signal.
% para(1) is the normalized center frequency in (0,1), i.e. 2*fc/fS.
% para(2) is the normalized bandwidth in (0,1) i.e. 2*fb/fS.
c = (tan(pi*para(2)/2)-1) / (tan(pi*para(2)/2)+1);
d = -cos(pi*para(1));
x = [0; 0];
x_1 = 0;
A = [-d*(1-c), c; 1, 0];
B = [1; 0];
C = [d*(1-c^2)/2, (1-c^2)/2];
D = (1-c)/2;
for n=1:length(audio)
    x_1 = A * x + B * audio(n);
    y(n) = C * x + D * audio(n);
    x = x_1;
end
```