First/Second-Order Allpass Filter Design

Yangang Cao

February 18, 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. A special class of parametric filter structures for lowpass, highpass, bandpass and bandreject filter functions was introduced in the following articles of this series, these filter structures are easily tunable by changing only one or two coefficients. They play important roles for real-time control with minimum computational complexity.

The basis for parametric first- and second-order IIR filters is the first- and second-order allpass filter, the allpass filters don't affect the amplitude, but the phase of signal. We discuss the first- and second-order allpass filters in this text.

A first-order allpass filter is given by the transfer function

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

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

the difference equations

$$x(n) = u(n) - cx(n-1)$$

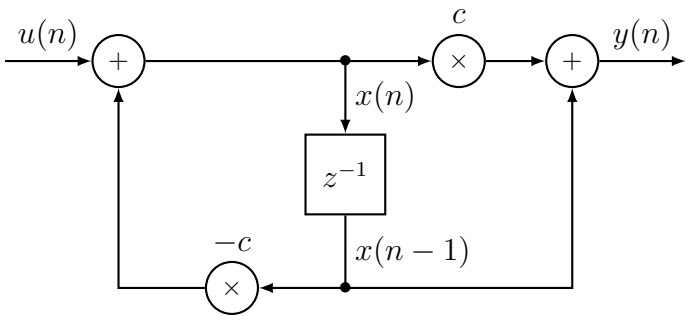
$$y(n) = cx(n) + x(n-1),$$

and corresponding state and output equations are

$$x(n) = -cx(n-1) + u(n)$$

$$y(n) = (1 - c^2)x(n-1) + cu(n).$$

which can be realized by the following block diagram.



A first-order allpass filter implementation can be obtained by the following **Matlab** code.

```
function y = firstallpassunit(audio, para)
% Applies a allpass filter to the input signal.
% para is the normalized cut-off frequency in (0,1)
c = (tan(pi*para/2)-1) / (tan(pi*para/2)+1);
x = 0;
x_1 = 0;
for n = 1:length(audio)
    x_1 = -c * x+ audio(n);
    y(n) = (1-c^2) * x + c * audio(n);
    x = x_1;
end
```

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

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

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

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

the difference equatons

$$x(n) = u(n) - d(1-c)x(n-1) + cx(n-2)$$

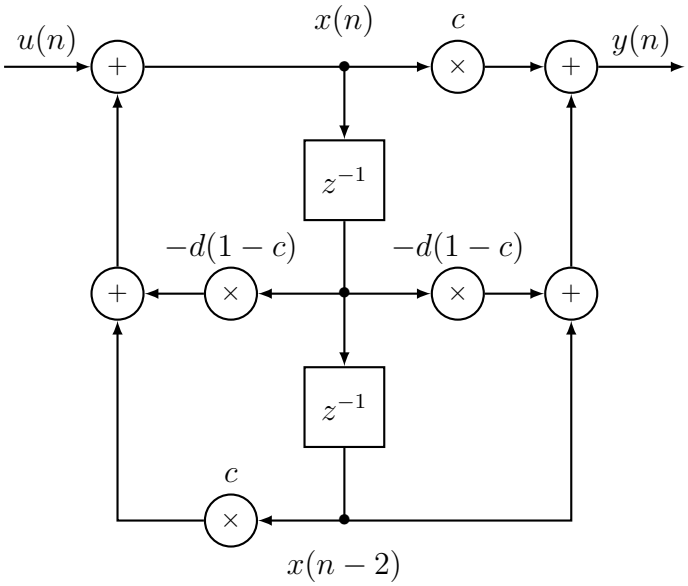
$$y(n) = -cx(n) + d(1-c)x(n-1) + x(n-2).$$

and corresponding state and output equations

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

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

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.



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


```
function y = secondallpassunit(audio, para)
% 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
```