CPE 367 – Experiment 5a – v4
Reverb Audio Effect – Dr F DePiero (28 Points)

Overview and Motivation

The signal processing in this experiment involves digital filters with particular delay characteristics and a with a specific style of impulse response. This is different than in many cases when the magnitude of the frequency response is the more critical aspect of the design.

Reverb is an effect that is naturally produced in a music hall or auditorium. It is caused when sound heard by the listener reflects from many surfaces. We will implement a digital filter that simulates the reverb effect. Our design is based on the original paper by Schroeder in 1962. The impulse response of Schroeder's reverb includes thousands of echoes per second. Schroeder used multiple stages (in parallel and cascade). See Fig 3. Each filter is IIR, thus generating many echoes before the response dies out. By combining multiple filters, each subsequent filter generates many echoes from the former, to generate the many echoes needed.

FYI, one of the filters used in the reverb is referred to as an 'all pass' meaning that it passes all frequencies without attenuation. This filter is designed specifically for its delay properties, allowing the full spectrum of the music to pass through.

Learning Objectives

- Find the equivalent difference equation for an IIR system
- Implement parallel and cascaded IIR filters
- Determine the impulse response for a simple IIR system algebraically

Prerequisite Learning Objectives

- Implement a digital filter in Python
- Python programming with WAV file I/O and some experience with MatLab

Procedures, Questions and Deliverables

1) Finding IIR difference equations

The reverb effect by Schroder includes two types of filters, a comb filter (Figure 1) and an all-pass (Figure 2). See M. R. Schroeder, "Natural Sounding Artificial Reverberation," J of the Audio Engineering Society, v10 n3, July 1962.

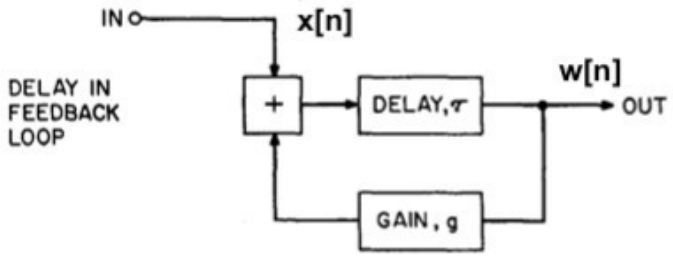


Figure 1. IIR comb filter with delay and feedback (from Schroeder, g = 0.7)

The comb filter does not have a flat frequency response. It eliminates certain frequencies, depending on the amount of delay. For example, consider if the delay was equal to half the period of a sinusoid. The feedback would result in a zero for the filter (adding trough to peak, for example).

- **1a) For the comb filter of Figure 1, find a difference equation an expression for w[n] in terms of x[] and w[] (4)**

$$w[n] = x[n] + x[n-1] \longrightarrow (1)$$

$$y[n] = w[n] - w[n-1] + w[n-2] \longrightarrow (2)$$

Applying z-Transform to equation (1)

$$W[z] = X(z) + z^{-1}X(z)$$

$$\frac{W(z)}{X(z)} = (1 + z^{-1})$$

Applying z-Transform to equation (2)

$$Y(z) = W(z) - z^{-1}W(z) + z^{-2}W(z)$$

$$\frac{Y(z)}{W(z)} = 1 - z^{-1} + z^{-2}$$

$$\frac{Y(z)}{X(z)} = \frac{Y(z)}{W(z)} \frac{W(z)}{X(z)} = (1 - z^{-1} + z^{-2})(1 + z^{-1})$$

$$= 1 + z^{-1} - z^{-1} - z^{-2} + z^{-2} + z^{-3}$$

$$\frac{Y(z)}{X(z)} = 1 + z^{-3}$$

$$Y(z) = X(z) + z^{-3}X(z)$$

Applying inverse z-Transform

$$\boxed{y[n] = x[n] + x[n-3]}$$

See the handout "hardware realizations.pdf" for examples and suggestions for finding a difference equation, given a hardware block diagram

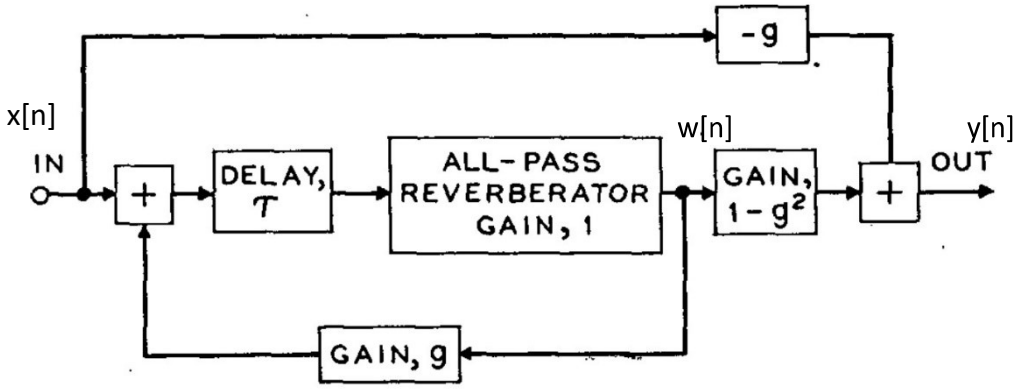


Figure 2. IIR all-pass filter. The "ALL-PASS REVERBERATOR GAIN" is a gain of 1. The other gains, g = 0.7, (from Schroeder). Consider calling the 'update' method at the input to the delay and calling the 'get' method at the output of the delay, for example.

- **1b)**

With N = 4 we obtain the transfer function

$$H(z) = \frac{1}{4}\left(z^{-1} + z^{-2} + z^{-3} + z^{-4}\right)$$

$$H(z) = \frac{1}{4}\,\frac{z^3+z^2+z+1}{z^4}$$

The are four poles at z = 0 and three zeros from the solution

$$z^3 + z^2 + z + 1 = \frac{1-z^4}{1-z} = 0$$

Since the transfer function is of the form

$$H(z) = \frac{1}{N}\,\frac{1-z^N}{z^N(1-z)}$$

the zeros are of the form

$$z = e^{jk\frac{2\pi}{N}},\ k = 1,\ \ldots,\ N-1$$

and the poles are all at $z = 0$.

The IIR all-pass has a flat frequency response. This can be best explained via Z transforms which will be covered later in the course. FYI, there is a pole-zero cancellation that results in a uniform response for all frequencies.

2) Reverb Implementation

Implement the reverb in Python. This requires 6 IIR filters. Each will need two FIFOs, one for the history of the input and one for the history of the output. (Although the four comb filters in parallel can share the FIFO storing their input history).

Parameters for the reverb – see Figure 3.

- Delays tau1-tau4 should be distinct values, chosen on the range of 30-45 mSec.
- Delays tau5 and tau6 should be 5 and 1.7 mSec.
- Gains g1 – g6 should all be 0.7
- Gain g7 can be adjusted to your liking, and to prevent clipping

- **2) Documentation: Paste the sections of your Python code that implements the six digital filters, using your FIFO (10)**

```
from scipy.signal import butter, freqz
import matplotlib.pyplot as plt
from math import pi
import numpy as np

f_s = 360    # Sample frequency in Hz
f_c = 45     # Cut-off frequency in Hz
order = 4    # Order of the butterworth filter

omega_c = 2 * pi * f_c       # Cut-off angular frequency
omega_c_d = omega_c / f_s    # Normalized cut-off frequency (digital)

# Design the digital Butterworth filter
b, a = butter(order, omega_c_d / pi)
print('Coefficients')
print("b =", b)                    # Print the coefficients
print("a =", a)

w, H = freqz(b, a, 4096)               # Calculate the frequency response
w *= f_s / (2 * pi)                    # Convert from rad/sample to Hz

# Plot the amplitude response
plt.subplot(2, 1, 1)
plt.suptitle('Bode Plot')
H_dB = 20 * np.log10(abs(H))           # Convert modulus of H to dB
plt.plot(w, H_dB)
plt.ylabel('Magnitude [dB]')
plt.xlim(0, f_s / 2)
plt.ylim(-80, 6)
plt.axvline(f_c, color='red')
plt.axhline(-3, linewidth=0.8, color='black', linestyle=':')

# Plot the phase response
plt.subplot(2, 1, 2)
phi = np.angle(H)                      # Argument of H
```

```
phi = np.unwrap(phi)                # Remove discontinuities
phi *= 180 / pi                     # and convert to degrees
plt.plot(w, phi)
plt.xlabel('Frequency [Hz]')
plt.ylabel('Phase [°]')
plt.xlim(0, f_s / 2)
plt.ylim(-360, 0)
plt.yticks([-360, -270, -180, -90, 0])
plt.axvline(f_c, color='red')

plt.show()
```

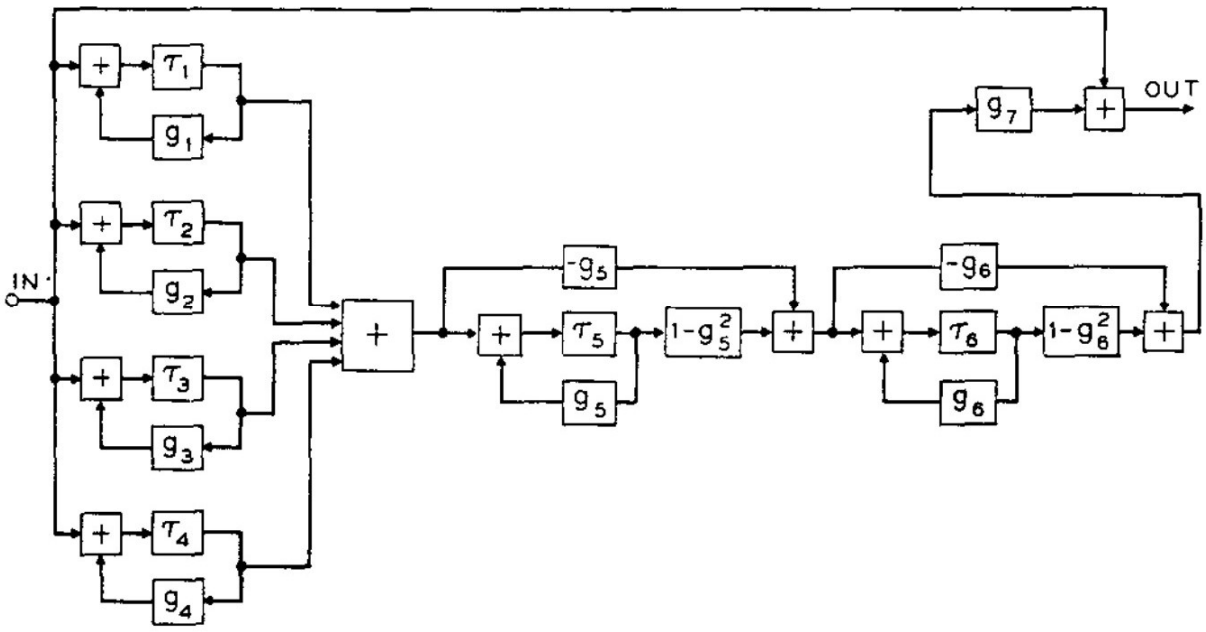


Figure 3. Reverb Effect (from Schroeder)

Suggestions

- To begin setting up the reverb effect, consider implementing and testing one stage of the comb filter or the all-pass first. A single stage should have negligible affect (the output should sound approximately the same as the input).
- Note the four comb filters are in parallel (on left) and share the same input
- Note the two all-pass filters are in cascade
- When implementing the whole reverb, consider computing the various signals working from the “IN” on the left and proceeding to the right.
- An example of the reverb is posted on the course Canvas site
- Use instances of your FIFOs as needed (at least six required). Consider calling the ‘update’ method to store the input of a delay unit. Call the ‘get’ method to access the output of the delay unit.

- See Schroeder's original paper, as you interest and time permit (posted on Canvas)

Unanswered Questions

- **How do we analyze the response of IIR filters, in particular the transient response?**

№	Name	Expression	Remark
1.	Input signal	$x(t) = \mathrm{Re}\left(\dot{\mathbf{X}}^{\mathrm{T}} e^{\mathbf{p}t}\right)$, $X(p) = \mathrm{Re}\left(\dot{\mathbf{X}}^{\mathrm{T}}\left[\frac{1}{p-p_n}\right]_N\right)$	$\dot{\mathbf{X}} = \left[\dot{X}_n\right]_N = \left[X m_n e^{-j\varphi_n}\right]_N$, $\mathbf{p} = [p_n]_N = [-\beta_n + j\omega_n]_N$
2.	Filter	$g(t) = \mathrm{Re}\left(\dot{\mathbf{G}}^{\mathrm{T}} e^{\mathbf{q}t}\right)$, $K(p) = \mathrm{Re}\left(\dot{\mathbf{G}}^{\mathrm{T}}\left[\frac{1}{p-\rho_m}\right]_M\right)$	$\dot{\mathbf{G}} = \left[\dot{G}_m\right]_M = \left[k_m e^{-j\phi_m}\right]_M$, $\mathbf{q} = [\rho_m]_M = [-\alpha_m + jw_m]_M$
3.	Time dependent transfer function	$K(p,t) = \mathrm{Re}\left(\dot{\mathbf{G}}^{\mathrm{T}}\left[\frac{1-e^{-(p-\rho_m)t}}{p-\rho_m}\right]_M\right)$	$K(p,t) = \int_0^t g(\tau) e^{-p\tau} d\tau$
4.	Forced components	$y_1(t) = \mathrm{Re}\left(\dot{\mathbf{Y}}^{\mathrm{T}} e^{\mathbf{p}t}\right)$	$\dot{\mathbf{Y}} = \mathrm{diag}(\dot{\mathbf{X}}) K(\mathbf{p})$
5.	Free components	$y_2(t) = \mathrm{Re}\left(\dot{\mathbf{V}}^{\mathrm{T}} e^{\mathbf{q}t}\right)$	$\dot{\mathbf{V}} = \mathrm{diag}(\dot{\mathbf{G}}) X(\mathbf{q})$
6.	Filter reaction	$y(t) = \mathrm{Re}\left(\dot{\mathbf{Y}}(t)^{\mathrm{T}} e^{\mathbf{p}t}\right)$	$\dot{\mathbf{Y}}(t) = \mathrm{diag}(\dot{\mathbf{X}}) K(\mathbf{p}, t)$

Table 1.

The first method is a complex amplitude method generalization for definition of forced and free components for filter reaction at semi-infinite or finite input signals. The advantages of this method are related to simple algebraic operations, which are used for determining the parameters of linear system reaction (filter, linear circuit) components to an input action described by a set of semi-infinite or finite damped oscillatory components. To analyze a filter

it is needed to use simple algebraic operations and operate a set of complex amplitudes and frequencies of forced and free filter reaction components. In this case, there are simple relations between complex amplitudes of output signal forced components and complex amplitudes of an input signal (item 4 Table 1), between complex amplitudes of output signal free components and complex amplitudes of a filter impulse function (item5).

The time-and-frequency approach in the second analysis method applies to a filter transfer function, i.e. time dependent transfer function of the filter is used [6,8]. In that case, instead of two sets of filter reaction components only one of them may be used.

Analysis methods given in the Table 1 enable to reduce effectively the computational costs when performing a filter analysis by using simple algebraic operations to determine the forced and free components of a filter reaction to an input action as a set of damped oscillatory components. Therefore, the considered analysis methods for linear systems (filters) can be effectively applied for performance analysis of signal processing by frequency filters.

- **How can we find poles and zeroes (Answer: Z Transform)?**

It is quite difficult to qualitatively analyze the Laplace transform and Z-transform, since mappings of their magnitude and phase or real part and imaginary part result in multiple mappings of 2-dimensional surfaces in 3-dimensional space. For this reason, it is very common to examine a plot of a transfer function's poles and zeros to try to gain a qualitative idea of what a system does.

Once the Z-transform of a system has been determined, one can use the information contained in function's polynomials to graphically represent the function and easily observe many defining characteristics. The Z-transform will have the below structure, based on Rational Functions:

X(z)=P(z)Q(z)Xz Pz Qz

The two polynomials, P(z)Pz and Q(z)Qz, allow us to find the poles and zeros of the Z-Transform.

zeros

The value(s) for zz where P(z)=0Pz 0.

The complex frequencies that make the overall gain of the filter transfer function zero.

poles

The value(s) for zz where Q(z)=0Qz 0.

The complex frequencies that make the overall gain of the filter transfer function infinite.

Below is a short program that plots the poles and zeros from the above example onto the Z-Plane.

```
% Set up vector for zeros
z = [j ; -j];

% Set up vector for poles
p = [-1 ; .5+.5j ; .5-.5j];

figure(1);
zplane(z,p);
title('Pole/Zero Plot for Complex Pole/Zero Plot Example');
```

- **How do poles and zeroes impact frequency response and what is pole + zero cancellation?**

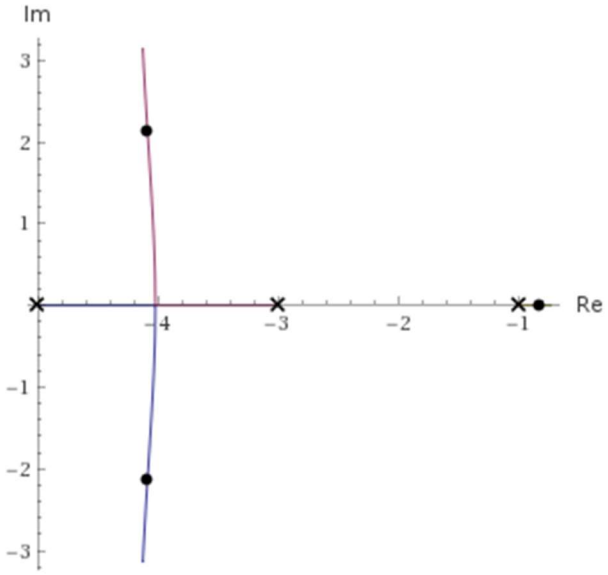
The cancellation is usually done when designing a controller to achieve some control goals (to increase the speed of the system, to reduce the tracking error, etc…). A common goal is to *cancel slow poles* (poles with negative real parts, thus stable, but situated near to the imaginary axis).

Practical control principles tell that you should add zeros with the controller's transfer function only to **cancel stable poles (have negative real part) which are quite far from the imaginary axis**.

The cancellation in practical terms is never exact, and so you should not try to cancel unstable poles (on the real positive *half plane (HP)*) or in the negative real half plane, but near the axis. If you apply cancellation to poles well inside the negative HP, usually there is no harm done to the systems' stability if the cancellation is not perfect (which is the practical case).

Under the hypothesis that you do a perfect zero cancellation, then in many cases you change a lot the shape of the root locus (RL). In fact the idea of designing a controller, under the analysis of the RL, is to change the paths of the RL such that the dominant pair of poles is located (by suitable values of the controller's parameters) in points of the ss-plane which satisfy the controlling goals. If you mess with (cancel) dominant poles, then you change the RL shape in the important parts (the paths of the dominant poles).

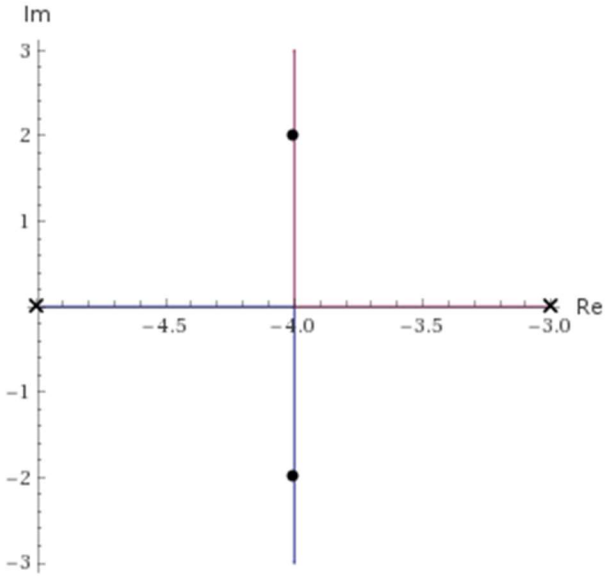
For instance, the root locus of
(s+1/2)(s+1)(s+3)(s+5)(s+1/2)(s+1)(s+3)(s+5)
is below, and it has a slow pole at s=−1s=−1 near the zero at s=−1/2s=−1/2:



(shown for gain between 0 and 10)

By cancelling the dominant pole with the zero after shifting it to the location of the pole, s=−1s=−1, the dominant poles scenario changes and the system is faster, without the pole at s=−1s=−1…

1(s+3)(s+5)1(s+3)(s+5)



(shown for gain between 0 and 10)