Sampling Rate Conversion

Overview

In discrete-time signal processing, it is often necessary to convert a signal from one sampling rate to another. A common example is the conversion from the sampling rate of a compact disk (CD); signal (44.1 KHz) to that of a Digital Audio Tape (DAT) signal (48 KHz). Another example is the aud io standard in High-Definition Television (HDTV) transmission, where at least three sampling rates are supported (32, 44.1, and 48 KHz). Although in principle we may convert the signal back to analog form and resample at the desired rate, it is usually preferable to perform t he entire conversion digitally. This is clue to many considerations including the fact that conversion to analog form often introduces noise in the signal, and that digital signal processing can be much more cost-effective and flexible.

This MATLAB project asks you to perform a sampling rate conversion on segments of audio signals. The input audio signals are quantized to 8 bits and sampled with a sampling frequency of 11,025 Hz. You are required to convert the signal to a sampling rate of 24,000 Hz in a computationally efficient manner. Although one conceptual way of realizing this sampling rate conversion process is to upsample the signal, lowpass filter, and downsample it, a more clever implementation can lead to an implementation that is many times more efficient. To do this, you can exploit various aspects of class to optimize the system, such as multistage filter implementation, filter design, and polyphase implementation. By the end of the project, you hopefully will have a much better understanding of both the theoretical aspect of the system as well as various issues in implementing a practical DSP system at a software level.

Project Goal

A sampling rate converter which produces an output signal with a sampling rate which is $\frac{L}{M}$ times the original sampling rate can be specified as shown in Figure 1.

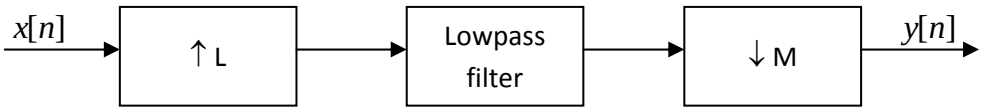

Figure 1: Sampling rate conversion system.

For an ideal sampling rate converter, the lowpass filter in Figure 1 is an ideal lowpass filter with cutoff frequency $\omega_c = \min\left(\frac{\pi}{M}, \frac{\pi}{L}\right)$ The goal of this project is to design an efficient DSP algorithm that implements the system in Figure 1 subject to the following constraints:

- The system performs the correct sampling rate conversion from 11,025 Hz to 24,000 Hz. In particular, do not assume that 11,025 Hz ≈ 11,000 Hz.

If the system in Figure 1 is an ideal sampling rate converter , when the input $x[n]$ is a unit impulse $\delta[n]$, the output $y[n]$ has a Fourier transform $Y(e^{j\omega})$ which corresponds to an ideal lowpass filter with a cutoff frequency at ω_c = (11,025/ 24,000)π. For an equivalent system which you are to implement, when the input *x[n]* is a unit impulse $\delta[n]$, the output *y[n]* must have a Fourier transform $Y(e^{j\omega})$ which is an approximation of a lowpass filter, and meets the specifications shown below in Table 1.

Passband Cutoff (ω_p)	$\frac{11{,}025\pi}{24{,}000}$
Passband Ripple	±0.1 dB or less
Stopband Frequency (ω_s)	$1.2\omega_p$
Stopband Attenuation	70 dB or more
Phase Constraints	|max grpdelay - min grpdelay| ≤ 720 in the passband

Table 1

If your sampling rate conversion system functions properly, you should meet the specifications in Table 1, and when you play the output audio signal at 24,000 Hz, it should sound the same as the input audio signal played at 11,025 Hz.

You should get a rough estimate of the efficiency of your design by determining how much computation was required to perform the sampling rate conversion on the `Wagner.wav` signal. The method which you will use to count the number of operations will be described shortly.

You are to write a MATLAB function `srconvert` such that the command `srconvert(in)` takes the input signal `in` with an associated sampling rate of 11,025 Hz, and returns an output signal at a sampling rate of 24,000 Hz. (See the section on writing MATLAB functions.) Once your function `srconvert.m` is finalized, run the command `y=srconvert([1 zeros(1,3000)]);`. This will produce a vector `y` which contains the response of your system to a unit impulse. Then call `verify(y)` to verify that your design meets the design specification.

The Files

The project zip file contains the audio files and MATLAB functions which you will need for this project. To access the zip file, click on the link below where you found this file on the web site.

In testing your system, you may find it helpful to first use a unit impulse as input to your system and then later try using real audio signals as inputs and listening to the outputs. To load a sound file into a vector `x`, type `x=wavread('filename.wav');`. To write a sound file for the MATLAB vector `x` into the current directory, type `wavwrite(x,sampfreq, 'filename.wav')`. You may test your system using any of the audio signals, although you should use the `Wagner.wav` signal to benchmark your system performance.

MATLAB Utilities

You may find the following MATLAB functions useful for your design.

`examlpf(h, wp, vs)` - allows you to examine the passband ripple, the group delay, as well as the stopband attenuation of the lowpass filter whose impulse response is `h`. It generates three simultaneous plots. The first one zooms in on the passband with cutoff frequency `wp`. The second plot measures the group delay in the passband, and the third plot is simply the magnitude response over the entire frequency range $(-\pi, \pi)$. The passband (`wp`) and stopband (`ws`) cutoff frequencies are normalized by $\pi/2$. That is, a cutoff frequency at $\pi/2$ should be entered as `0.5`.

`poly1(h, M)` - returns a matrix `E` whose i^{th} row corresponds to the i^{th} polyphase components of the FIR filter `h`, obtained via type I decomposition with downsampling factor `M`.

`fftfilt(h , x)` - convolves the signal `x` with the filter `h` using the FFTs. Beware that this command might not always leads to a more efficient implementation than `conv(h ,x)`, depending on the length of the signals.

`upsample(h ,L)` - returns an upsampled version of `h` by a factor `L`, without any interpolation.

`downsample(h ,M)` - downsamples `h` by `M`.

Computation Counts

MATLAB does not automatically record the number of floating point operations (flops) in your algorithm, so to measure your computational efficiency, you'll have to determine the required number of multiplies and adds yourself. You can make this process somewhat automated by inserting a counter to be updated each time the `filter` function is called, but you should also do a hand calculation to double check.

Do not include in the flops count the computation involved in designing any filters you need.

Example MATLAB Function

To give you an example of how to write a MATLAB function, here is the code from the file `poly1.m`.

```
function E=poly1(h,M)
%
% Performs type I polyphase decomposition of h in M components.
% The ith row of E corresponds to the ith polyphase component.
% Assumes that the first point of h is index 0.
%
h = [h zeros(1, ceil(length(h)/M)*M-length(h))];
E = reshape(h, M, length(h)/M);
```

Tips

The old homework problem in the Appendix might be useful as a guideline to efficient implementation of an interpolation filter. You may also find useful page 85 of the Vaidyanathan paper cited below.

> P. P. Vaidyanathan, ``Multirate digital filters, filter banks, polyphase networks, and applications: A tutorial,'' Proceedings of the IEEE, vol. 78, pp. 56-92, Jan. 1990.
>
> http://www.systems.caltech.edu/dsp/ppv/papers/ProcIEEEmultirateTUTExtra.pdf (Available 7/08)

Appendix – An Old Homework Problem:

Let $x[n]$ represent a signal which is obtained by sampling a continuous time audio signal $x_c(t)$ using an ideal C/D converter shown in Figure A1. Assume that the sampling rate is $1/T = 44.1$ kHz .

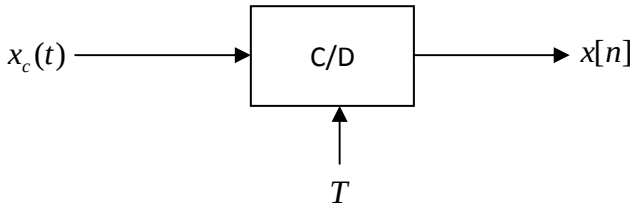

Figure A1

We wish to design a 4x (4 times) oversampling digital FIR interpolation filter. One way to do this is to use the single-stage design of System 1 shown in Figure A2.

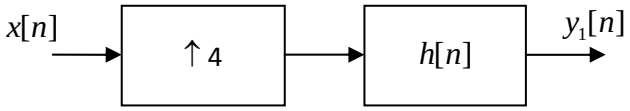

Figure A2: System 1

The filter $h[n]$ with Fourier transform $H(e^{j\omega})$ is designed using the window method with a Kaiser window. Suppose the filter $h[n]$ is designed to meet Specification 1:

Specification 1:

$$1-\delta \le \left|H(e^{j\omega})\right| \le 1+\delta, \qquad |\omega| \le \omega_1$$

$$\left|H(e^{j\omega})\right| \le \delta, \qquad \omega_2 \le |\omega| \le \pi$$

where $\omega_1 = 0.23\pi$, $\omega_2 = 0.27\pi$, and $\delta = 10^{-4}$.

(a) Using a Kaiser window, estimate the length of the filter $h[n]$ which meets Specification 1.

(b) Using the filter length estimate for $h[n]$ in part(a), estimate the number of multiplications per second required in System 1 if the system is implemented in polyphase form using four polyphase components.

An alternate way to design the 4x oversampling interpolation filter is to use the two-stage design of System 2 shown in Figure A3.

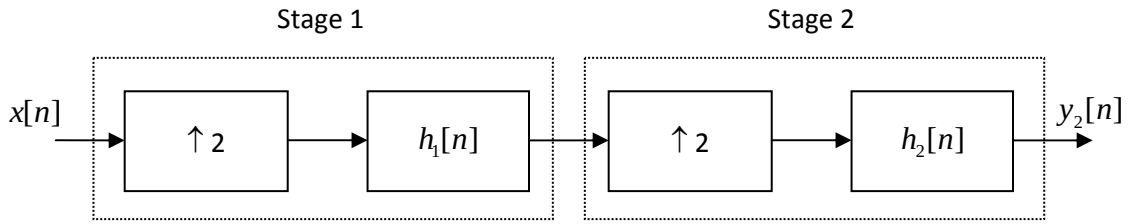

Figure A3: System 2

The filters $h_1[n]$ and $h_2[n]$ with frequency responses $H_1(e^{j\omega})$ and $H_2(e^{j\omega})$, respectively, are designed using the window method with Kaiser windows. Suppose that these filters are designed to meet Specification 2.

Specification 2:

$$1-\delta' \le \left|H_1(e^{j\omega})\right| \le 1+\delta', \qquad |\omega| \le 2\omega_1$$
$$\left|H_1(e^{j\omega})\right| \le \delta', \qquad 2\omega_2 \le |\omega| \le \pi$$
$$1-\delta' \le \left|H_2(e^{j\omega})\right| \le 1+\delta', \qquad |\omega| \le \omega_1$$
$$\left|H_2(e^{j\omega})\right| \le \delta', \qquad \pi-\omega_1 \le |\omega| \le \pi$$

where $\delta' = \sqrt{1+\delta} - 1$, ω_1, ω_2, and δ are given previously.

(c) Show that System 2 is equivalent to System 1 with

$$H(e^{j\omega}) = H_1(e^{j\omega})H_2(e^{j\omega})$$

(d) Show that if the filters $h_1[n]$ and $h_2[n]$ are designed using Kaiser windows to meet Specification 2, then the resulting equivalent system in System 1 will have a filter $H(e^{j\omega})$ which meets Specification 1. (You may assume that the filters designed have magnitude responses which are monotonically decreasing in the transition band from the passband to the stopband).

(e) Using Kaiser windows, estimate the lengths of the filters $h_1[n]$ and $h_2[n]$ which meet Specification 2.

(f) Using the filter length estimates for $h_1[n]$ and $h_2[n]$ in part (e), estimate the number of multiplications per second required in System 2 if each stage is separately implemented in polyphase form using two polyphase components. What are the computational savings in multiplications, if any, over System 1?