ECE 438 - Laboratory 4b
Interpolation and Decimation

Last updated on February 4, 2022

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import soundfile as sf
import IPython.display as ipd
from scipy import signal
```

In [2]:
```
# make sure the plot is displayed in this notebook
%matplotlib inline
# specify the size of the plot
plt.rcParams['figure.figsize'] = (16, 6)

# for auto-reloading extenrnal modules
%load_ext autoreload
%autoreload 2
```

1. Discrete-Time Interpolation

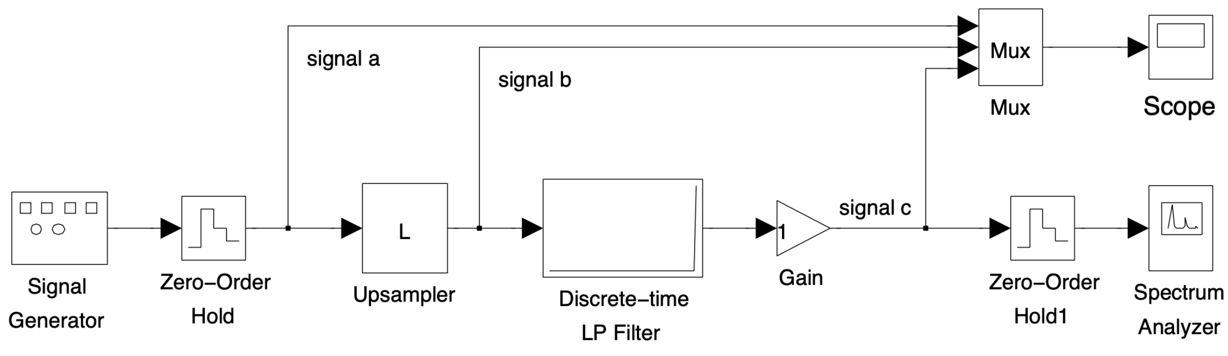

Figure 1: Simulink model for discrete-time interpolation.

In the previous lab (Lab 4a), we saw that the frequency content of a signal must be limited to half the sampling rate in order to avoid aliasing effects in the reconstructed signal. However, reconstruction can be difficult if the sampling rate is chosen to be just above the Nyquist frequency. Reconstruction is much easier for a higher sampling rate because the sampled signal will better “track” the original analog signal.

From another perspective, the analog output filter must have a very sharp cutoff in order to accurately reconstruct a signal that was sampled just above the Nyquist rate. Such filters are difficult and expensive to manufacture. Alternatively, a higher sampling rate allows the use analog output filters that have a slow roll-off. These filters are much less expensive. However, a high sampling rate is not practical in most applications, as it resultsin unnecessary samples and excessive storage requirements.

A practical solution to this dilemma is to *interpolate* the digital signal to create new (artificial) samples between the existing samples. This may be done by first upsampling the digital representation, and then filtering out unwanted components using a discrete-time filter. This discrete-time filter serves the same purpose as an analog filter with a sharp cutoff, but it is generally simpler and more cost effective to implement.

Upsampling a signal by a factor of L is simply the process of inserting $L-1$ zeros in between each sample. The frequency domain relationship between a signal $x(n)$ and its upsampled version $z(n)$ can be shown to be the following

$$Z(\omega) = X(\omega L). \tag{1}$$

Therefore the DTFT of $z(n)$ is simply $X(\omega)$ compressed in frequency by a factor of L. Since $X(\omega)$ has a period of 2π, $Z(\omega)$ will have a period of $2\pi/L$. All of the original information of $x(n)$ will be contained in the interval $[-\pi/L, \pi/L]$ of $Z(\omega)$, and the new aliases that are created in the interval $[-\pi, \pi]$ are the unwanted components that need to be filtered out. In the time domain, this filtering has the effect of changing the inserted zeros into artificial samples of $x(n)$, commonly known as *interpolated* samples.

Figure 1 shows a Simulink model that demonstrates discrete-time interpolation. The interpolating system contains three main components: an upsampler which inserts $L-1$ zeros between each input sample, a discrete-time low pass filter which removes aliased signal components in the interpolated signal, and a gain block to correct the magnitude of the final signal. Notice that **signal a** is the input discrete-time signal while **signal c** is the final interpolated discrete-time signal.

Open the experiment by double clicking on the icon labeled ***Discrete Time Interpolator***. The components of the system are initially set to interpolate by a factor of 1. This means that the input and output signals will be the same except for a delay. Run this model with the initial settings, and observe the signals on the ***Scope***.

Simulink represents any discrete-time signal by holding each sample value over a certain time period. This representation is equivalent to a sample-and-hold reconstruction of the underlying discrete-time signal. Therefore, a continuous-time ***Spectrum Analyzer*** may be used to view the frequency content of the output **signal c**. The ***Zero-Order Hold*** at the ***Gain*** output is required as a buffer for the ***Spectrum Analyzer*** in order to set its internal sampling period.

The lowest frequency component in the spectrum corresponds to the frequency content of the original input signal, while the higher frequencies are aliased components resulting from the sample-and-hold reconstruction. Notice that the aliased components of **signal c** appear at multiples of the sampling frequency of 1 Hz. Print the output of the ***Spectrum Analyzer***.

Exercise 1

1.Submit your plot of signal c and its frequency spectrum. Circle the aliased components in your plot.

insert your answer here

Next modify the system to upsample by a factor of 4 by setting this parameter in the ***Upsampler***. You will also need to set the ***Sample time*** of the DT filter to 0.25. This effectively increases the sampling frequency of the system to 4 Hz. Run the simulation again and observe the behavior of the system. Notice that zeros have been inserted between samples of the input signal. After you get an accurate plot of the output frequency spectrum, print the output of the ***Spectrum Analyzer***.

Notice the new aliased components generated by the upsampler. Some of these spectral components lie between the frequency of the original signal and the new sampling frequency, 4 Hz. These aliases are due to the zeros that are inserted by the upsampler.

2.Submit your plot of signal c and its frequency spectrum. On your frequency plot, circle the first aliased component and label the value of its center frequency. Comment on the shape of the envelope of the spectrum.

insert your answer here

Notice in the previous ***Scope*** output that the process of upsampling causes a decrease in the energy of the sample-and-hold representation by a factor of 4. This is the reason for using the ***Gain*** block.

Now determine the gain factor of the ***Gain*** block and the cutoff frequency of the ***Discrete-time LP filter*** needed to produce the desired interpolated signal. Run the simulation and observe the behavior of the system. After you get an accurate plot of the output frequency spectrum, print the output of the ***Spectrum Analyzer***. Identify the change in the location of the aliased components in the output signal.

3.Submit your plot of signal c and its frequency spectrum. Give the values of the cutoff frequency and gain that were used. On your frequency plot, circle the location of the first aliased component. Explain why discrete-time interpolation is desirable before reconstructing a sampled signal

insert your answer here

2. Discrete-Time Decimation

In the previous section, we used interpolation to increase the sampling rate of a discretetime signal. However, we often have the opposite problem in which the desired sampling rate is lower than the sampling rate of the available data. In this case, we must use a process called *decimation* to reduce the sampling rate of the signal.

Decimating, or *downsampling*, a signal $x[n]$ by a factor of D is the process of creating a new signal $y[n]$ by taking only every Dth sample of $x[n]$. Therefore $y[n]$ is simply $x[Dn]$. The frequency domain relationship between $y[n]$ and $x[n]$ can be shown to be the following:

$$Y(\omega) = \frac{1}{D}\sum_{k=0}^{D-1} X\left(\frac{\omega - 2\pi k}{D}\right). \tag{2}$$

Notice the similarity of (2) to the sampling theorem equation in (1) in Lab 4a. This similarity should be expected because decimation is the process of sampling a discrete-time signal. In this case, $Y(\omega)$ is formed by taking $X(\omega)$ in the interval $[-\pi, \pi]$ and expanding it in frequency by a factor of D. Then it is repeated in frequency every 2π, and scaled in amplitude by $1/D$. For similar reasons as described for equation (1) in Lab 4a, aliasing will be prevented if in the interval $[-\pi, \pi]$, $X(e^{j\omega})$ is zero outside the interval $[-\pi/D, \pi/D]$. Then (2) simplifies to

$$Y(\omega) = \frac{1}{D} X\left(\frac{\omega}{D}\right) \quad \text{for } \omega \in [-\pi, \pi]. \tag{3}$$

A system for decimating a signal is shown in Figure 2. The signal is first filtered using a low pass filter with a cutoff frequency of $\pi/2$ rad/sample. This insures that the signal is band limited so that the relationship in (3) holds. The output of the filter is then subsampled by removing every other sample.

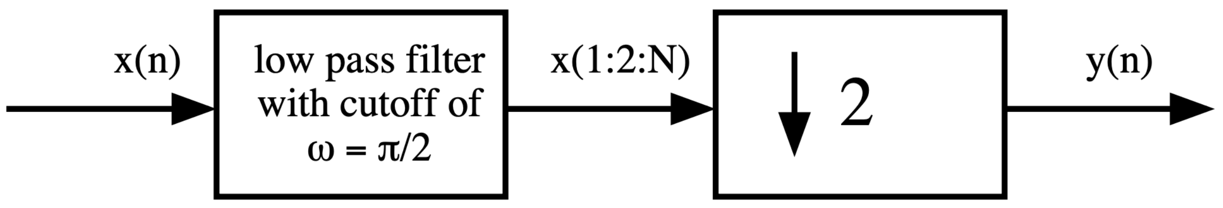

Figure 2: This system decimates a discrete-time signal by a factor of 2

Exercise 2

1. Read in the signal contained in `music.au` using `music, fs = sf.read("music.au")`, and then play it back with sound using `ipd.Audio(music, rate=fs)`.

In [3]:
```python
# insert your code here
```

The signal contained in `music.au` was sampled at 16 kHz, so it will sound much too slow when played back at the default 8 kHz sampling rate.

2. To correct the sampling rate of the signal, form a new signal, `sig1`, by selecting every other sample of the music vector. Play the new signal, and listen carefully to the new signal.

In [4]:
```python
# insert your code here
```

3. Compute a second subsampled signal, `sig2`, by first low pass filtering the original music vector using a discrete-time filter of length 20, and with a cutoff frequency of $\pi/2$. Then decimate the filtered signal by 2, and listen carefully to the new signal.

Hint: You can create a low-pass filter by using the function `signal.firwin` (https://docs.scipy.org/doc/scipy/reference/reference/generated/scipy.signal.firwin.html) from `scipy` and use the following syntax:

```
signal.firwin(M, W/np.pi)
```

which return a length `M` low-pass filter with cutoff frequency `W` rad/sample.

Use `np.convolve` (https://numpy.org/doc/stable/reference/generated/numpy.convolve.html) to get the convolution of two signals.

In [5]:
```python
# insert your code here
```

4. Comment on the quality of the audio signal generated by using the two decimation methods. Was there any noticeable distortion in sig1? If so, describe the distortion.

insert your answer here