EE386 Digital Signal Processing Lab **Jul-Dec 2021**

Experiment 2

Author: Prasad Fidelis D'sa *Email:* prasadfidelisdsa.191mt032@nitk.edu.in

1 Introduction

This exercise is based on the concepts of sampling and aliasing. We begin by plotting and summing signals of various frequencies to form a cumulative signal. We then sample the signal at different rates and investigate and compare the results. This is followed by their interpolation to reconstruct the original signal. Finally we plot their Energy Density Spectrum. Generation of tones and the effect of sampling rate is then investigated followed by the downsampling and upsampling of a given track. In our example we have $\alpha = 1 + mod(260, 4) = 1$.
All the code for this exercise and the relevant files are included in the `.zip` file submitted along with this report.

2 Problems

1. Sampling and frequency-domain aliasing

(Part 1: Plotting component signals over $\frac{1}{\alpha}$ second duration)

(Solution)

For our choice of $\alpha = 1$ the three waveforms are

- $\cos(10\pi t)$
- $0.5\cos(12\pi t)$
- $0.25\cos(20\pi t)$

The three waveforms are plotted over the interval $t \in [0, 1]$ in Figure 1

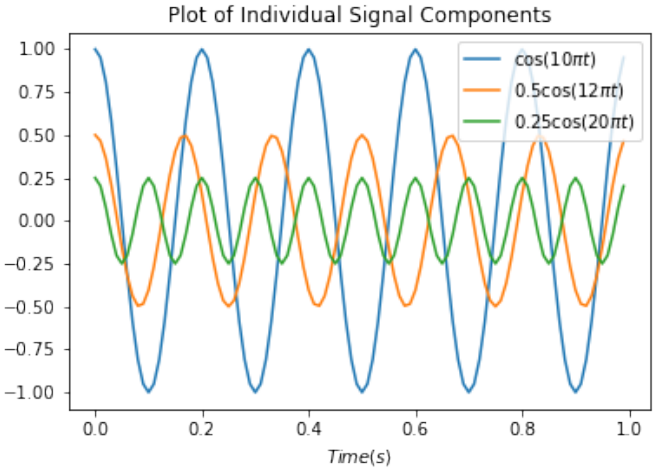

Figure 1: Individual signal components that make up the continuous signal in Figure 2

(Part 2: Plotting the cumulative signal)

(Solution)

The cumulative signal $y(t)$ is the sum of the three component signals in Figure 1 and is continuous over the interval $t \in [0, 1]$

$$y(t) = \cos(10\pi t) + 0.5\cos(12\pi t) + 0.25\cos(20\pi t) \tag{2.1}$$

Figure 2 shows the plot of signal $y(t)$

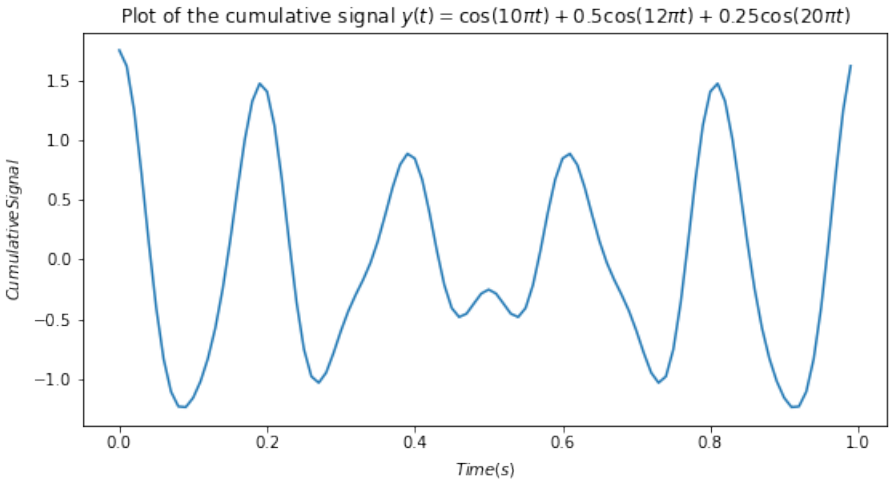

Figure 2: The cumulative signal comprising of the sum of the individual waveforms in Figure 1

(Part 3: Sampling the waveform at different sampling rates)

(Solution)

Consider an analog sinusoidal signal of the form

$$x_a(t) = A\cos(2\pi Ft + \theta) \tag{2.2}$$

which, when sampled periodically at a rate $F_s = \frac{1}{T}$ samples per second. yields

$$x_a(nT) \equiv x(n) = A\cos(2\pi FnT + \theta) = A\cos(\frac{2\pi nF}{F_s} + \theta) \tag{2.3}$$

we note that the frequency variable F and discrete frequency variable f are linearly related as

$$f = \frac{F}{F_s} \tag{2.4}$$

The relation in Eq (2.4) justifies the name relative or normalized frequency, which is sometimes used to describe the frequency variable f and implies that we can use f to determine the frequency F in hertz only if the sampling frequency F_s is known.
We recall that for discrete-time sinusoids

$$-\frac{1}{2} \leqslant f < \frac{1}{2} \tag{2.5}$$

or equivalently

$$-\frac{F_s}{2} \leqslant F < \frac{F_s}{2} \tag{2.6}$$

Periodic sampling of a continuous-time signal implies a mapping of the infinite frequency range for the variable F into a finite frequency range for the variable f. Since the highest frequency in a discrete-time signal is $\omega = \pi$ or $f = \frac{1}{2}$, it follows that with a sampling rate F_s, the corresponding highest value of F is

$$F_{max} = \frac{F_s}{2} = \frac{1}{2T} \tag{2.7}$$

This introduces an ambiguity as we shall see. Any frequency above $\frac{F_s}{2}$ or below $-\frac{F_s}{2}$ results in samples that are identical with a corresponding frequency in the range $-\frac{F_s}{2} \leqslant F < \frac{F_s}{2}$
As a consequence of this we can say that the frequencies

$$F_k = F + kF_s \tag{2.8}$$

where k is an integer are indistinguishable from the frequency F after sampling and hence they are aliases of F.

In our example the cumulative signal of Figure 2 is sampled at rates

- $F_s = 14\alpha = 14$ samples/second
- At the Nyquist rate of the signal $F_{ny} = 20\alpha = 20$ samples/second
- At a sampling rate 3α such that 6α Hz is aliased to 3α Hz.

We calculate 3α as the sampling rate by setting $k = 1$ in Eq (2.8)
Figure 3 shows the sampled signal at different rates. We note that the sampling rates are less than Nyquist frequency for the first and third plots.

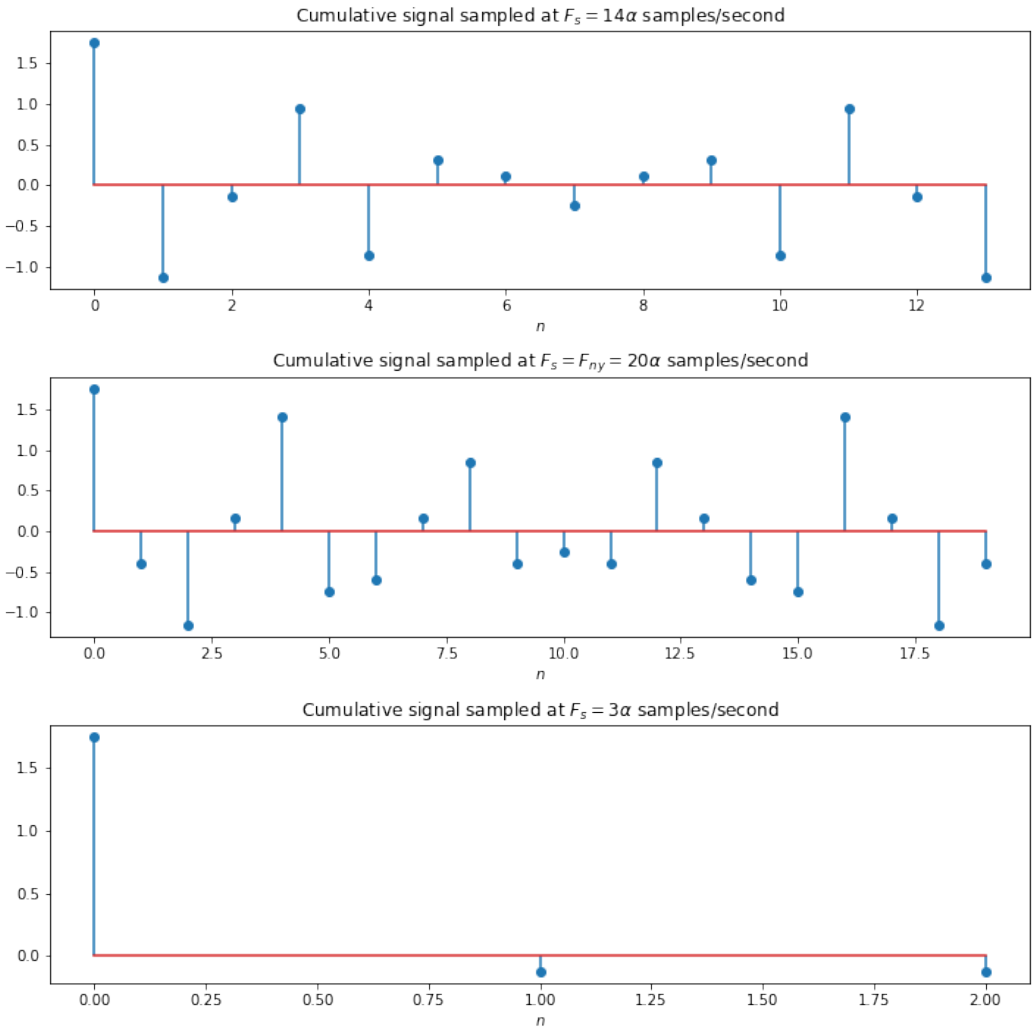

Figure 3: Sampling of the cumulative signal in Figure 2 at different sampling rates

(Part 4: Linear Interpolation of the sampled responses)
(Solution)
Interpolation is the process of reconstructing a Continuous Time signal $x_a(t)$ from its samples

$x(n) = x_a(nT)$. The ideal interpolation given the correct sampling of a bandlimited signal will result in sinc interpolation. However, the interpolation methods that are more often used in practice: nearest neighbor interpolation, linear interpolation and spline interpolation.
Linear interpolation works by effectively drawing a straight line between two neighboring samples and returning the appropriate point along that line. The `plt.plot(x,y)` in Python performs this and hence we use it to reconstruct out signal from the samples in Figure 3.
Figure 4 shows the reconstructed signals corresponding to different sampling rates.

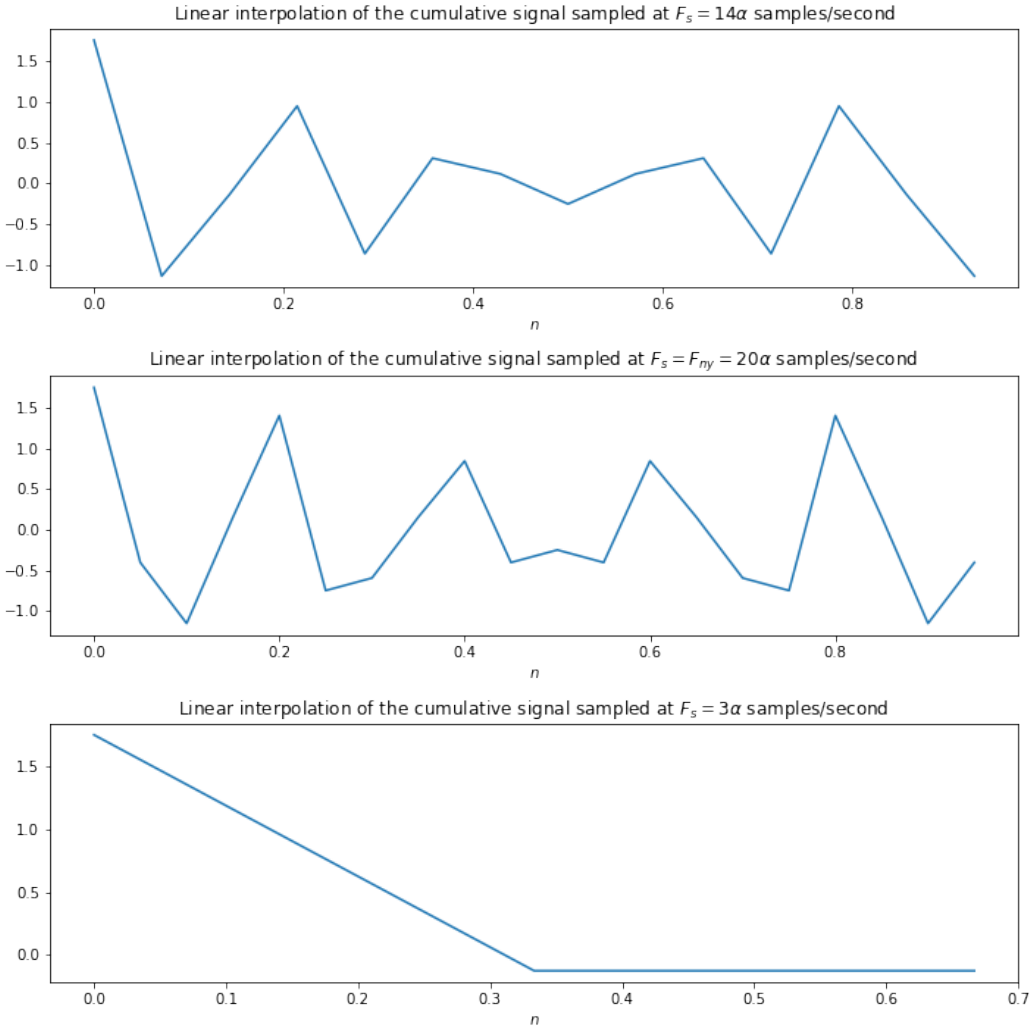

Figure 4: Linear interpolation of the samples in Figure 3 at different sampling rates

We observe that as the sampling frequency decreases, the signal separation also decreases. When the sampling frequency drops below the Nyquist rate, the frequencies will crossover and cause aliasing. This aliasing will result in the reconstructed signal not matching the original signal.
When the signal is sampled at the Nyquist frequency and above the data points produced by sampling will continue to retain the cyclic nature of the analog signal. Hence the reconstruction

bears resemblance to our continuous signal in Figure 2 and we can see that the the frequency of the interpolated wave is identical to the frequency of the original signal.
However, as soon as we reduce the sampling frequency to the point at which there are fewer than F_{ny} samples per cycle, this statement can no longer be made. At $F_s = 14\alpha$ samples/second and $F_s = 3\alpha$ samples/second the discrete-time waveform has acquired fundamentally new cyclical behaviour. Full repetition of the sampled pattern requires more than one sinusoid cycle, the frequencies of which are investigated in the next sub problem. The effect of insufficient sampling frequency makes it somewhat difficult to interpret the original signal. If we knew nothing about the original sinusoid and performed an analysis using the discrete-time waveform resulting from these samples, we would form seriously erroneous ideas about the frequency of the original signal. Furthermore, if all we have is the discrete data, it is impossible to know that frequency characteristics have been corrupted. Sampling has created a new frequency that was not present in the original signal, but we don't know that this frequency was not present.

In conclusion, when we sample at frequencies below the Nyquist rate, information is permanently lost, and the original signal cannot be perfectly reconstructed.

(Part 5: Energy Density Spectrum of the sampled responses using FFT)
(Solution)
FFT
A fast Fourier transform (FFT) is an algorithm that calculates the discrete Fourier transform (DFT) of some sequence efficiently. FFTs commonly change the time domain into the frequency domain. In Python FFT is calculated using `numpy.fft.fft(signal)`

To draw the Energy Density Spectrum using FFT we will utilise Parseval's relation for discrete-time aperiodic signals with finite energy.

$$E_X = \sum_{n=-\infty}^{\infty} |x(n)|^2 = \frac{1}{2\pi} \int_{-\pi}^{\pi} |X(\omega)|^2 \, d\omega \tag{2.9}$$

As in the case of continuous-time signals, the quantity

$$S_{XX}(\omega) = |X(\omega)|^2 \tag{2.10}$$

represents the distribution of energy as a function of frequency, and it is called the Energy Density Spectrum of $x(n)$.
Suppose now that the signal $x(n)$ is real, then it follows that

$$S_{XX}(-\omega) = S_{XX}(\omega) \tag{2.11}$$

Indeed, if we know $X(\omega)$ in the range $0 \leq \omega \leq \pi$, we can determine it for the range $-\pi \leq \omega < 0$ using the symmetry properties given above. Therefore the frequency-domain description of a real discrete-time signal is completely specified by its spectrum in the frequency range $0 \leq \omega \leq \pi$ or $0 \leq F \leq \frac{F_s}{2}$

Figure 5 shows the Energy Density Spectrum for the sampled signals in Figure 3

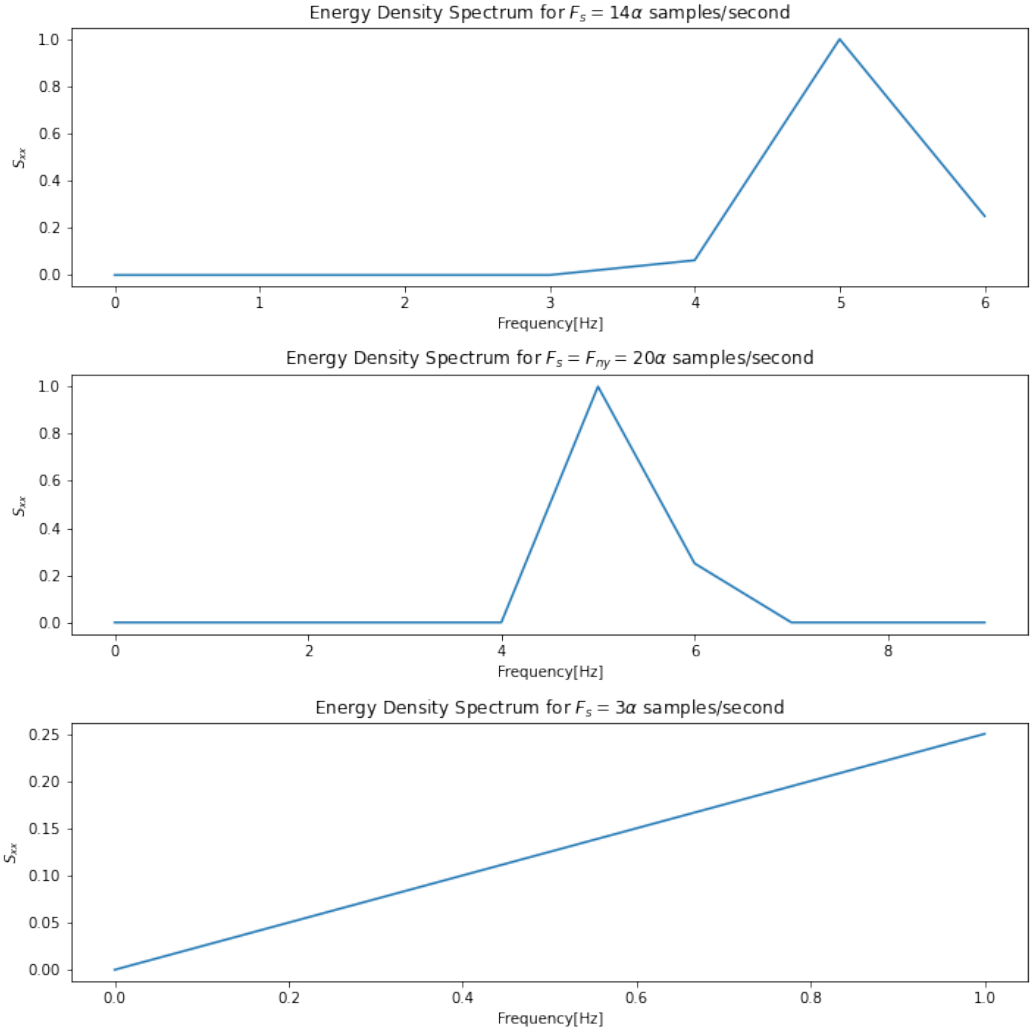

Figure 5: The Energy Density Spectrum of the samples in Figure 3

At the Nyquist frequency and above no frequency is aliased and components that are present will have the original frequencies at 5 Hz, 6 Hz and 10 Hz respectively.

For $F_s = 14$ samples/second the 5 Hz and 6 Hz signals will remain as they are. However, the 10 Hz frequency component will get aliased to $F - F_s = 10 - 14 = -4$ Hz or 4 Hz as seen in Figure 5

For $F_s = 3$ samples/second the 5 Hz, 6 Hz and 10 Hz frequency components will get aliased to $F - F_s$ i.e. to 2 Hz, 3 Hz for 5 Hz and 6 Hz respectively and $F - 3F_s = 1$ Hz for the 10 Hz component

2. Generating digital music

(Generation of tones at different sampling rates)

(Solution)

We generate the tones utilising the frequencies for equal tempered scale A_4= 440 Hz. Therefore We generate these tones at three different sampling rates of $F_s = 3000$ samples/second, $F_s = 22000$ samples/second and $F_s = 44100$ samples/second.

The `.wav` files are found with the format `A_Major_Scale_<samplingrate>.wav` in the `.zip` file submitted along with this report.

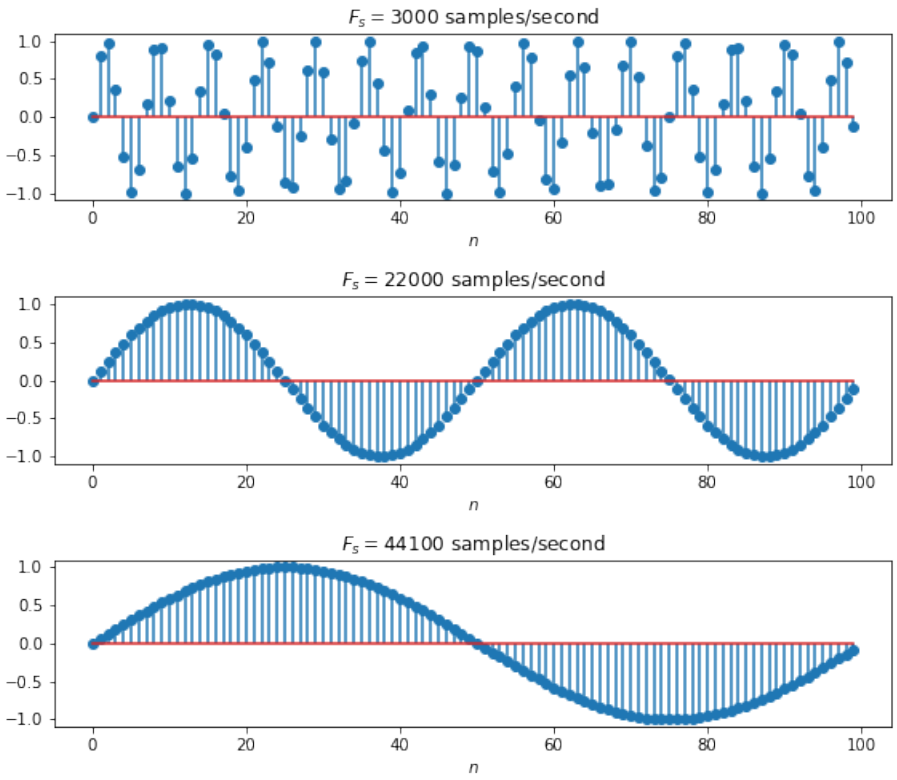

Figure 6: First hundred samples of the tones generated at sampling rate $F_s = 3000$ samples/second, $F_s = 22000$ samples/second, $F_s = 44100$ samples/second

Ideally once the sampling takes place at a rate above the Nyquist rate, further increases in the sampling frequency do not improve the quality of the reconstructed signal. This is true because of the ideal low-pass filter. In real-world applications, sampling at higher frequencies results in better reconstructed signals. In our example $F_{Ny} = 1760$ Hz. As we increase the sampling the rate the frequency resolution improves and the audio gets clearer.

3. Resampling

(Part 1: Downsampling of `Track001.wav`)

(Solution)

Downsampling (or subsampling) is the process of reducing the sampling rate of a signal. This is usually done to reduce the data rate or the size of the data. In our example to perform downsampling by an integer factor M, we decimate the signal by M i.e. keep only every Mth sample and discard the M-1 samples in between.

The Figure 7 below demonstrate downsampling for values of M equal to 2, 3 and 4. The sample numbers get divided by a factor of M as expected. The corresponding audio `.wav` files are included with the format `track_<samplingrate>.wav`. We can notice a clear difference in the audio for the different sampling rates. As we decrease the sampling rate the audio becomes less clear and noisy. The higher frequency notes get muffled indicating that some aliasing is taking place. However, the file size decreases.

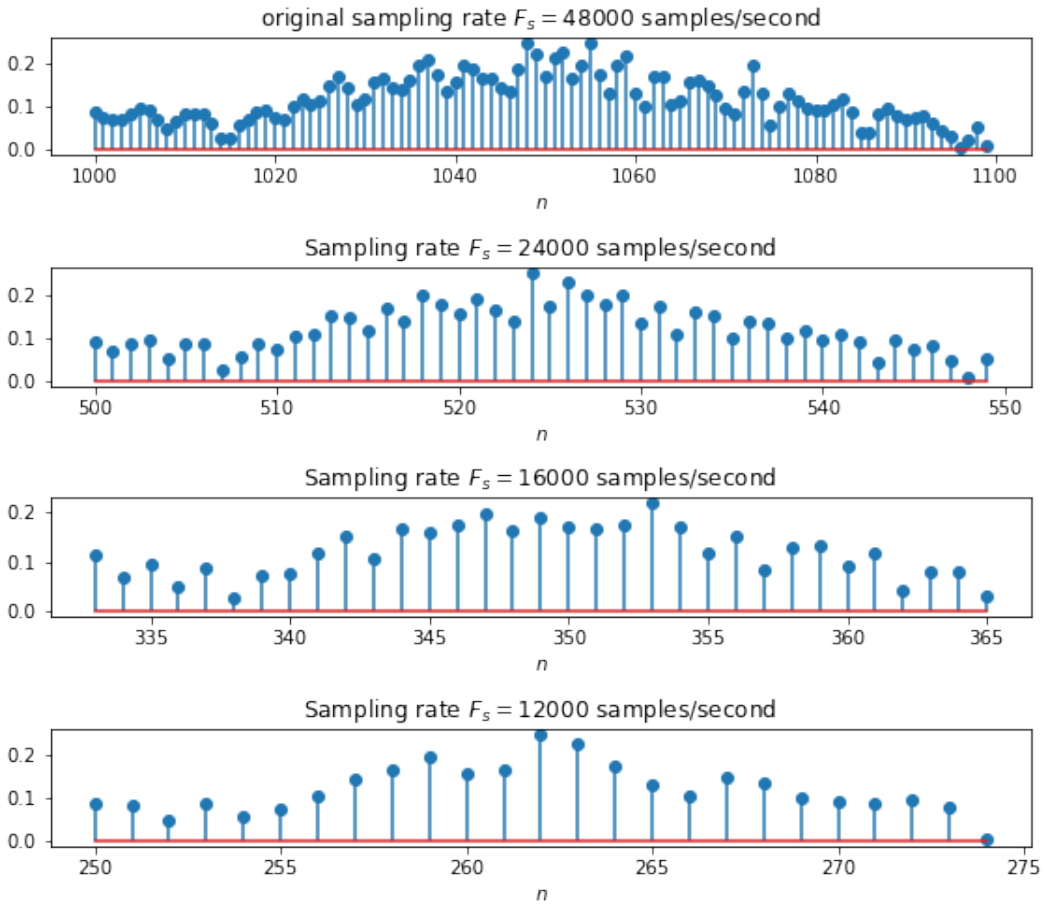

Figure 7: Track001.wav downsampled by M = 2, M = 3 and M =4

(Part 2: Upsampling of `Track001.wav`)

(Solution)

Upsampling is the process of inserting zero-valued samples between original samples to increase the sampling rate. Upsampling adds to the original signal undesired spectral images which are centered on multiples of the original sampling rate. Interpolation is the process of upsampling followed by filtering because the filtering removes the undesired spectral images. The goal is to create "in-between" samples from the original samples.

The final result is as if you had just originally sampled your signal at the higher rate.

We perform resampling using `scipy.signal.resample(x,num)` to resample x to num samples using Fourier method along the given axis.

The resampled signal starts at the same value as x but is sampled with a spacing of `len(x) /num*(spacingofx)`. Because a Fourier method is used, the signal is assumed to be periodic.

The Figure 8 below show the upsampling and interpolation of `Track001.wav` by a factor of 2 and 2.5 without any change in the frequency content.

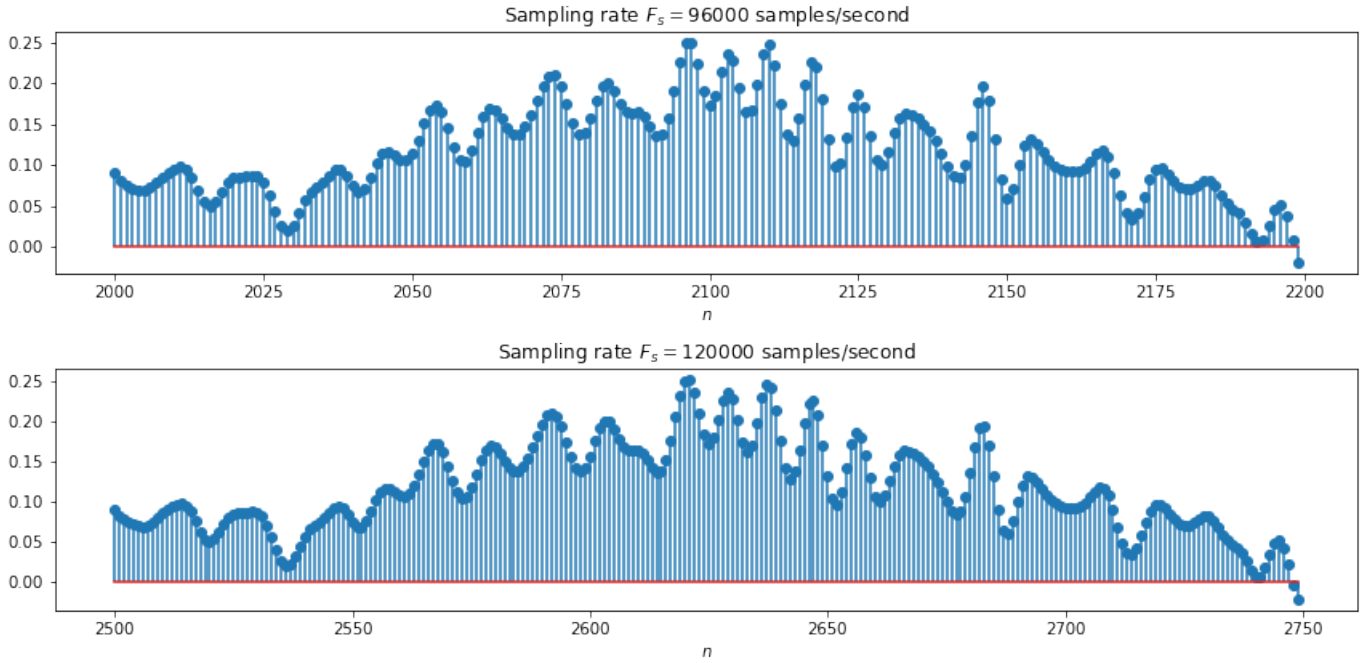

Figure 8: Track001.wav upsampled by factor of 2 and 2.5