Interpolation Methods for Audio Upsampling

Anand Kamble

Department of Scientific Computing
Florida State University

1 Introduction

In the domain of signal processing, upsampling plays a crucial role. Its significance lies in enhancing the quality of signals that were originally captured at a lower sample rate, as well as restoring older signals that need to be upsampled for compatibility with modern systems and softwares [1] [2].

In the context of audio, the need for upsampling or downsampling arises due to the existence of various standards for audio sample rates. This adjustment is made based on specific requirements. The following table lists some common sample rates employed for the storage and playback of audio [3].

Table 1: Sample Rates and Their Common Uses

Rate (kHz)	Use(s)
8	Telephone, walkie-talkie, wireless intercom, and microphones
32	digital video camcorder, Voice quality, digitizing FM radio
44.1	CD, DAT, digital recording software/hardware
48	Video, digital recording software/hardware
96	Digital recording software/hardware while recording/editing
192	Digital recording software/hardware while recording/editing

It is clear from the above table that audio can be in any format and needs to be converted to a suitable format depending on the application. For example, if an audio sample was recorded over a camcorder and needs to be distributed using CDs, it needs to be upscaled from 32kHz to 44.1kHz.

2 Methodology

In this project, our approach involves downsampling an audio signal followed by reconstruction to a higher sample rate using various interpolation methods. Our objective is to compare the outcomes and determine which method yields results closest to the original audio signal.

In this project, we have selected a sample audio, specifically the piano piece 'River Flows In You' composed by Yiruma, with an initial sample rate of 44.1 kHz.[4] We are downsampling this audio by a factor of 5 to 8820 Hz. It's worth noting that the chosen downsampled rate is slightly higher than the standard sample rate for telephones. For the sake of simplicity in this project, we are opting for downsampling by an integer factor rather than a fractional one.

2.1 Visualization

When visualizing the audio, we encounter approximately 1.6 to 7 million data points, posing challenges in observing data changes following downsampling. Figure 1, illustrates the plot of the downsampled audio data points.
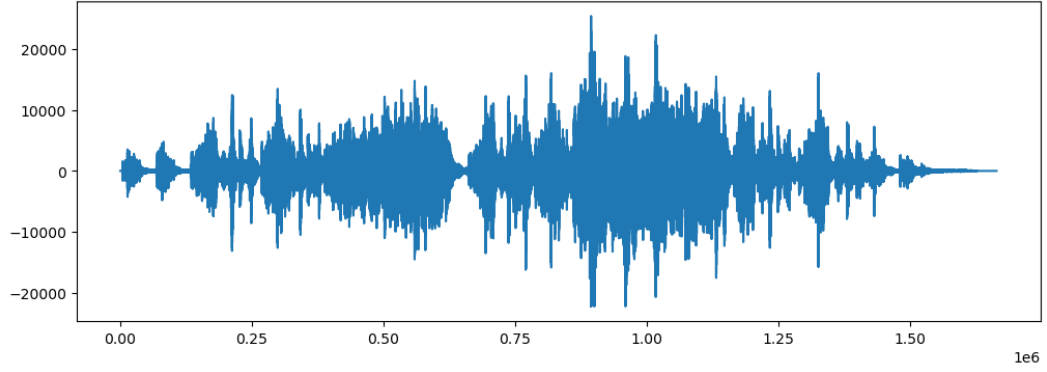

Figure 1: Downsampled audio

That being so, our focus will be narrowed to a specific region within the audio for enhanced visualization. This defined region will serve as our focal point for visualization throughout the project and it is represented by Figure 2.
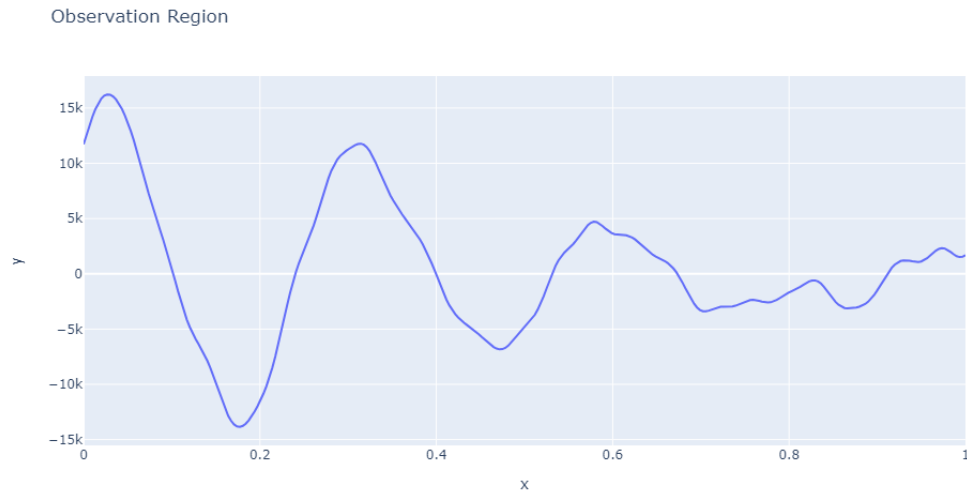

Figure 2: Observation Region

2.2 Down Sampling

From the original audio, we generate a downsampled version intended for use as a low-bitrate audio suitable for interpolation and subsequent comparison.

The downsampling process involves creating a new array, where each element corresponds to every n-th element in the original audio array. Let's denote the original audio matrix by A_o with length p and the downsampled audio as A_n with length q, where n is the downsampling factor.

$$A_n[i] = A_o[n.i] \ , \quad 0 \leq i \leq \frac{p}{n}$$

The down-sampling function is implemented as follows:

Listing 1: Down Sampling Function

```python
def downsample_audio(original_audio, n):
    A_n = []
    new_sample_rate = original_audio[0] // n

    # Iterate through the original audio data and select every nth
        element
    for i in tqdm(range(len(original_audio[1]))):
        if i % n == 0:
            A_n.append(original_audio[1][i])

    return new_sample_rate, A_n
```

Figure 3 illustrates the disparity between the original audio signal and its downsampled counterpart by plotting the data points within the specified observation region.
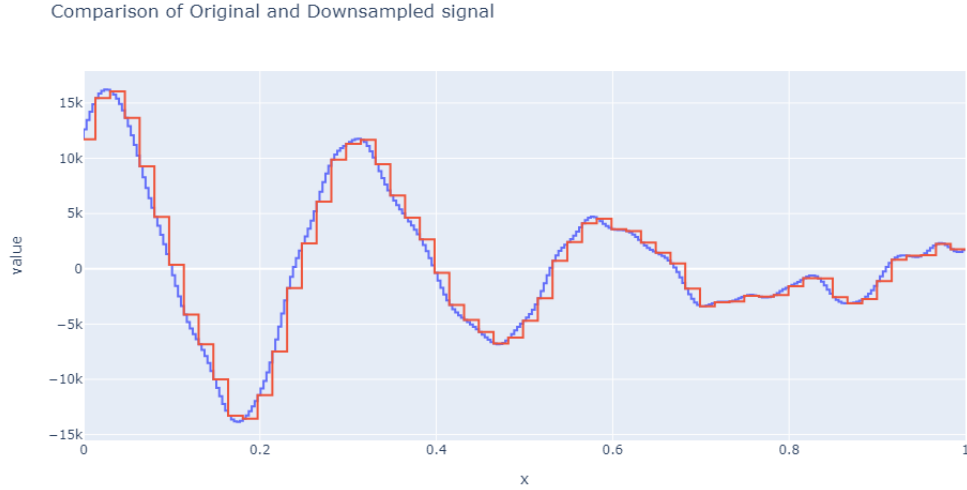

Figure 3: Comparison of Original and Downsampled signal

2.3 Linear Interpolation

Linear Interpolation, a fundamental method for reconstructing downsampled audio to a higher sample rate, offers simplicity and effectiveness in bridging the gaps between data points. This technique aims to create a linear transition between consecutive observations, providing a straightforward yet valuable approach to upsampling.

The linear interpolation process commences with the creation of arrays `x_observed` and `y_observed`, representing the observed data points from the audio. Subsequently, a new set of x values (`x`) is generated based on the target sample rate. The `np.interp` function is then employed to perform the linear interpolation, filling in the gaps and producing the upsampled audio data.

Listing 2: Linear Interpolation Function

```
def Linear_Interpolate(audio, target_sample_rate):
    x_observed = np.linspace(0.0, 1.0, len(audio[1]))
    y_observed = audio[1]
    x = np.linspace(min(x_observed), max(x_observed),
                    num=(len(audio[1]) * (target_sample_rate // audio
                        [0])))
    y = np.interp(x, x_observed, y_observed)
    return target_sample_rate, y
```

This function takes the original audio as input along with the target sample rate and performs linear interpolation to generate the upsampled audio.

2.3.1 Results

The application of Linear Interpolation is showcased in Figure 4, providing a visual comparison of the original audio, the downsampled version, and the linearly interpolated audio. The root mean squared error calculation indicates that the upscaled audio differs from the original by 2.42%.[5]

Despite its simplicity, linear interpolation proves to be a reliable method for upsampling, offering a balance between computational efficiency and preservation of audio integrity.
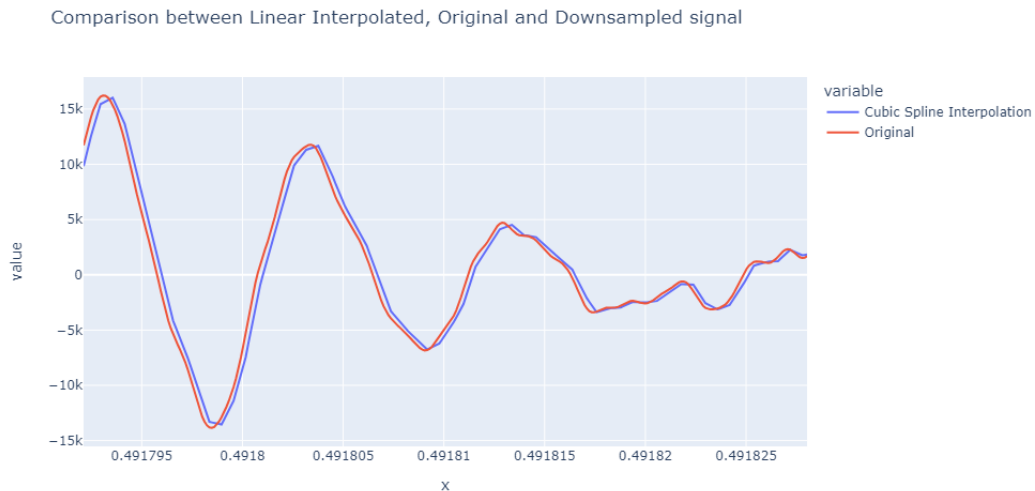

Figure 4: Comparison of Linear interpolated and Original signal

4

2.4 Piecewise cubic Hermite interpolating polynomial

The Piecewise Cubic Hermite Interpolating Polynomial (Pchip) method is employed in this project for reconstructing the downsampled audio to a higher sample rate. Pchip interpolation is particularly useful in achieving smooth transitions between data points, ensuring a continuous and differentiable interpolation.

The Pchip interpolation process begins by creating arrays `x_observed` and `y_observed`, representing the observed data points from the audio. Subsequently, a new set of x values (`x`) is generated based on the target sample rate. The `pchip_interpolate` function from the `scipy.interpolate` module is then applied to perform the interpolation and obtain the upsampled audio data.

Listing 3: Piecewise cubic Hermite interpolating polynomial Function

```python
from scipy.interpolate import pchip_interpolate

def Pchip_interpolate(audio,target_sample_rate):
    x_observed = np.linspace(0.0, 1.0, len(audio[1]))
    y_observed = audio[1]
    x = np.linspace(min(x_observed), max(x_observed), num=(len(audio
        [1])*(target_sample_rate//audio[0])))
    y = pchip_interpolate(x_observed, y_observed, x)
    return target_sample_rate,y
```

2.4.1 Results

The application of Piecewise Cubic Hermite Interpolating Polynomial is visualized in Figure 5, providing a comparative analysis of the original audio, the downsampled version, and the Pchip interpolated audio. The root mean squared error calculation indicates that the upscaled audio differs from the original by 2.418%. [5]

The Pchip method excels in preserving the shape and smoothness of the original audio signal, making it a valuable technique for applications where maintaining the continuity of the signal is essential.
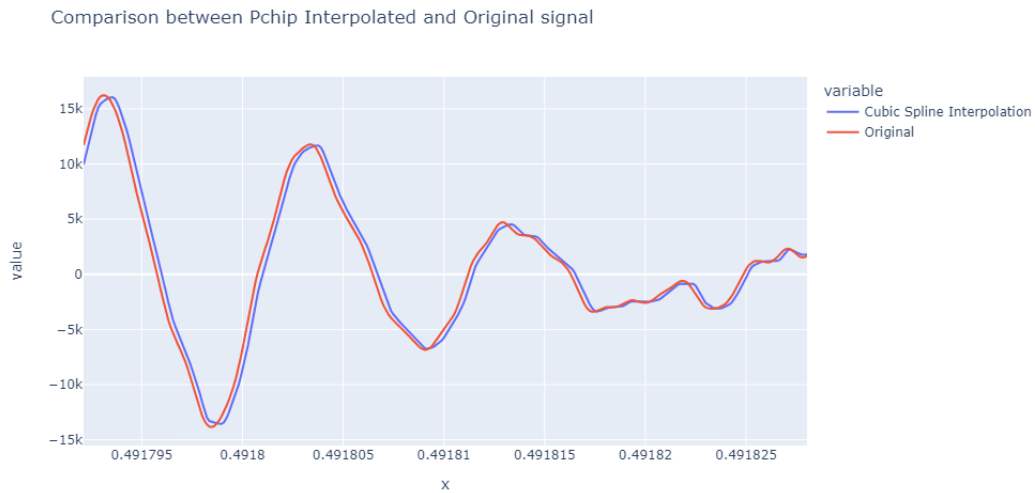

Figure 5: Comparison of Pchip Interpolated and Original signal

2.5 Cubic Spline Interpolation

Cubic Spline Interpolation, a method employed in reconstructing downsampled audio to a higher sample rate, leverages the flexibility of cubic polynomials to ensure a smooth and continuous interpolation. The implementation involves the utilization of the `CubicSpline` function from the `scipy.interpolate` module.

Listing 4: Cubic Spline Interpolation

```python
from scipy.interpolate import CubicSpline

def CubicSpline_interpolate(audio, target_sample_rate):
    x = np.linspace(0,1,len(audio[1]))
    y = audio[1]
    cs = CubicSpline(x, y)
    xs = np.linspace(0, 1, (len(audio[1])*(target_sample_rate//audio
        [0])))
    return target_sample_rate, cs(xs)
```

2.5.1 Results

The effectiveness of Cubic Spline Interpolation is illustrated in Figure 6, highlighting the comparison between the original audio, the downsampled version, and the Cubic Spline interpolated audio. The root mean squared error calculation indicates that the upscaled audio differs from the original by 2.418%.[5]

The smoothness achieved by Cubic Spline Interpolation contributes to its ability to reconstruct the downsampled audio, making it a viable method for applications where preserving the continuity of the signal is crucial.
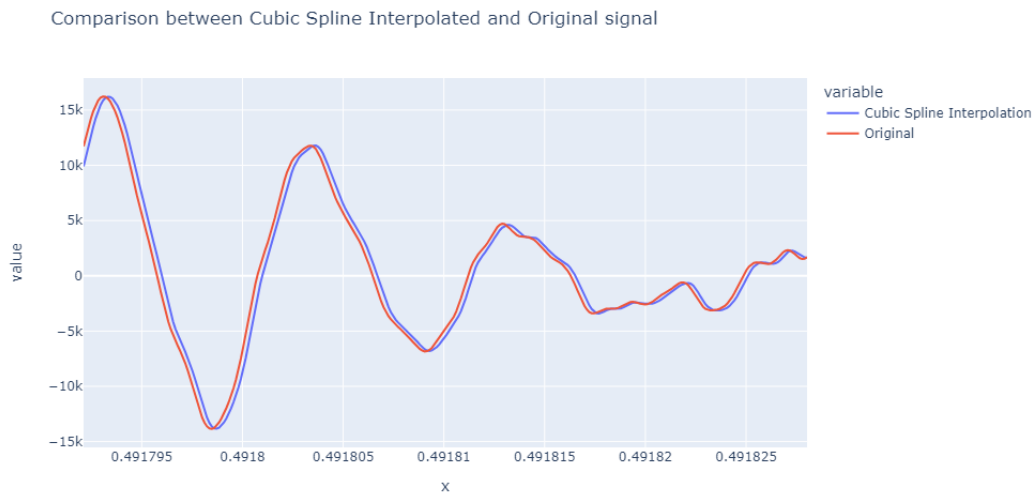

Figure 6: Comparison of Cubic Spline Interpolated and Original signal

3 Conclusion

In the pursuit of enhancing audio quality through upsampling methods, this project investigated various interpolation techniques applied to a downsampled audio signal. The selected interpolation methods—Linear Interpolation, Piecewise Cubic Hermite Interpolating Polynomial (Pchip), and Cubic Spline Interpolation—were evaluated based on their ability to reconstruct the downsampled audio close to the original.

The visual comparisons and quantitative analysis revealed that linear interpolation exhibited the least error, resulting in a more faithful reconstruction of the downsampled audio compared to the original signal. This observation aligns with the goal of minimizing distortion and preserving the integrity of the audio waveform.
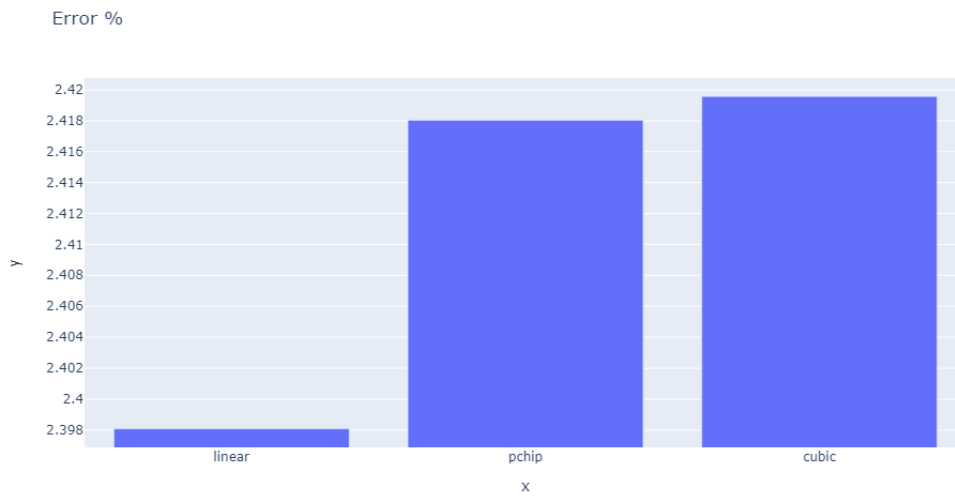

Figure 7: Errors in different methods

While the choice of interpolation method depends on specific use cases and desired outcomes, the findings of this study underscore the effectiveness of linear interpolation in achieving accurate audio upsampling. Further exploration into the interplay of different factors, such as downsampling ratios and signal characteristics, could provide valuable insights for refining audio upsampling techniques in future applications.

References

[1] Wikipedia. *Sampling (signal processing), `https: // en. wikipedia. org/ wiki/ Sampling_ (signal_ processing)`*. 2023.

[2] Wikipedia. *Upsampling, `https: // en. wikipedia. org/ wiki/ Upsampling`*. 2023.

[3] Jeffrey Hass. *Introduction to Computer Music. `https: // cmtext. indiana. edu/ digital_ audio/ chapter5_ rate. php`*. Indiana University, 2023.

[4] Universal Music Studio. *Solo, `https: // www. prostudiomasters. com/ album/ page/ 104572`*. September 24, 2021.

[5] Wikipedia. *Root-mean-square deviation, `https: // en. wikipedia. org/ wiki/ Root-mean-square_ deviation`*. 2023.