Scientific Report

Comparative Signal Analysis of String Instruments (Violin, Guitar, and Bass) Using Fourier Analysis and Cubic Spline Interpolation

Robben Wijanathan

Abstract

This report presents a comparative signal analysis of three string instruments (violin, guitar, and bass) using Fourier Transform and Cubic Spline Interpolation methods. Fourier analysis was done to identify dominant frequency components, while cubic spline interpolation was used to reconstruct and smooth the discrete waveform data. The results highlight the unique harmonic structures and tonal differences among the instruments, demonstrating the effectiveness of combining interpolation and frequency-domain analysis for digital signal characterization.

Keywords: *signal analysis, Fourier analysis, cubic spline interpolation, digital audio processing, waveform reconstruction, time-domain analysis*

1. Introduction

1.1. Background

Different musical notes correspond to different frequencies. For example, the note A4 vibrates at 440 Hz, while C5 vibrates at 523 Hz. These frequencies form the basis of musical pitch. Figure 1 shows a reference chart of musical notes and their corresponding frequencies.

	Octaves				
Notes (Hertz)	1	2	3	4	5
C	32	65	130	261	523
C#	34	69	138	277	554
D	36	73	146	293	587
D#	38	77	155	311	622
E	41	82	164	329	659
F	43	87	174	349	698
F#	46	92	185	369	739
G	49	98	196	392	784
G#	52	104	208	415	830
A	55	110	220	440	880
A#	58	116	233	466	932
B	61	123	246	493	987

Figure 1. Frequencies of Musical Notes

String instruments such as the **violin**, **guitar**, and **bass** produce sound through vibrating strings under tension. When a string is plucked or bowed, it generates a **fundamental frequency** that determines the played note, along with higher **harmonics** that shape the instrument's unique *timbre*.

In this study, we analyze recordings from these three instruments and apply **Fourier Analysis** to examine their frequency content. Converting the time-domain signal into the frequency domain allows us to identify the fundamental note, observe its harmonics, and understand how each instrument produces its characteristic sound.

1.2. Objectives

- Record digital sound signals of the violin, guitar, and bass using audio synthesis tools.
- Visualize each instrument's waveform in the time domain.
- Apply **cubic spline interpolation** to obtain a smooth continuous signal from the sampled data.
- Use the **Fourier Series** to enforce periodicity and the **Fourier Transform** to convert each waveform into the frequency domain.
- Identify the fundamental and harmonic frequencies that characterize each instrument's tone.

2. Methodology

2.1. Signal Function

The audio signals used in this analysis were generated digitally using plugins within the Digital Audio Workstation (DAW) *FL Studio*. The recorded sounds include the following variations:

1. Bass on A4 (real note A2)
2. Guitar on A4
3. Violin on A4
4. Guitar A Major chord
5. Violin A4–E5 (perfect fifth)
6. Guitar A Major + Violin A4–E5
7. Combined: Bass A4, Guitar A Major, and Violin A4–E5

Each of these audio files was visualized using `matplotlib` in Python to generate waveform plots. These plots were then processed through the WebPlotDigitizer tool (https://automeris.io/wpd/) to extract key coordinate points, such as extrema (peaks and troughs), from the visualized waveforms.

2.2. Cubic Spline Interpolation

After extracting the key extremum points from each waveform, we apply **cubic spline interpolation** to obtain a smooth approximation of the signal. A cubic spline is a set of cubic polynomials defined on intervals $[x_i, x_{i+1}]$ such that each segment $S_i(x)$ passes through the data points:

$$S_i(x_i) = y_i, \qquad S_i(x_{i+1}) = y_{i+1}.$$

To ensure the segments join smoothly, the first and second derivatives match at every internal point:

$$S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}), \qquad S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}).$$

This yields a continuously differentiable curve that preserves the waveform's local extrema and smoothness, providing the reconstructed signal used for subsequent **Fourier Analysis**.

2.3. Fourier Analysis

After obtaining a smooth, continuous representation of the waveform from the cubic spline interpolation, the next step is to analyze its frequency content using **Fourier Analysis**. This method decomposes a complex time-domain signal into a series of sinusoidal components, each corresponding to a specific frequency, amplitude, and phase. Such decomposition provides a mathematical framework for understanding the harmonic structure of musical sounds.

2.3.1. Fourier Series Representation

For a periodic signal $s(t)$ with period T, the **Fourier Series** expresses the waveform as a weighted sum of sine and cosine functions:

$$s(t) = a_0 + \sum_{n=1}^{\infty} \left[a_n \cos(n\omega_0 t) + b_n \sin(n\omega_0 t) \right],$$

where $\omega_0 = \frac{2\pi}{T}$ is the fundamental angular frequency. The coefficients a_n and b_n determine the amplitude of each harmonic component and are obtained from:

$$a_n = \frac{2}{T}\int_0^T s(t)\cos(n\omega_0 t)\,dt, \quad b_n = \frac{2}{T}\int_0^T s(t)\sin(n\omega_0 t)\,dt.$$

These coefficients quantify how much of each sinusoidal frequency is present in the original signal. In the context of string instruments, the fundamental component corresponds to the pitch of the note, while higher-order harmonics define the unique timbre and color of the sound.

2.3.2. Fourier Transform

When dealing with non-periodic or transient signals, the discrete harmonics of the Fourier Series become a continuous spectrum. This leads to the **Fourier Transform**, defined as

$$S(\omega) = \int_{-\infty}^{\infty} s(t)\,e^{-i\omega t}\,dt,$$

which represents the signal in the frequency domain. Here, $S(\omega)$ describes how the energy of the signal is distributed across different frequencies.

Applying the Fourier Transform to the interpolated waveform reveals the dominant frequencies and harmonic patterns of each instrument. This frequency-domain view deepens the understanding of how string vibrations translate into tonal perception, enabling a quantitative comparison among the violin, guitar, and bass signals.

3. Result & Discussion

3.1. Visualizing the Signal

Each audio file is converted/read as a WAV signal and visualized in Python using `matplotlib`. Below is an example for the **bass on A4 (real note A2)**.

```
# Bass A4 (A2)

sample_rate, data = wavfile.read("./audio/Bass A4 (A2).wav")

if data.ndim > 1:
    data = data[:, 0]  # use left channel

time = np.arange(len(data)) / sample_rate

start_time = 0.45
end_time = 0.5

start_index = int(start_time * sample_rate)
end_index = int(end_time * sample_rate)

segment = data[start_index:end_index]
segment_time = time[start_index:end_index]

plt.figure(figsize=(10, 6))
plt.plot(segment_time, segment, color=plt.cm.cool(0.5), linewidth=1.5)
plt.title("Bass A4 (A2) (0.45 - 0.5 seconds)", fontsize=16, weight='bold')
plt.xlabel("Time [s]", fontsize=14)
plt.ylabel("Amplitude", fontsize=14)
plt.grid(True, linestyle='--', alpha=0.6)
plt.tick_params(axis='both', labelsize=12)
plt.locator_params(axis='x', nbins=20)
plt.locator_params(axis='y', nbins=20)
plt.gcf().patch.set_facecolor('#f9f9f9')
plt.tight_layout()
plt.show()
```

Bass on A4 (A2) — waveform visualization

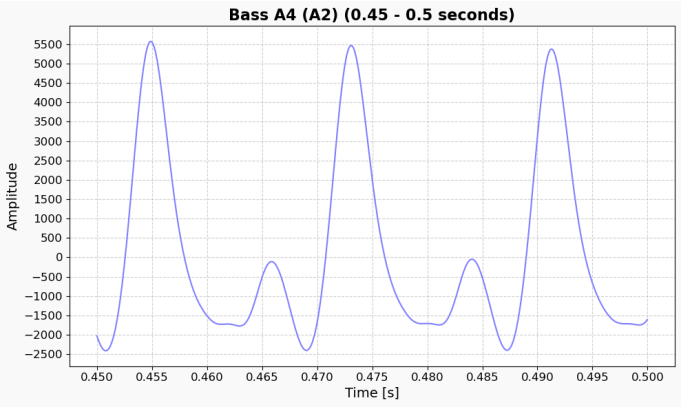


Figure 2. Waveform of Bass on A4 (real note A2) over a 50 ms segment.

The same procedure is applied to the remaining six audio files (violin A4, guitar A4, A-major chord, violin A4–E5, guitar A-major + violin A4–E5, and the combined signal).

- Guitar on A4

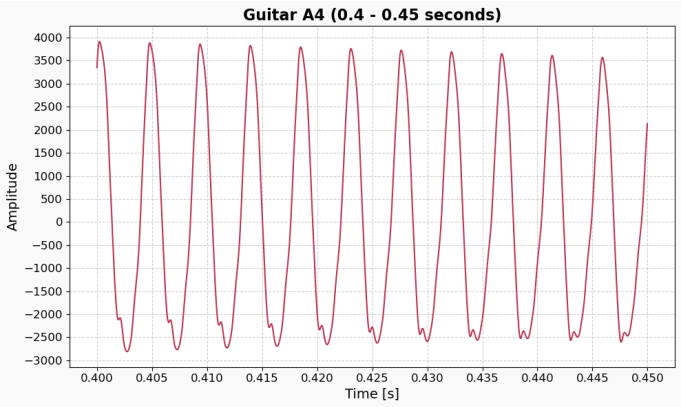


Figure 3. Guitar on A4

- Violin on A4

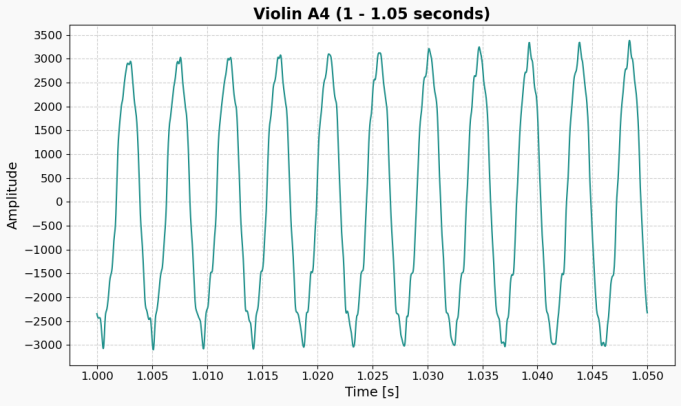


Figure 4. Violin on A4

- Guitar A Major chord

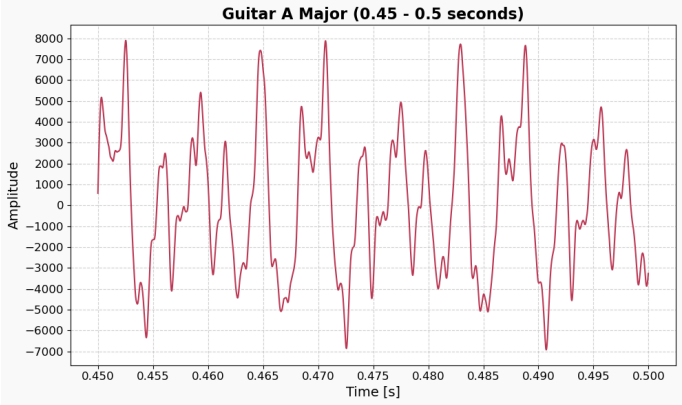


Figure 5. Guitar A Major Chord

- Violin A4–E5 (perfect fifth)

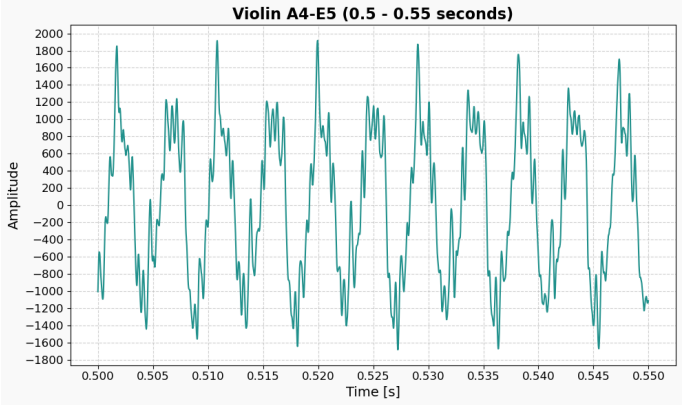


Figure 6. Violin A4-E5 (perfect fifth)

- Guitar A Major + Violin A4–E5

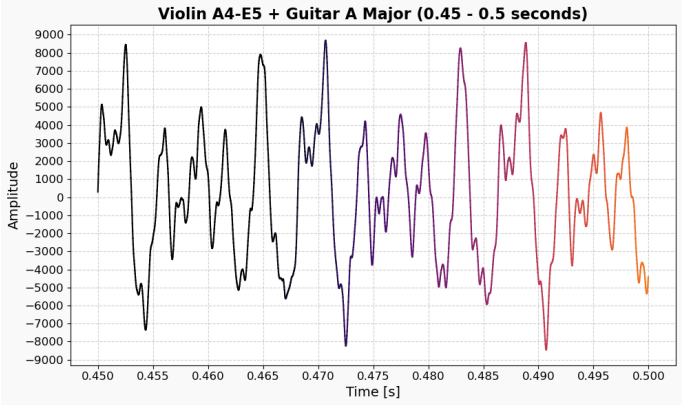


Figure 7. Guitar A Major + Violin A4–E5

- Combined: Bass A4, Guitar A Major, and Violin A4–E5

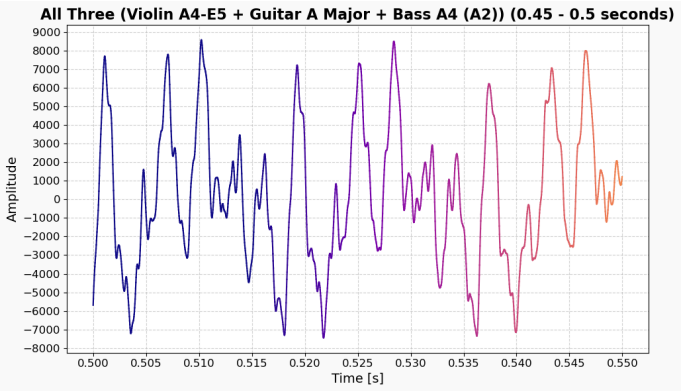


Figure 8. Combined: Bass A4, Guitar A Major, and Violin A4–E5

3.2. Key Points of the Graph

After obtaining the waveform visualizations, each plot is processed using **WebPlotDigitizer** (https://automeris.io/wpd/). The axes are first calibrated according to the time and amplitude scales, then coordinate points are placed on key extrema, peaks, and troughs that define the shape of the waveform. These extracted coordinates are later used for **cubic spline interpolation** to reconstruct a continuous signal function.

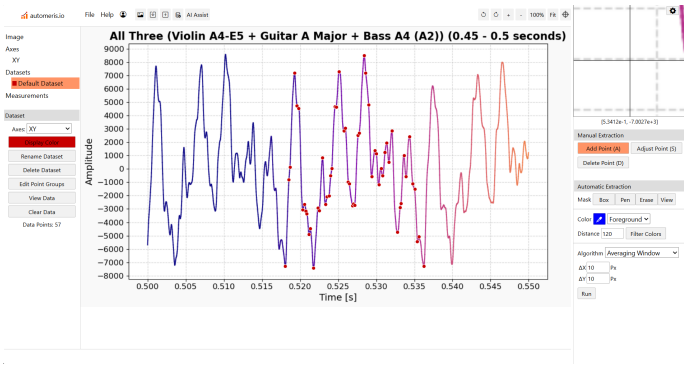


Figure 9. Example of coordinate point extraction using WebPlotDigitizer for the combined signal (Bass A4, Guitar A Major, Violin A4–E5).

3.3. Cubic Spline Interpolation

The coordinate points extracted from WebPlotDigitizer are exported as `.csv` files, each representing the time-amplitude data of a single waveform segment. An example of the data format for the **Bass A4 (A2)** signal is shown below:

```
0.450754069   -2403.165243
0.454779117    5582.033968
0.461066829   -1739.885481
0.462030318   -1707.382145
0.462994377   -1755.361522
0.465817089     -96.50511572
0.469007996   -2397.288727
```

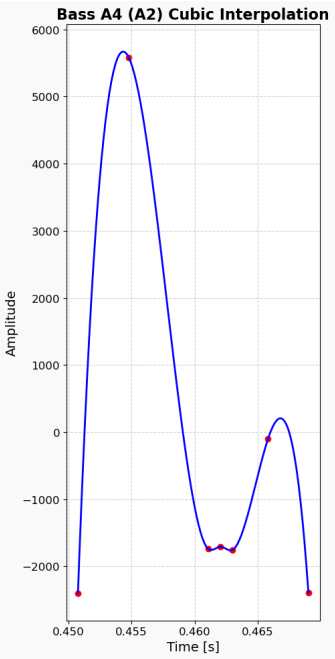
Bass A4 (A2).csv

Using these discrete points, a **cubic spline interpolation** is performed to reconstruct a smooth and continuous approximation of the waveform. The Python implementation employs the `CubicSpline` function from the `scipy.interpolate` library, as shown below:

```
# Bass A4 (A2)

data = pd.read_csv("./plot_data/Bass A4 (A2).csv
    ", header=None)
```

```
x = data.iloc[:, 0].values
y = data.iloc[:, 1].values

sorted_idx = np.argsort(x)
x, y = x[sorted_idx], y[sorted_idx]
cs = CubicSpline(x, y)

x_smooth = np.linspace(min(x), max(x), 500)
y_smooth = cs(x_smooth)

plt.figure(figsize=(6, 8))
plt.scatter(x, y, color='red', label='Data Points')
plt.plot(x_smooth, y_smooth, color='blue', linewidth=2, label='Cubic Spline')
plt.title("Bass A4 (A2) Cubic Spline Interpolation", fontsize=16, weight='bold')
plt.xlabel("Time [s]", fontsize=14)
plt.ylabel("Amplitude", fontsize=14)
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()
plt.tight_layout()
plt.show()

print(cs.c)
```

Bass A4 (A2) Cubic Spline Interpolation



The Python script reads the extracted time–amplitude data from a `.csv` file and applies a **cubic spline interpolation** using the `CubicSpline` function from `SciPy`. The data points are first sorted, then interpolated to create a smooth curve that accurately represents the waveform. Finally, the script plots both the original data points and the fitted spline for visual comparison, and prints the spline coefficients that define the cubic equations for each segment.

The figure shows the reconstructed waveform using cubic spline interpolation. The curve smoothly connects the discrete data points, preserving the signal's oscillatory pattern. This representation will later be used for frequency-domain analysis via Fourier Transform.

Figure 10. Cubic Spline Interpolation for Bass A4 (A2)

```
[[ 4.50159900e+10  4.50159900e+10 -2.44623944e+11  2.15578015e+11
  -6.51897989e+10 -6.51897989e+10]
 [-9.50718199e+08 -4.07143612e+08  4.41999132e+08 -2.65077936e+08
   3.58411718e+08 -1.93624401e+08]
 [ 5.08125860e+06 -3.84200623e+05 -1.65039152e+05  5.42238573e+03
   9.54016412e+04  5.60548811e+05]
 [-2.40316524e+03  5.58203397e+03 -1.73988548e+03 -1.70738214e+03
  -1.75536152e+03 -9.65051157e+01]]
```

Interpolation Coefficients (Matrix c)

The resulting coefficients define the cubic polynomial for each interval between data points, producing a continuous analytical model of the waveform. The same interpolation process is applied to the remaining six audio signals to obtain their continuous representations for subsequent frequency-domain analysis.

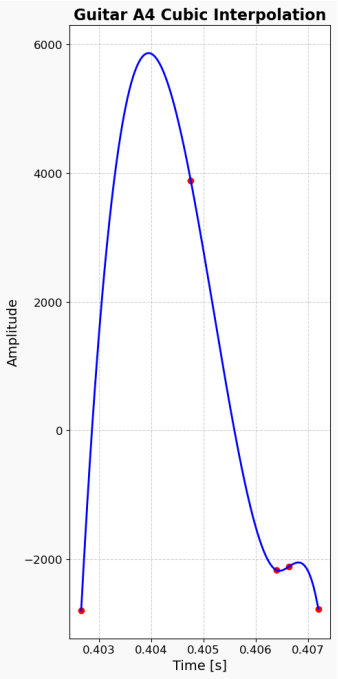


Figure 11. Guitar A4 Cubic Spline Interpolation

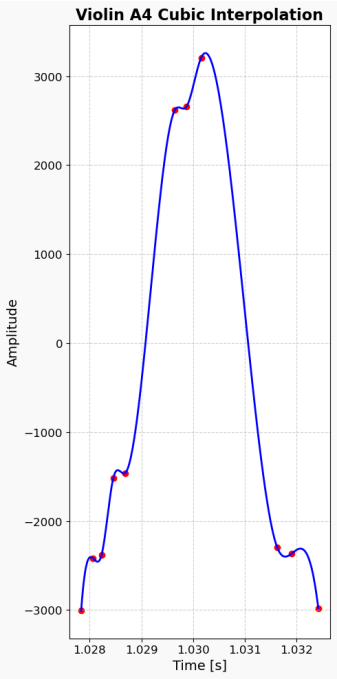


Figure 12. Violin A4 Cubic Spline Interpolation

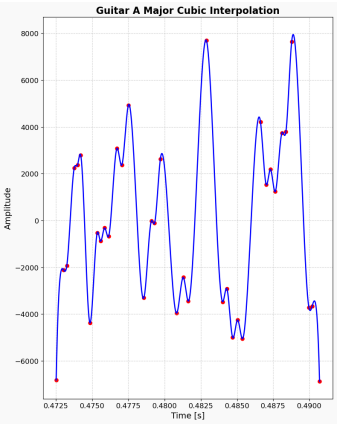


Figure 13. Guitar A Major Cubic Spline Interpolation

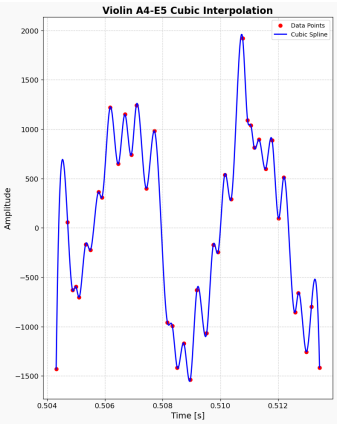


Figure 14. Violin A4–E5 Cubic Spline Interpolation

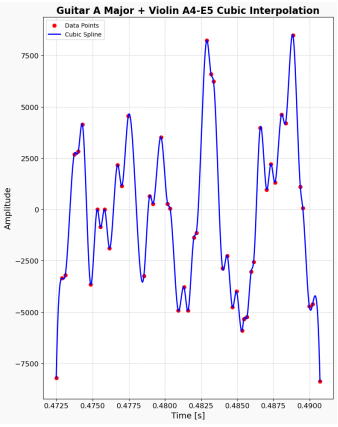


Figure 15. Guitar A Major + Violin A4–E5 Cubic Spline Interpolation

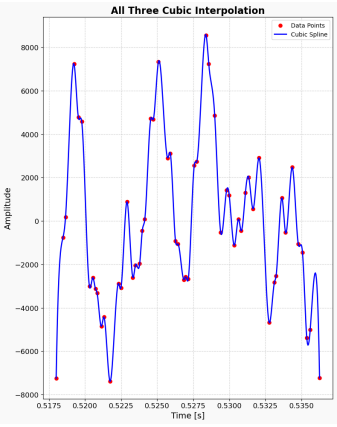


Figure 16. All Three Cubic Spline Interpolation

3.4. Fourier Analysis

3.4.1. Fourier Series

Using Cubic Spline Interpolation, the signal $s(t)$ can be represented as a piecewise function. In each interval, the piece of the signal is represented by a third-degree polynomial with four coefficients. Thus, for n intervals, the interpolated signal is expressed as

$$s(t) = \sum_{i=1}^{n} s_i(t)$$

where $s_i(t)$ is a cubic spline at the i-th interval, whose value is non-zero only within said interval.

We are now to determine the fourier series representation of the signal $s(t)$:

$$s(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos\left(\frac{2\pi kt}{T_2 - T_1}\right) + b_k \sin\left(\frac{2\pi kt}{T_2 - T_1}\right)$$

We obtain the coefficients of the series a_0, a_k, b_k with the following formulas:

$$a_0 = \frac{1}{T_2 - T_1}\int_{T_1}^{T_2} s(t)\mathrm{d}t,$$

$$a_k = \frac{2}{T_2 - T_1}\int_{T_1}^{T_2} s(t)\cos\left(\frac{2\pi kt}{T_2 - T_1}\right)\mathrm{d}t,$$

$$b_k = \frac{2}{T_2 - T_1}\int_{T_1}^{T_2} s(t)\sin\left(\frac{2\pi kt}{T_2 - T_1}\right)\mathrm{d}t$$

We then plug in the cubic spline representation of $s(t)$ into the formulas above. Note that for n intervals, T_2 is now denoted as T_{n+1}:

$$a_0 = \frac{1}{T_{n+1} - T_1}\int_{T_1}^{T_{n+1}} \left(\sum_{i=1}^{n} s_i(t)\right)\mathrm{d}t,$$

$$a_k = \frac{2}{T_{n+1} - T_1}\int_{T_1}^{T_{n+1}} \left(\sum_{i=1}^{n} s_i(t)\right)\cos\left(\frac{2\pi kt}{T_{n+1} - T_1}\right)\mathrm{d}t,$$

$$b_k = \frac{2}{T_{n+1} - T_1}\int_{T_1}^{T_{n+1}} \left(\sum_{i=1}^{n} s_i(t)\right)\sin\left(\frac{2\pi kt}{T_{n+1} - T_1}\right)\mathrm{d}t$$

Since each spline s_i is non-zero only in the interval (T_i, T_{i+1}) we can modify the integral of sums to a sum of integrals.

$$a_0 = \frac{1}{T_{n+1} - T_1}\sum_{i=1}^{n}\int_{T_i}^{T_{i+1}} s_i(t)\mathrm{d}t,$$

$$a_k = \frac{2}{T_{n+1} - T_1}\sum_{i=1}^{n}\int_{T_i}^{T_{i+1}} s_i(t)\cos\left(\frac{2\pi kt}{T_{n+1} - T_1}\right)\mathrm{d}t,$$

$$b_k = \frac{2}{T_{n+1} - T_1}\sum_{i=1}^{n}\int_{T_i}^{T_{i+1}} s_i(t)\sin\left(\frac{2\pi kt}{T_{n+1} - T_1}\right)\mathrm{d}t$$

Since each spline s_i is a 3rd degree polynomial of the form $\alpha_i(t - T_i)^3 + \beta_i(t-T_i)^2 + \gamma_i(t-T_i) + \delta_i$, we can determine the integrals involving s_i through, and by extension, the coefficients a_0, a_k, b_k through a generalized approach.

1. Determining the formula of a_o:
   We first determine the general integral:

$$\int_{T_i}^{T_{i+1}} s_i(t)\mathrm{d}t$$

$$= \int_{T_i}^{T_{i+1}} \left(\alpha_i(t - T_i)^3 + \beta_i(t - T_i)^2 + \gamma_i(t - T_i) + \delta_i\right)\mathrm{d}t$$

$$= \left(\frac{\alpha_i}{4}(t - T_i)^4 + \frac{\beta_i}{3}(t - T_i)^3 + \frac{\gamma_i}{2}(t - T_i)^2 + \delta_i(t - T_i)\Big|_{T_i}^{T_{i+1}}\right)$$

$$= \frac{\alpha_i}{4}(T_{i+1} - T_i)^4 + \frac{\beta_i}{3}(T_{i+1} - T_i)^3 + \frac{\gamma_i}{2}(T_{i+1} - T_i)^2 + \delta_i(T_{i+1} - T_i)$$

Hence, a_o is determined by the formula:

$$\boxed{a_0 = \frac{1}{T_{n+1} - T_1}\sum_{i=1}^{n}\left[\frac{\alpha_i}{4}(T_{i+1} - T_i)^4 + \frac{\beta_i}{3}(T_{i+1} - T_i)^3 + \frac{\gamma_i}{2}(T_{i+1} - T_i)^2 + \delta_i(T_{i+1} - T_i)\right]}$$

2. Determining the formula of a_k, b_k:

If $L = T_{n+1} - T_1, \quad L_i = (T_i,\ T_{i+1}), \quad \omega_k = \dfrac{2\pi k}{L}$
We have

$$a_k = \frac{2}{L}\sum_{i=1}^{n}\int_{L_i} s_i(t)\cos(\omega_k t)\mathrm{d}t$$

$$b_k = \frac{2}{L}\sum_{i=1}^{n}\int_{L_i} s_i(t)\sin(\omega_k t)\mathrm{d}t$$

$$a_k = \frac{2}{L}\sum_{i=1}^{n}\int_{L_i}\left(\alpha_i(t - T_i)^3 + \beta_i(t - T_i)^2 + \gamma_i(t - T_i) + \delta_i\right)\cos(\omega_k t)\,dt$$

$$= \frac{2}{L}\sum_{i=1}^{n}\left(\alpha_i\int_{L_i}(t - T_i)^3\cos(\omega_k t)\,dt + \beta_i\int_{L_i}(t - T_i)^2\cos(\omega_k t)\,dt + \gamma_i\int_{L_i}(t - T_i)\cos(\omega_k t)\,dt + \delta_i\int_{L_i}\cos(\omega_k t)\,dt\right).$$

$$b_k = \frac{2}{L}\sum_{i=1}^{n}\int_{L_i}\left(\alpha_i(t - T_i)^3 + \beta_i(t - T_i)^2 + \gamma_i(t - T_i) + \delta_i\right)\sin(\omega_k t)\,dt$$

$$= \frac{2}{L}\sum_{i=1}^{n}\left(\alpha_i\int_{L_i}(t - T_i)^3\sin(\omega_k t)\,dt + \beta_i\int_{L_i}(t - T_i)^2\sin(\omega_k t)\,dt + \gamma_i\int_{L_i}(t - T_i)\sin(\omega_k t)\,dt + \delta_i\int_{L_i}\sin(\omega_k t)\,dt\right).$$

Let

$$C_n = \int (t - T_i)^n\ \cos(\omega_k t)\mathrm{d}t,$$

$$S_n = \int (t - T_i)^n\ \sin(\omega_k t)\mathrm{d}t,$$

Then, by applying integration by parts, we obtain the following recursive formulas.

$$C_n = \frac{(t - T_i)^n\ \sin(\omega_k t)}{\omega_k} - \frac{n}{\omega_k}S_{n-1},$$

$$S_n = -\frac{(t - T_i)^n\ \cos(\omega_k t)}{\omega_k} + \frac{n}{\omega_k}C_{n-1},$$

with base cases

$$C_0 = \frac{\sin(\omega_k t)}{\omega_k} + C$$

$$S_0 = -\frac{\cos(\omega_k t)}{\omega_k} + C$$

Let $\tau_i = T_{i+1} - T_i$. Using the formulas above, we get

$$\begin{aligned} a_k = \frac{2}{L}\sum_{i=1}^{n}\Bigg[& \alpha_i\left(\frac{\tau_i^3\sin(\omega_k T_{i+1})}{\omega_k} + \frac{3\tau_i^2\cos(\omega_k T_{i+1})}{\omega_k^2} - \frac{6\tau_i\sin(\omega_k T_{i+1})}{\omega_k^3} - \frac{6(\cos(\omega_k T_{i+1}) - \cos(\omega_k T_i))}{\omega_k^4}\right) \\ & + \beta_i\left(\frac{\tau_i^2\sin(\omega_k T_{i+1})}{\omega_k} + \frac{2\tau_i\cos(\omega_k T_{i+1})}{\omega_k^2} - \frac{2(\sin(\omega_k T_{i+1}) - \sin(\omega_k T_i))}{\omega_k^3}\right) \\ & + \gamma_i\left(\frac{\tau_i\sin(\omega_k T_{i+1})}{\omega_k} + \frac{\cos(\omega_k T_{i+1}) - \cos(\omega_k T_i)}{\omega_k^2}\right) \\ & + \delta_i\left(\frac{\sin(\omega_k T_{i+1}) - \sin(\omega_k T_i)}{\omega_k}\right)\Bigg] \end{aligned}$$

$$\begin{aligned} b_k = \frac{2}{L}\sum_{i=1}^{n}\Bigg[& \alpha_i\left(-\frac{\tau_i^3\cos(\omega_k T_{i+1})}{\omega_k} + \frac{3\tau_i^2\sin(\omega_k T_{i+1})}{\omega_k^2} + \frac{6\tau_i\cos(\omega_k T_{i+1})}{\omega_k^3} - \frac{6(\sin(\omega_k T_{i+1}) - \sin(\omega_k T_i))}{\omega_k^4}\right) \\ & + \beta_i\left(-\frac{\tau_i^2\cos(\omega_k T_{i+1})}{\omega_k} + \frac{2\tau_i\sin(\omega_k T_{i+1})}{\omega_k^2} + \frac{2(\cos(\omega_k T_{i+1}) - \cos(\omega_k T_i))}{\omega_k^3}\right) \\ & + \gamma_i\left(-\frac{\tau_i\cos(\omega_k T_{i+1})}{\omega_k} + \frac{\sin(\omega_k T_{i+1}) - \sin(\omega_k T_i)}{\omega_k^2}\right) \\ & + \delta_i\left(-\frac{\cos(\omega_k T_{i+1}) - \cos(\omega_k T_i)}{\omega_k}\right)\Bigg] \end{aligned}$$

We then construct a Python function to compute the Fourier coefficients using the formulas derived above. The cubic–spline interpolation coefficients are passed as inputs to the function, and the function returns the values of a_0, a_k, and b_k.

```python
def getFourierCoeffs(coefs, T, K=50):
    coefs = np.asarray(coefs, dtype=float)
    T = np.asarray(T, dtype=float)
    n = coefs.shape[0]

    alpha = coefs[:, 0]
    beta  = coefs[:, 1]
    gamma = coefs[:, 2]
    delta = coefs[:, 3]

    tau = T[1:] - T[:-1]
    L = T[-1] - T[0]
    if L == 0:
        raise ValueError("Total length L = T[-1] - T[0] must be nonzero.")

    a0 = (1.0 / L) * np.sum(
        alpha * (tau**4) / 4.0
        + beta  * (tau**3) / 3.0
        + gamma * (tau**2) / 2.0
        + delta * (tau)
    )

    ak = np.zeros(K, dtype=float)
    bk = np.zeros(K, dtype=float)

    T_i = T[:-1]
    T_ip1 = T[1:]

    for k in range(1, K+1):
        omega = 2.0 * np.pi * k / L
        w1 = omega
        w2 = omega**2
        w3 = omega**3
        w4 = omega**4

        sin_Ti   = np.sin(w1 * T_i)
        sin_Tip1 = np.sin(w1 * T_ip1)
        cos_Ti   = np.cos(w1 * T_i)
        cos_Tip1 = np.cos(w1 * T_ip1)

        term_alpha_a = (
            alpha * (
                (tau**3 * sin_Tip1) / w1
                + (3.0 * tau**2 * cos_Tip1) / w2
                - (6.0 * tau * sin_Tip1) / w3
                - (6.0 * (cos_Tip1 - cos_Ti)) / w4
            )
        )
        term_beta_a = (
            beta * (
                (tau**2 * sin_Tip1) / w1
                + (2.0 * tau * cos_Tip1) / w2
                - (2.0 * (sin_Tip1 - sin_Ti)) / w3
            )
        )
        term_gamma_a = (
            gamma * (
                (tau * sin_Tip1) / w1
                + (cos_Tip1 - cos_Ti) / w2
            )
        )
        term_delta_a = (
            delta * ( (sin_Tip1 - sin_Ti) / w1 )
        )

        sum_a_intervals = np.sum(term_alpha_a + term_beta_a + term_gamma_a + term_delta_a)
        ak[k-1] = (2.0 / L) * sum_a_intervals

        term_alpha_b = (
            alpha * (
                - (tau**3 * cos_Tip1) / w1
                + (3.0 * tau**2 * sin_Tip1) / w2
                + (6.0 * tau * cos_Tip1) / w3
                - (6.0 * (sin_Tip1 - sin_Ti)) / w4
            )
        )
        term_beta_b = (
            beta * (
                - (tau**2 * cos_Tip1) / w1
                + (2.0 * tau * sin_Tip1) / w2
                + (2.0 * (cos_Tip1 - cos_Ti)) / w3
            )
        )
        term_gamma_b = (
            gamma * (
                - (tau * cos_Tip1) / w1
                + (sin_Tip1 - sin_Ti) / w2
            )
        )
        term_delta_b = (
            delta * ( - (cos_Tip1 - cos_Ti) / w1 )
        )

        sum_b_intervals = np.sum(term_alpha_b + term_beta_b + term_gamma_b + term_delta_b)
        bk[k-1] = (2.0 / L) * sum_b_intervals

    return a0, ak, bk
```

Fourier Series Coefficients

Using the computed Fourier coefficients, we then construct the Fourier series representation of the function, as shown in the following code.

```
def fourier_series(t, terms, a0, ak, bk, L):
    t = np.asarray(t, dtype=float)
    y = np.full_like(t, a0, dtype=float)
    for k in range(1, terms+1):
        omega = 2 * np.pi * k / L
        y += ak[k-1] * np.cos(omega * t) + bk[k
    -1] * np.sin(omega * t)
    return y
```

Fourier Series

The function `fourier_series` evaluates the Fourier series at a set of points t. It initializes the output with the constant term a_0, then iteratively adds each harmonic using the cosine and sine terms with coefficients a_k and b_k. The angular frequency $\omega_k = 2\pi k/L$ is computed for each term, and the resulting series values are returned as the array y.

We then use Matplotlib to visualize the resulting Fourier series, illustrating the periodic behavior of the reconstructed function.

```
def visualizeFourier(data):
    x = data.iloc[:, 0].values
    y = data.iloc[:, 1].values

    cs = CubicSpline(x, y)

    x_smooth = np.linspace(min(x), max(x), 500)
    y_smooth = cs(x_smooth)
    cubicSpline = np.transpose(cs.c)

    a0, ak, bk = getFourierCoeffs(coefs=
    cubicSpline, T=x, K=50)

    L = x[-1] - x[0]

    terms = 50

    t_plot_long = np.linspace(x[0] - L, x[-1] +
    L, 4000)
    y_fourier_long = fourier_series(t_plot_long,
     terms, a0, ak, bk, L)

    plt.figure(figsize=(10,5))
    plt.plot(t_plot_long, y_fourier_long,
                label=f"Fourier Series (K={terms
    })", linewidth=2)

    plt.scatter(x, y, color='red', s=12, label="
    Original Data", zorder=5)

    plt.axvline(x[0], color='gray', linestyle='
    --', alpha=0.5)
    plt.axvline(x[-1], color='gray', linestyle='
    --', alpha=0.5)
    plt.axvline(x[0] - L, color='gray',
    linestyle='--', alpha=0.3)
    plt.axvline(x[-1] + L, color='gray',
    linestyle='--', alpha=0.3)

    plt.title("Fourier Series Reconstruction (
    Extended to Show Periodicity)")
    plt.xlabel("t")
    plt.ylabel("s(t)")
    plt.grid(True, alpha=0.3)
    plt.legend()
    plt.show()
```

Fourier Series Visualisation

Below are the Fourier Series visualizations for each waveform:

- Bass on A4 (A2)

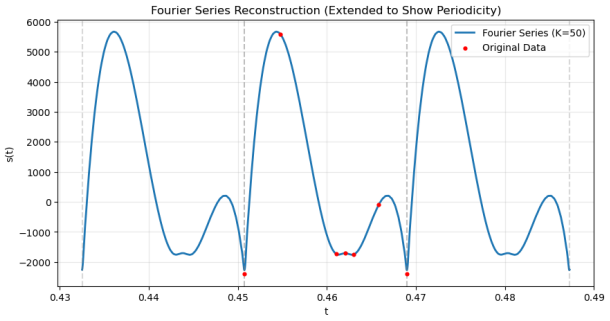


Figure 17. Bass on A4 (A2)

- Guitar on A4

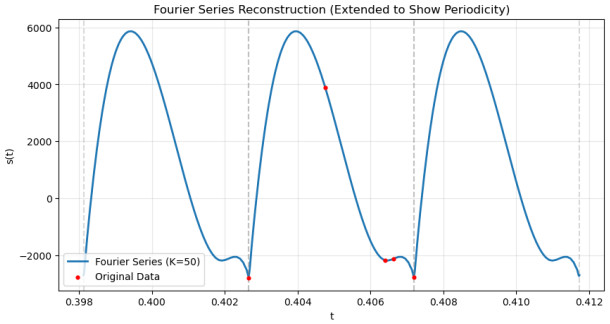


Figure 18. Guitar on A4

- Violin on A4

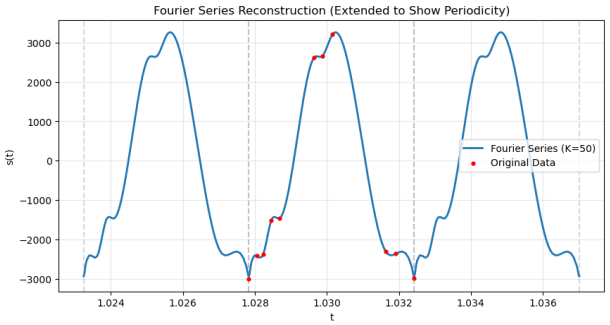


Figure 19. Violin on A4

- Guitar on A Major

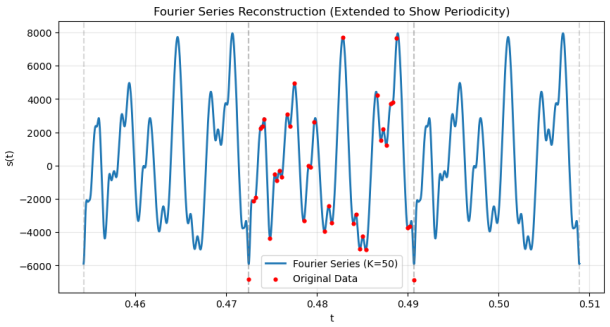


Figure 20. Guitar on A Major

- Violin on A4-E5

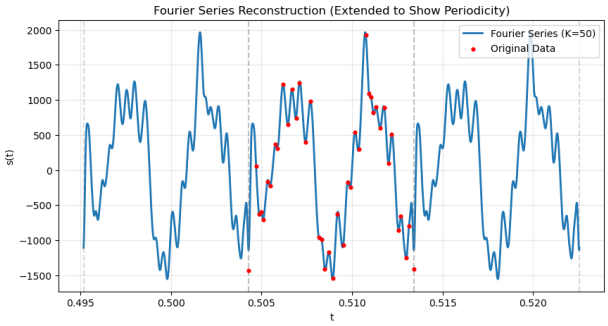


Figure 21. Violin on A4-E5

- Guitar A Major + Violin A4-E5

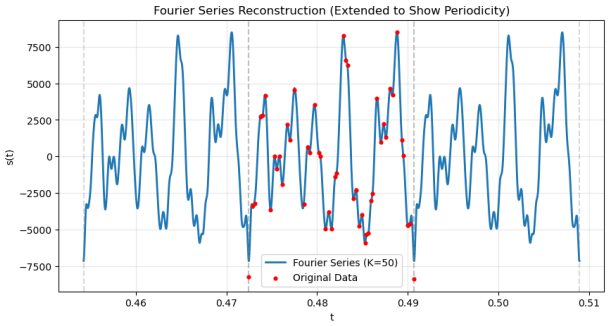


Figure 22. Guitar A Major + Violin A4-E5

- All Three

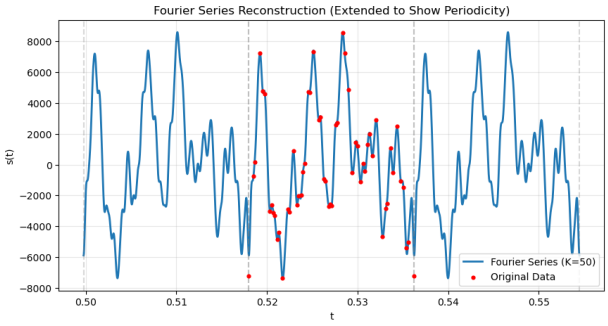


Figure 23. All Three

Reconstructed audio from the Fourier series of all seven waveforms: https://binusianorg-my.sharepoint.com/personal/robben_wijanathan_binus_ac_id/_layouts/15/guestaccess.aspx?share=El4K6H1ZURZFmG523N5sMPgBzuK3FupaKarmDc9NqVcY5A&e=T5x83c.

3.4.2. Fourier Transform

After obtaining the Fourier series representation of the signal:

$$s(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos\left(\frac{2\pi kt}{\Delta T}\right) + b_k \sin\left(\frac{2\pi kt}{\Delta T}\right),$$

We determine the dominant frequency of each waveform to identify its principal notes. This is done by applying the Fourier Transform:

$$\mathcal{F}\{s(t)\}(\omega) = \int_{-\infty}^{\infty} s(t)\, e^{-i\omega t}\, \mathrm{d}t,$$

Which shows the contribution of each frequency. The dominant component is found by locating the peak magnitude in the spectrum. The results are as follows:

$$\mathcal{F}\{s(t)\}(\omega) = \int_{-\infty}^{\infty} \left(a_0 + \sum_{k=1}^{\infty} a_k \cos\left(\frac{2\pi kt}{\Delta T}\right) + b_k \sin\left(\frac{2\pi kt}{\Delta T}\right)\right) e^{-i\omega t}\, \mathrm{d}t$$

$$= \int_{-\infty}^{\infty} a_0\, e^{-i\omega t}\, \mathrm{d}t + \int_{-\infty}^{\infty} \left(\sum_{k=1}^{\infty} a_k \cos\left(\frac{2\pi kt}{\Delta T}\right) + b_k \sin\left(\frac{2\pi kt}{\Delta T}\right)\right) e^{-i\omega t}\, \mathrm{d}t$$

$$= \int_{-\infty}^{\infty} a_0\, e^{-i\omega t}\, \mathrm{d}t + \sum_{k=1}^{\infty} \int_{-\infty}^{\infty} a_k \cos\left(\frac{2\pi kt}{\Delta T}\right) e^{-i\omega t}\, \mathrm{d}t + \sum_{k=1}^{\infty} \int_{-\infty}^{\infty} b_k \sin\left(\frac{2\pi kt}{\Delta T}\right) e^{-i\omega t}\, \mathrm{d}t$$

Although the formulas for the coefficients a_0, a_k, and b_k are complicated, each coefficient is simply a scalar and does not affect the integration, allowing them to be factored out of the integrals.

$$\mathcal{F}\{s(t)\}(\omega) = a_0 \int_{-\infty}^{\infty} e^{-i\omega t}\, \mathrm{d}t + \sum_{k=1}^{\infty} a_k \int_{-\infty}^{\infty} \cos\left(\frac{2\pi kt}{\Delta T}\right) e^{-i\omega t}\, \mathrm{d}t + \sum_{k=1}^{\infty} b_k \int_{-\infty}^{\infty} \sin\left(\frac{2\pi kt}{\Delta T}\right) e^{-i\omega t}\, \mathrm{d}t$$

Let $\omega_k = \dfrac{2\pi k}{\Delta T}$. We can use the standard Fourier Transforms

$$\mathcal{F}\{1\} = 2\pi\delta(\omega),$$
$$\mathcal{F}\{\cos(\omega_k t)\} = \pi\left[\delta(\omega - \omega_k) + \delta(\omega + \omega_k)\right],$$
$$\mathcal{F}\{\sin(\omega_k t)\} = -i\pi\left[\delta(\omega - \omega_k) - \delta(\omega + \omega_k)\right]$$

So, it becomes

$$\mathcal{F}\{s(t)\}(\omega) = 2\pi a_0 \delta(\omega) + \sum_{k=1}^{\infty} a_k \pi\left[\delta(\omega - \omega_k) + \delta(\omega + \omega_k)\right] - \sum_{k=1}^{\infty} b_k \pi i\left[\delta(\omega - \omega_k) - \delta(\omega + \omega_k)\right]$$

We can collect the coefficients to simplify the formula and subsequently make it easier to graph.

$$\boxed{\mathcal{F}\{s(t)\}(\omega) = 2\pi a_0 \delta(\omega) + \sum_{k=1}^{\infty} \pi(a_k - ib_k)\delta(\omega - \omega_k) + \pi(a_k + ib_k)\delta(\omega + \omega_k)}$$

This can be interpreted as the resulting function $S(\omega)$ as having different forms for three different cases regarding the value of ω.

- $\omega = 0$

$$S(0) = 2\pi a_0$$

- $\omega = \omega_k$

$$S(\omega_k) = \pi(a_k - ib_k)$$

- $\omega = -\omega_k$

$$S(-\omega_k) = \pi(a_k + ib_k)$$

After deriving the general Fourier Transform formula, we implement it in Python to visualize the frequency spectrum of each waveform and automatically identify its dominant frequencies.

The function shown below processes a recorded signal by first reconstructing it using a cubic spline, then computing its Fourier series coefficients, and finally evaluating the Fourier Transform using a Gaussian approximation of the Dirac delta function. A user-defined threshold is applied to highlight only the significant peaks in the spectrum, corresponding to the fundamental frequency and its harmonics. This approach allows us to efficiently analyze the harmonic structure of different string instruments. When the function is applied to each waveform, it produces a clear frequency-domain plot along with the dominant frequencies detected, making it easier to interpret and compare the acoustic characteristics of each instrument.

```python
def visualizeFourierTransform(data, name, threshold, sigma=2.0, w_limit=4000, K=50, n_freq=10000):

    x = np.asarray(data.iloc[:, 0])
    y = np.asarray(data.iloc[:, 1])
    order = np.argsort(x)
    x = x[order]
    y = y[order]

    if x.size < 2:
        raise ValueError("Need at least two x samples.")
    if x.size != y.size:
        raise ValueError("x and y must have same length.")

    cs = CubicSpline(x, y)
    cubicSpline = np.transpose(cs.c)

    T_scalar = x[-1] - x[0]
    if T_scalar <= 0:
        raise ValueError("Invalid x range (no span).")

    a0, ak, bk = getFourierCoeffs(cubicSpline, x, K=K)

    ak = np.asarray(ak, float)
    bk = np.asarray(bk, float)
    if ak.size != bk.size:
        raise ValueError("ak and bk must have the same length after getFourierCoeffs.")
    if ak.size == 0:
        raise ValueError("No harmonics returned by getFourierCoeffs.")

    Kret = ak.size
    ks = np.arange(1, Kret + 1)
    omega0 = 2.0 * np.pi / T_scalar
    omega = np.linspace(0, w_limit, n_freq)

    def delta_approx(w, w0):
        return np.exp(-0.5 * ((w - w0) / sigma) ** 2)

    FT = np.zeros_like(omega, dtype=np.complex128)
    FT += 2.0 * np.pi * a0 * delta_approx(omega, 0.0)

    for i, k in enumerate(ks):
        wk = k * omega0
        FT += np.pi * (ak[i] - 1j * bk[i]) * delta_approx(omega,  wk)
        FT += np.pi * (ak[i] + 1j * bk[i]) * delta_approx(omega, -wk)

    mag = np.abs(FT)
    dom_mask = mag > threshold
    dom_omega = omega[dom_mask]

    if dom_omega.size:
        print(f"\nDominant Frequencies (|F(  )| > {threshold}):")
        for w in dom_omega:
            f = w / (2.0 * np.pi)
            print(f"   = {w:.2f} rad/s   f = {f:.4f} Hz")
    else:
        print(f"\nNo frequencies with |F(  )| > {threshold} found in   {w_limit} rad/s.")

    plt.figure(figsize=(10, 4))
    plt.plot(omega, mag)
    plt.xlim([0, w_limit])
    plt.xlabel("Frequency    (rad/s)")
    plt.ylabel("|F(  )|")
    plt.title("Fourier Transform Magnitude " + name)
    plt.grid(True)
    plt.show()

    return {
        "omega": omega,
        "FT": FT,
        "a0": a0,
        "ak": ak,
        "bk": bk,
        "omega0": omega0,
        "T": T_scalar
    }
```

Fourier Transform Visualisation

When this function is applied to each waveform, it produces the following frequency-domain results:

- Bass on A4 (A2)

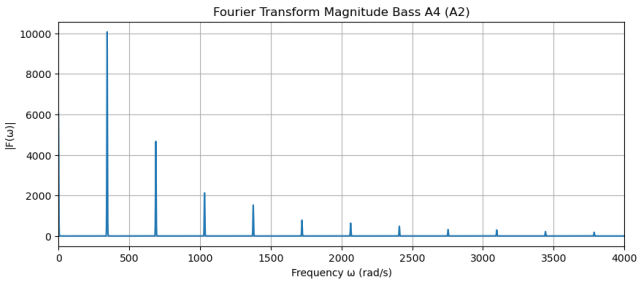


Figure 24. Bass on A4 (A2)

```
Unique Dominant Frequencies (Approx.)

omega ~ +/- 347 rad/s      ->    f ~ +/- 55.3 Hz
omega ~ +/- 690 rad/s      ->    f ~ +/- 109.9 Hz

```

Dominant Frequencies

From the dominant frequency output, we observe two significant frequency components: approximately 55.3 Hz and 109.9 Hz. Referring to the musical note frequency chart in Figure 1, these frequencies correspond closely to the notes **A1** and **A2**. This indicates that the recorded bass waveform contains not only the fundamental note but also its strong harmonic.

The presence of both frequencies is expected for string instruments: when a string vibrates, it naturally produces a fundamental frequency along with harmonics at integer multiples of that frequency. These harmonics enrich the sound and contribute to the instrument's characteristic timbre. In this case, the bass produces a strong second harmonic (A2) alongside its fundamental (A1), demonstrating the harmonic structure typical of plucked string vibrations.

- Guitar on A4

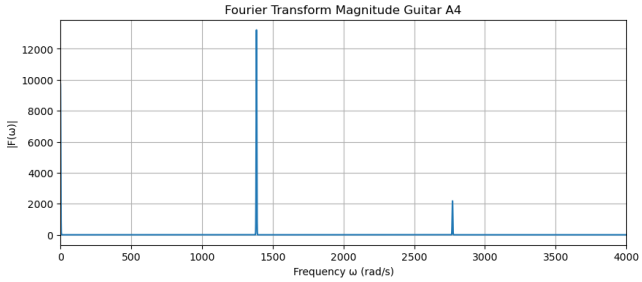


Figure 25. Guitar on A4

```
Unique Dominant Frequencies (Approx.)

omega ~ +/- 1386 rad/s      ->    f ~ +/- 221.0 Hz
omega ~ +/- 2771 rad/s      ->    f ~ +/- 441.0 Hz
```

Dominant Frequencies

Identified Notes:

- 221 Hz ≈ 220 Hz → **A3**
- 441 Hz ≈ 440 Hz → **A4**

- Violin on A4

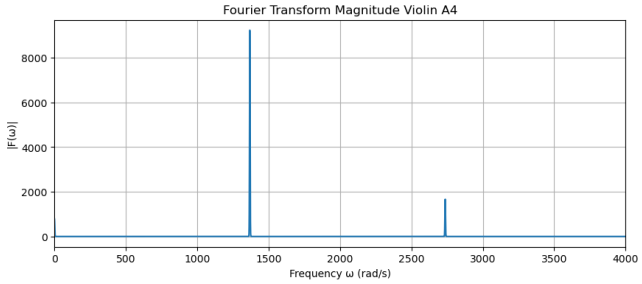


Figure 26. Violin on A4

```
Unique Dominant Frequencies (Approx.)

omega ~ +/- 1368 rad/s      ->    f ~ +/- 217.8 Hz
omega ~ +/- 2737 rad/s      ->    f ~ +/- 435.6 Hz
```

Dominant Frequencies

Identified Notes:

- 218 Hz ≈ 220 Hz → **A3**
- 436 Hz ≈ 440 Hz → **A4**

- Guitar on A Major

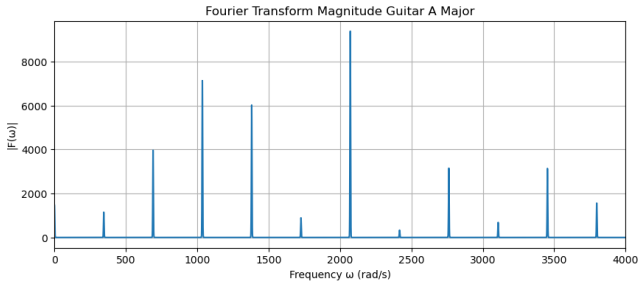


Figure 27. Guitar on A Major

```
Unique Dominant Frequencies (Approx.)

omega ~ +/- 692 rad/s       ->    f ~ +/- 110.2 Hz
omega ~ +/- 1037 rad/s      ->    f ~ +/- 165.1 Hz
omega ~ +/- 1382 rad/s      ->    f ~ +/- 220.0 Hz
omega ~ +/- 2072 rad/s      ->    f ~ +/- 330.0 Hz
omega ~ +/- 2762 rad/s      ->    f ~ +/- 439.8 Hz
omega ~ +/- 3453 rad/s      ->    f ~ +/- 549.7 Hz
```

Dominant Frequencies

Identified Notes:

- 110.2 Hz ≈ 110 Hz → **A2**
- 165.1 Hz ≈ 164 Hz → **E3**
- 220.0 Hz → **A3**
- 330.0 Hz ≈ 329 Hz → **E4**
- 439.8 Hz ≈ 440 Hz → **A4**
- 549.7 Hz ≈ 554 Hz → **C#5**

A Major contains A, C#, and E, matching the identified notes.

- Violin on A4-E5

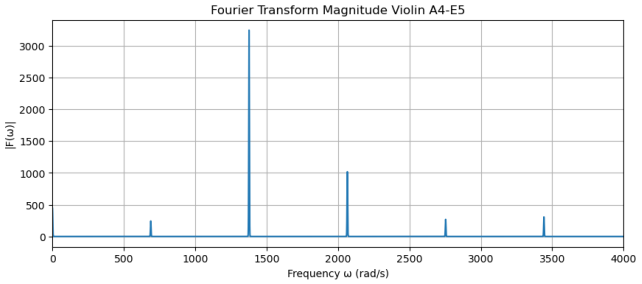


Figure 28. Violin on A4-E5

```
Unique Dominant Frequencies (Approx.)

omega ~ +/- 1378 rad/s      ->    f ~ +/- 219.5 Hz
omega ~ +/- 2065 rad/s      ->    f ~ +/- 328.7 Hz
```

Dominant Frequencies

Identified Notes:

- 219.5 Hz ≈ 220 Hz → **A3**
- 328.7 Hz ≈ 329 Hz → **E4**

- Guitar A Major + Violin A4-E5

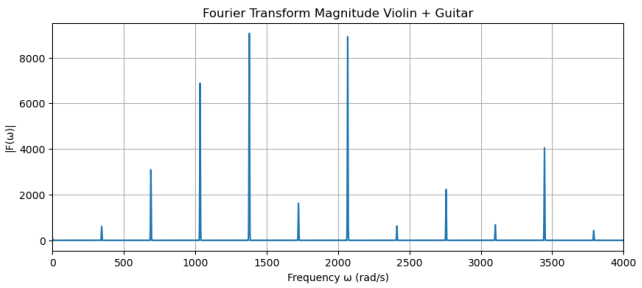


Figure 29. Guitar A Major + Violin A4-E5

```
Unique Dominant Frequencies (Approx.)

omega ~ +/- 688 rad/s       ->    f ~ +/- 110.0 Hz
omega ~ +/- 1034 rad/s      ->    f ~ +/- 164.7 Hz
omega ~ +/- 1379 rad/s      ->    f ~ +/- 219.6 Hz
omega ~ +/- 1723 rad/s      ->    f ~ +/- 274.3 Hz
omega ~ +/- 2067 rad/s      ->    f ~ +/- 329.3 Hz
omega ~ +/- 2757 rad/s      ->    f ~ +/- 438.9 Hz
omega ~ +/- 3446 rad/s      ->    f ~ +/- 548.5 Hz
```

Dominant Frequencies

Identified Notes:

- 110.0 Hz → **A2**
- 164.7 Hz ≈ 164 Hz → **E3**
- 219.6 Hz ≈ 220 Hz → **A3**
- 274.3 Hz ≈ 277 Hz → **C#4**
- 329.3 Hz ≈ 329 Hz → **E4**
- 438.9 Hz ≈ 440 Hz → **A4**
- 548.5 Hz ≈ 554 Hz → **C#5**

In the combined case of Guitar A Major + Violin A4-E5, we identified all the notes of the A major chord, which already contains in itself the notes of the Violin A4-E5 that we've identified: A3 and E4. However, it is interesting that a new note was identified here, namely the C#4, which is still part of the A Major chord.

- All Three

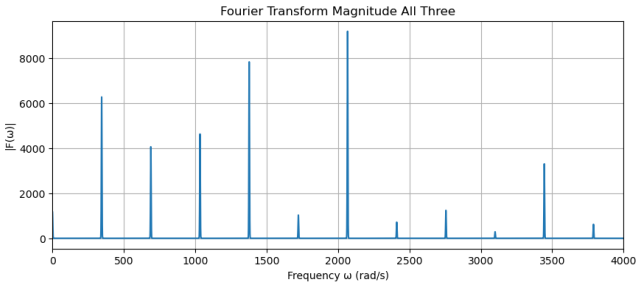


Figure 30. All Three

```
Unique Dominant Frequencies (Approx.)

omega ~ +/- 346 rad/s      ->   f ~ +/- 55.1 Hz
omega ~ +/- 690 rad/s      ->   f ~ +/- 109.9 Hz
omega ~ +/- 1033 rad/s     ->   f ~ +/- 164.4 Hz
omega ~ +/- 1378 rad/s     ->   f ~ +/- 219.5 Hz
omega ~ +/- 1722 rad/s     ->   f ~ +/- 274.1 Hz
omega ~ +/- 2066 rad/s     ->   f ~ +/- 328.8 Hz
omega ~ +/- 2756 rad/s     ->   f ~ +/- 438.7 Hz
omega ~ +/- 3444 rad/s     ->   f ~ +/- 548.5 Hz
```

Dominant Frequencies

Identified Notes:

- 55.1 Hz ≈ 55 Hz → **A1**
- 109.9 Hz ≈ 110 Hz → **A2**
- 164.4 Hz ≈ 164 Hz → **E3**
- 219.5 Hz ≈ 220 Hz → **A3**
- 274.1 Hz ≈ 277 Hz → **C#4**
- 328.8 Hz ≈ 329 Hz → **E4**
- 438.7 Hz ≈ 440 Hz → **A4**
- 548.5 Hz ≈ 554 Hz → **C#5**

From the Fourier transform visualizations, we observe that higher frequencies correspond to higher musical notes, while lower frequencies correspond to lower notes. We also see that an octave occurs when a note's frequency is doubled.

4. Conclusion

Based on the results of the Fourier Transform analysis, it can be concluded that the Fourier method is highly effective for identifying the dominant frequency components within a signal. This makes it a powerful and reliable tool for determining the musical notes present in digitally recorded waveforms.

However, it is important to note that the signals in this project imitate real musical instruments. As a result, the Fourier Transform does not capture only the fundamental frequency (the played note), but also the accompanying harmonics that naturally arise in vibrating strings. These harmonics often exhibit significant amplitude and, in some cases, may appear stronger than the fundamental component itself.

Consequently, several of the Fourier spectra obtained show a dominant peak corresponding to a lower harmonic (such as A3), even when the intended play note is A4 at 440 Hz. This apparent shift does not represent an error in the Fourier Transform, but rather reflects the inherent acoustic characteristics of string instruments, where strong harmonic content interacts with the fundamental tone.

Overall, the analysis demonstrates that the Fourier Transform accurately reveals the spectral structure of musical signals. Nonetheless, the presence of strong natural harmonics can cause the dominant peak to align with a harmonic instead of the fundamental frequency. This behavior highlights the complexity of real-world timbres and emphasizes the need to consider harmonic content when interpreting frequency-domain results. Furthermore, the results clearly show that higher frequencies correspond to higher musical notes, while lower frequencies correspond to lower notes, and that an octave occurs when the frequency of a note is doubled.

References

[1] M.J. Roberts, *Signals and Systems: Analysis Using Transform Methods & MATLAB 3rd Edition*, New York: McGraw-Hill Higher Education, 2017.