

Audio Analysis and Symbolic Music Note Generation

Sarvesh Satish

Student (Grade 12), Heartfulness International School, Hyderabad, India

Abstract

This paper documents an analysis of a short Guitar recording, covering the full pipeline from raw waveform to a playable Musical Instrument Digital Interface (MIDI) file and, eventually, engraved sheet music. The Fast Fourier Transform (FFT) was used to examine the overall spectral content of the recording, and a Short-Time Fourier Transform (STFT) spectrogram was used to observe how that content changed over time. The pYIN algorithm provided a frame-by-frame estimate of the fundamental frequency (f_0), which, combined with peak-picking on the static FFT spectrum, was used to identify musical notes. Mel-frequency cepstral coefficients (MFCCs), spectral centroid, spectral bandwidth, and a chromagram were also extracted to characterize the timbre and harmonic content of the recording.

The f_0 contour was grouped into fifteen discrete note events with defined start times, end times, and durations, which were then written out as a MIDI file using mido and imported into GarageBand to produce sheet music. The pipeline worked end-to-end, but the results were not clean: the f_0 estimate stayed close to a single low pitch for much of the clip even though the note table shows a wider melodic range, short notes near the 0.1 second mark were borderline between real events and tracking noise, and the last 1.8 seconds of the recording, mostly breath and decay rather than a clear pitch, still registered as a sustained F4. These issues, and possible remedies, are discussed in Sections 4 and 5.

Keywords: Audio Analysis, Fast Fourier Transform, Spectrogram, Pitch Tracking, pYIN, MIDI, Music Transcription, Feature Extraction

1 Introduction

1.1 Background

Audio-to-notation conversion was selected as the topic of this project because it sits at an interesting intersection: a musical instrument produces a continuous sound wave, but sheet music is discrete, with fixed pitches, fixed durations, and fixed onsets. Something has to decide where one note ends and the next begins, and the aim was to understand how that decision is actually made in code rather than accepting that music applications such as GarageBand or Melodyne "just work". Audio signal processing techniques, the Fourier transform in particular, are the common thread linking speech recognition, music production, and this kind of automatic transcription, so the project also served as a hands-on way to learn those techniques.

1.2 Problem Statement

The goal was to take a single audio recording and extract its musical content (which notes are played, when, and for how long) using signal-processing techniques rather than by ear. This required converting

a continuous digital waveform into frequency information, tracking how that frequency information changes from moment to moment, and then judging where the boundaries between notes actually lie. That last step turned out to be the hardest part of the project and is discussed at length in Section 4.

1.3 Objectives

The objectives of this analysis are as follows:

1. **Fundamental audio concepts:** establish a solid understanding of sampling rate, frequency, amplitude, and the distinction between time-domain and frequency-domain representations.
2. **Time-frequency transformation:** apply the Fast Fourier Transform (FFT) and the Short-Time Fourier Transform (STFT) to convert audio from the time domain to the frequency domain, enabling spectral analysis and spectrogram generation.
3. **Dominant frequency and note identification:** identify prominent frequencies and corresponding musical notes using both static FFT analysis and dynamic pitch tracking (pYIN).
4. **Feature extraction:** extract Mel-frequency cepstral coefficients (MFCCs), spectral centroid, spectral bandwidth, and chromagram to characterize timbral and harmonic properties.
5. **Note quantization and MIDI generation:** convert the estimated fundamental frequency (f_0) contour into discrete notes with defined start times and durations, and generate a Musical Instrument Digital Interface (MIDI) file.
6. **Reporting:** compile findings, methodology, challenges, and future directions into a structured research paper, referencing sheet music as an outcome of the MIDI conversion process.

2 Methodology

The analytical process involved several distinct stages, each employing specific techniques and libraries to dissect the audio signal and reconstruct its musical components. The primary library used for audio processing was librosa [1], complemented by SciPy [2] for the Fast Fourier Transform, Matplotlib [3] for visualization, and mido for MIDI file generation.

2.1 Audio Data Acquisition

An example audio file, specifically a "guitar" sample from librosa.ex(), was loaded into the Python environment. The librosa.load() function was used to retrieve the audio time series (y) and its corresponding sampling rate (sr). The chosen sampling rate was 22,050 Hz, which is standard for many audio applications and sufficient to capture the relevant frequency content of musical instruments.

2.2 Fundamental Audio Concepts

- **Sampling rate:** an sr of 22,050 Hz means 22,050 samples were taken per second, allowing accurate representation of frequencies up to the Nyquist frequency of 11,025 Hz.
- **Frequency:** measured in Hertz (Hz); the key parameter for identifying musical pitch, with detected frequencies mapped to standard musical notes.
- **Amplitude:** represented by the numerical values in the time-domain series y ; amplitude variations correspond to changes in loudness.
- **Time-domain representation:** the raw signal y represents the amplitude of the sound over time, visualized directly as a waveform.
- **Frequency-domain representation:** obtained via FFT, shows the distribution of energy across frequencies, crucial for spectral analysis.

2.3 Time-Domain Analysis

The initial step involved visualizing the raw audio waveform in the time domain using librosa, display,

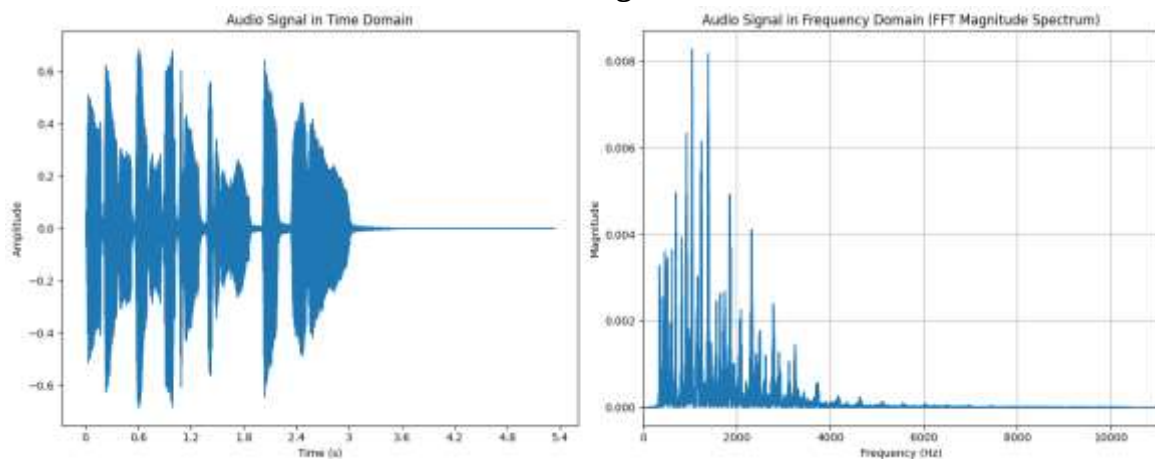
waveshow(). This provided a direct visual representation of the amplitude variations of the sound over its duration, offering insight into its overall dynamic structure.

2.4 Frequency-Domain Analysis (FFT)

To understand the static spectral content of the audio, a Fast Fourier Transform [4] was applied to the entire audio signal y using `scipy.fft.fft()`. The magnitude spectrum was calculated as given in Equation (1), where N is the number of samples in the signal, and plotted against frequency (`scipy.fft.fftfreq()`) to identify dominant frequencies present throughout the clip. The x-axis of the plot was limited to the Nyquist frequency, $sr \div 2$.

$$\text{Magnitude} = (2.0 \div N) \times |yf[0 : N \div 2]| \quad (1)$$

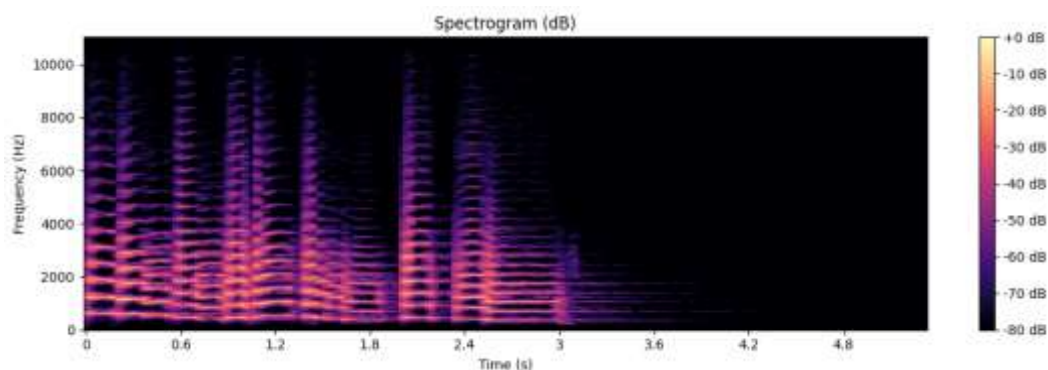
Figure 1: Time-Domain Waveform (Left) and FFT Magnitude Spectrum (Right) of the Analyzed Audio Signal



2.5 Spectrogram Generation

To observe how frequency content evolved over time, a spectrogram was generated using the Short-Time Fourier Transform. The function `librosa.stft()` was applied with `n_fft = 2048` (FFT window size) and `hop_length = 512` (samples between successive frames). The resulting amplitude spectrogram was converted to a decibel scale with `librosa.amplitude_to_db()` for improved visual dynamic range, and visualized with `librosa.display.specshow()` (Figure 2).

Figure 2: Decibel-Scaled Spectrogram (STFT) Showing Energy Distribution across Frequency and Time



2.6 Dominant Frequency and Note Identification

Two complementary methods were used for note identification.

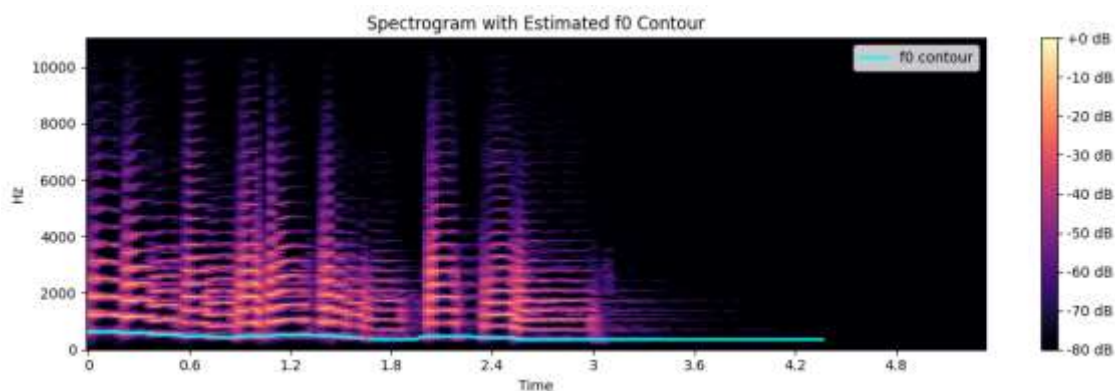
2.6.1 Static Peak Detection (from FFT)

Peaks in the overall FFT magnitude spectrum were identified using `scipy.signal.find_peaks()`, with a dynamic height threshold set at 10% of the maximum magnitude to filter out minor components. The frequencies corresponding to these peaks were mapped to musical note names using `librosa.hz_to_note()`.

2.6.2 Dynamic Pitch Tracking (pYIN Algorithm)

For a more accurate, time-varying estimate of the fundamental frequency, the `librosa.pyin()` algorithm was applied [5], which builds on the YIN estimator [6]. This algorithm robustly tracks the pitch contour over time, even in the presence of harmonics. The estimated f_0 contour was overlaid on the spectrogram to visually correlate pitch changes with spectral energy (Figure 3).

Figure 3: Spectrogram with the pYIN-Estimated Fundamental-Frequency (f_0) Contour Overlaid in Cyan

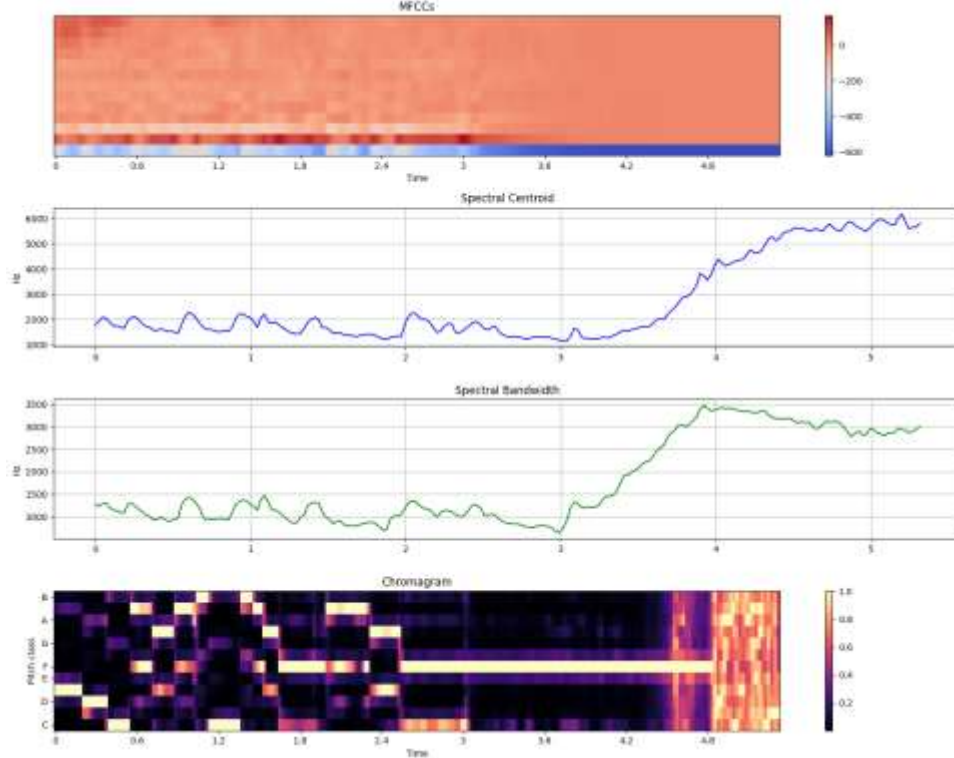


2.7 Feature Extraction

Beyond basic frequency analysis, several additional features were extracted to provide a comprehensive picture of the characteristics of the audio (Figure 4):

- **Mel-frequency cepstral coefficients (MFCCs)** [7]: computed with `librosa.feature.mfcc(n_mfcc=13)`; describe the spectral envelope, crucial for distinguishing timbral qualities, and visualized as a spectro-temporal map.
- **Spectral centroid**: computed with `librosa.feature.spectral_centroid()`; indicates the "brightness" of the sound, plotted over time.
- **Spectral bandwidth**: computed with `librosa.feature.spectral_bandwidth()`; measures the spread of the spectrum around the centroid, reflecting richness or fullness of the sound.
- **Chromagram**: computed with `librosa.feature.chroma_stft()`; represents the twelve pitch classes present in the audio irrespective of octave, useful for harmonic analysis and chord detection.

Figure 4: Extracted MFCCs, Spectral Centroid, Spectral Bandwidth, and Chromagram over Time



2.8 Note Quantization and MIDI Generation

To convert the continuous f0 contour into a symbolic musical representation, a custom script was developed to quantize notes:

1. **Note event identification:** the f0 contour and voiced_flag (from pYIN) were iterated through; where a voiced segment was detected, the frequency was mapped to a note name using `librosa.hz_to_note()`.
2. **Grouping and duration calculation:** consecutive segments identified as the same note were grouped, and the start time, end time, and duration of each stable note event were recorded.
3. **MIDI file creation:** the mido library was used to construct `generated_music.mid`. For each quantized note event, `note_on` and `note_off` messages were created, with `delta_time` computed from the start time and duration of the note at a default tempo of 120 beats per minute (BPM) and a default velocity of 100.

2.9 The Role of MIDI in Sheet Music Generation

A MIDI file is not an audio file itself but a symbolic representation of musical information, that is, a set of instructions for a musical instrument or software [8]. It contains data about:

- **Note on/off events:** when a note starts and ends
- **Pitch:** which note is played (e.g., C4, F#5)
- **Velocity:** how hard or soft a note is played (loudness)
- **Duration:** how long a note is held
- **Other parameters:** tempo changes, instrument changes, and other musical parameters

Because MIDI files store musical events symbolically, they are the ideal format for communicating musical ideas to music notation software. Instead of dealing with the complex waveforms of an audio

file, notation software can directly interpret the note events, pitches, durations, and velocities within a MIDI file to reconstruct and display the music as sheet music.

2.10 Workflow for Generating Sheet Music from MIDI

1. **Export the MIDI file:** the Python script generates `generated_music.mid`, containing all musical event data (notes, timings, velocity).
2. **Import into notation software:** the `.mid` file can be imported into MuseScore (free/open-source), Finale, Sibelius, Logic Pro X / GarageBand, or Cubase / Ableton Live. These programs interpret note-on/note-off messages and durations to graphically render notes on a staff, automatically determining note heads, stems, beams, and rests. For this project, GarageBand (macOS) was used.
3. **Refinement and customization:** an initial rendering typically requires manual adjustment, such as quantization correction, key/time-signature assignment, rhythm and beam grouping, dynamics/articulation markings, and layout/formatting.
4. **Printing and exporting:** the refined score can be printed or exported to PDF, image files, or MusicXML for sharing.

2.11 Limitations of Direct Sheet Music Generation in Python

While Python is well suited to audio analysis and MIDI generation, directly generating high-quality visual sheet music (e.g., PDF) is complex and generally outside the focus of audio-processing libraries, for several reasons:

- **Complex layout engines:** music notation involves intricate layout rules (collision avoidance, beam angles, stem directions, complex symbols, lyric alignment) that require sophisticated rendering engines.
- **Musical semantics:** converting raw MIDI data into musically intelligent notation (tuplets, grace notes, phrasing, optimal beaming) requires music-theory understanding and often AI/ML techniques.
- **Font and symbol management:** sheet music relies on specialized musical fonts and symbols not readily available in standard Python plotting libraries.
- **Interactivity and editing:** Python scripts typically produce static output, whereas notation software offers interactive editing and playback.

The most practical workflow therefore uses Python for the analytical heavy lifting and MIDI creation, and dedicated notation software for the final visual rendering and refinement of sheet music.

3 Results and Analysis

3.1 Time-Domain Observations

The initial visualization of the audio waveform in the time domain revealed the overall amplitude envelope of the signal. For the "guitar" example, the waveform showed distinct amplitude variations corresponding to the onset and decay of notes, helping to establish the dynamic range and the presence of silence or sustained notes within the clip, and providing a foundational visual context for subsequent frequency analysis.

3.2 FFT Analysis

Applying the FFT to the entire audio signal provided a static spectral overview. The magnitude-spectrum plot (frequency versus magnitude) displayed prominent peaks at specific frequencies, which were mapped to musical notes using `librosa.hz_to_note()`. For example, a clear peak around 348.75 Hz was identified as F4, and several other notes (G#4, A#4, C5, and D5) were also detected. This static

analysis highlighted the main pitch components consistently present throughout the audio, serving as an initial step toward note identification.

3.3 Spectrogram Interpretation

The spectrogram gave a much clearer picture than the static FFT of what was actually happening in the recording. Distinct vertical striations are visible roughly every 0.2–0.3 seconds through the first three seconds of the clip, lining up closely with the note onsets later recovered in Table 1. Past about the 3-second mark, though, the pattern changes noticeably: the dense harmonic bands thin out, and by 3.6–4 seconds the spectrogram is dominated by broadband, noise-like energy rather than clean horizontal bands. That shift matters for the feature analysis below, because it is exactly where the spectral centroid and bandwidth plots spike. This is very likely the decay and instrument noise (breath, valve movement) of the recording rather than a sustained musical pitch, even though the note-quantization step still labeled it as an F4 (see Section 3.5 and the discussion in Section 4).

3.4 Extracted Feature Insights

- **MFCCs:** the MFCC map is fairly stable for the first 3 seconds or so, then flattens out noticeably afterward, which is consistent with the recording moving from a clearly pitched tone into a decay/noise tail with little spectral structure left to describe.
- **Spectral centroid:** stays in a fairly narrow 1,000–2,300 Hz band while notes are being played, then jumps sharply after about 3.6 s, climbing to roughly 6,000 Hz by the end of the clip. Since brightness this high is not typical of a sustained guitar note, this jump lines up with the noise-dominated tail rather than a genuine bright pitch.
- **Spectral bandwidth:** follows almost the same shape as the centroid, hovering under 1,500 Hz during the melodic section, then widening to about 3,500 Hz after 3.6 s. A sudden widening like this usually means the sound has stopped being a clean tone and become closer to broadband noise, which fits the spectrogram observation above.
- **Chromagram:** shows clean, well-separated pitch-class activity through the first 4.5 seconds or so, matching the note sequence in Table 1, but becomes noticeably smeared across almost all twelve pitch classes at once in the final second, which is another sign that this final segment is not really a single identifiable pitch.

3.5 Quantized Notes and MIDI Output

Based on the pYIN f0 estimation, a sequence of discrete musical notes with start times, end times, and durations was successfully identified and quantized; the process filtered out very short, potentially artifactual note events. The complete quantized note sequence extracted from the analyzed clip is listed in Table 1.

Table 1: Quantized Note Events (Pitch, Start Time, End Time, and Duration) Extracted from the f0 Contour

Note	Start (s)	End (s)	Duration (s)
D#5	0.00	0.19	0.19
D5	0.19	0.39	0.21
C5	0.39	0.56	0.16
A#4	0.56	0.74	0.19

G#4	0.74	0.88	0.14
A#4	0.88	1.04	0.16
B4	1.04	1.14	0.09
C5	1.14	1.35	0.21
B4	1.35	1.46	0.12
A#4	1.46	1.56	0.09
G#4	1.56	1.65	0.09
F4	1.67	1.97	0.30
A#4	2.00	2.30	0.30
G#4	2.32	2.55	0.23
F4	2.55	4.39	1.83

This sequence of musical events was used to generate a MIDI file, generated_music.mid, serving as a symbolic representation of the analyzed audio that captures pitch and timing information. The .mid file was then imported into GarageBand to automatically generate visual sheet music, transforming the raw audio into a readable musical score. Figure 5 shows the resulting engraved excerpt, including the triplet groupings and accidentals recovered from the quantized note data.

Figure 5: Final Sheet Music Excerpt Engraved in GarageBand from the Generated MIDI File, Showing Triplet Groupings, Accidentals, and Rhythmic Values Recovered from the Quantized Notes in Table 1



The generated sheet music, derived directly from this MIDI file, is a practical outcome of the analytical pipeline, demonstrating the translation from sound waves to musical notation. Minor manual refinement (beam grouping and triplet bracketing) was applied within the notation software, consistent with the workflow described in Section 2.10.

4 Discussion of Challenges

A few problems showed up while working through this pipeline that are worth calling out specifically, rather than as generic pitfalls of audio analysis.

- **The f0 contour looks flatter than the note table suggests it should be.** In Figure 3, the cyan pYIN contour sits close to a single low band (roughly 400–500 Hz) for most of the clip, with only small dips and rises. But Table 1 lists notes spanning D#5 down to F4, which is more than an octave of range. The most likely explanation is that pYIN locked onto a lower harmonic of the true pitch for

stretches of the recording, a known failure mode when a strong overtone is nearly as loud as the fundamental. It was not possible to verify this against ground truth (e.g., a known score for the recording) within the available time, so it is flagged here as a real limitation of the results rather than something fully resolved.

- **Several detected notes are very short, some under 0.1 seconds.** The B4 at 1.04–1.14 s and the G#4 at 1.56–1.65 s both last 0.09 s, which is short enough that it cannot be stated with full confidence that they are real, deliberate notes rather than transient artifacts from the pitch tracker jumping briefly between values. The quantization script already discards the shortest, most obviously spurious segments, but the threshold for what counts as "real" is somewhat arbitrary, and a stricter cutoff would have merged these into neighboring notes instead.
- **The final 1.8 seconds were labeled as a single sustained F4, but the spectral evidence says otherwise.** As discussed in Section 3.3, the spectral centroid and bandwidth both spike sharply after 3.6 seconds, and the spectrogram shows broadband noise rather than a clean harmonic stack in that region. This is almost certainly the decay and breath noise of the guitar, not a genuinely sustained pitch, but because pYIN still reported a low, relatively stable f_0 value there, the note-grouping logic merged it into one long "note". This is the single result in this paper that is trusted the least.
- **Parameter choices were not tuned rigorously.** The FFT peak-detection threshold (10% of the maximum magnitude) and the STFT settings ($n_{\text{fft}} = 2048$, $\text{hop_length} = 512$) were reasonable defaults rather than values tested systematically against alternatives. Different choices, especially a smaller hop length, would likely change the exact note boundaries in Table 1, even if the overall melodic shape stayed similar.

5 Future Directions

Rather than listing generic next steps, the following ideas are direct responses to the specific problems identified in Section 4:

- **A voicing/energy check before trusting f_0 .** Cross-referencing the pYIN f_0 estimate against the spectral centroid and bandwidth could catch cases like the noisy decay tail at the end of this clip. If the bandwidth is unusually wide for a given segment, that is a signal that the "note" there probably is not real and should be trimmed or discarded rather than kept as a full-length note.
- **Octave-error correction.** Since the flat-looking f_0 contour in Figure 3 appears to be a possible octave-tracking issue, a post-processing step that compares candidate f_0 values against the harmonic peaks in the FFT (Section 2.6.1) could help decide whether the pitch tracker has locked onto a subharmonic.
- **A confidence-weighted minimum note duration,** instead of a single fixed cutoff, so that very short segments are kept only when the voicing probability of pYIN is high. This would directly address the borderline 0.09-second notes.
- **Testing on a recording with a known, written-out melody** would allow the note accuracy of the pipeline to be checked against ground truth instead of relying on a by-ear judgment of whether the output looks reasonable.

6 Conclusion

Going into this project, the expectation was that the hard part would be the mathematics: FFTs, decibel scales, and cepstral coefficients. In practice, the transforms themselves were the easy part; librosa and

SciPy handle the heavy lifting in a couple of function calls. What actually required the most thought was the step that gets the least attention in most tutorials: deciding, from a noisy, continuous f0 estimate, where one note stops and another begins. That judgment call is where most of the interesting problems in this paper originated.

The pipeline did work: a real, playable MIDI file and a piece of engraved sheet music (Figure 5) were obtained that had not been written by hand. However, a close look at the results makes it clear that the output is not fully trustworthy everywhere. The f0 contour in Figure 3 is flatter than the wide melodic range in Table 1 would suggest, which points to a likely octave-tracking error rather than a genuinely narrow melody. Moreover, the last note in the table, a 1.83-second F4, is almost certainly the decay and breath noise of the recording mislabeled as a sustained pitch, based on how sharply the spectral centroid and bandwidth spike in that same window (Section 3.4). The first 3 seconds of Table 1 can be trusted much more than the last two entries.

If this project were extended, the voicing-check and octave-correction ideas in Section 5 would be prioritized over anything more ambitious such as deep-learning timbre synthesis, because fixing the existing blind spots of the pipeline seems more valuable than adding new capabilities on top of a note-detection step that is not yet fully trusted. Even with those gaps, the project did what it set out to do: turn a recording into a symbolic score using signal-processing tools that were assembled and understood step by step, rather than a black-box application.

Acknowledgement

I gratefully acknowledge Dr. Sumit Soman, PhD, Senior IEEE Member, for mentoring this work and serving as an advisor for the research paper.

References

1. McFee, B., Raffel, C., Liang, D., Ellis, D. P. W., McVicar, M., Battenberg, E., & Nieto, O. (2015). librosa: Audio and music signal analysis in Python. In Proceedings of the 14th Python in Science Conference (SciPy 2015).
2. Virtanen, P., Gommers, R., Oliphant, T. E., et al. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17, 261–272.
3. Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3), 90–95.
4. Cooley, J. W., & Tukey, J. W. (1965). An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90), 297–301.
5. Mauch, M., & Dixon, S. (2014). pYIN: A fundamental frequency estimator using probabilistic threshold distributions. In IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2014).
6. de Cheveigné, A., & Kawahara, H. (2002). YIN, a fundamental frequency estimator for speech and music. *Journal of the Acoustical Society of America*, 111(4), 1917–1930.
7. Davis, S., & Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4), 357–366.
8. MIDI Manufacturers Association. (1996). The Complete MIDI 1.0 Detailed Specification. La Habra, CA: MMA.