Musical Instrument Extraction Using Cepstral Techniques

Jason Bissias & Matthew Farris

December 2025; Revised March 2026

1. Introduction

A major part of the modern musician's toolbox is that of samples from other songs. In general, there are several techniques to extract a certain sound from a symphony of unwanted sounds- whether it is carefully applying equalization on the track to bring out only the desired sounds, finding multitrack versions of songs from which to record "stems" (the individual tracks of a song), or using an AI stem splitting tool. [1] At the present moment, AI stem splitters have taken over as one of the quickest ways music producers build up their soundscapes from existing songs. However, AI is a system that requires intense energy and economic resources [2], which drives the question: how can we reduce AI implementation to just the tools which absolutely necessitate its use? Thus, in this report we present a method to extract certain instruments and sounds from a complex combination of overlapping sounds.

The main benefit of the cepstrum perspective of analyzing sound information is that the "fingerprint" of each sound is independent of pitch, it essentially presents the timbre of the sound [3].

2. Mathematical Background and Methodology

To explore the mathematics of what makes cepstral based instrument separation effective, an abstract view of the algorithm is necessary.

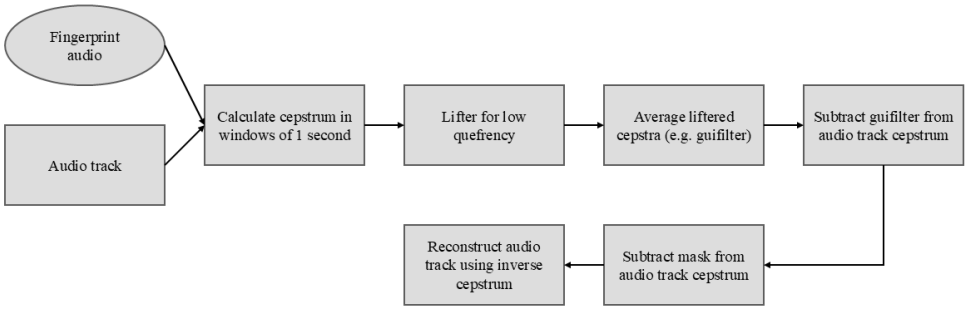

Figure 1 Flowchart of algorithm

The flow chart above describes each step in the algorithm from the point of view of the input data. “Fingerprint audio” refers to an audio sample of the target instrument. The other input is an audio track containing the desired sample. The final output is the audio track in its entire with the target instrument isolated or made far more prominent compared to other instruments.

The first step is calculating the cepstrum of both the audio track and the audio fingerprint in windows of 1 second. 1 second is chosen as it provides a sufficient amount of time for most instruments to produce several cycles of their waveforms, while also being small enough to not include cofounding data of the instruments.

For the fingerprint, the cepstral data is liftered to target low quefrency components. Low Quefrency corresponds to high frequency in the frequency domain. It’s often used in speaker recognition as it contains slow changing information unique to each speaker. In the context of instrument extraction, timber, or the unique sound profile of an instrument, is contained in low quefrency portions of the cepstrum.

These liftered cepstrum are averaged together to remove variance associated with pitch, which is located at higher quefrencies and changes much more rapidly. The final result is a cepstral fingerprint of an instrument’s timbre. Ideally the instrument used for fingerprinting is an audible match for the target instrument in the song.

The audio track is then split into 1 second segment and the cepstrum is calculated for each segment. The fingerprint cepstrum is then subtracted from the audio track cepstrum for each segment. This produces a “mask” that removes or significantly suppresses the target instrument. This mask is subtracted again from the original audio track cepstrum to leave only the target instrument behind. The subtracted cepstrum has its inverse cepstrum calculated and each 1 second window is concatenated into a continuous time domain signal.

Before the mask can be applied to the original track, a scaling factor is applied to ensure that an appropriate “amount” of the mask is used. The target instrument might not be present in the same

magnitude throughout the track. The scaling factor solves this problem by measuring how much of the target instrument's timbre is contained in a given segment.

This scaling factor is determined by calculating the root mean square error of the fingerprint and audio track. When the target instrument has a larger presence in the segment, the error will decrease, and the mask subtraction won't be as prominent as segments with less presence.

$$Scaling\ Factor = \frac{RMSE(Audio\ Track - Fingerprint)}{10}$$

Cepstral techniques were preferred as the main tool for the extraction as they feature separability of frequency components via simple addition and subtraction. This property originates from the logarithmic operator in the formula for calculating the cepstrum seen below.

$$C(x) = F|\log(|F(x)|^2)|^2$$

Conceptualizing the combination of the target instrument with the rest of the symphony as a signal with a noise term, we can separate these components using the technique previously mentioned. Noise separation by using a reference track is not an original concept and in [4], wind noise from recordings of voices is reduced by first building a dictionary of wind noises and voices.

$$X = DH$$

In this paper, separation through the use of non-negative matrix factorization is discussed. The equation above describes the approach of the paper. The D matrix refers to the dictionary of wind noise and voices from a recording. H is the code matrix, which provides information on which elements from the dictionary matrix would be used at a given time. The product of these two matrices is understood to constitute the short time Fourier transform of the signal.

This technique does not rely on the cepstrum for its analysis, but building a dictionary and a control component for where it is used and to what magnitude is applicable to the technique used in this experiment.

In [3], cepstral techniques are used in the identification of instruments. In Figures 1 and 2 of the paper, we see a three-dimensional plot of a flute and clarinet's power spectrums changing over time. This idea was also borrowed for the cepstral fingerprint technique. Finally, in [5], spectra are analyzed using an average over the entire recording of an instrument. In this paper, a sustained

note was played for each instrument to better acquire the unique attributes of the instrument's spectrum.

3. Experiments and Results

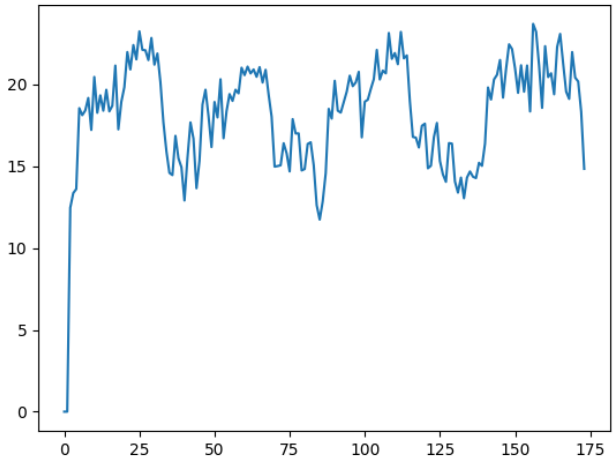

Figure 2: RMSE over song length

The first experiment conducted to test the use of a cepstral fingerprint for instrument extraction is illustrated in Figure 2. The figure shows the RMSE calculated between a cepstral fingerprint and an audio track over 1.5 second windows. The fingerprint was simply the first 1.5 seconds of the track. No other alteration was made to it.

The error appears to take on a periodic pattern as the track progresses. This is due to the fact that the fingerprint contains a section of the track that repeats in these sections. Specifically, a combination of a guitar riff and a drum pattern that would play in the intro and in the chorus of the track. The error can be considered a metric of how similar the two cepstrums appeared and was evidence that further refinement could allow for a viable experiment.

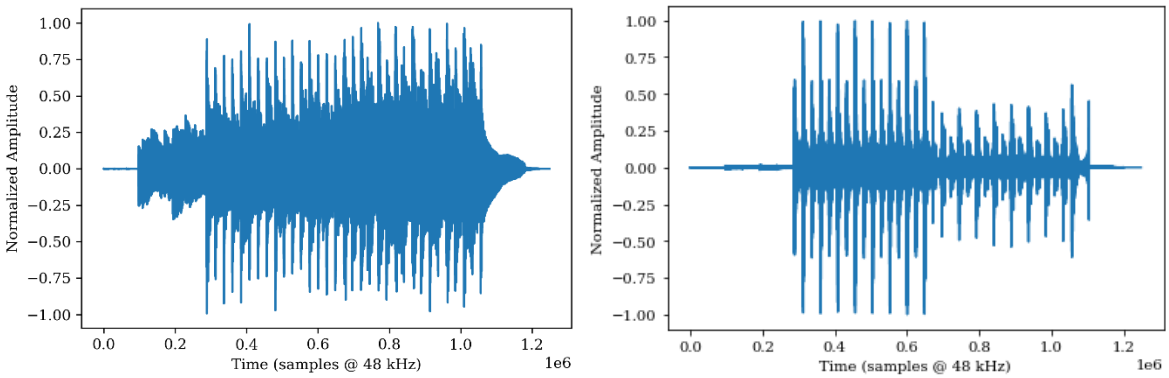

Figure 3: Original waveform (left) and generated mask (right)

Figure 3 is the inverse cepstrum of the masked audio track and the waveform of the original track. The opening to this track features a guitar riff, which was also the target instrument for extraction. In the first subtraction, we can see this riff all but eliminated in the opening. A significant portion of the waveform is also removed in the first subtraction.

Audibly, the guitar is very faint in the track, but other instruments such as the drums and keyboard mostly maintain their sound, but a noticeable hiss is present. This is likely due to high frequency components being missed during the subtraction. A sort of residue of the guitar.

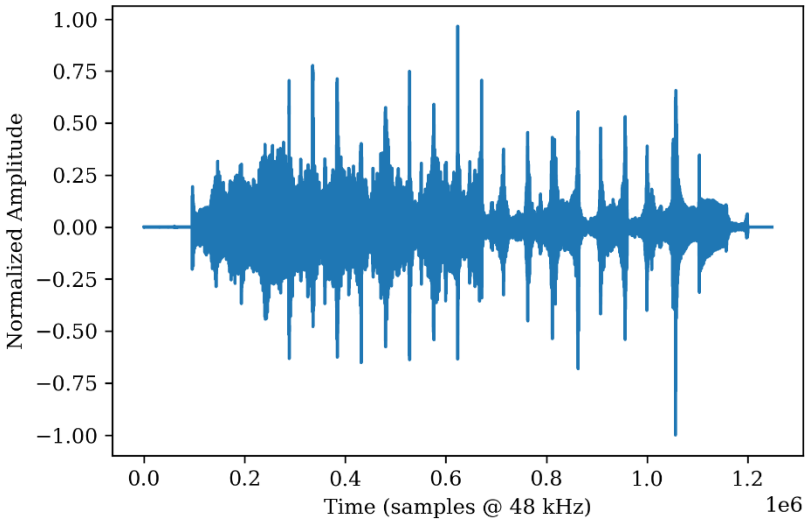

Figure 4: Final output

Figure 4 is the second subtraction and contains, ideally, the isolated guitar track. The opening riff is visible, and the periodic spikes of the drums and keyboards are suppressed. This is reflected

when listening to the track itself. The drums and keyboards suppressed significantly while the guitar was somewhat muted but still very prominent throughout the track.

4. Conclusion

Instrument extraction through the use of cepstral techniques is clearly effective at sometimes isolating or enhancing target instruments in audio tracks. In its current form the algorithm can't produce audio of sufficient quality for proper sampling in most use cases. It could enhance the effectiveness of other techniques for instrument extraction such as standard filtering or machine learning and AI based techniques.

The algorithm itself could be improved by changing the scaling factor to better reflect a quantized amount of the target instrument present. This would prevent artifacts of the fingerprint from imprinting onto the final output. This was a common issue in the early phases of development.

Additionally, liftering more precisely and excluding pitch in forming the fingerprint could enhance the performance of the extraction by not favoring specific pitches accidentally included.

References

1. Tracklib (2025), *Music Sampling: A Beginner's Guide*, Tracklib, https://www.tracklib.com/blog/music-sampling-guide.
2. IEA (2025), *Energy and AI*, IEA, https://www.iea.org/reports/energy-and-ai.
3. Eronen, A. & Klapuri, A. (2000), *Musical instrument recognition using cepstral coefficients and temporal features*, IEEE, https://doi.org/10.1109/ICASSP.2000.859069.
4. Schmidt, M., Larsen, J., & Hsiao, F-T. (2007) *Wind Noise Reduction using Non-negative Sparse Coding*, IEEE, https://doi.org/10.1109/MLSP.2007.4414345.
5. Brown, J. C. “Computer identification of musical instruments using pattern recognition with cepstral coefficients as features.” J. Acoust. Soc. Am. lOS(3) 1933-1941.

Appendix

```
from scipy.io import wavfile
from core import rceps
import numpy as np
from sklearn.metrics import mean_squared_error
from scipy import signal, datasets, ndimage
from utils import find_closest_ind
import matplotlib.pyplot as plt
import wave
from acoustics.cepstrum import complex_cepstrum, inverse_complex_cepstrum
samplerate, data = wavfile.read('Song_1.wav')
samplerate2, data2 = wavfile.read('chime_1.wav')
#data=np.mean(data, axis=1, dtype=data.dtype)
ceps_sum=np.zeros(int(round(samplerate2 *1))-1)
print(samplerate2)
windows = np.zeros(int(round(samplerate *1)))
for i in range(1, round((len(data2)/int(round(samplerate*1))))):
    FingerPrintAudio=data2[int(round(samplerate2*1))*(i-1):int(round(samplerate2*1))*i-
1]
    ceps,ndelay=complex_cepstrum(FingerPrintAudio)
    L=int(len(ceps)*0.4)
    lifter = np.zeros_like(ceps)
    lifter[:L] = 1
    lifter[-L+1:] = 1
    ceps=ceps * lifter
    ceps_sum=ceps_sum+ceps
    phase_sum=phase_sum+ndelay
ceps_avg=ceps_sum/(round((len(data2)/int(round(samplerate*1))))-1)
ndelay=phase_sum/(round((len(data2)/int(round(samplerate*1))))-1)
#error=np.zeros(round((len(data)/int(round(samplerate*0.5)))))
#FingerPrintquef, FingerPrintC = rceps(FingerPrintAudio,qmin_ind=0, qmax_ind=None,
sr=samplerate)
#print(FingerPrintC)
output=np.zeros(len(data))
output2=np.zeros(len(data))
```

```
for i in range(1, round((len(data)/int(round(samplerate*1))))):
    print(i)
    testAudio=data[1+int(round(samplerate*1))*(i-1):int(round(samplerate*1))*i]
    ceps_test,ndelay2=complex_cepstrum(testAudio)
    processed=ceps_test-ceps_avg
    processed2=ndelay2-ndelay
    final2=ndelay2-processed2
    diff=(processed)**2
    square_mean = np.mean(diff)
    root_mean_square=np.sqrt(square_mean)
    final=ceps_test-processed*(root_mean_square/10)
    final2=ndelay2-processed2
    reconstructed=inverse_complex_cepstrum(ceps_avg, ndelay)
    reconstructed2=inverse_complex_cepstrum(processed, ndelay2)
    output[1+int(round(samplerate*1))*(i-1):int(round(samplerate*1))*i]=reconstructed
    output2[1+int(round(samplerate*1))*(i-
1):int(round(samplerate*1))*i]=reconstructed+reconstructed2
output=output/np.max(np.abs(output))
output2=output2/np.max(np.abs(output2))
output_filename = "guifilter.wav"
wavfile.write(output_filename, samplerate, output.astype(np.float32))
wavfile.write("recombined.wav", samplerate, output2.astype(np.float32))


plt.plot(output)
#plt.plot(data)
plt.show()
```