# Beat Detection Algorithm for Popular Music with Fast Fourier Transform

Ishan Balakrishnan
*University of California, Berkeley*
*Computer Science and Business Administration*
Berkeley, CA, United States
ishan.balakrishnan(at)berkeley.edu

Sukhm Kang
*University of Chicago*
*Computational and Applied Mathematics*
Chicago, IL, United States
sukhmkang(at)uchicago.edu

***Abstract*—This paper presents a method for extracting the tempo of a piece of music using the Fast Fourier Transform and a noise-filtration algorithm. Upon receiving an audio signal, the program breaks the signal into seven discrete frequency ranges and uses an iterative approach to distinguish musical beats from noise. The algorithm was implemented using Python, trained on the Top 100 songs on Spotify from 2016 and 2018, and tested on 300 songs from various genres. The algorithm achieved an 86.8% success rate at identifying a song's tempo in beats per minute (BPM). These results were sustained across popular music from different genres and songs of various tempos.**

***Index Terms*—beat detection, peak detection, signal processing, tempo detection, computational musicology**

## I. Introduction

A beat is defined as a perceivable pulse within a piece of music, usually occurring at regular intervals and providing a sense of rhythm. A slightly less rigorous but more accessible definition of the beat of a piece of music, provided by Tzanetakis, et al., is "the regular. . . sequence of pulses corresponding to where a human would tap his [or her or their] foot while listening to the music" [1]. Beats are defined in relation to a piece's meter, which defines the intervals, cycles, and patterns of musical beats. In this paper, music with a clearer beat and metrical clarity is referred to as "clappable." "Clappability" is by no means an objective metric of clarity, as listeners from different backgrounds may interpret rhythmic structures in different ways. However, "clappability" is still a broadly useful indication of the presence of a beat in a piece of music. The underlying beat of a piece of music is closely related to its tempo, which is defined as the speed at which a piece of music is played; in particular, musical tempo is usually measured in beats per minute (BPM). The beat of a song, especially in the context of Western popular music, is one of its most recognizable and essential components. The salience of the beat within music has led to significant academic efforts into developing algorithms for beat and tempo detection within an audio signal. This paper aims to add to the discussion of beat detection algorithms by providing a novel methodology for finding the tempo of a piece of music and by focusing specifically on tailoring this methodology to modern popular music.

Beat and tempo detection have several significant applications, most notably within music curation and classification algorithms, fields that are becoming increasingly relevant in the age of digital streaming platforms such as Spotify, Apple Music, and Pandora. In addition, both music transcription and visualization software relies heavily on being able to detect beats, drum events, and tempo within a piece of music. Beat detection algorithms can also have applications outside of music, in fields such as cardiac arrhythmia detection, which relies on beat detection in order to find periodicity (or a lack of periodicity) within a signal from a heart monitor.

### A. Literature Review

Computational musicology–the combination of music theory with music informatics–is a relatively new field of research with numerous unexplored applications. Recent efforts have focused largely on computationally studying musical patterns in real-time, with several new studies attempting to track audio tempo and measure BPM instantaneously. Most of these real-time techniques, however, have yet to be applied to popular music; for example, Mottaghi et al. [2] work with ballroom music, Jensen, et al. [3] use computer-generated audio tracks, and Cheng et al. [4] analyze music from a variety of genres and sources including classical and cinematic music. Furthermore, these studies lack a comprehensive review of their algorithmic accuracy when measuring the BPM of music. Therefore, it is uncertain if their results are transferable to the modern era of popular music.

There is also a significant body of literature discussing the applications of tempo-detection even if the detection was not completed in real-time, but these studies similarly lack clearly quantified results and do not use popular music as training or testing datasets [5]. The foundation for much of the work that algorithmically detects the beat of music is audio compression and transformation; thus, several studies explore the applications of the Discrete Fourier Transform and Huffman Coding on music analysis [4], [5], [6], [7], [**?**].

Goto and Muraoka [**?**] study popular music and develop an algorithm with upwards of 86% accuracy to track BPM in real-time, but their study is from over two decades ago and may not apply to the popular music of today. Similarly, the work of Zhu and Wang [**?**] employs an approach incorporating the Huffman Coded Domain to track the tempo of popular music with 84% accuracy.[1] While both of these works offer promising findings, popular music has changed considerably in recent years, with the integration of new audio technologies– modern popular music incorporates a different combination of sound, mastering, drums, and techniques than popular music from a decade or two decades ago. These modifications to music may have led to an increase in ”noise,” which we define as the collection of sounds, instruments, or audio effects in a piece of music that are unhelpful when searching for the beat. Building off of the work started by Goto and Muraoka [**?**] and Zhu and Wang [**?**], we aim to create a beat-detection algorithm which can filter such noise by training it on datasets of modern popular music.

A more accurate, reliable, and updated approach to measuring the BPM of popular music can be integrated into existing research efforts such as the work of Kirovski and Attias [**?**], who propose a quantitative identifier for music using its BPM and rhythmic structure; such techniques could pave the way for new classifications and categorizations of modern music.

### *B. Objective*

Our primary objective is to test and design a noise-reducing beat-detection algorithm which accurately measures the tempo of modern popular music using Python. Such an algorithm must be able to:

1) Break down the audio signal of a piece of music into distinct frequency ranges using the Discrete Fourier Transform
2) Develop a method to identify the peaks across these distinct frequency ranges
3) Create a process for eliminating peaks associated with noise so that only the beats within a piece of music are left
4) Measure the amount of time between detected beats to calculate the BPM

To do this, we adopt a two-pronged ”training and testing” approach to iteratively build the broader algorithm, comprised of the listed components above. This requires training the algorithm and using its output as feedback to repeatedly adjust our methods related to Steps 2 and 3 until a fully optimized detection algorithm is created.

### *C. Hypothesis*

We put forth the following hypotheses about the algorithm's performance:

1) The algorithm performs better on genres with a more ”clappable” beat such as Pop, whereas the algorithm is less effective at detecting the tempo of genres characterized by more complex rhythms and greater presence of noise such as Rock.
2) The algorithm performs better on music with a faster tempo than songs with a slower tempo. If there is more time between two beats, there are more opportunities for the algorithm to identify an incorrect beat due to noise. Additionally, slower songs are generally less ”clappable” than faster ones.

## II. Methodology

The algorithm consists of three phases. First, upon input of an audio file (in .wav format[2]), the Fast Fourier Transform (FFT) separates the audio signal encoded in the .wav file into seven distinct frequency ranges. Second, the algorithm identifies peaks within each frequency range (which, we argue, correspond to beats in the music) and analyzes the distance between the peaks using a moving window. By distance, we refer to the amount of time (in seconds) between peaks in the audio signal. Finally, the program calculates the tempo, in beats per minute, of the piece of music by using the groups of peaks that are spaced out most evenly. We search for the peaks with the most regular spacing since these peaks are most likely to represent beats within a piece of music. The parameters of the algorithm were trained on two sets of 100 popular songs (Spotify Top 100 from 2016 and 2018) and the algorithm was then tested on two sets of 100 popular songs (Billboard Hot 100 from 2021 year-end and April 2022) and one set of 100 rock songs (Spotify '00s Rock Anthems Playlist). A full pseudocode of the algorithm is included in the Appendix and a top-level view of the algorithm's design is included in Figure 1.

### *A. Assumptions*

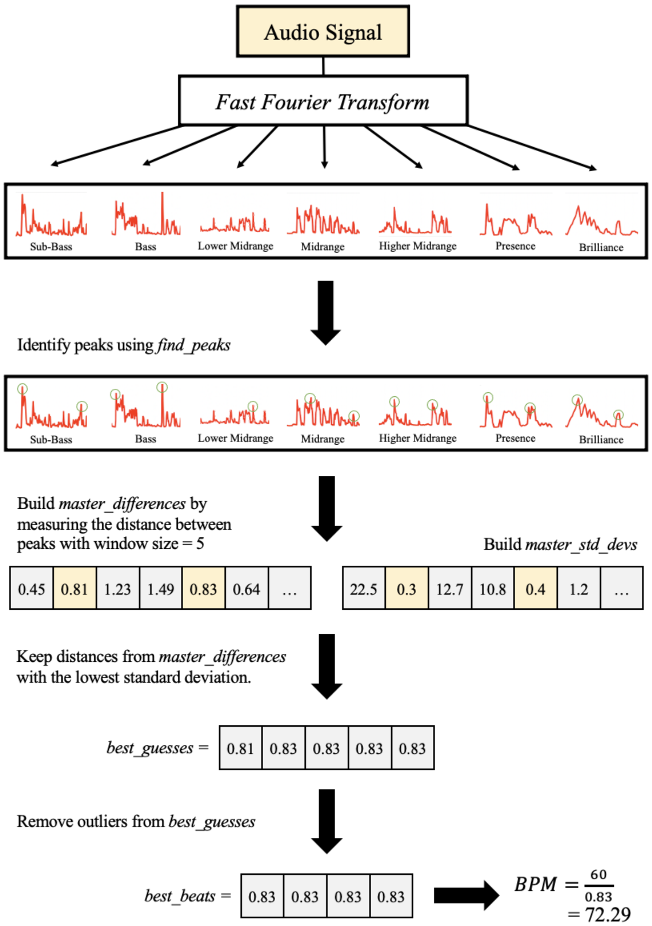
We make several assumptions about the music that the algorithm is used for. First, we assume that the analyzed music has a constant tempo and a time signature that does not change throughout the piece. Second, the logic of the algorithm is based on the premise that at least one frequency range will contain a regular half-note, quarter-note, or eighth-note pulse corresponding to the beat of the song. Though this is not a radical assumption for most music, it may cause the algorithm to fail for pieces of music with unusual beats or for slow ballads that consist solely of piano and vocals and therefore lack a clear, ”clappable” beat. Similarly, the algorithm implicitly assumes that no other instruments will be playing beats at regular intervals other than the BPM of the song.[3] Third, the algorithm assumes that the distances between noisy peaks that do not correspond to the music's true tempo will always have less regularity than the peaks corresponding to beats. This assumption lays the foundation for our noise-filtration process. Finally, the algorithm is designed to analyze

[1] The algorithm correctly identified the tempo of 21 out of 25 popular songs from commercial CDs.

[2] A .wav file is a lossless audio format which leaves the original analog audio uncompressed [**?**]. The file format was created by Microsoft. The algorithm uses .wav files because of the availability of the Wave module [**?**] in Python, which allows the computer to efficiently read data from .wav audio files.

[3] While the phenomenon of a piece of music overlaying two meters over each other simultaneously (“polymeter”) is unheard of in popular music, making our algorithm's assumption reasonable, polymeter is used in some non-popular music. For an example of polymeter, see “5/4” by Gorillaz, in which the pulses in the drum beat suggest a 4/4 time signature, whereas the use of accents in the guitar suggest a 5/4 time signature.

Fig. 1. Beat Detection Algorithm Design




music consisting of two channels (left and right) with a sampling rate of 48 KHz.[4]

### B. Fast Fourier Transform Background

It is beyond the scope of this paper to provide an in-depth derivation of the Fast Fourier Transform, or FFT, since its use in music analysis and beat detection is well-established. However, this paper provides some of the mathematical intuition behind the FFT in order to contextualize its use in the beat detection algorithm.

The Fast Fourier Transform is a particular application of the Discrete Fourier Transform, or DFT. A brief summary of the Discrete Fourier Transform, when applied to signals, is that it transfers a signal from the time domain to the frequency domain. Specifically, the DFT takes an input vector $f$, with each element in the vector constituting a sample, and returns a vector of frequencies $\hat{f}$ that make up the input vector. Each element $\hat{f}_i$ of $\hat{f}$ is a complex number representing the phase and magnitude of the frequency corresponding to index $i$, with the real component representing the magnitude and the imaginary component representing the phase.

[4]Though the algorithm is currently designed for audio files matching these criteria, it can be easily modified to take in mono audio and/or accommodate a sampling rate of 44.1 KHz, another common sampling rate for audio files.

In particular, the DFT can be calculated using the following sum:

$$\hat{f}_k = \sum_{j=0}^{n-1} f_j \cdot e^{-i2\pi jk/n} \tag{1}$$

For simplicity, we use the following common notation:

$$\omega_n = e^{-2\pi i/n} \tag{2}$$

The sum in (1) can be trivially rewritten as the following (square) matrix-vector multiplication, assuming $f$ is a vector with $n$ elements:

$$\hat{f} = \begin{bmatrix} 1 & 1 & \dots & 1 & 1 \\ 1 & \omega_n & {\omega_n}^2 & \dots & {\omega_n}^{n-1} \\ 1 & {\omega_n}^2 & {\omega_n}^4 & \dots & {\omega_n}^{2(n-1)} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & {\omega_n}^{n-1} & {\omega_n}^{2(n-1)} & \dots & {\omega_n}^{(n-1)^2} \end{bmatrix} f \tag{3}$$

The innovation of the Fast Fourier Transform, or FFT, is that the matrix multiplication in (3) can be decomposed into a nested sequence of less computationally intensive matrix multiplications [?]. Letting $F_n$ be the $n$ by $n$ matrix in (3), the matrix-vector multiplication can be written as:

$$F_{2n} = \begin{bmatrix} I_n & D_n \\ I_n & -D_n \end{bmatrix} \begin{bmatrix} F_n & 0 \\ 0 & F_n \end{bmatrix} P \tag{4}$$

The matrix $P$ is a permutation matrix that reorders the elements of the outputted vector, as the matrix multiplication in (4) results in the correct coefficients but in the incorrect order. $D_n$ is a diagonal matrix:

$$D_n = \begin{bmatrix} 1 & & & & \\ & \omega & & & \\ & & \omega^2 & & \\ & & & \dots & \\ & & & & \omega^{n-1} \end{bmatrix} \tag{5}$$

Multiplication by a diagonal matrix such as $D_n$ is less costly than multiplying by the $n$ by $n$ matrix in (3).

The matrix decomposition in (4) is the core of the Fast Fourier Transform algorithm. The FFT requires that the number of elements in $f$ is a power of two, namely because $F_{2n}$ is decomposed into $F_n$ block matrices recursively until the matrix multiplication is reduced to non-costly 2 by 2 matrix multiplications. Thus, the FFT is considered an $O(n \log(n))$ run-time operation, compared to the raw matrix multiplication of the DFT, which is of $O(n^2)$ time-complexity [?]. For large enough $n$, this discrepancy leads to significant computational benefits.

To summarize, the FFT is a computationally cheap way to decompose a signal into the frequencies that make up the signal.

### C. Frequency Breakdown Using Fast Fourier Transform

The goal of the first section of the algorithm is to break down an inputted song (a .wav file) into seven different frequency ranges. The purpose of breaking down the audio signal into seven frequency ranges is the intuitive notion that the presence of beats may be more apparent in certain frequency ranges (e.g., within certain instruments that are playing on the downbeats of a song) than in the audio signal as a whole. Using the FFT, the algorithm separates an audio file into the following seven audio ranges:

TABLE I
FREQUENCY RANGES

| Name | Frequency Range |
|---|---|
| Sub-Bass | 16-60 Hz |
| Bass | 60-250 Hz |
| Lower Midrange | 250-500 Hz |
| Midrange | 500-2000 Hz |
| Higher Midrange | 2000-4000 Hz |
| Presence | 4000-6000 Hz |
| Brilliance | 6000-20000 Hz |

Source: CUI Devices. [?]

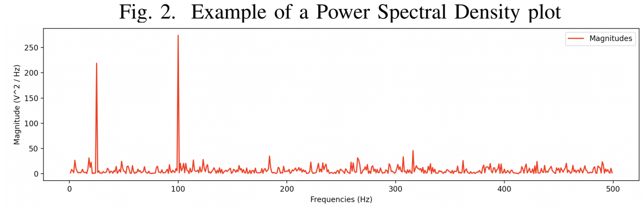
Our usage of the FFT for beat detection is similar to that of Cheng, et al. [4] and Scheirer, et al. [?], who both employ the FFT to break down audio signals into several discrete frequency ranges before identifying beats in the signal.

A .wav file contains a sequence of bytes in little-endian format, with the encoded integers representing an audio signal. Each element in the sequence of bytes is referred to as a "sample," which can be thought of as the value of the audio signal at an instantaneous moment in time. The music analyzed by this algorithm had a sampling rate $s$ of 48 KHz, meaning every second of music consists of 48,000 equally spaced samples.

The chunk size $c$, which is the amount of samples that the algorithm reads from the audio file at a time, for our algorithm was 1024 samples. Using a chunk size that is a power of two maximizes the computational benefits provided by the FFT algorithm, which is most efficient when the length of the input vector is a power of two.[5] Furthermore, 1024 is also the chunk size adopted by Mottaghi, et al. in their implementation of the Fast Fourier Transform for beat detection [5].

For each chunk $f$ of audio data, we computed its Fast Fourier Transform, returning a vector $\hat{f}$ of complex numbers representing the frequencies that make up the audio signal. For the purposes of beat detection, it is not necessary to distinguish between the magnitude and phase components of the complex numbers in $\hat{f}$. Thus the vector $\hat{f}$ was converted to $\hat{f}_{magnitudes}$ by multiplying each element of $\hat{f}$ by its complex conjugate. This multiplication results in what is called the Power Spectral Density, or PSD, of a signal. To be precise, the PSD is measured in units $V^2$/Hz and indicates how much of a particular frequency is present in the inputted chunk $f$. The following figure is an example of the Power Spectral Density of a signal consisting of the sum of two sin functions, one with a frequency of 100 Hz and the other with a frequency of 25 Hz.

Fig. 2. Example of a Power Spectral Density plot




As shown in the figure, the peaks in the plot correspond to the frequencies with the most representation in the inputted sample. The same calculation represented in Figure 1 is performed on each chunk of 1024 samples from the inputted piece of music.

Thus, for each chunk $f$ of audio data, we calculated a vector $\hat{f}_{magnitudes}$ of the relative power of each frequency that makes up the audio chunk. In order to understand which elements of $\hat{f}_{magnitudes}$ corresponded to which frequency ranges, we employed the following equation, where $h_i$ is the frequency corresponding to element $i$ of $\hat{f}_{magnitudes}$, $s$ is the sampling rate, and $c$ is the chunk size:

$$h_i = i \cdot \frac{s}{c} \tag{6}$$

Using equation (6), we categorized the elements of $\hat{f}_{magnitudes}$ into the seven frequency ranges listed in Table I.

In particular, for each frequency range, we take the average of all elements of $\hat{f}_{magnitudes}$ with indexes falling within that particular frequency range. Equation (7) is an example to make this step clearer and less abstract. As an example, $p_{brilliance}$, the average power level in the brilliance category, can be calculated with the following equation:

$$mean(\{\hat{f}_i \ s.t. \ 6000 \leq i \cdot \frac{s}{c} \leq 20000\}) \tag{7}$$

By performing this calculation for each chunk in the song and saving the power levels within each frequency range in separate arrays, the algorithm is able to reconstruct a graph of the power level (measured in units $V^2$/Hz) of each frequency range over time. To make this clearer, Figure 3 depicts the power level of the seven frequency ranges that make up Katy Perry's "Teenage Dream" plotted over time.

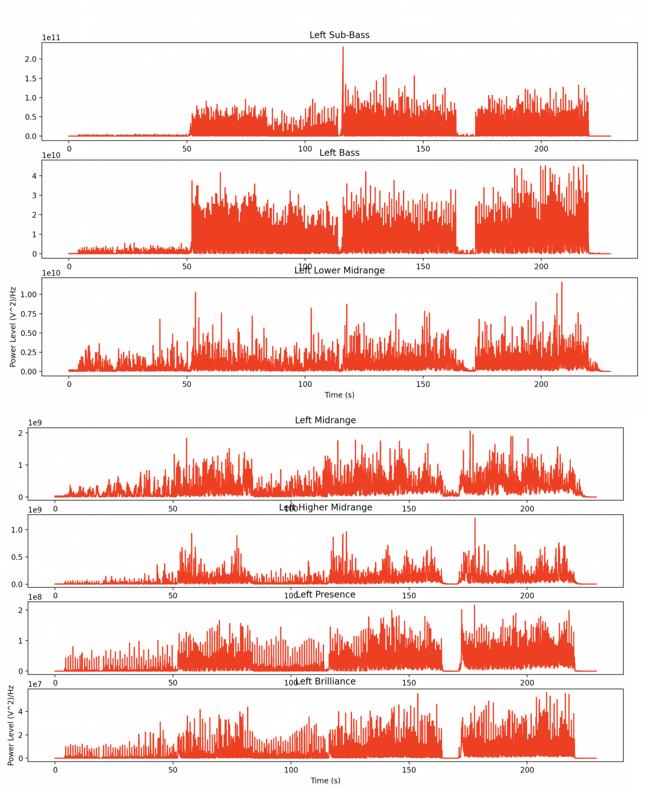
Once the piece of music is broken down into 7 frequency ranges, the data moves to the next stage of the algorithm.

### D. Peak Detection in Frequency Ranges

The goal of this component of the algorithm is to find peaks in the power levels of each frequency range over time. This is driven by the assumption that peaks in power correspond to beats in a song. Therefore, finding a set of peaks within each

[5]Though minimizing runtime is not the primary objective of this algorithm's design, having a fast runtime is helpful for running an algorithm on hundreds of songs using different combinations of parameters, which was necessary for the training and testing of the algorithm. The program takes one to two seconds to generate a song's BPM, depending on the length of the song.

Fig. 3. Plot of Power Level of Frequency Ranges Over Time for "Teenage Dream"




frequency range is a key step to determining the tempo of a piece of music.

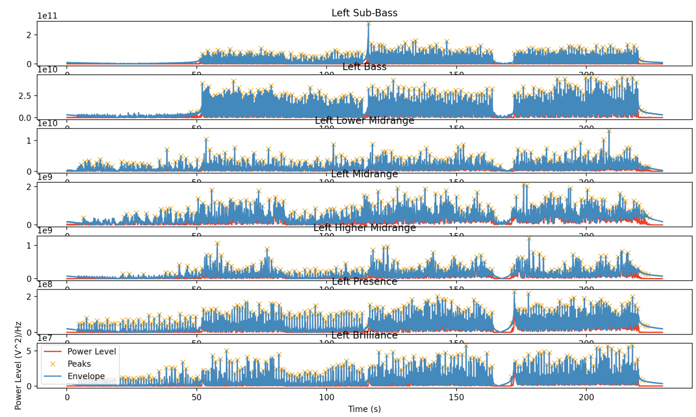
The algorithm first uses the *hilbert* function from the SciPy module to find the upper envelope of the audio signal within each frequency range [?]. The upper envelope of an audio signal is a smooth curve that intersects a signal's peaks, acting as an upper boundary for the signal [?].[6] Analyzing the envelope provides a smoother estimation of the peaks of the audio data within each frequency range. For further examples of usage of the envelope in peak detection, see the work of Cheng, et al. [4] and Scheirer [?], which both use the envelope in a similar manner to our algorithm.

A point $x_o$ is defined as a local maximum, or peak, of a function $f$ if and only if:

$$\exists \delta \; s.t \; \forall x \; s.t \; d(x, x_o) < \delta \; we \; have \; f(x) < f(x_o) \tag{8}$$

The algorithm employs the *find_peaks* function offered by the SciPy module in order to find peaks in the power of each frequency range over time [?]. The *find_peaks* uses a methodology similar to that of equation (8) to identify peaks in a dataset. *find_peaks* offers multiple parameters, namely the minimum distance between peaks and the minimum height for a point to be considered a peak. After tuning the minimum distance and minimum height parameters using the training data, we find that a minimum distance of 0.938 seconds and a minimum height of the mean power level over the entire song led to the most accurate results.We elaborate on this process and how we selected these specific values in the Training and Testing section. Plotting the envelope and the detected peaks leads to a graph as seen in Figure 4, which shows the envelope, the signal, and the detected peaks for Katy Perry's "Teenage Dream."

Fig. 4. Plot of Power Level of Frequency Ranges Over Time for "Teenage Dream" with Envelope and Detected Peaks




### E. Distinguishing Beats from Noise

We find that the *find_peaks* function often detects peaks that do not correspond to beats within a song. This is because audio signals, especially those of polyphonic music, contain large amounts of noise that may result in peaks in the song's power level that do not necessarily correspond to beats in the song. Furthermore, the design of *find_peaks* is relatively simplistic, as the algorithm labels any data point that is greater than the two points adjacent to it as a peak. According to the documentation of the function, if a minimum distance between peaks is inputted, the *find_peaks* function removes the lowest peaks until all remaining peaks satisfy the minimum distance requirement [?]. *find_peaks*, due to its simplicity, is highly susceptible to identifying noisy peaks in the data, creating the need for an approach to distinguish noisy peaks from peaks that correspond to beats in the music.

We adopt the following iterative methodology to begin the noise-reduction process:

1) Create a window of the first five peaks detected
2) Calculate the distance between each adjacent peak in the window
3) Compute the mean and standard deviation of the distances between the peaks in the windows and store the mean and standard deviation in separate arrays
4) Shift the window over by one peak and repeat steps (1) through (4)

We conduct this process for every frequency range, which creates two arrays: $master\_differences$ and $master\_standard\_deviations$. $master\_differences$ holds

[6] Note that our definition of the envelope is not mathematically rigorous. For a rigorous treatment of how the envelope is calculated, see the SciPy documentantion for the *hilbert* function.

the mean distances between the peaks from each window and $master\_standard\_deviations$ holds the standard deviation of the distances between the peaks from each window.

In order to separate beats from noise, the algorithm only keeps the five mean differences with the lowest standard deviations. In other words, the algorithm only keeps data from the five windows where the distance between the peaks has the highest regularity. We posit that these windows, because of the high periodicity of the peaks, are most likely to constitute "beats" rather than noise. This means that, after the filtration process, the algorithm determines the tempo of a piece of music based on approximately 8.8 to 25 seconds of audio.[7]

Finally, left with an array, hereafter referred to as $best\_beats$, of the five average distances with the highest regularity, we remove all elements from $best\_beats$ that are outside of one standard deviation of the mean of $best\_beats$. This ensures that, if $best\_beats$ contains both double the song's tempo and the song's actual tempo, one of the two are filtered out, leaving one guess for the song's tempo.

Once outliers are removed from the $best\_beats$ array, the average of the $best\_beats$ array is taken as the algorithm's final guess for the song's tempo in beats per minute.

Table II displays the efficacy of the noise-removal process for the first 10 songs on Spotify's Top Songs of 2016 playlist, which was part of the training dataset used for the algorithm. The "BPM Before Filter" column shows what the algorithm would have outputted for the song's tempo without filtering out windows with higher standard deviations. The "BPM After Filter" displays the algorithm's output for the song's tempo after filtering out noise by using the standard deviation of the distance between peaks. Table III showcases the efficacy of the noise-removal process across the training data by comparing the percent error before and after filtering out noise. Here, percent error refers to the average percent error between the algorithm's calculated tempos and the actual tempos of the songs in the dataset.

TABLE II
BPM OUTPUT BEFORE AND AFTER NOISE FILTRATION

| **Title** | ***Before*** | ***After*** | ***Actual*** |
|---|---|---|---|
| Closer ft. Halsey | 83.5 | 95.0 | 95 |
| Love Yourself | 77.3 | 100.4 | 100 |
| One Dance | 80.6 | 104.2 | 104 |
| Starboy | 88.2 | 93.1 | 93 |
| Hello | 74.0 | 126.1 | 79 |
| Panda | 145.3 | 144.4 | 143 |
| Hurts So Good | 156.4 | 119.9 | 120 |
| Cheap Thrills ft. Sean Paul | 83.1 | 89.9 | 90 |
| Work | 82.7 | 52.6 | 92 |
| I Know What You Did Last Summer | 75.8 | 113.7 | 114 |

*All numbers in this table are in units of BPM.

TABLE III
PERCENT ERROR BEFORE AND AFTER NOISE FILTRATION

| **Dataset** | **Before** | ***After*** |
|---|---|---|
| Spotify 2016 | 25.4% | 4.15% |
| Spotify 2018 | 26.4% | 2.06% |

*Peak distance held constant at 0.94 seconds.

Brief analysis of Tables II and III suggests that the noise filtration algorithm is broadly effective at distinguishing between beats and noise, or otherwise random peaks in the audio data. More detailed exploration of the effectiveness of the algorithm is included in the Results and Discussion sections.

### F. Training and Testing

Upon implementing a naive Python version of the algorithm, we utilized two training datasets to build and refine our algorithm. Our training datasets were the Spotify Top 100 Songs of 2016 and the Spotify Top 100 Songs of 2018, as these playlists consist of unique popular music from a variety of artists and genres including pop, rock, hip-hop, country, and rap. These playlists are ideal training datasets as they contain no overlapping music and their artists are largely distinctive, meaning that although there are some artists with multiple songs in the 2016 and 2018 training sets, redundancies are majorly minimized. Between the two datasets, our algorithm was trained using feedback from 200 distinct songs and more than 150 unique artists or combinations of artists [NOTE TO SELF: check numbers].

Our training approach modifies the underlying logic or parameters of the following algorithms: Peak-Detection and Peak Analysis.

*1) Training the Peak Detection Algorithm:* As mentioned in Section D, the $find\_peaks$ function offers two parameters for manipulation: the minimum distance between peaks and the minimum height of a peak.

Through our training process, both of these parameters were altered to optimize for algorithmic accuracy.

(a) The optimal minimum distance value of the $find\_peaks$ function is 0.938 seconds. By enforcing this minimum restriction, the algorithm's calculated BPM output is capped at approximately 63.96, but in doing so, the algorithm is able to exactly detect one-fourth or one-half of the music's BPM with improved precision, in the case that the music's tempo exceeds 63.96 BPM. At the start of the algorithm's training, the minimum distance was enforced at a value of 0.320 seconds. Under these constraints, the algorithm could theoretically compute a BPM as high as 187.5. We started at this value, as the vast majority of popular music has a tempo between 60 and 180 BPM. Therefore, we set the minimum distance such that it could compute a BPM across this range. The naive algorithm was able to output BPM values within the desired range, (60, 180) but the results lacked precision. Through numerous tests (as displayed in Table IV and Table V), the minimum distance value was incremented, until the algorithm was fully optimized.

[7]These values were calculated by estimating how many seconds of audio corresponds to five windows of five beats in music ranging from 60 to 170 beats per minute. The exact number of seconds for 5 beats depends on the BPM of the song. Though this may seem like an overly strict threshold, other algorithms such as Cheng, et al. use as little as 2.2 seconds from a piece of music to calculate the BPM.

TABLE IV
$find\_peaks$ MINIMUM DISTANCE VS. ALGORITHMIC ACCURACY

| **Minimum Distance** | ***Algorithmic Accuracy*** |
|---|---|
| 0.32 seconds | 53/100 |
| 0.43 seconds | 71/100 |
| 0.53 seconds | 73/100 |
| 0.64 seconds | 74/100 |
| 0.75 seconds | 83/100 |
| 0.85 seconds | 87/100 |
| 0.87 seconds | 86/100 |
| 0.89 seconds | 88/100 |
| 0.92 seconds | 87/100 |
| 0.94 seconds | 88/100 |
| 0.96 seconds | 85/100 |
| 1.1 seconds | 82/100 |

*From the Spotify 2016 Top 100 dataset.

TABLE V
$find\_peaks$ MINIMUM DISTANCE VS. ALGORITHMIC ACCURACY

| **Minimum Distance** | ***Algorithmic Accuracy*** |
|---|---|
| 0.32 seconds | 52/100 |
| 0.43 seconds | 75/100 |
| 0.53 seconds | 75/100 |
| 0.64 seconds | 85/100 |
| 0.75 seconds | 88/100 |
| 0.85 seconds | 90/100 |
| 0.87 seconds | 89/100 |
| 0.89 seconds | 90/100 |
| 0.92 seconds | 91/100 |
| 0.94 seconds | 93/100 |
| 0.96 seconds | 92/100 |
| 1.1 seconds | 88/100 |

*From the Spotify 2018 Top 100 dataset.

The intuition behind the results of Table IV and Table V is that a larger value for the minimum distance acts as a noise-blocker, especially in songs with a slower tempo. Increasing the minimum distance parameter decreases the total number of data points the *find_peaks* function labels as a peak. However, the peaks that are removed have lower power levels and therefore are more likely to be noise. As a result, a higher minimum distance threshold makes it easier to locate beats. In simple terms, the algorithm is vulnerable to being overwhelmed by noisy peaks and, as a result, it has optimal performance for larger distance intervals. This approach, however, has diminishing returns, as shown by Tables IV and V.

(b) We used the mean power level of the piece of music as the minimum height value of the $find\_peaks$ function and did not test any additional values. We selected this value because a beat should intuitively have higher energy than the mean energy of a song.

*2) Training the Peak Interpretation Process:* After building the plot of peaks from the $find\_peaks$ function, we trained three aspects of our approach to peak analysis to achieve optimal results.

(a) *Moving Windows*. We constructed two types of moving windows: First, windows with a window size of 5 and a window jump of 5 peaks, resulting in no overlap between neighboring windows. Second, windows with a window size of 5 and a window jump of 1 peak, resulting in an intersection of 4 peaks between neighboring windows. As seen in Table VI, overlapping windows lead to a higher accuracy rate.

TABLE VI
WINDOW TYPE VS. ALGORITHMIC ACCURACY

| **Dataset** | **Window Type** | ***Accuracy*** |
|---|---|---|
| Spotify 2016 | Overlapping | 88/100 |
| Spotify 2016 | Non-Overlapping | 85/100 |
| Spotify 2018 | Overlapping | 93/100 |
| Spotify 2018 | Non-Overlapping | 87/100 |

*Peak distance held constant at 0.94 seconds.

We hypothesize that non-overlapping windows achieve lower accuracy because placing the data into non-overlapping groups of peaks risks splitting up regions of the song with regularly spaced peaks. This could lead to the loss of data that reflects the true BPM.

(b) *Noise Filtration*. Using the corresponding standard deviation value stored in $master\_std\_devs$, we devised an approach to filtering the $master\_differences$ array such that only our $best\_guesses$ were left. Initially, we attempted to filter by percentile, keeping only the distance values associated with the lowest 5%, 2%, and 1% of standard deviation values present in the master array. This approach proved to be futile, however, as even across just 1% of data, a considerable quantity of noisy peaks were still present. To enhance the filtration process, we altered our approach and filtered by rank, keeping only 3, 4, and 5 distance values (based on lowest standard deviation). Ultimately, using this approach, we find that keeping the 5 best distance values based on standard deviation leads to optimal results.

(c) *Outlier Classification*. The final step in the algorithmic process creates the $best\_beats$ array from the $best\_guesses$ array by removing all outliers. This necessitates selecting a threshold of standard deviations for classifying data in the $best\_beats$ array as an outlier. We first tried using a value of 2 standard deviations before reducing the threshold to 1 standard deviation for optimal results.

*3) Summary of Parameter Training:* Throughout the training period, the following values were tested for each parameter:

- *Minimum Distance Between Peaks*: 0.32, 0.43, 0.53, 0.64, 0.75, 0.85, 0.87, 0.89, 0.92, 0.938, 0.96, 1.1 seconds
- *Minimum Peak Height*: Average of the power level of the song
- *Window Type*: Overlapping and Non-Overlapping
- *Standard Deviation Threshold*: By percentile (Bottom 1, 2, and 5 percentile windows in terms of standard deviation) and by numerical cutoff (Bottom 3, 4, and 5 windows in terms of standard deviation)
- *Outlier Classification*: 2 standard deviations from mean, 1 standard deviation from mean

After training, we determined that the following parameters lead to optimal accuracy:

- *Minimum Distance Between Peaks*: 0.938 seconds
- *Minimum Peak Height*: Average of the power level of the song
- *Window Type*: Overlapping
- *Standard Deviation Threshold*: Bottom 5 windows in terms of standard deviation
- *Outlier Classification*: 1 standard deviation from mean

## III. Results

For the purposes of this paper, the algorithm's output tempo is considered correct if it is exactly one half (½) or one fourth (¼) of the piece's accepted tempo. This is because, for music in common time (4/4), a tempo of any factor of two of the accepted tempo is equivalent. The accepted tempo was retrieved from TuneBat, an online music database which lists the BPM of more than 70 million songs [?], and compared to the BPM determined by the algorithm.

To account for the equivalence of tempos across factors of 2, when assessing the accuracy of the algorithm, we multiply the algorithm's calculated BPM values by a factor of 1, 2 or 4, depending on if the algorithm calculates the exact BPM of the inputted song, half (½) of the BPM, or one fourth (¼) of the BPM, respectively. We refer to this new value as "Adjusted BPM." The BPM adjustment process is displayed in Table VII.

TABLE VII
BPM Adjusted Results

| Title | *Calc. BPM* | *Adjusted BPM* | *Actual BPM* |
|---|---|---|---|
| Mood | 45.47291835 | 90.9458367 | 91 |
| Blinding Lights | 42.74316109 | 170.9726444 | 171 |
| The Good Ones | 45 | 90 | 90 |
| Every Chance I Get | 37.5 | 150 | 150 |
| Forever After All | 38.00675676 | 152.027027 | 152 |

*From the 2021 BILLBOARD HOT 100 dataset.

For each song in our training and testing datasets, the adjusted BPM value is classified as correct if the percent error between the calculated value and the accepted value is less than 0.05. The aggregated results from our training and testing datasets, after applying the BPM adjustment process referenced above and enforcing the percent error bound, are included in Table VIII.

TABLE VIII
Algorithm Results

| Dataset | *Type* | *Correct* | *Percent Error* |
|---|---|---|---|
| SPOTIFY Top 100 2016 | Training | 88/100 | 4.15% |
| SPOTIFY Top 100 2018 | Training | 93/100 | 2.06% |
| BILLBOARD Hot 100 2021 | Testing | 86/100 | 3.79% |
| BILLBOARD Hot 100 2022 | Testing | 86/100 | 3.82% |
| SPOTIFY '00s Rock Anthems | Testing | 81/100 | 5.22% |

The success of our algorithm persists across all three testing datasets and is able to retain accuracy levels similar to those achieved for the training datasets. The algorithm's success rate, across training and testing data, was 86.8 percent, with an average percent error of 3.79 percent.

We then tested the algorithm's success by genre. To classify the genre of songs within our testing datasets, we used the Spotify API which, when given the ID of a song, produces an ID for the corresponding artist along with a list of their common genres [?]. From this list, we grouped songs into six distinct categories: Pop, Hip Hop, R&B, Country, Rock, and Other (which includes Latin music and film music, among other miscellaneous categories). For our purposes, artists with music that is labeled as a subset of one of our categories is classified under the broader category; for example, music corresponding to dance pop is included in the Pop category. The algorithm's accuracy across the six genre categories is displayed in Table IX.

TABLE IX
Algorithm Testing Accuracy by Genre

| Genre | *Correct* | *Incorrect* | *Success Rate* |
|---|---|---|---|
| Pop | 49 | 12 | 80.3% |
| Hip Hop | 56 | 4 | 90.3% |
| R&B | 12 | 4 | 75.0% |
| Country | 36 | 3 | 92.3% |
| Rock | 86 | 19 | 81.2% |
| Other | 14 | 5 | 73.7% |
| **Overall** | **253** | **47** | **84.3%** |

*Data compiled from the testing datasets.

Next, we tested the ability of the algorithm to identify the BPM across distinct tempo ranges of music. We constructed 4 tempo ranges which spanned the range of BPM values present across our datasets. Table X tracks the accuracy of the algorithm for both the 2021 Billboard Hot 100 and 2022 Billboard Hot 100 testing sets across these distinct tempo ranges, showing how the algorithm retains accuracy irrespective of the tempo of the music.

TABLE X
Algorithm Testing Accuracy by Tempo Range

| Tempo Range | *Success Rate* |
|---|---|
| 0 to 100 BPM | 83.1% |
| 100 to 120 BPM | 87.1% |
| 120 to 140 BPM | 86.7% |
| 140 BPM and Higher | 90.6% |

The algorithm's testing results also demonstrate consistency across tempo ranges constructed relative to the value of the minimum distance parameter. As explained in Section F of the Methodology, the optimal distance of the $find\_peaks$ algorithm is equivalent to a time interval of 0.938 seconds. This places an upper bound on the maximum tempo the algorithm can compute (63.96 BPM). Table XI explores the effect this constraint has on the algorithm's ability to accurately measure the BPM, especially for pieces of music with a faster tempo. Each song was categorized into Range A or Range B based on its accepted BPM using the following bounds:

Range A: 0 to 127.93 BPM

Range B: 127.93 BPM

The value 127.93 was calculated by multiplying the maximum BPM the algorithm can compute, 63.96, by a factor of 2. This segments our songs into two groups: Range A for which the algorithm could theoretically detect either one-half of the tempo or one-quarter of the tempo, and Range B for which the algorithm can only detect one-quarter of the tempo.

TABLE XI
ALGORITHM TESTING ACCURACY BY TEMPO RANGE

| Dataset | *Range A* | *Range B* |
| --- | --- | --- |
| BILLBOARD Hot 100 2021 | 86.4% | 87.8% |
| BILLBOARD Hot 100 2022 | 86.9% | 84.6% |

Finally, we tested our hypothesis that the algorithm is more effective at detecting the tempo of songs with a "clappable" beat. To determine if this was the case, we tested whether the algorithm performs better with songs that receive a higher ranking from TuneBat's "Danceability" score.[8] According to TuneBat, the "Danceability" score measures on a scale from 0 to 100 "how appropriate [a] track is for dancing based on overall regularity, beat strength, rhythm stability, and tempo" [?]. We view "Danceability" as a proxy for the difficulty of perceiving, identifying, and computing the beat of a piece of music, similar to the "power-difference measure" proposed by Goto [?]. Table XII compares the average "Danceability" score (referred to as D) of the songs that the algorithm correctly calculates to the average "Danceability" of the songs that the algorithm misses.

TABLE XII
ALGORITHM TESTING ACCURACY BY DANCEABILITY

| Dataset | *Avg. D. of Correct* | *Avg. D. of Incorrect* |
| --- | --- | --- |
| BILLBOARD 2021 | 67.8 | 62 |
| BILLBOARD 2022 | 69.0 | 59.4 |

*From the BILLBOARD HOT 100 datasets.

Table XII suggests that, on average, the algorithm performs better with songs with a higher "Danceability" score. The songs that the algorithm correctly detected were on average 12.7 percent more "Danceable", according to TuneBat's index. These results support our hypothesis that music with a clearer beat is easier to track algorithmically.

Finally, we tested the statistical significance of the findings presented in Tables IX, X, and XII.

We sought to determine whether the variations in the algorithm's performance across genres were statistically significant. We conducted z-tests comparing the algorithm's performance within genre groups to the algorithm's performance across all 500 songs from the training and testing datasets. We found no statistically significant difference in the algorithm's performance across genres, except for Hip Hop: the difference between the algorithm's performance for Hip Hop (2.24 percent error on average) and the algorithm's performance across all genres (3.79 percent error on average) was statistically significant, with a p-value of 0.0224.

Similarly, to determine if the algorithm's performance across tempo ranges was statistically consistent, we performed distinct z-tests for each tempo range listed in Table X. Each test compared the algorithm's percent error in a specific tempo range to the algorithm's overall percent error across all tempo ranges. The variations in the algorithm's percent error in every tested range was found to be statistically insignificant, with the z-tests generating p-values of 0.1989, 0.4940, 0.2048, and 0.3200, respective to the ordering in Table X.

Lastly, to evaluate the effect "clappability" had on algorithmic accuracy, we conducted a z-test on the data included in Table XII. The test compared the algorithm's performance for the top quintile of songs (based on their TuneBat Danceability scores) to the algorithm's performance across all songs. The difference between the algorithm's performance for highly Danceable songs[9] compared to its overall performance was statistically significant with a p-value of 0.0321. The algorithm achieved an average percent error of 1.92 percent and a success rate of 92.8 percent for highly Danceable songs.

To further explore the effect the Danceability of music has on the algorithm's performance, we conducted a two-sample t-test comparing the algorithm's effectiveness for the top quintile of songs by Danceability to the bottom quintile of songs by Danceability. The algorithm's average percent error for the bottom quintile of Danceable songs was 8.4 percent, with a success rate of 72.1 percent. The t-test revealed that the difference in the algorithm's performance for the top quintile in comparison to the bottom quintile was statistically significant with a p-value of 0.006.

# IV. DISCUSSION

## A. Overview

The algorithm exhibited consistent results across training and testing data. In the Billboard Hot 100 2021, Billboard Hot 100 2022, and Spotify '00s Rock Anthems playlist, the algorithm achieved a success rate of 86 percent, 86 percent, and 81 percent, respectively. Furthermore, the results from Tables II and III suggest that the noise-filtration section of the algorithm is effective at differentiating between peaks in an audio signal that represent noise and peaks that represent a beat, as the algorithm was able to reduce percent error by 88 percent compared to a naive approach that considered all peaks in the audio signal with equal weight when calculating the tempo of a piece of music. These results are extremely encouraging and suggest that the algorithm is broadly effective at identifying beats in a piece of music.

The algorithm was also consistent across music with different genres and tempos.

[8] For the purposes of this paper, "Danceability" and "clappability" are synonymous and will be referred to interchangeably.

[9] We consider a song to be highly Danceable if its TuneBat Danceability score is in the top quintile of songs. For reference, this corresponds to a TuneBat Danceability score of 77 or higher.

Contrary to our hypothesis the algorithm is most effective for Pop music, the algorithm saw the highest success rates in the genres of Hip Hop (90.3 percent) and Country (92.3 percent). The high success in the Hip Hop category was statistically significant and may be because of the presence of trap beats with clear snare sounds in Hip Hop songs. On the other hand, the algorithm's success for detecting the tempo of Country music was not statistically significant.

Similarly, the algorithm's results did not support our hypothesis that the algorithm is more effective for Pop music and less effective for Rock music. We found no statistically significant difference in the algorithm's performance in either genre. The algorithm unexpectedly saw similar success rates for Pop (80.3 percent) and Rock (81.2 percent) music, contrary to our hypothesis that the increased levels of noise in Rock music would result in a lower success rate. While the algorithm's accuracy rate in detecting the beat for Pop and Rock music was below the overall average, we found that this difference was not statistically significant. This provides evidence for the algorithm's consistency across genres.

The algorithm performed consistently across the four tempo ranges, with a slight upward trend in accuracy as the tempo increased. However, the variations in the algorithm's performance across tempo ranges were not statistically significant. These results do not match our initial hypothesis that the algorithm encounters greater difficulties measuring the BPM of music with a slower tempo.

Furthermore, the algorithm's ability to detect the beat of music across different tempo ranges was not constrained by the parameter value chosen for the minimum distance between peaks. This indicates that the algorithm's design is equally effective at calculating one-half of the tempo and one-quarter of the tempo.

Lastly, our investigation of the algorithm's performance across the TuneBat "Danceability" index provides support for our hypothesis that the algorithm is more effective for songs with a clearer beat and stronger sense of rhythm. Analyzing the success of the algorithm in relation to the Danceability score reflects how the algorithm mimics the way in which a human listener would find the beat of a piece of music. The same way a human listener may struggle with less "clappable" songs, the algorithm is similarly constrained by low Danceability. This finding matches the results of Scheirer, whose algorithm achieved performance "similar to the performance of human listeners in a variety of musical situations." [**?**].

The success of the algorithm across genre, tempo, and "clappability" suggests that the algorithm's parameters and design were not overfit to the songs in the training sets, which mostly consisted of popular music. Instead, the algorithm's results suggest that the algorithm's design is effective for music across a variety of cross-sections.

### *B. Limitations*

The success of our FFT beat-detection algorithm is partially dependent on three key factors, with an ideal music track having the following characteristics:

1) The music has high audibility of the beat (often characterized by a large presence of drums).
2) The music has no tempo changes.
3) The music is in a constant time signature.

There were a few pieces of music within the testing datasets which were suboptimal in these categories, exemplifying key shortcomings in our algorithm. While these limitations do not apply to the majority of modern popular music, it is still a notable limitation of our approach and the design of our algorithm.

First, in some situations, our algorithm's performance suffers if the music does not have consistent drums present in the track. For example, throughout our training tests, our algorithm was regularly able to detect the BPM of music by dance pop artists such as Dua Lipa, whose music is characterized by a clear, "clappable" beat, but it outputted an incorrect tempo for Lonely by Justin Bieber, possibly due to the lack of drums and "clappability" in the track. This shortcoming, however, was not universal across all tracks lacking drums, as our algorithm in many cases was still able to detect the beat.[10]

Second, the algorithm fails to account for tempo changes in music. Notably, the algorithm correctly identified the tempo of Travis Scott's "SICKO MODE" as 154.96 beats per minute despite a tempo change in the song. This result matches the tempo of the song's first and second verses (which have a BPM of 155), but does not match the song's introduction, which has a tempo of roughly 139 BPM. The presence of a clear electronic drumbeat in the first and second verses likely caused the algorithm to pick up the tempo of those verses with higher confidence than the tempo of the introduction section, leading to the output of 155 BPM.

While the algorithm correctly identified one of the tempos of Scott's "SICKO MODE," the algorithm was unable to identify either of the tempos of "Welcome to the Black Parade" by My Chemical Romance, which has a tempo of 75 BPM in its introduction section and 97 BPM in the rest of the piece. The algorithm, however, outputted 125 BPM, which does not match either tempo present in the piece.

These case studies demonstrate the variable performance of the algorithm on pieces with changes in tempo.

Third, the algorithm is unable to account for changes in the time signature. For Billie Eilish's "Happier Than Ever," which includes both a tempo change and a time signature change, the algorithm's output did not match the correct tempo for the song. Notably, the algorithm calculated a BPM that differed from both of the correct BPM values, indicating that the error may have been caused by the music's variable time signature and not the change in tempo. In modern popular music, however, the vast majority of music is made in a constant time signature of four beats per measure.

[10]For an example, see "changes" by XXXTentacion, a piece of music that is relatively "unclappable" due to its lack of drums and slow tempo. However, the algorithm correctly detected the tempo of the song at 60 BPM. This can be explained by the fact that the piano part of "changes" falls exactly on the downbeat, with little overall noise in the track.

Finally, we acknowledge that many of the "ideal" characteristics needed for the algorithm to accurately calculate the tempo of a song are characteristics most commonly found in Western music. For example, the algorithm is unlikely to perform well on complex polyrhythms found in South African music [**?**]. This bias towards Western music and rhythmic structures reflects a lack of inclusivity in our algorithm and beat detection algorithms as a whole.

These limitations reflect an opportunity for improvement in our algorithm. These improvements may be made with the use of advanced machine learning techniques which could, for example, identify a tempo change in music and accordingly calculate multiple BPM values for a given track of music with a certainty score attached to each guess for the BPM value. Adopting these changes would require further research, which may build upon our algorithm to develop an even more accurate approach to tempo measurement. For examples of beat-detection algorithms employing machine learning, see Souza, et al. [**?**] and Böck and Schedl [**?**].

Our algorithm can also be improved through greater exploration of the window size, window jump, and minimum height parameters. The window size was held constant at 5 peaks, the window jump was only tested with values of 1 and 5, and the minimum height was held constant at the mean power level of the piece of music. Further research could experiment with different values for these parameters, possibly leading to a refined version of this algorithm with higher accuracy.

## V. Conclusion

Computational musicology is a field of growing importance with new applications in streaming, music curation, and transcription. In this paper, we contribute to the discussion by developing an algorithm using the Fast Fourier Transform and a noise-filtration approach to find the tempo of a piece of music. Our findings indicate that even a simple approach (relative to other published beat-detection algorithms) can achieve high accuracy rates when accompanied by a rigorous noise reduction algorithm.

## References


[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, "Title of paper if known," unpublished.

[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.


## Appendix

### A. Pseudocode

**Algorithm 1** Construction of Frequency Ranges

**Input:** Array of audio data $\{audio\_input\}$ in bytes and array of indices $\{[sub\_bass\_indices], [bass\_indices], etc.\}$ corresponding to each frequency range in Table I calculated using Equation (6)

**Output:** Array of power levels of frequency ranges over time

$chunk \leftarrow 1024$

$data \leftarrow audio\_input.readframes(chunk)$ $\{readframes$ is a command from the Wave module that reads $chunk$ bytes from an audio file at a time. The while loop ends once the algorithm has read through the full audio file.$\}$

**while** $len(data) > 0$ **do**

  $chunk\_fft \leftarrow fft(data)$ $\{chunk\_fft$ is an array of complex numbers calculated for each chunk of data$\}$

  $fft\_magnitudes \leftarrow chunk\_fft \cdot conj(chunk\_fft)$

  $sub\_bass \leftarrow mean(fft\_magnitudes[sub\_bass\_indices])$

  $bass \leftarrow mean(fft\_magnitudes[bass\_indices])$

  $low\_mid \leftarrow mean(fft\_magnitudes[low\_mid\_indices])$

  $mid \leftarrow mean(fft\_magnitudes[mid\_indices])$

  $high\_mid \leftarrow mean(fft\_magnitudes[high\_mid\_indices])$

  $pres \leftarrow mean(fft\_magnitudes[pres\_indices])$

  $bril \leftarrow mean(fft\_magnitudes[bril\_indices])$

  $data \leftarrow input.readframes(chunk)$

  $sub\_bass\_array.append(sub\_bass)$

  $bass\_array.append(bass)$

  ... {Value for each frequency range is appended onto a separate array.}

  $brilliance\_array.append(bril)$

**end while**

**return** $sub\_bass\_array$, $bass\_array$, $low\_mid\_array$, $mid\_array$, $high\_mid\_array$, $pres\_array$, $brilliance\_array$

**Algorithm 2** Detection of Peaks with Moving Window

**Input:** $sub\_bass\_array$, $bass\_array$, $low\_mid\_array$, $mid\_array$, $high\_mid\_array$, $pres\_array$, $brilliance\_array$

**Output:** Master_differences (an array of the average distances between peaks) and Master_std_devs (an array of the std. deviation of the distance between peaks within each window)

$master\_differences \leftarrow []$
$master\_std\_devs \leftarrow []$
{All of the analysis below this comment is performed on every frequency range, but only one frequency range is included in the pseudocode for brevity.}
$sub\_bass\_envelope \leftarrow hilbert(sub\_bass\_array)$
$sub\_bass\_peaks \leftarrow find\_peaks(sub\_bass\_envelope)$
{Both $hilbert$ (a function for finding the envelope of a signal) and $find\_peaks$ are available in the SciPy module, as stated before. $find\_peaks$ returns the indexes of the inputted array that correspond to peaks.}
**for** $i = 0$, $i$++, while $i < length(sub\_bass\_peaks)$ **do**
  $window \leftarrow sub\_bass\_peaks[i : i + 6]$
  $diff\_array \leftarrow [window[4] - window[3], window[3] - window[2], window[2] - window[1], window[1] - window[0]]$
  $avg\_diff \leftarrow mean(diff\_array)$
  $std\_dev \leftarrow std(diff\_array)$
  $master\_differences.append(avg\_diff)$
  $master\_std\_devs.append(std\_dev)$
**end for**
{Notice that $master\_differences$ and $master\_std\_devs$ are indexed such that element $i$ of $master\_differences$ is the average distance between the peaks of window $i$, and element $i$ of $master\_std\_devs$ is the standard deviation of the distance between the peaks of window $i$}
**return** $master\_differences$, $master\_std\_devs$

**Algorithm 3** Selection of Best Guesses for Peaks

**Input:** $master\_differences$, $master\_std\_devs$

**Output:** Tempo of the $audio\_input$ in beats per minute

$best\_indexes \leftarrow$ [$i$ such that $master\_std\_devs[i]$ is in the 5 lowest standard deviations]
$best\_guesses \leftarrow master\_differences[best\_indexes]$
$best\_guesses \leftarrow remove\_outliers(best\_guesses)$
**return** mean($best\_guesses$)

### B. Full Algorithm Results

TABLE XIII: Billboard Hot 100 2021 Results

| Title | *Calc. BPM* | *Adj. BPM* | *Actual BPM* |
|---|---|---|---|
| Levitating | 102.6 | 102.6 | 103.0 |
| Save Your Tears Remix | 117.7 | 117.7 | 118.0 |
| Blinding Lights | 85.5 | 171.0 | 171.0 |
| Mood | 90.9 | 90.9 | 91.0 |
| Good 4 U | 84.2 | 168.4 | 167.0 |
| Kiss Me More | 74.0 | 74.0 | 111.0 |
| Leave The Door Open | 74.0 | 148.0 | 148.0 |
| Drivers License | 95.7 | 95.7 | 144.0 |
| Montero | 89.6 | 89.6 | 89.0 |
| Peaches | 89.9 | 89.9 | 90.0 |
| Butter | 110.3 | 110.3 | 110.0 |
| Stay | 85.0 | 169.9 | 170.0 |
| Deja Vu | 117.9 | 117.9 | 91.0 |
| Positions | 72.3 | 144.6 | 144.0 |
| Bad Habits | 125.6 | 125.6 | 126.0 |
| Heat Waves | 80.6 | 80.6 | 81.0 |
| Without You | 92.2 | 92.2 | 93.0 |
| Forever After All | 76.0 | 152.0 | 152.0 |
| Go Crazy | 93.8 | 93.8 | 94.0 |
| Astronaut In The Ocean | 75.0 | 150 | 150.0 |
| 34+35 | 110.3 | 110.3 | 110.0 |
| What You Know Bout Love | 84.0 | 84.0 | 84.0 |
| My Ex's Best Friend | 62.5 | 125.0 | 125.0 |
| Industry Baby | 74.9 | 149.7 | 150.0 |
| Therefore I Am | 93.8 | 93.8 | 94.0 |
| Up | 83.0 | 165.9 | 166.0 |
| Fancy Like | 79.9 | 79.9 | 80.0 |
| Dakiti | 110.0 | 110.0 | 110.0 |
| Best Friend | 93.8 | 93.8 | 145.0 |
| Rapstar | 108.2 | 108.2 | 81.0 |
| Heartbreak Anniversary | 65.0 | 65.0 | 89.0 |
| For The Night | 63.1 | 126.1 | 126.0 |
| Calling My Phone | 52.6 | 105.1 | 105.0 |
| Beautiful Mistakes | 99.0 | 99.0 | 99.0 |
| Holy | 86.8 | 86.8 | 87.0 |
| On Me | 78.1 | 78.1 | 78.0 |
| You Broke Me First | 124.4 | 124.4 | 124.0 |
| Traitor | 100.1 | 100.1 | 101.0 |
| Back In Blood | 73.4 | 146.9 | 147.0 |
| I Hope | 75.0 | 75 | 76.0 |
| Dynamite | 76.0 | 76.0 | 114.0 |
| Wockesha | 85.0 | 85.0 | 82.0 |
| You Right | 64.5 | 129.0 | 129.0 |
| Beat Box | 80.1 | 160.3 | 160.0 |
| Laugh Now Cry Later | 67.0 | 133.9 | 134.0 |
| Need To Know | 65.0 | 129.9 | 130.0 |
| Wants And Needs | 67.9 | 135.9 | 136.0 |
| Way 2 Sexy | 67.9 | 135.9 | 136.0 |
| Telepatia | 84.0 | 84.0 | 84.0 |
| Whoopty | 119.3 | 119.3 | 140.0 |
| Lemonade | 70.0 | 139.9 | 140.0 |

| | | | |
|---|---|---|---|
| Good Days | 121.8 | 121.8 | 121.0 |
| Starting Over | 89.0 | 89.0 | 89.0 |
| Body | 93.8 | 93.8 | 94.0 |
| Willow | 84.0 | 84.0 | 81.0 |
| Bang! | 70.1 | 140.3 | 140.0 |
| Better Together | 108.6 | 108.6 | 138.0 |
| You're Mines Still | 89.0 | 89.0 | 87.0 |
| Every Chance I Get | 75.0 | 150 | 150.0 |
| Essence | 106.3 | 106.3 | 104.0 |
| Chasing After You | 66.0 | 132.0 | 132.0 |
| The Good Ones | 90.0 | 90 | 90.0 |
| Leave Before You Love Me | 119.7 | 119.7 | 120.0 |
| Glad You Exist | 104.2 | 104.2 | 104.0 |
| Lonely | 120.1 | 60.0 | 79.0 |
| Beggin' | 67.0 | 133.9 | 134.0 |
| Streets | 45.0 | 90.0 | 90.0 |
| What's Next | 65.0 | 130.0 | 130.0 |
| Famous Friends | 102.0 | 102.0 | 102.0 |
| Lil Bit | 119.7 | 119.7 | 120.0 |
| Thot S**t | 65.0 | 129.9 | 130.0 |
| Late At Night | 98.7 | 98.7 | 99.0 |
| Kings & Queens | 114.2 | 114.2 | 130.0 |
| Anyone | 116.2 | 116.2 | 116.0 |
| Track Star | 113.3 | 113.3 | 130.0 |
| Time Today | 68.6 | 137.2 | 137.0 |
| Cry Baby | 65.0 | 129.9 | 130.0 |
| All I Want For Christmas Is You | 75.0 | 150 | 150.0 |
| No More Parties | 78.3 | 156.7 | 157.0 |
| What's Your Country Song | 86.9 | 173.9 | 174.0 |
| One Too Many | 83.0 | 83.0 | 83.0 |
| Arcade | 72.1 | 72.1 | 72.0 |
| Yonaguni | 90.1 | 90.1 | 90.0 |
| Good Time | 68.8 | 137.6 | 138.0 |
| If I Didn't Love You | 92.2 | 92.2 | 92.0 |
| Knife Talk | 73.1 | 146.1 | 146.0 |
| POV | 66.0 | 132.0 | 132.0 |
| Just The Way | 91.6 | 91.6 | 90.0 |
| Take My Breath | 121.2 | 121.2 | 121.0 |
| We're Good | 67.0 | 133.9 | 134.0 |
| Hell Of A View | 99.0 | 99.0 | 99.0 |
| Rockin' Around The Christmas Tree | 70.7 | 70.7 | 67.0 |
| Put Your Records On | 91.2 | 91.2 | 91.0 |
| Happier Than Ever | 107.1 | 107.1 | 81.0 |
| Single Saturday Night | 82.0 | 82.0 | 82.0 |
| Things A Man Oughta Know | 70.0 | 139.9 | 140.0 |
| Throat Baby | 66.0 | 132.0 | 132.0 |
| Tombstone | 85.0 | 85.0 | 85.0 |
| Drinkin' Beer. Talkin' God. Amen. | 100.1 | 100.1 | 100.0 |
| Todo De Ti | 124.6 | 124.6 | 128.0 |

TABLE XIV: Billboard Hot 100 2022 Results

| **Title** | ***Calc. BPM*** | ***Adj. BPM*** | ***Actual BPM*** |
|---|---|---|---|
| Heat Waves | 80.6 | 80.6 | 81.0 |
| Stay | 85.0 | 169.9 | 170.0 |

| | | | |
|---|---|---|---|
| Super Gremlin | 73.1 | 73.1 | 73.0 |
| Abcdefu | 122.3 | 122.3 | 122.0 |
| Ghost | 77.1 | 154.1 | 154.0 |
| We Don't Talk About Bruno | 102.6 | 102.6 | 103.0 |
| Enemy | 77.1 | 77.1 | 77.0 |
| That's What I Want | 118.0 | 118.0 | 88.0 |
| Woman | 108.2 | 108.2 | 108.0 |
| Easy On Me | 114.6 | 114.6 | 142.0 |
| Big Energy | 106.1 | 106.1 | 106.0 |
| Bad Habits | 125.6 | 125.6 | 126.0 |
| Shivers | 70.5 | 141.0 | 141.0 |
| Cold Heart PNAU Remix | 115.7 | 115.7 | 116.0 |
| Need To Know | 65.0 | 129.9 | 130.0 |
| Levitating | 102.6 | 102.6 | 103.0 |
| Save Your Tears Remix | 117.7 | 117.7 | 118.0 |
| Til You Can't | 80.1 | 160.3 | 160.0 |
| Pushin' P | 77.5 | 77.5 | 78.0 |
| One Right Now | 97.0 | 97.0 | 97.0 |
| Industry Baby | 74.9 | 149.7 | 150.0 |
| What Happened To Virgil | 70.1 | 140.3 | 140.0 |
| I Hate U | 80.1 | 80.1 | 107.0 |
| Hrs And Hrs | 70.1 | 140.3 | 140.0 |
| Sweetest Pie | 124.4 | 124.4 | 124.0 |
| Mamiii | 93.7 | 93.7 | 94.0 |
| Good 4 U | 84.2 | 168.4 | 167.0 |
| Ahhh Ha | 78.1 | 156.2 | 156.0 |
| Light Switch | 92.1 | 92.1 | 92.0 |
| Doin' This | 114.8 | 114.8 | 115.0 |
| You Right | 64.5 | 129.0 | 129.0 |
| Fingers Crossed | 110.3 | 110.3 | 109.0 |
| Buy Dirt | 89.0 | 89.0 | 89.0 |
| Bam Bam | 94.9 | 94.9 | 95.0 |
| Surface Pressure | 90.0 | 90 | 90.0 |
| Fancy Like | 79.9 | 79.9 | 80.0 |
| Blick Blick | 70.0 | 139.9 | 140.0 |
| Sand In My Boots | 70.1 | 70.1 | 70.0 |
| Love Nwantiti | 93.1 | 93.1 | 93.0 |
| Never Say Never | 70.1 | 140.3 | 140.0 |
| Aa | 104.2 | 104.2 | 104.0 |
| Boyfriend | 119.7 | 119.7 | 90.0 |
| Thinkin' With My D**k | 81.2 | 81.2 | 81.0 |
| Drunk | 119.7 | 119.7 | 120.0 |
| Beers On Me | 73.1 | 146.1 | 146.0 |
| Knife Talk | 73.1 | 146.1 | 146.0 |
| Broadway Girls | 75.0 | 75 | 75.0 |
| Shes All I Wanna Be | 80.0 | 160.0 | 160.0 |
| Numb Little Bug | 85.1 | 85.1 | 85.0 |
| Nobody Like U | 104.6 | 104.6 | 105.0 |
| The Motto | 117.7 | 117.7 | 118.0 |
| Slow Down Summer | 78.1 | 156.2 | 156.0 |
| 23 | 98.0 | 98.0 | 98.0 |
| Peru | 30.9 | 123.6 | 108.0 |
| The Family Madrigal | 70.0 | 140.0 | 141.0 |
| Handsomer | 81.8 | 81.8 | 82.0 |
| Sometimes | 122.1 | 122.1 | 122.0 |
| To Be Loved By You | 73.1 | 146.1 | 146.0 |

| | | | |
|---|---|---|---|
| Heart On Fire | 119.7 | 119.7 | 120.0 |
| Circles Around This Town | 75.0 | 150 | 150.0 |
| Computer Murderers | 86.5 | 173.1 | 173.0 |
| Petty Too | 77.1 | 154.1 | 154.0 |
| Do We Have A Problem | 119.7 | 119.7 | 120.0 |
| To The Moon | 106.5 | 106.5 | 144.0 |
| Me Or Sum | 80.4 | 160.7 | 161.0 |
| Nail Tech | 75.0 | 150 | 150.0 |
| Flower Shops | 85.2 | 85.2 | 128.0 |
| No Interviews | 79.0 | 158.0 | 158.0 |
| Never Wanted To Be That Girl | 74.0 | 74.0 | 74.0 |
| Banking On Me | 67.6 | 135.2 | 135.0 |
| Barbarian | 109.2 | 109.2 | 98.0 |
| Beautiful Lies | 85.2 | 85.2 | 84.0 |
| Bones | 114.3 | 114.3 | 114.0 |
| By Your Side | 79.0 | 158.0 | 158.0 |
| City Of Gods | 75.0 | 150 | 147.0 |
| Closer | 74.0 | 74.0 | 95.0 |
| Comeback As A Country Boy | 76.0 | 76.0 | 76.0 |
| Dos Orugitas | 127.1 | 127.1 | 94.0 |
| Freaky Deaky | 104.2 | 104.2 | 104.0 |
| Ghost Story | 95.3 | 95.3 | 143.0 |
| Give Heaven Some Hell | 63.1 | 126.1 | 126.0 |
| Golden Child | 80.1 | 80.1 | 80.0 |
| Half Of My Hometown | 68.9 | 137.9 | 138.0 |
| High | 90.0 | 90.0 | 135.0 |
| I Love You So | 76.0 | 76.0 | 76.0 |
| I'm Tired | 123.5 | 123.5 | 132.0 |
| Idgaf | 98.0 | 98.0 | 98.0 |
| If I Was A Cowboy | 80.1 | 80.1 | 80.0 |
| Maybe | 87.9 | 87.9 | 88.0 |
| Money So Big | 69.1 | 138.2 | 138.0 |
| Over | 104.2 | 104.2 | 103.0 |
| P Power | 119.7 | 119.7 | 119.0 |
| Pressure | 72.1 | 144.2 | 144.0 |
| Rumors | 80.1 | 160.3 | 160.0 |
| She Likes It | 63.5 | 127.0 | 127.0 |
| Smokin' Out The Window | 81.8 | 81.8 | 82.0 |
| Smoking & Thinking | 80.8 | 161.6 | 162.0 |
| Tom's Diner | 98.7 | 98.7 | 98.0 |
| Waiting On A Miracle | 127.3 | 127.3 | 95.0 |
| What Else Can I Do | 119.7 | 119.7 | 120.0 |

TABLE XV: Spotify '00s Rock Anthems Results

| **Title** | ***Calc. BPM*** | ***Adj. BPM*** | ***Actual BPM*** |
|---|---|---|---|
| Ocean Avenue | 115.3 | 115.3 | 174 |
| Hysteria | 93.4 | 93.4 | 93 |
| Boulevard Of Broken Dreams | 83.7 | 167.4 | 167 |
| Can't Stop | 90.7 | 90.7 | 91 |
| What I've Done | 119.7 | 119.7 | 120 |
| Chop Suey | 125.2 | 125.2 | 125 |
| Seven Nation Army | 124.4 | 124.4 | 124 |
| Mr. Brightside | 98.7 | 98.7 | 148 |
| Kryptonite | 98.7 | 98.7 | 99 |

| | | | |
|---|---|---|---|
| Sugar, We're Going Down | 81.1 | 162.1 | 162 |
| Misery Business | 86.5 | 173.1 | 173 |
| Dani California | 95.3 | 95.3 | 96 |
| Holiday | 73.2 | 146.5 | 147 |
| Last Resort | 90.5 | 90.5 | 91 |
| In The End | 104.9 | 104.9 | 105 |
| I Miss You | 110.3 | 110.3 | 110 |
| Californication | 97.0 | 97.0 | 96 |
| You're Gonna Go Far Kid | 126.1 | 126.1 | 126 |
| Toxicity | 75.8 | 75.8 | 117 |
| Dance, Dance | 114.3 | 114.3 | 114 |
| The Middle | 108.2 | 108.2 | 162 |
| The Kill | 122.5 | 122.5 | 91 |
| I Write Sins Not Tragedies | 117.7 | 117.7 | 170 |
| Take A Look Around | 100.8 | 100.8 | 102 |
| Bring Me To Life | 95.1 | 95.1 | 95 |
| Savior | 112.5 | 112.5 | 112 |
| Numb | 109.9 | 109.9 | 110 |
| Best Of You | 111.3 | 111.3 | 130 |
| The Diary Of Jane | 83.7 | 167.4 | 167 |
| Take Me Out | 103.8 | 103.8 | 105 |
| Welcome To The Black Parade | 125.7 | 125.7 | 97 |
| I Hate Everything About You | 89.3 | 89.3 | 89 |
| Sex On Fire | 76.4 | 152.7 | 153 |
| I'm Not Okay | 89.9 | 89.9 | 90 |
| Like A Stone | 108.2 | 108.2 | 108 |
| The Pretender | 86.5 | 173.1 | 173 |
| When You Were Young | 113.9 | 113.9 | 130 |
| Face Down | 92.8 | 92.8 | 93 |
| It's Been Awhile | 117.7 | 117.7 | 117 |
| Island In The Sun | 114.8 | 114.8 | 115 |
| She Hates Me | 110.3 | 110.3 | 110 |
| Uprising | 127.1 | 127.1 | 128 |
| Yellow | 86.5 | 173.1 | 173 |
| Smooth Criminal | 127.3 | 127.3 | 127 |
| American Idiot | 93.1 | 93.1 | 93 |
| The Anthem | 88.7 | 88.7 | 89 |
| Thnks Fr Th Mmrs | 103.0 | 103.0 | 155 |
| Paralyzer | 106.1 | 106.1 | 106 |
| All My Life | 84.0 | 167.9 | 168 |
| The Reason | 82.7 | 82.7 | 83 |
| No One Knows | 125.6 | 125.6 | 171 |
| In Too Deep | 115.3 | 115.3 | 116 |
| Supermassive Black Hole | 120.2 | 120.2 | 120 |
| First Date | 95.8 | 95.8 | 96 |
| Youth Of The Nation | 98.0 | 98.0 | 98 |
| Rollin' | 95.0 | 95.0 | 96 |
| Complicated | 78.1 | 78.1 | 78 |
| Gives You Hell | 100.1 | 100.1 | 100 |
| Fake It | 66.2 | 132.4 | 132 |
| Joker And The Thief | 88.4 | 88.4 | 78 |
| Miss Murder | 119.0 | 119.0 | 144 |
| Higher | 78.0 | 155.9 | 156 |
| I'm Just A Kid | 109.9 | 109.9 | 110 |
| Beverly Hills | 87.9 | 87.9 | 88 |
| Dear Maria Count Me In | 90.5 | 90.5 | 91 |

| | | | |
|---|---|---|---|
| Use Somebody | 114.0 | 114.0 | 137 |
| Through Glass | 105.7 | 105.7 | 106 |
| Wish You Were Here | 84.8 | 169.6 | 170 |
| Last Nite | 104.2 | 104.2 | 104 |
| Steady As She Goes | 124.9 | 124.9 | 124 |
| Are You Gonna Be My Girl | 104.9 | 104.9 | 105 |
| Want You Bad | 105.7 | 105.7 | 106 |
| Welcome Home | 77.1 | 154.1 | 154 |
| Reptilia | 79.0 | 158.0 | 158 |
| Hate To Say I Told You So | 67.8 | 135.5 | 136 |
| Jerk It Out | 67.0 | 133.9 | 134 |
| I Bet You Look Good On The Dancefloor | 104.7 | 104.7 | 103 |
| Float On | 100.8 | 100.8 | 101 |
| Your Touch | 115.3 | 115.3 | 116 |
| A-Punk | 87.5 | 175.1 | 175 |
| You Know You're Right | 86.9 | 173.7 | 168 |
| Rock N Roll Train | 117.2 | 117.2 | 118 |
| Obstacle 1 | 119.8 | 119.8 | 121 |
| Banquet | 100.6 | 100.6 | 150 |
| Hey There Delilah | 107.9 | 107.9 | 104 |
| Little Sister | 81.6 | 163.2 | 162 |
| Change In The House Of Flies | 68.6 | 137.2 | 142 |
| Dig | 77.6 | 155.1 | 135 |
| Woman | 113.3 | 113.3 | 113 |
| The Bitter End | 122.5 | 122.5 | 93 |
| Judith | 110.3 | 110.3 | 82 |
| There There | 125.6 | 125.6 | 126 |
| The Taste Of Ink | 98.3 | 98.3 | 98 |
| Everyday Is Exactly The Same | 72.1 | 144.2 | 144 |
| Cute Without The E | 96.6 | 96.6 | 95 |
| Fell In Love With A Girl | 96.9 | 96.9 | 96 |
| Stop Crying Your Heart Out | 74.9 | 149.7 | 150 |
| Beautiful Day | 68.1 | 136.3 | 136 |
| How You Remind Me | 86.0 | 172.0 | 172 |