DAVIDE PISANÒ

AN ALGORITHM FOR MU-SIC RECOGNITION

Contents

1 Music and physics

A sound is a vibration that propagates through the air and can be understood by ears. MP3 players or computers use headphones or built-in speakers to produce vibrations and make "sounds". Music is just a kind of signal which can be reproduced using various sinusoidal waveforms and like that must be treated.

1.1 Signals scrambling via sines sums

A sine wave is characterized by two parameters:

- *Frequency* $\Longrightarrow$ the number of cycles per unit of time, measured in Hertz
- *Amplitude* $\Longrightarrow$ the height of the cycle

A generic sine wave is described by the function:

$$y(t) = A \cdot sin(2\pi f t + \varphi)$$

where:

- A is the amplitude
- f is the frequency
- t is the time
- φ is the phase

In the figure 1.1 the sine wave is described by the function:

$$y(t) = 1 \cdot sin(2\pi 20t + 0) = sin(40\pi t)$$

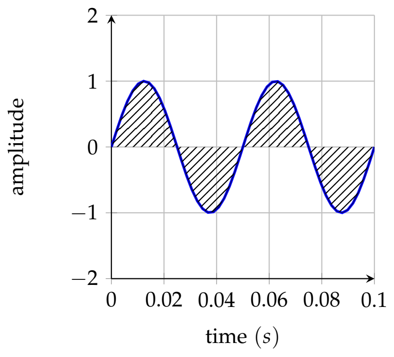

Figure 1.1: A 20Hz sine wave

But, in the real world, almost any signal is a weighted sum of sines. For example, take three different sine waves, as follows:

- Frequency 10Hz and amplitude 1 $\implies a(t) = sin(20\pi t)$
- Frequency 30Hz and amplitude 2 $\implies b(t) = 2 \cdot sin(60\pi t)$
- Frequency 60Hz and amplitude 3 $\implies c(t) = 3 \cdot sin(120\pi t)$

These three functions can be plotted separately as shown in the figure 1.2, or they can be summed up in just a single function to represent a more realistic sound (figure 1.3).

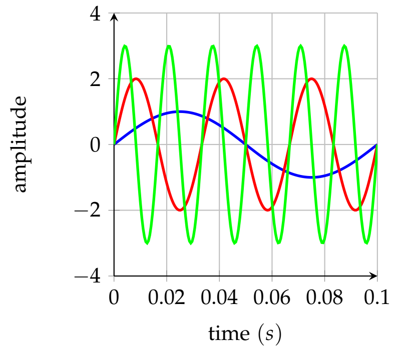

Figure 1.2: Three different sine waves

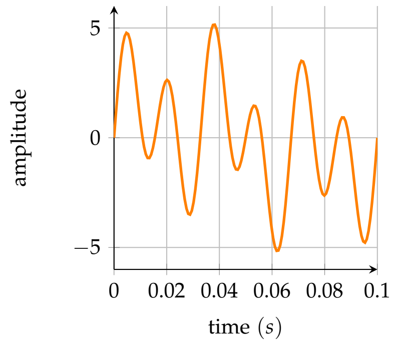

Figure 1.3: Sum of the three sine waves

1.2 *Spectrogram*

A **spectrogram** (figure 1.4) is a visual representation of how the frequencies of a signal vary with time. Usually, a spectrogram is made up of three axis (making a 3D graph):

- Time on the horizontal axis (x)
- Frequencies on the vertical axis (y)
- Amplitude of a given frequency at a given time (described by a color)

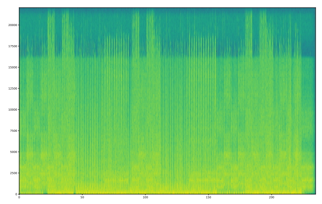

Figure 1.4: Spectrogram of a real song

1.3 *Digitalization*

Nowadays the most common way to listen to music is using a digital file[1]. However, a sound is an analog phenomenon that needs to be converted into a **digital** representation, in order to be easily recorded and stored.

[1] Such as an MP3 file or a FLAC file and so on

1.3.1 *Sampling*

Analog signals are continuous signals, which means that given two boundaries in the signals there is always a point between them. But in the digital world is not so affordable to store an infinite amount of data, so the analog signal needs to be reduced to a **discrete-time signal**. This process is called **sampling**. It is quite simple: an instantaneous value of the continuous signal is taken every **T** seconds. T is called **sampling period** and it should be short enough so that the digital song sounds like the analog one.

The **sampling rate** or **sampling frequency** f_s is the number of samples obtained in one second, given by the formula:

$$f_s = \frac{1}{T}$$

The standard sampling rate for digital music is usually 44100 Hz. The reason behind that number lies in the Nyquist-Shannon theorem which can be expressed as:

1.3.1.1 Nyquist-Shannon theorem

Suppose the highest frequency component for a given analog signal is f_{max}, the sampling rate must be at least $2f_{max}$.

There is enough theory involved in the theorem, but it states that given an analog signal, it needs at least 2 points per cycle to be correctly identified. So, since the human ears can listen to signals whose frequency is between 20 Hz and 20 kHz, taking the highest boundary and multiplying it by 2, it will give 40 kHz, which is close enough to the standardized 44.1 kHz (figure 1.5).

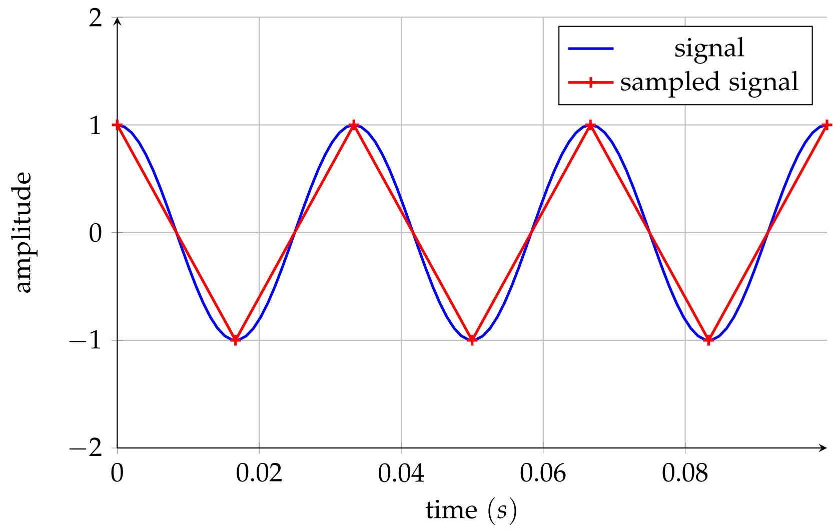

Figure 1.5: Sampling example of a 30 Hz signal

1.3.2 Quantization

With the sampling process, the signal is not fully digital: the time resolution became discrete but what about signal amplitude? The amplitude of a signal represents its loudness, and in an analog world it is continuous, so there must be a way to make it discrete. This process is called **quantization**. The quantization resolution is measured in bits, also known as **bit depth** of a song.

Taking a depth of 3 bits means that the loudness of the song can vary between 0 and $2^3 - 1$, so there are just 8 quantization levels to represent the loudness of the whole song (figure 1.6).

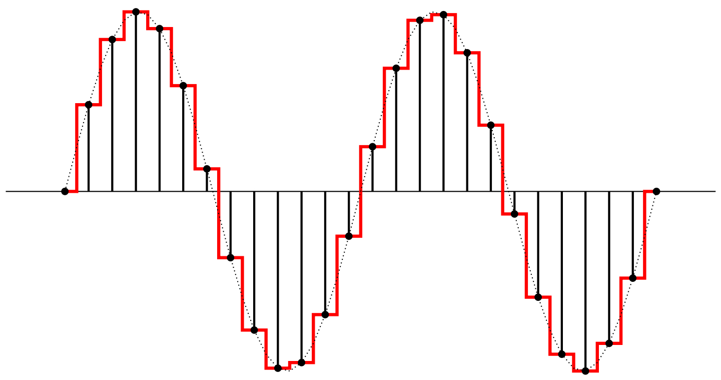
Figure 1.6: 3-bit quantization example

The higher the bit depth, the better the amplitude is approximated. The standard quantization is coded on **16 bits**. For instance, in the same previous analog signal with a 16-bit quantization, the amplitude can assume 2^{16} values, which will give a much more precise accuracy (figure 1.7).

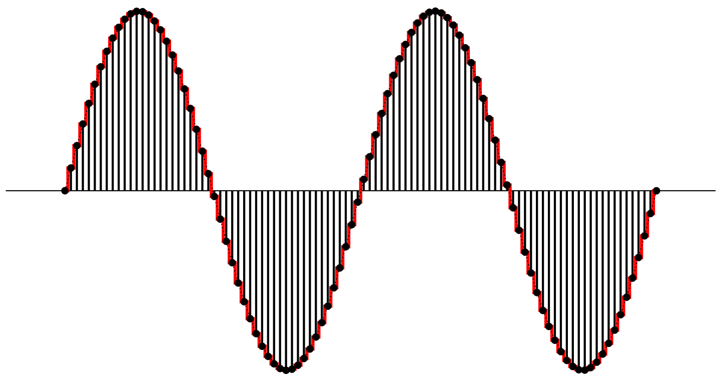
Figure 1.7: 16-bit quantization example

1.4 *Divide and conquer: the math way*

The previous sections should have given enough information to proceed to the real problem: how to break down a complex audio signal into pure sine waves with their own parameters.

1.4.1 *The Taylor series*

A **Taylor series** is a series expansion of a function about a point.The Taylor series of a *real function* $f(x)$ about a point a is given by:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \tag{1.1}$$

If $a = 0$ then the expansion (1.1) is known as a **Maclaurin series**, given by:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!}x^n$$

If the upper limit of the summation is not an ∞ but a generic N then a Taylor polynomial is being calculated.

The Taylor's theorem states that any function can be expressed as a Taylor series as long as the function can be derived N times in a **neighborhood** of a. But this is not always possible for a normal audio signal, for instance, seeing the figure 1.8, the function represented is neither analytic nor continuous, so it cannot be approximated by a Taylor polynomial.

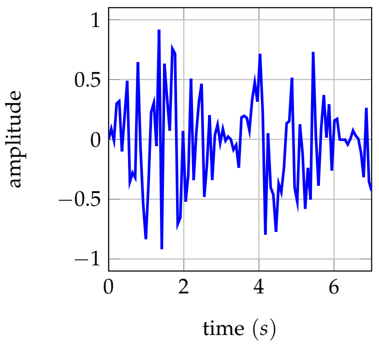

Figure 1.8: Audio signal example

1.4.2 *The Fourier series*

Joseph Fourier, a French mathematician, discovered a way to approximate a discontinuous function using a so-called **trigonometric polynomial** or a **trigonometric series**.

1.4.2.1 Def: Trigonometric polynomial

A trigonometric polynomial of order n is a function like:

$$P_n(x) = \tfrac{a_0}{2} + \sum_{k=1}^{n} (a_k \cos kx + b_k \sin kx), \text{ with } k \in \mathbb{N}$$

The real constants a_k and b_k are independent from x and they are called **polynomial coefficients**.

The term $a_k \cos kx + b_k \sin kx$ is a sinusoidal function whose period is $T_k = \frac{2\pi}{k}$.

So a trigonometric polynomial is a linear combination of sinusoidal functions, whose common period is $T = 2\pi$, multiple of all the main periods. It is a periodic function, but not necessarily sinusoidal.

1.4.2.2 Def: Trigonometric series

A trigonometric series is a series of functions like:

$$\tfrac{a_0}{2} + \sum_{k=1}^{+\infty} (a_k \cos kx + b_k \sin kx), \text{ with } k \in \mathbb{N}$$

a_0, a_k and b_k are the **coefficients** of the series.

Now, take a generic periodic function $f(x)$ with $T = 2\pi$, integrable in $[-\pi, \pi]$. Then surely $f(x) \cos nx$ and $f(x) \sin nx$ with $n \in \mathbb{N}$ will be integrable in $[-\pi, \pi]$.

1.4.2.3 Def: Fourier coefficients of $f(x)$

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\,dx$$

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos nx\,dx$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin nx\,dx$$

with $n \in \mathbb{N}^\star$

1.4.2.4 Def: Fourier series

Given a function $f(x)$ with $T = 2\pi$ and integrable in $[-\pi, \pi]$, the following trigonometric series is called **Fourier series**:

$$\frac{a_0}{2} + \sum_{n=1}^{+\infty} (a_n \cos nx + b_n \sin nx)$$

whose coefficients are the Fourier coefficients.

If the function $f(x)$ to be expanded is even or odd the Fourier series can be simplified as follows:

- $f(x)$ is even, Fourier series is a cosine series[2]

$$\frac{a_0}{2} + \sum_{n=1}^{+\infty} a_n \cos nx$$

  where:

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\,dx = \frac{2}{\pi}\int_{0}^{\pi} f(x)\,dx$$

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos nx\,dx = \frac{2}{\pi}\int_{0}^{\pi} f(x)\cos nx\,dx$$

- $f(x)$ is odd, Fourier series is a sine series

$$\sum_{n=1}^{+\infty} b_n \sin nx$$

  where:

$$b_n = \frac{2}{\pi}\int_{0}^{\pi} f(x)\sin nx\,dx$$

[2] The product of two even functions is an even function; the product of two odd functions is an even function.
Integration by parts is often used to solve those integrals.

1.5 *Discrete Fourier Transform*

The **DFT** (Discrete Fourier Transform) applies to discrete signals and gives a discrete spectrum.

$$X(n) = \sum_{t=0}^{N-1} x(t)e^{-i\frac{2\pi tn}{N}} \tag{1.2}$$

Where:

- N is the size of the **window**: the number of samples that composed the signal
- $X(n)$ is the **nth bin of frequencies**
- $x(t)$ is the **tth sample of the audio signal**

The interpretation is that the vector x represents the signal level at various points in time, the vector X represents the signal level at various frequencies. What the formula (1.2) states is that the signal level at frequency n is equal to the sum of the signal level at each time t multiplied by a complex exponential (figure 1.9).

For example, take an audio signal with 512-sample window, this formula must be applied 512 times:

- Once for $n = 0$ to compute the 0^{th} bin of frequencies
- Once for $n = 1$ to compute the 1^{st} bin of frequencies
- ...
- Once for $n = 511$ to compute the 511^{th} bin of frequencies

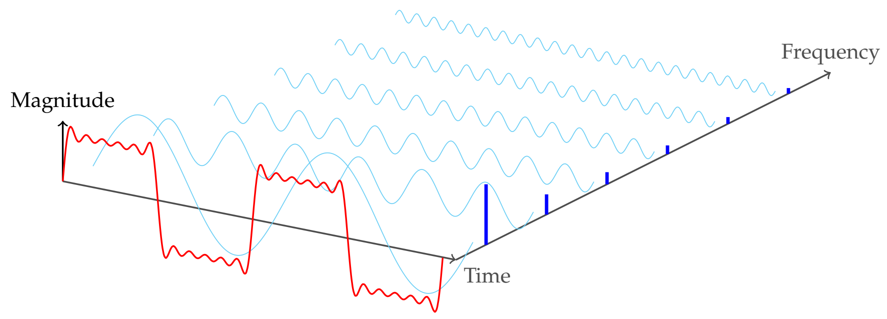

Figure 1.9: Visual representation of FT

A bin of frequencies is a group of frequencies among two boundaries.

The reason why the DFT can compute bins of frequencies and not exact frequencies is quite simple: the DFT gives a **discrete spectrum**. A bin of frequencies is the smallest unit of frequencies the DFT can compute and the size of the bin is called **spectral resolution** or **frequency resolution** which is given by the formula:

$$B_S = \frac{F_S}{N}$$

Where:

- B_S is the **bin size**

- F_S is the **sampling rate**

- N is the **size of the window**

For instance, taking a sampling rate of 8000 Hz and a window size of 512, the bin size will be of **15.6 Hz**, so:

- The 0^{th} bin contains the frequencies between 0 Hz and 15.6 Hz

- The 1^{st} bin contains the frequencies between 15.6 Hz and 31.2 Hz

- And so on

A particularity for a real-valued signal[3] is that **only half of the bins computed by the DFT are needed** since the output of the DFT is symmetric. In this case, *fewer calculations* can be made by exploiting this property, which goes under the name of **conjugate complex symmetry**.

[3] Such as an audio recording

1.5.0.1 Theorem: Conjugate complex symmetry of the DFT

If a function $x(t)$ is real valued then:

$$X(N-n) = X^*(n) \tag{1.3}$$

where:

- $X(\odot)$ is the output of the DFT applied to $x(t)$
- $(\odot)^*$ denotes the conjugate

1.5.0.2 Proof: Conjugate complex symmetry of the DFT

Insert in (1.2) the property (1.3):

$$\begin{aligned}
X(n) &= \sum_{t=0}^{N-1} x(t)e^{-i\frac{2\pi tn}{N}} && (1.4)\\
X(N-n) &= \sum_{t=0}^{N-1} x(t)e^{-i\frac{2\pi t(N-n)}{N}} && (1.5)\\
&= \sum_{t=0}^{N-1} x(t)e^{-i2\pi t}e^{i\frac{2\pi tn}{N}} \\
&= \sum_{t=0}^{N-1} x(t)e^{i\frac{2\pi tn}{N}} && (1.6)\\
&= \left(\sum_{t=0}^{N-1} x(t)e^{-i\frac{2\pi tn}{N}}\right)^* && (1.7)\\
&= X^*(n)
\end{aligned}$$

Where:

(1.4) is simply the plain DFT definition

(1.5) is the DFT evaluated in $N-n$

(1.6) uses that $e^{-i2\pi t} = 1\,\forall t$

(1.7) exploits that $x(t)$ is real

Hence, if the window size is equal to 512:

	0	1	2	3	...	508	509	510	511
$X =$					...	$X^*[3]$	$X^*[2]$	$X^*[1]$	$X^*[0]$

So, the DFT algorithm needs to be repeated only half times the window size[4].

To be accurate, most real-DFT implementations outputs an $N/2+1$ length array, where N is the window size. Taking, as always, a sampling rate of 8000 Hz and a window size of 512[5]:

- The 0th bin contains the so-called DC component or offset, being the sum of each sample in the window (see 1.5.0.3)
- The 1st bin contains the frequencies between 0 Hz and 15.6 Hz
- The 2nd bin contains the frequencies between 15.6 Hz and 31.2 Hz
- And so on

[4] 256 times in this example

[5] With the bin size being 15.6 Hz

1.5.0.3 Proof: DC component

Calculate the DFT (1.2) with $n = 0$:

$$\begin{aligned} X(n)\,|_{n=0} &= \sum_{t=0}^{N-1} x(t)e^{-i\frac{2\pi tn}{N}}\,|_{n=0} \\ &= \sum_{t=0}^{N-1} x(t) \end{aligned}$$

The DC component is simply ignored by the algorithm implementation and, in most cases, equal to 0.

1.6 *Window function*

Now the problem is partially solved: the DFT can be used to obtain the frequencies amplitude for (just to say) the first $\frac{1}{10}$ second part of the song, for the second, the third and so on.

The problem is that in this way a rectangular function is implicitly applied: a function that equals 1 for the song portion under analysis and 0 elsewhere (figure 1.10).

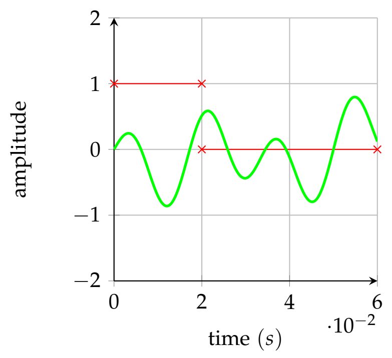

Figure 1.10: Rectangular window function example

By windowing the audio signal, the audio signal is multiplied by a window function which depends on the piece of the audio signal under analysis. The usage of a window function produces **spectral leakage**. Spectral leakage is the appearance of new frequencies that does not exist inside the audio signal. The power of the real frequencies is leaked to others frequencies.

Spectral leakage cannot be avoided but it can be controlled and reduced by choosing the right window function: there are many different window functions besides the rectangular one. Just to name a few: *triangular*, *Blackman*, *Hamming*, *Hann* and many others.

When analyzing unknown very noisy data best choice is the Hann window function, defined by the following formula:

$$w(n) = \frac{1}{2}\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right)$$

Where:

- N is the size of the window
- $w(n)$ is the value of the window function at n

The aim of this window function is to decrease the amplitude of the discontinuities at the boundaries of a given piece of an audio signal (see figure 1.11).

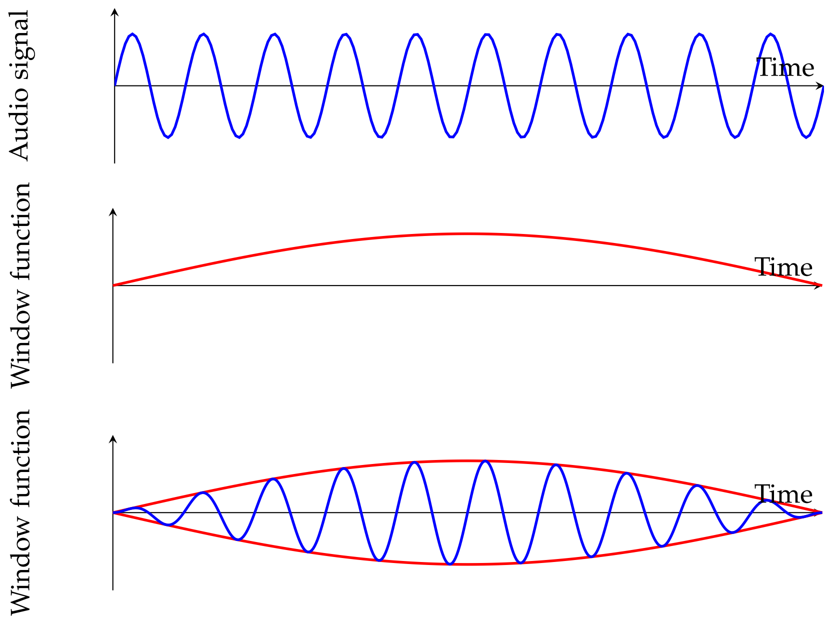

Figure 1.11: Shape the signal with a window function

1.7 *Downsampling and window size*

Downsampling is the process of reducing the sampling rate of a signal. Also, the window size can be reduced accordingly, taking the highest possible acceptable value[6]. For instance, resample a song from 44.1 kHz to 8 kHz and use a window size of just 512: in this way, there are fewer samples to analyze and the DFT is applied less frequently.

[6] The value which gives an acceptable frequency resolution

The only difference is that the resampled song will only have frequencies from 0 to 4 kHz (Nyquist-Shannon theorem 1.3.1.1), but the most important part of the song is still present in this range of frequencies.

2 The actual algorithm

Now it is the time to put everything together and start coding the algorithm. The aim of the software is to record a small portion of a song and find its title.

The following sections will cover a global overview of the algorithm, then the actual process involved in the song scoring.

Take into account that, at this abstraction level, some obvious parts[1] will not be described, since they are standard pieces of code.

[1] Such as the track file reading routines

2.1 Global overview

An **audio fingerprint** is a digital summary that can be used to identify an audio sample. In the figure 2.1 a simplified architecture of the scoring algorithm is represented.

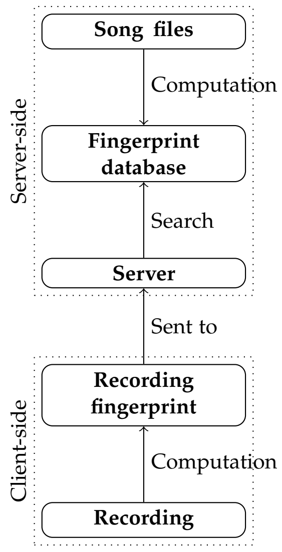

Figure 2.1: General overview scheme

On the **server side**:

- The algorithm computes the fingerprints of the input tracks
- The computed fingerprints are stored in the database

On the **client side**:

- The current playing music is recorded for a couple of seconds
- The algorithm computes the fingerprints of the recording
- The fingerprints are sent to the server
- The server analyzes the fingerprints and possibly outputs a song title

In the following sections the algorithm is being described.

2.2 *Spectrogram creation*

The first step to analyze the audio is to create the spectrogram. The process is described in the following lines:

- A 512-sized Hann window function is computed
- The audio signal is divided in 512-sized windows with a 50% overlap (figure 2.2)
- Each audio window is multiplied by the Hann precomputed window
- The DFT is computed for each audio window and added into a vector which represents the spectrogram

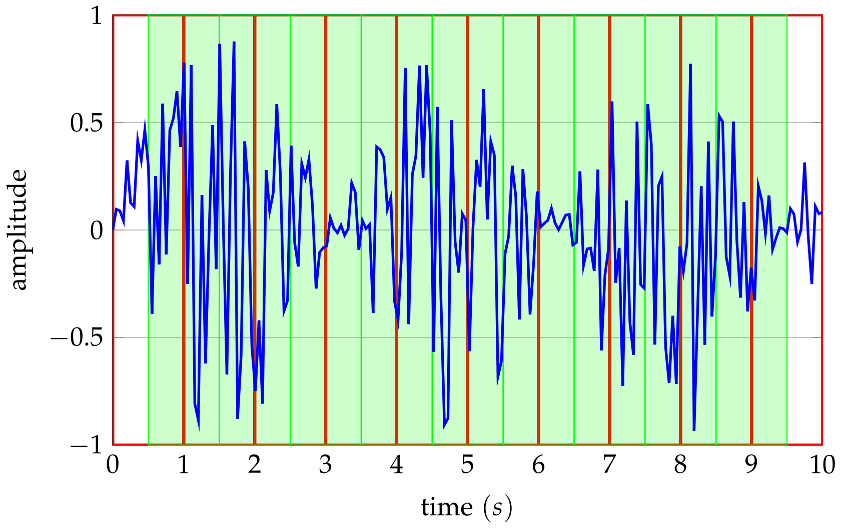

Figure 2.2: Audio windowing with overlap

2.3 *Peaks finder and fingerprinting*

The spectrogram is then processed in order to obtain its most significant information. It is divided into a sort of a grid where each cell of the grid has the following size:

- Width equals to C
- Height equals to a range of frequencies (called **band**)

Take the simplified figure 2.3 where:

- $C = 4$
- The band size is always 100 Hz

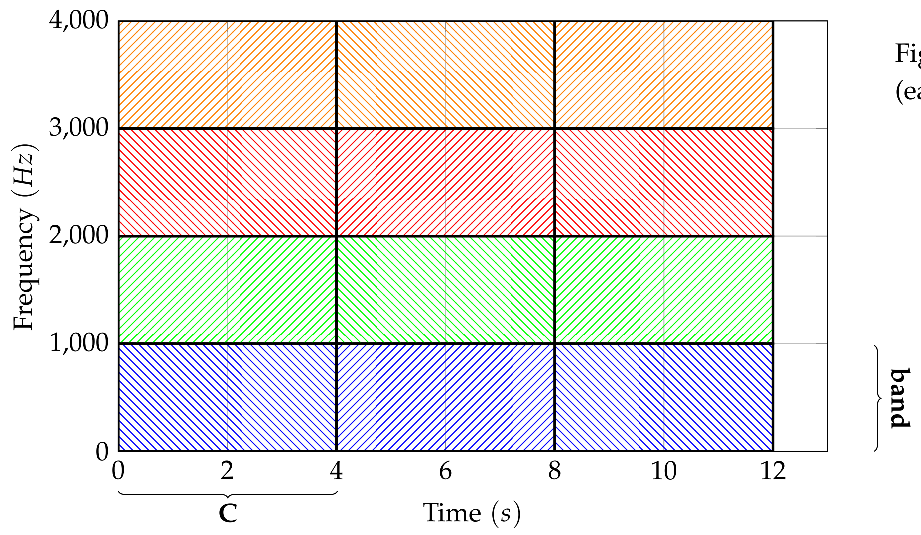

Figure 2.3: Spectrogram division (each color is a band)

In the actual algorithm $C = 32$ and the bands lengths follow a logarithmic scale (see Dividing the spectrogram in bands).

For each cell the algorithm finds and stores the 3 most powerful frequencies in a vector (figure 2.4).

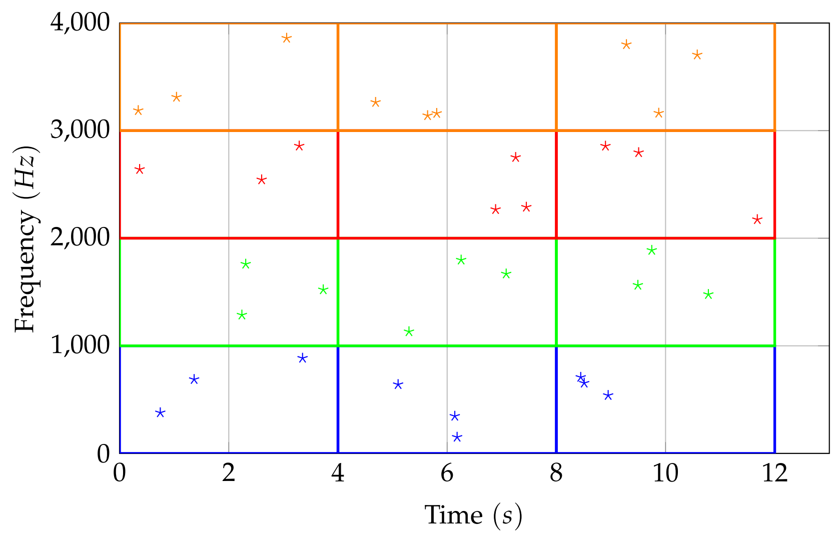

Figure 2.4: Three points per cell spectrogram

2.3.1 *Dividing the spectrogram in bands*

There exist at least a few ways to divide the spectrogram in meaningful frequency ranges, but the most interesting one is the **mel scale**. It was experimentally discovered that the higher the frequency the hardest for a human hear to notice the difference between relatively close but different frequencies.

In other words, humans perceive frequencies on a *logarithmic scale*, so that the difference between 500 and 1000 Hz is noticeable, but this is not true for the difference between 10000 and 10000 Hz.

The formula to convert between the Hertz (linear) scale and the mel (logarithmic) scale is defined as:

$$f(m) = 700\left(10^{\frac{m}{2595}} - 1\right) \tag{2.1}$$

where:

- **m** is the mel frequency
- **f** is the Hertz frequency

The plotted formula is represented in figure 2.5.

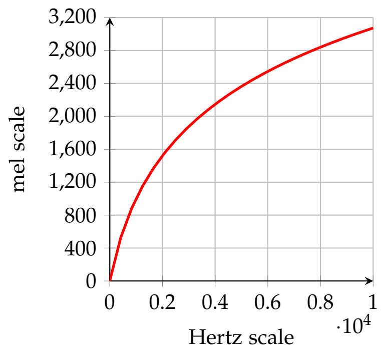

Figure 2.5: Mel scale to Hertz mapping

The algorithm follows this approach:

1. A starting mel value α is chosen
2. A mel band size δ is chosen
3. Compute the mel bands boundaries as:
$$\{0, \alpha, \alpha + \delta, \alpha + 2\delta, ..., \alpha + k\delta\}$$
4. Convert the mel band boundaries back to Hertz with formula (2.1):
$$\{0, f(\alpha), f(\alpha + \delta), f(\alpha + 2\delta), ..., f(\alpha + k\delta)\}$$

Each band will contain roughly the same "amount of information" from a listener point of view.

2.4 *Links structure*

Now, having a way to compute a frequency summary of an audio signal, the recognition process should be straight enough: each point of the recording should be compared with the points of the full song. Though it works well, this simple approach requires a lot of computation time.

Instead of comparing each point one by one, the idea is to look for multiple points at the same time. This group of points is called **links**, and it represents a relationship between a point and some others. In order to be sure that both the recording and the full song will generate the same links, a rule must be defined: spectrogram points must be sorted in ascending order according to the window they belong to.

So, given a point α (called **address**) and an initially empty set of points A, the algorithm will put in A all the other points β whose absolute window difference compared to the point α is between 1 and 3 and are in the same band as α. In symbols:

$$\beta \in A \Leftrightarrow (1 \leq |\beta.window - \alpha.window| < 3) \wedge (\beta.band = \alpha.band)$$

For each point β belonging to A a **link** is computed and added to a list (figure 2.6). The link structure is the following (figure 2.7):

$$link \begin{cases} .hash = h\left(\delta_w, \delta_f, \alpha.frequency\right) \\ .window = \alpha.window \end{cases}$$

Where:

- $\delta_w = \beta.window - \alpha.window$
- $\delta_f = \beta.frequency - \alpha.frequency$
- h is a generic hash function

The produced links are quite reproducible, even in the presence of noise and codec compression. Another advantage is that in this way all the times are relative.

On the server side each link is stored in a database along with the song information.

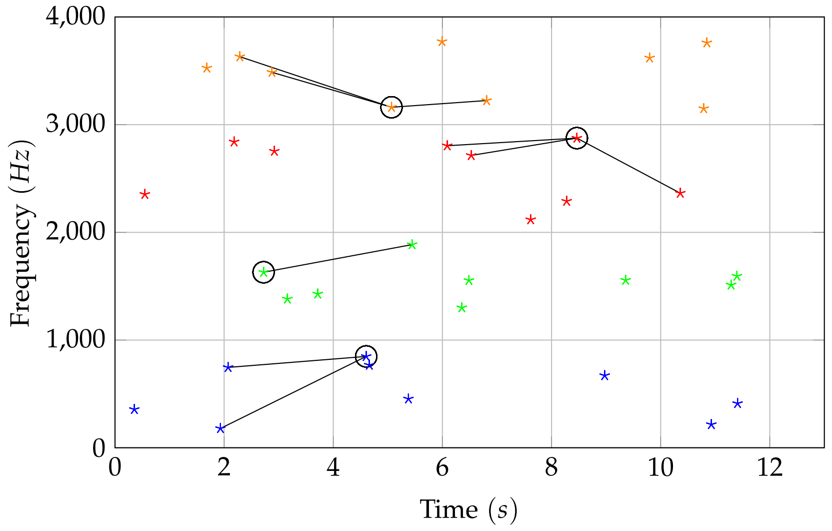

Figure 2.6: Links creation

2.5 Scoring

It is assumed that the full song links are in the database. Now a client, after recording a small piece of song, computes on its own the links of the recording and sends them to the server. The server has to:

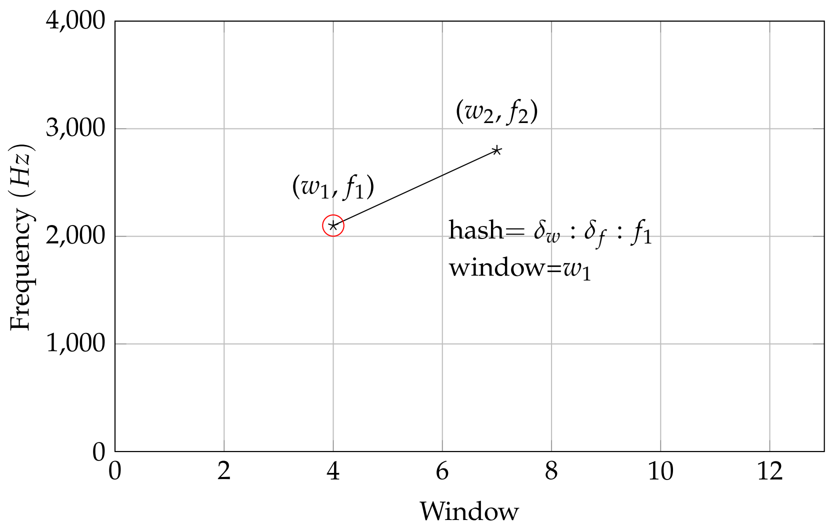

Figure 2.7: Link structure

- Put the recording links in a temporary in-memory table
- Relate the full songs links table with the recording table if they share the same hash
- Count the tuples grouped by the time difference between the recording links and the full song links[2] and the song id
- Descending sort the tuples according to the count field
- The topmost tuple obtained contains the id of the most likely matching

[2] Links of the same song share the same time difference

Webliography and bibliography

- Le serie di Fourier - Matematica.verde - Modulo Epsilon - Le serie, le serie di Fourier e la trasformata di Laplace
- `https://helpx.adobe.com/audition/using/digitizing-audio.html`
- `http://zone.ni.com/reference/en-XX/help/370524T-01/siggenhelp/fund_nyquist_and_shannon_theorems/`
- `http://mathworld.wolfram.com/TaylorSeries.html`
- `http://mathworld.wolfram.com/RectangleFunction.html`
- `https://en.wikipedia.org/wiki/Window_function`
- `https://www.sfu.ca/sonic-studio-webdav/handbook/Mel.html`
- `https://tex.stackexchange.com/questions/127375/replicate-the-fourier-transform-time-frequency-domains-correspondence-illustrati`
- `https://stackoverflow.com/questions/7560860/are-tables-created-with-create-temporary-table-in-memory-or-on-disk?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa`
- `https://stackoverflow.com/questions/8269916/what-is-sliding-window-algorithm-examples`
- `https://dsp.stackexchange.com/questions/19776/is-it-necessary-to-apply-some-window-method-to-obtain-the-fft-java`
- `https://stackoverflow.com/questions/6740545/understanding-fft-output`
- `https://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf`