# Audio Processing - Onset Detection and Tempo Estimation

## Introduction

For this project I decided to create an Onset Detection and Tempo Estimation algorithm using energy calculations. The program consists of one script which uses numpy, matplotlib.pyplot, soundfile and sounddevice which will need to be installed. The script needs a .wav file it can analyze, which is loaded on line 7 using the file name as a string. When the script is run it detects onsets and estimates tempo, it prints a list of timestamps where it detected onsets and prints the estimated tempo in BPM. Then it plots the detected onsets in a diagram that shows the delta of the energy windows and marks where onsets are detected. When onset diagram is plotted, it will start playing back the audio file with beeps where onsets where detected. After the audio file has been played back, it plots a mel-spectrogram and a tempogram and waits for a button press which will close the plots and the script will be done. The project is on GitHub and the repository can be accessed with the following link: https://github.com/mattiHedegaard/AudioProcessingOnsetTempo.git

The audio file used for this project is from GTZAN (Tzanetakis & Cook, 2002).

## Implementation

This section outlines the implementation of the algorithm. It is divided into two parts: the practical implementation—which covers onset detection and tempo estimation—and the analytical component, which includes visualizations such as plots. Additionally, the theoretical basis will be discussed alongside the implementation.

### Onset Detection

The onset detection is calculated using energy calculation in overlapping windows. The idea is to check for energy changes and see if and if they are significant enough it will be counted as an onset.

```
# Parameters
N = 1024        # window size
H = 512         # hop size
threshold_ratio = 0.3  # onset threshold ratio
```

To start of the window size $N$ and the hop size $N$ is decided and saved as a parameter. Here the following statement needs to be true; $N > H$. If $N < H$ the windows would not overlap. Then a threshold ratio for the onsets is chosen, for how much change in energy there needs to be.

```
# Compute energy in overlapping windows
energies = []
positions = []
for m in range(0, len(y) - N, H):
    window = y[m:m + N]
    energy = np.sum(window ** 2)
    energies.append(energy)
    positions.append(m)
```

Then to calculate the energy for each window this this for loop then uses this formula:

$$E_k = \sum_{n=0}^{N-1} x_k[n]^2$$

Where $E_k$ is the short-time energy in window $k$, $N$ is the window size and $x_k[n]^2$ is $n$'th sample in the $k$'th window. So, for each sample in the window, the sample is squared and added to the total energy of the window, the result for the window will then be the sum of the samples squared in window $k$. And the results are saved in a list of the energy results and a list of the position of the windows.

```
# Compute difference in energy
delta_energy = np.diff(energies, prepend=energies[0])
```

This computes the first-order differences of energy which represent the delta in energy using this formula:

$$\Delta E[k] = E[k] - E[k-1]$$

So, it checks the difference in energy for each window compared to the previous.

```
# Detect onsets
threshold = np.max(delta_energy) * threshold_ratio
onset_indices = np.where(delta_energy > threshold)[0]
onset_sample_positions = positions[onset_indices]
onset_times = onset_sample_positions / sr
```

Here the algorithm decides a threshold using the threshold ratio. It calculates it as a percentage of the biggest delta. And then checks all $\Delta E_k$ to see if the energy change is greater or significantly big to be an onset and stores all onsets sample positions and timestamps in lists. Where the timestamps are converting to time using this formula:

$$t = \frac{sample\ imdex}{sample\ rate}$$

## Tempo Estimation

```
# Estimate tempo
if len(onset_times) > 1:
    intervals = np.diff(onset_times)
    avg_interval = np.mean(intervals)
    tempo_bpm = 60.0 / avg_interval
else:
    tempo_bpm = 0.0
```

To estimate the tempo of the audio, the list with the timestamp for each onset is used to calculate the average interval between onsets and then a BPM is calculated using this formula:

$$BPM = \frac{60}{avarage\ interval\ in\ seconds}$$

This assumes a regular pulse and calculates how many beats fit into one minute.

## Plots

### *Mel-Spectrogram*

I Tested the algorithm with a 30 second disco song which produced this mel-spectrogram:

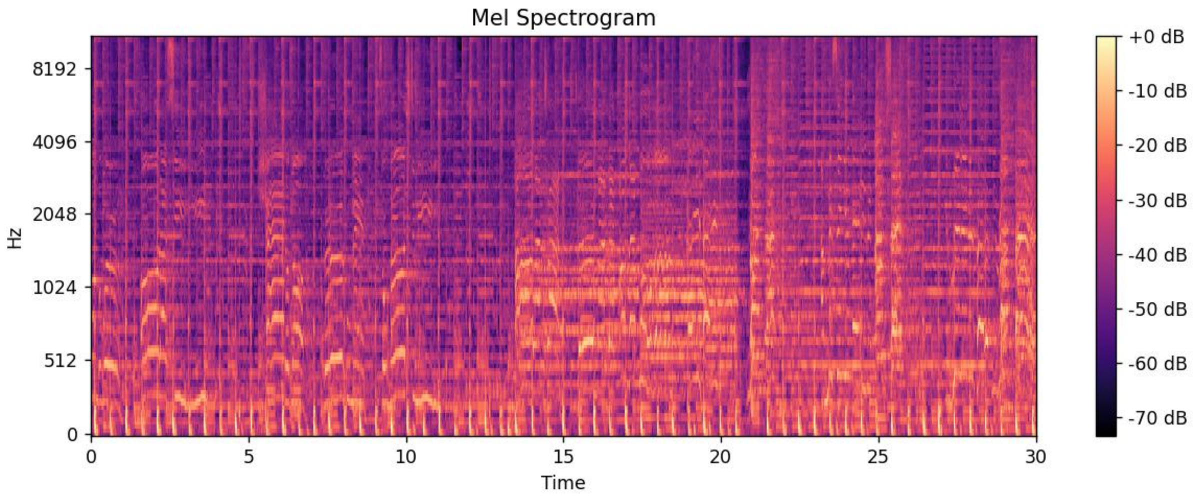

*Figure 1: Mel-Spectrogram from a 30 second disco audio file*

In a mel-spectrogram there is time as the x-axis, the frequency, on the mel scale, as the y-axis, and the intensity of the color represents how loud the sound is in decibel (dB). So, to read this we look at a time on the x-axis and look for the brightest colors and see which frequency's those sounds are played at.

This mel-spectrogram in figure 1 shows:

- the drumbeats at the bottom of the graph; drumbeats are loud and consist of low-frequency sounds therefore they are showing up at the bottom in a bright orange color.
- Around 14-20 we can hear background vocals singing a higher pitch which shows around 512-1024 Hz.
- Around 1, 3 6, 8, 10, 22, 25 and 30 seconds, there are louder sounds around the mid-range frequency, likely corresponding to the lead singer.
- And around 21 there is a moment of near silence which corresponds to a musical break or transition just before the chorus.

```
S = librosa.feature.melspectrogram(y=y_librosa, sr=sr, n_fft=N,
hop_length=H, n_mels=128)
S_dB = librosa.power_to_db(S, ref=np.max)
```

The mel spectrogram is computed using this function from the Librosa library which splits the waveform into overlapping windows and a Fourier Transform is computed for each using this formula:

$$X(k,m) = \sum_{n=0}^{N-1} x[n+mH] \cdot w[n] \cdot e^{-j2\pi kn/N}$$

Where:

- $x[m]$ is the input signal
- $w[m]$ is the window function
- $N$ is the window size
- $H$ is the hop size
- $m$ is the frame index

- $k$ is the frequency bin

Then when each window has been Fourier Transformed, we compute the squared magnitude using this formula:

$$|X(k,m)|^2$$

Then, a mel filterbank is applied to collapse the frequency spectrum into $n\_mels$ bands using triangular filters. These triangular filters reduce dimensionality and simulate how humans perceive pitch. The filters overlap and allow for smooth transitions between neighboring frequency bands. On the mel scale, the filter spacing is approximately linear at lower frequencies and logarithmic at higher frequencies, which mirrors how human hearing is.

After applying the filterbank, each value is converted to decibels using the formula:

$$s_{dB} = 10 \cdot \log_{10}\left(\frac{s}{ref}\right)$$

### *Tempogram*

Using the same audio file this tempogram was computed:

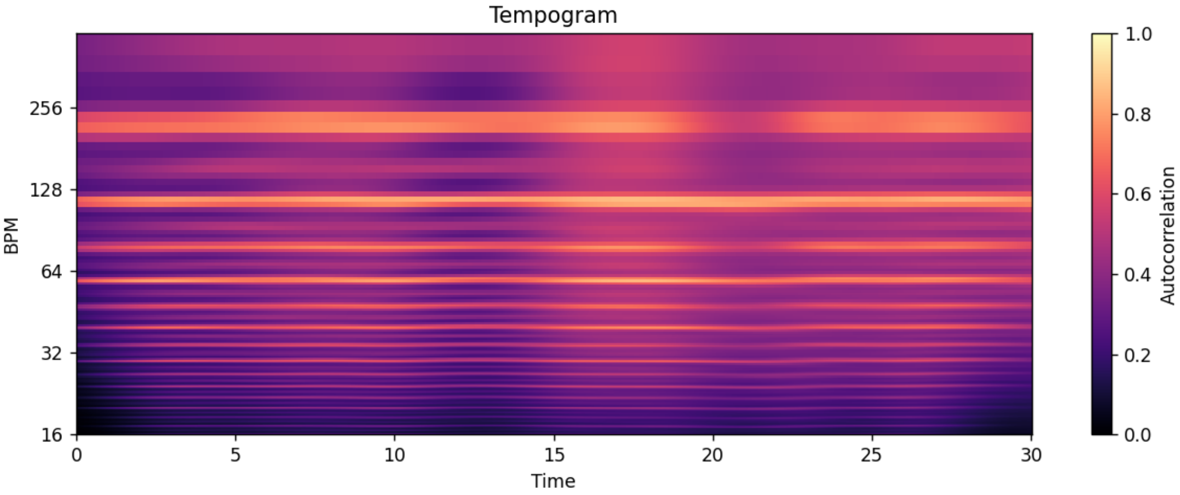

*Figure 2: Tempogram from a 30 second a disco audio file*

A tempogram shows the changes of tempo in BPM over time. On the x-axis there is time in seconds and in the y-axis is the BPM and the color represents the autocorrelation that is calculated throughout the song.

This tempogram in figure 2 shows:

- Around 120 is a very clear and bright line which indicates the beat is around 120 BPM which would be the beat we feel/hear.
- 240 BPM may represent a perceived double-time rhythm, often present in fast disco patterns.
- The line at 60 BPM may indicate slower subdivisions or a halftime feel

```
# Onset strength envelope
onset_env = librosa.onset.onset_strength(y=y_librosa, sr=sr,
hop_length=H)
# Tempogram
tempogram = librosa.feature.tempogram(onset_envelope=onset_env, sr=sr,
hop_length=H)
```

To compute a tempogram, it needs a onset envelope which is computed using spectral flux which captures increases In energy across frequency bins which usually signal new musical events:

$$onset\_env[m] = \sum_{k} \max\left(0, |X(k,m)| - |X(k,m-1|\right)$$

Then the like the other calculation, we split it into windows where the autocorrelation of each window is computed, which measures how well the onset envelope aligns with itself over different $\tau$:

$$T(m,t) = \sum_{l} onset\_env[m+l] \cdot onset\_env[m+l+\tau]$$

Where $m$ is the current time frame and $\tau$ is the lag. Then peaks in the autocorrelation correspond to repeating rhythmic patterns.

# References

Tzanetakis, G., & Cook, P. (2002). *Musical genre classification of audio signals*. IEEE Transactions on Speech and Audio Processing, 10(5), 293–302. https://doi.org/10.1109/TSA.2002.800560