Analyzing musical signals with BPM detection
Creating a BPM detection algotithm

by Julian Wösten,
HKU – 2nd year – juni 2018

To create audio-visual interactive systems one of the parameters that could be useful to detect is the BPM of a music piece. To study this aspect further I have tried to make an application that would analyze an audio signal in real time and return the approximated BPM of the signal.
This report provides a concise explanation of the steps I took to create my own BPM detection algorithm.

Plain and simple:
BPM: (Beat Per Minute) means how many beats are present in a minute of audio signal. Depending on the audio signal these beats can have different time intervals, however for my specific case I will make the assumption that all the beat intervals are equally spaced. Meaning every next beat interval will be the same size as the last one.

How to calculate BPM?
In this case the BPM can be calculated using the equation:

Equation 1:
60/ time interval = BPM

For example 120 BPM, means that the beat interval calculation would be:
60(seconds in a minute)/0.5 (interval in seconds)= 120 BPM
Now that we know how to calculate the BPM from an audio signal, the following step will be to figure out how to collect data necessary for this calculation.
For the BPM algorithm, that will be figuring out how to extract the beats of a signal and measuring the time intervals in between them. Once I have that information our basic beat tracking system is done.

So the question is how to extract a beat from an audio signal?

To do this we first have to look at some of the things that make up a beat.
Take for example a kick sample in the figure below.

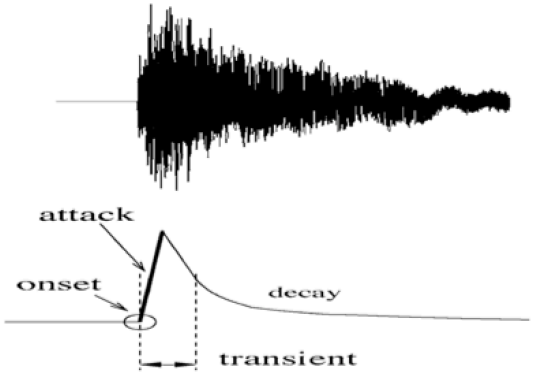


figure 1. onset visualized (source: see *references*)

In audio terms your could say that the kick has an ADSR (attack decay sustain release) The attack starts with the onset, then a short peak high amplitude. The onset and attack together are called a transient and as shown in the picture after that comes the decay of the signal.

Detecting the transient of a signal is interesting for BPM detection.
To determine the BPM of a signal it's very common to look for these transients. That's because knowing the transient/onset interval, between two following beats in a rhythm provides the data needed to calculate your BPM, as previously shown in *equation 1*.
But what exactly is this thing we are trying to detect?
To answer this we need to further into what music is, and how we can analyze music.

Analyzing music
Music is sound and sound is energy, music is always definable as a collection of slow and fast moving vibrations that can contain either low or high energy.
When a sound moves from quiet to loud what actually changes is the energy level of the signal, from low energy to high energy.
Just like the transient of a musical pulse.
That is what we are looking for: a way to detect big energy changes in a musical signal.
Which bring us to the next problem to solve:

How to detect energy changes in music?

An important element we will need for this BPM detection algorithm.
I will be using The Fast Fourier Transform for this purpose.

Note: You could skip reading this part if your just interested in the beat tracking algorithm, this was a little side research I did into the deeper working of the Fast Fourier Transform. To skip reading about the FFT go to page 6.

The Fast Fourier Transform.
To explain what the Fast Fourier Transform (FFT in short) we should first take a look at the Discrete Fourier Transform (DFT), leaving the "fast" part out for now.
To give a simple and accurate explanation:
"The Fourier Transform decomposes a signal into the frequency components that make it up."
The DFT does exactly this but instead of processing an infinite amount of data it processes a finite amount of data. This is done using the mathematical formula given below which is a big topic of its own I am not going to go into right now.
But in short it's a mathematical formula that looks like this:

$$X(k) = \sum_{N=0}^{N-1} x(n) e^{-i2\pi kn/N}$$ **Equation 2:DFT formula**

X(k) = the input signal, which gets a sweep from 0 to n-1.
(source: see references)

To explain the Fourier Transform, a practical example:
If we take an audio signal using the formula above we can now in theory split up the signal into all of the smaller frequencies that made up the original sound.
The Fourier Transform then puts the derived frequencies into what are called "bins".

To explain this concept the best you could imagine bins are buckets that the FFT fills up with the derived frequencies from the original signal. So imagine for example having a different low, mid, high frequency bucket filed with the corresponding frequencies. When we want to use FT bins it useful to think of the following formula:

$$n\ frequencies\ per\ bin = \frac{Sample\ rate}{n\ of\ bin\ used}$$ **Equation 3: Bin size calculation**

If you have a maximal sample rate of 44100Hz and the amount of FT bins set to 1024, this means that every individual bin stores 43Hz.
Because:
44100/1024 = 43
(This spectrum is not logarithmically divided like a more common musical spectrum. The spectrum of bin is linear. Which is an important detail to keep in mind.)

This also means that we would need to use 43 data points in a buffer to store 1 second of audio data. Theoretically now we should be able to monitor quite accurately which frequencies are present in our musical signal depending on how big we set our bin size. That is really useful if you want to detect i.e. the bass kick in a song and measure its amplitude!

Unfortunately in real life it's not that simple, because the DFT calculation is computed using the square of N. (^N) (see equation 2). Meaning to calculate 1000 data points using the DFT this requires millions of complex multiplications.
If sigma N is a big number (when you for ex. give a big chunk of audio data) the computational effort needed to Fourier Transform the signal increases drastically.

Luckily two smart mathematicians solved this problem for us (J. W. Cooley and J. W. Tukey in 1965).
They thought of a way to make the process of calculating the DFT faster by using a sorting algorithm to speed up the calculations.
And that's what brings me back to the FFT algorithm. This is very simplified, but basically the FFT is the DFT combined with a divide and concur sorting algorithm. There are different versions of the FFT but the one we will be looking at is the Cooley Tukey algorithm.

This algorithm uses a "butterfly diagram" to ease the computational effort the computer make. The simplified version of how this works is: while the DFT is being calculated the sorting algorithms makes sure that data points that are already previously calculated are reused again later via the butterfly algorithm. This allows the FFT to compute a signal faster than the DFT because it needs to compute less data. The sorting algorithm fills in the blanks.

The picture below shows the butterfly diagram:

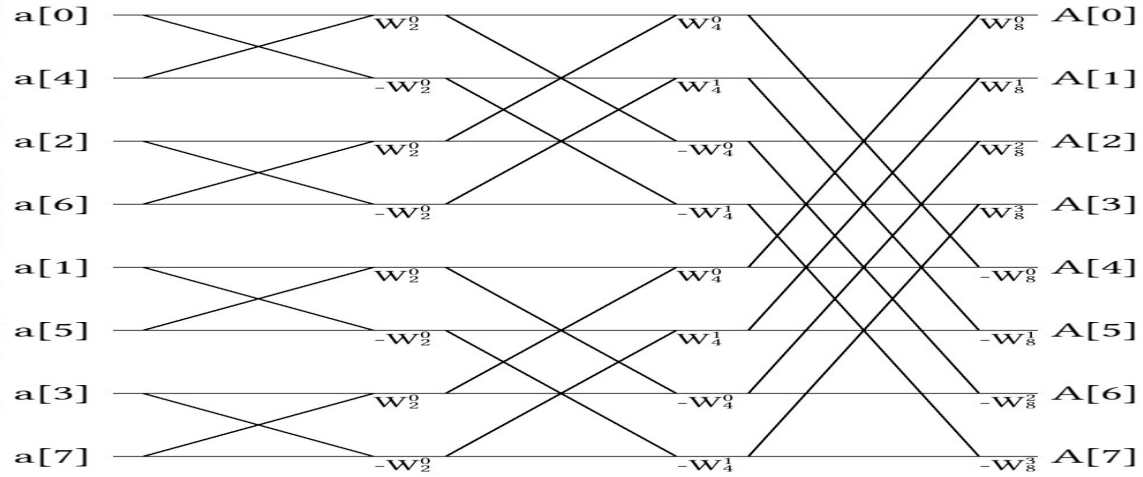
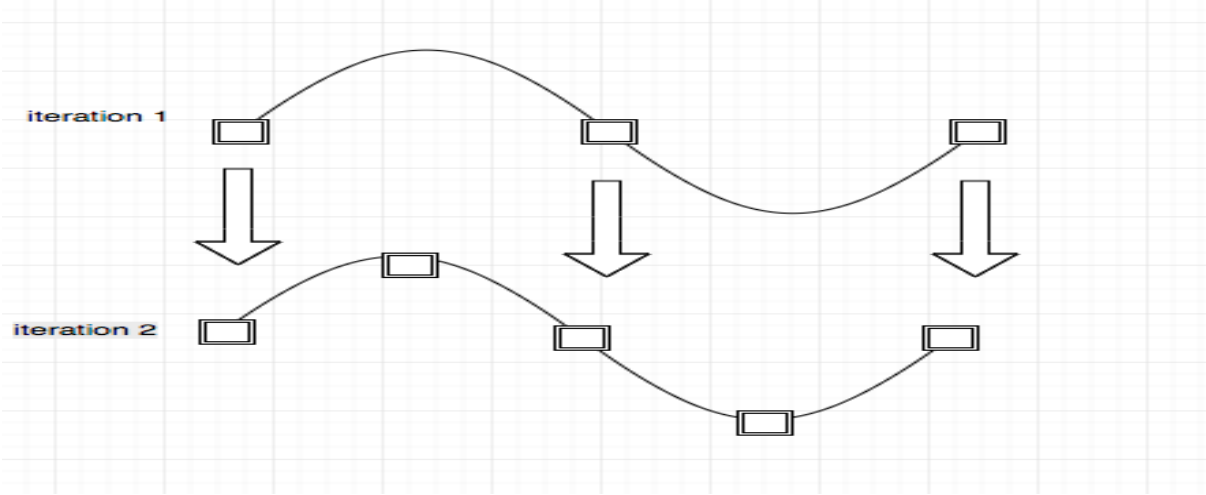


figure 3: a butterfly diagram (visually depicting how samples are processed by the algorithm)

If you give the FFT a large chunk of data, the butterfly diagram starts of with a number Fourier Transformations of 2 data points which are then further combined into 4 data points, then into 8 and then into16 etc. until you only have one array of data left. This array is the final FFT transformation. (As shown in the diagram above)
In this case the number of iterations trough the data is: n log base 2 of n, instead of the square of n (equation 2.)
Now we can see that this is a big improvement in speed over the DFT in the calculations below:

DFT: 10 ^10= 10.000.000.000
FFT: 10 log(10) = 23.023 (approximated)
(These equations calculate the number of interactions the FT has to do to get to its final result.)

To explain a bit better I made a graphical representation, (really simplified) as I personally understood it from an audio perspective.
The picture below shows 2 iterations of the FFT. In the first one 3 points of the signal are sampled, marked by the first 3 squares on the sine wave.
In the next iteration these points are used again in combination with the new sampled audio data points detecting another part of the wave by reusing the data from the previous calculations. And thus reducing the amount of data needed to compute the FT.



When we compute an FFT you give a set of data to the algorithm and the FFT spits out a Fourier transformed signal into different FFT bins. However there is a problem in combining sound with an FFT, because audio is not just a static data sheet of numbers to be analyzed by the algorithm. Besides the amplitude and phase information it also has a very important time dimension. You can't play a sound if there is no time dimension to distribute the signal in. The FFT algorithm only calculates two aspects of the data: Amplitude and Phase.
This means that to get the real time FFT of a song, we would repeatedly have to feed small chunks of audio data to the FFT to compute. That way chunk by chunk the audio gets analyzed but we don't lose the time based aspect of our sound signal. (Although of course this method reduces the signal flow depending on the size of the audio chunks.)
This technique of feeding data chunks to the FFT is what is called "Windowing". (Think of small audio sample windows that open up and are fed into the FFT.)
There are a lot of different windows that you could use for an FFT. If no specific window is used, the FFT automaticly uses a rectangular window. (because of the audio chunks given to the FFT periodicly)

When a rectangular window is used a problem could arise when a signal is sudenly cut of by the the window closing. A discontinous signal is then computed by the FFT which results in a high frequency that was not pressent in the origional signal.
To eliminate this problem a general window that is often used is the "Hamming window" Depicted in the picture below.

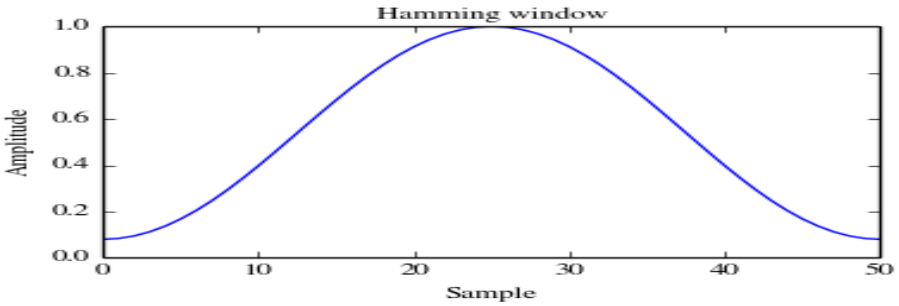


figure 4. a hamming window (source: see references)
the x axis represents the samples procesed by the FFT, and the Y axis is the amplitude of the signal.

The hamming window eliminates the discontinued signal problem mentioned earlier because the amlitude of the origional signal is smoothed out in the beginning and end of the window function.
using the FFT with a windowing function is called a STFT(Short-Time Fourier Transform) this is used a lot in audio analysis.

And there you go a FFT implementation that we can use for our beat tracking system!

That's basically what I needed to explain about the FFT algorithm. Now back to the beat tracking algorithm.
The next step is the calculation of the *"spectral flux".*

To quote Wikipedia for a "quick and dirty" explanation:
"Spectral flux is a measure of how quickly the power spectrum of a signal is changing, calculated by comparing the power spectrum for one frame against the power spectrum from the previous frame."
Source: https://en.wikipedia.org/wiki/Spectral_flux

Now remember how earlier on I was talking about how we wanted to find the transients in a musical signal, and that a transient is a positive rise in energy. Now this is where the two fit together perfectly, with the spectral flux of a signal we could detect a musical transient (rise in energy).
To calculate the spectral flux of our audio signal we could compare two subsequent FFT bin frames. And if there is a big peak in the signal it means a transient is detected.

$$s = \sum_{n=0}^{n-1} (k - k\,(n-1))$$

The spectral flux calculation.
(S) = the spectral flux, (K) is the current spectrum and (N) indicates the data point of (k) that is used.
(k (n-1)) is the previous spectrum data.

Great, we are almost there for our basic algorithm.
If we combine everything I discussed so far, the FFT and the spectral flux. We could try to detect the spectral fluctuations in the lower part of the frequency spectrum to find the kick drum of an audio track.
We could generalize that the kick drum of a piece of audio is found in the range of 60-150Hz.
Earlier I touched upon bin sizes and how the frequency spectrum is divided. So with a bin amount of 1024, and a sample rate of 44100, we would have to look in bin 2,3 and 4 to find the kick drum.
Because: bin 1 = 0-43hz, bin 2 = 43-86Hz, and so on.
To find the exact bin with the kick drum in real time I thought of the following technique in c++:

```
peak = 0;

for (unsigned int i = 0; i< 10; i++){
     //if current value is bigger than the last measured biggest peak make the
     current one the biggest
        if (peakFinder[i] > peak){
            peak = peakFinder[i];

            biggestBinNumber = i;

}

//reset peak finder buffer to recalculate
peakFinder.clear();
```

In the code above `peakFinder[i]` is a vector containing the first 10 FFT bins.
This code finds the bin that has the highest value and passes the bin number into `biggestBinNumber`.

The code is repeated over and over again, every time the biggest bin number and highest peak value are stored and the average value is calculated.
Using this code, the system dynamically calculates which bin it has to monitor to find the kick drum and it also gets the average highest energy peak from the kick signal.
The average peak energy calculation can be used as a threshold to know above what amplitude a signal must be to be perceived as a transient.

That's what we need for onset detection. To find the BPM, like mentioned in the beginning al we need is: the interval between two subsequent transients. (Equation 1.)
When the data of the first two onsets is detected you can create what is called a "pulse train". A pulse train checks if there is a periodically reoccurring interval between onset intervals, this would confirm a steady BPM pulse and when this is detected, the algorithm can output the calculated BPM.

As a last step to make the system a bit more accurate, we could add a small margin of error when checking for the steady BPM pulse interval. For example an error margin of 4% of the onset interval could make the system more accurate when detecting the BPM of music where the beat is not always strictly in sync with the rhythm.

Summary:

The BPM of a signal could be of good use when making the translation between audio-visual systems. To be able to detect the BPM of a signal I developed a basic algorithm using FFT and Spectral Flux. The FFT divides the frequencies of the original signal into bins. The spectral fluctuation of the bins is then calculated. Next the peak finder algorithm starts checking the amplitude threshold and bin number to monitor for the kick drum and finally the BPM is calculated.
In this study of finding the BPM of a signal we can now use BPM as a useful parameter in audio-visual programming.

Acknowledgments:
Just a small remark for one of my teachers Ciska, I was really having a hard time writing a piece about my research struggling where to begin and how in depth to go into the topics I wanted to touch upon.
But after the tip I received of writing it for somebody who would have very little knowledge of the subject. Which would put me in the teaching position. The writing became effortless and a very fun process.
It was a simple advice, but I have to say I helped me enormously writing this piece.

And of course I want to thank my girlfriend for proofreading my text☺

My final remarks:
And I hope this text will contain useful knowledge for anybody who stumbles upon it using the internet, my research into audio analysis is maybe not the best paper out there, but I hope it will provide a good clear starting point for anybody who is looking to dive into the subject☺

references:

Inspiration and ideas:
https://www.badlogicgames.com/wordpress/?p=161

Fourier Transform research:
http://www.versci.com/fft/index.html

FFT:
https://www.algorithm-archive.org/chapters/FFT/cooley_tukey.html
http://www.drdobbs.com/cpp/a-simple-and-efficient-fft-implementatio/199500857

Windowing:
http://download.ni.com/evaluation/pxi/Understanding%20FFTs%20and%20Windowing.pdf

Spectral flux:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.332.989&rep=rep1&type=pdf
https://www.badlogicgames.com/wordpress/?p=161

Beat tracking:
https://www.parallelcube.com/2018/03/30/beat-detection-algorithm/s

Wikipedia:
https://nl.wikipedia.org/wiki/Fast_Fourier_transform
https://en.wikipedia.org/wiki/Spectral_flux

Books(general info):
Computer music tutorial – Curtis Roads
Musis data analysis – Claus Weihs
Introduction to digital signal processing- Tae Hong Park
An introduction to audio content analysis – Alexander Lerch

Figure 1:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.332.989&rep=rep1&type=pdf

Equation 2:
http://practicalcryptography.com/miscellaneous/machine-learning/intuitive-guide-discrete-fourier-transform/

Figure 3:
https://www.algorithm-archive.org/chapters/FFT/cooley_tukey.html

Figure 4:
https://docs.scipy.org/doc/scipy0.15.1/reference/generated/scipy.signal.hamming.html