# Music loop extractor based on autocorrelation

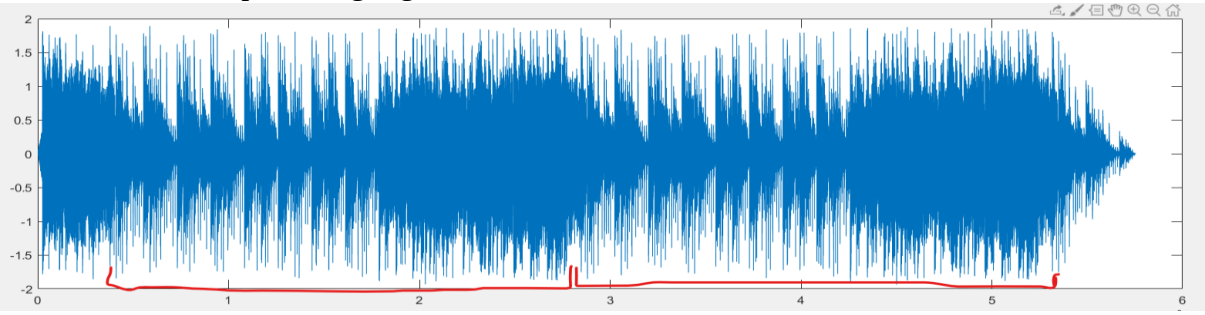
## By Wajd Boulos

During game development, I faced an interesting problem.
Since video games music is repeatedly played while the game is running, I need to use music that is periodic. Video games music that is not periodic will not sound good to the ear the moment the music is looped because we can clearly hear a discontinuity.

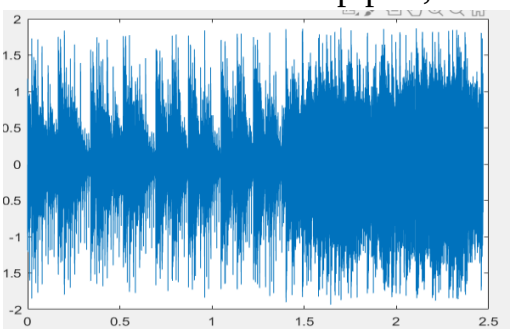
The way that video games music is presented in mp3 files is usually the following:
Intro-loop-loop-outro. (Could be more than 2 loops)
**The goal is to extract the loop from the audio file so that it can be played repeatedly when the game is running**. If we did not extract the loop the game engine will play the music in this manner:
Intro-loop-loop-outro-Intro-loop-loop-outro… which is NOT what we want!
What we want is the game to play the music in a natural manner: loop-loop-loop-loop-loop…
One of the benefits of this is also file compression, since we are only storing 1 loop in the MP3 file instead of 2 or 3.
While it's possible to use an audio editing program such as Audacity to do this, it is time consuming to determine the loop start and end with sample perfection. It is also tedious to do this manually for a large number of files (speaking from experience)

The solution: I propose a solution that solves this problem in O(NlogN), where N is the number of samples.
Time complexity is crucial here, the example below is 2 minutes and 10 seconds long, and it has exactly 5756016 samples! Solutions with worse time complexity like O(N^2) would take forever to complete and are impractical.

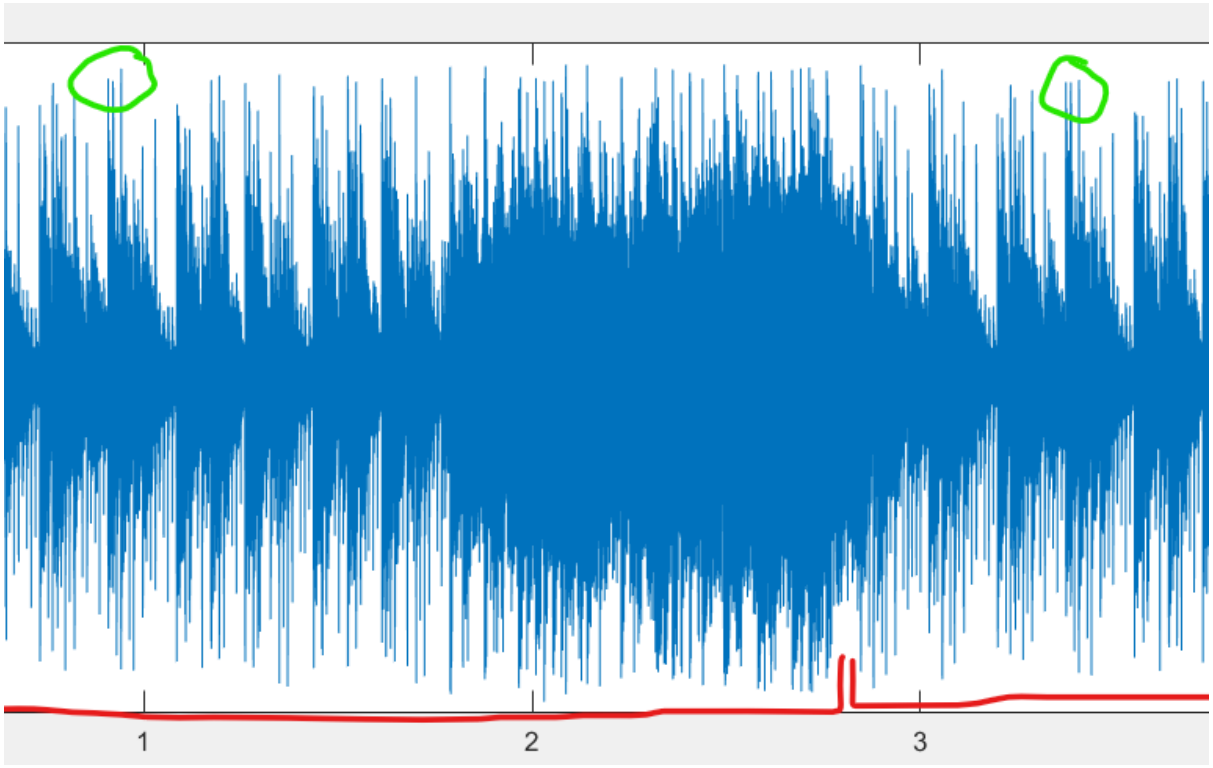
Let's start with an example.
As you can see in this audio file, the music starts with an intro, loops twice and fades away. The 2 identical loops are highlighted in red.




Our goal is to extract the loop part, which is this:



Note that due to mp3's lossy compression algorithm, the 2 loops aren't exactly identical, as seen in the green circles. You can think of this as an added noise. Thus, pretending that we're just dealing with an array and use comparisons between index values will not work. See the green circles for example.




So how do we solve this problem?
Answer: Autocorrelation!
Correlation between two signals help us determine how closely these signals are related.
Correlation is defined as:

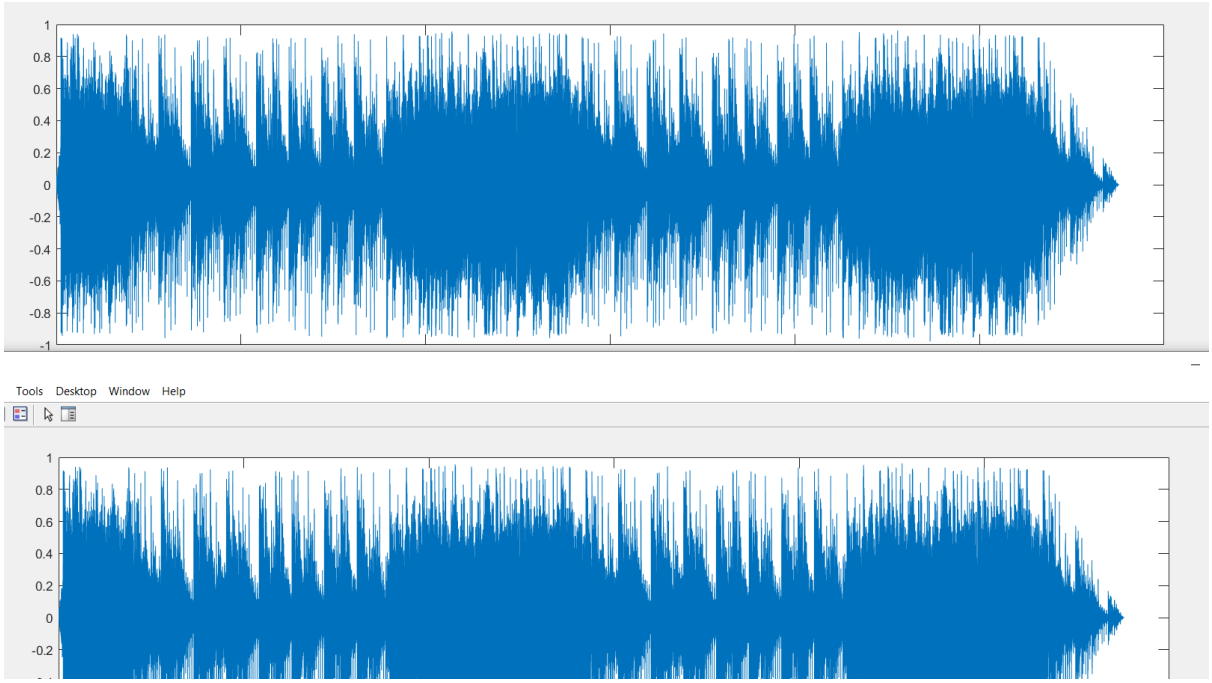
$$(f \star g)[n] \triangleq \sum_{m=-\infty}^{\infty} \overline{f[m-n]}\, g[m]$$

If we have a signal g and a signal f and we are trying to find if g has a part where it is similar to the signal f, we can simply apply cross correlation and find the maximum point of the correlation. The x part of that point will tell us where is the signal inside g that is similar to f is located.

But the problem here is that we only have 1 signal here which is our audio file, how can we still locate that loop if we don't already have it?

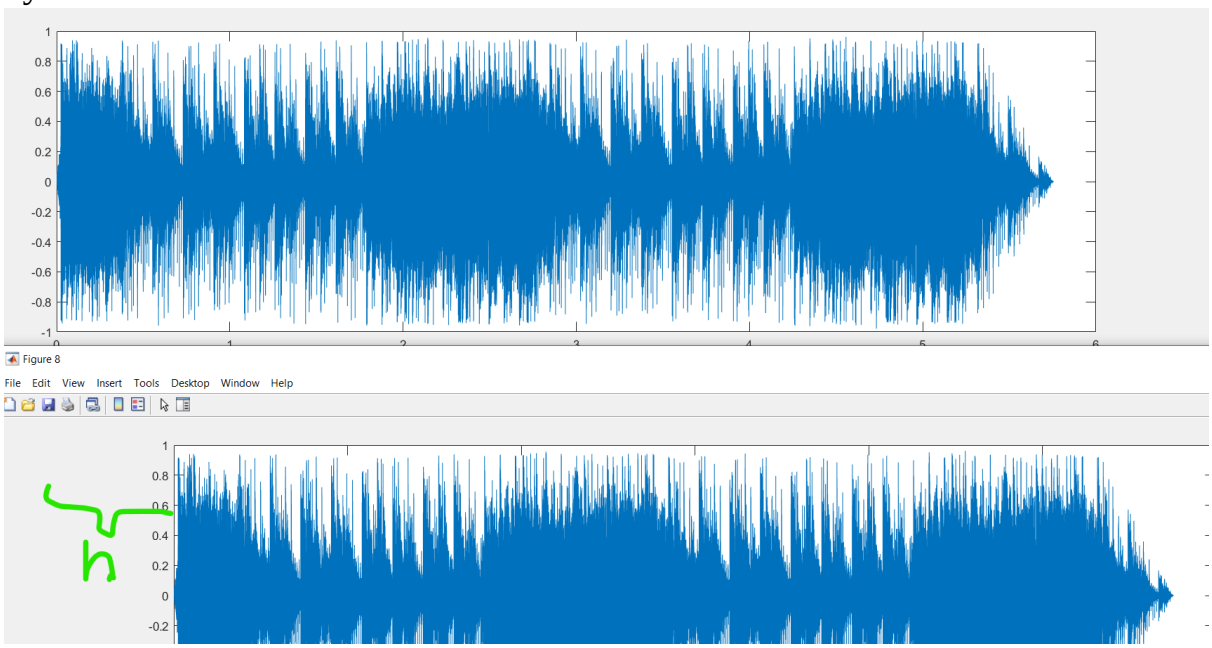
The trick here is to correlate our audio file with itself.

First, align the audio file with a copy version of itself, name the original f1 and the copied f2. Let T be the loop length.

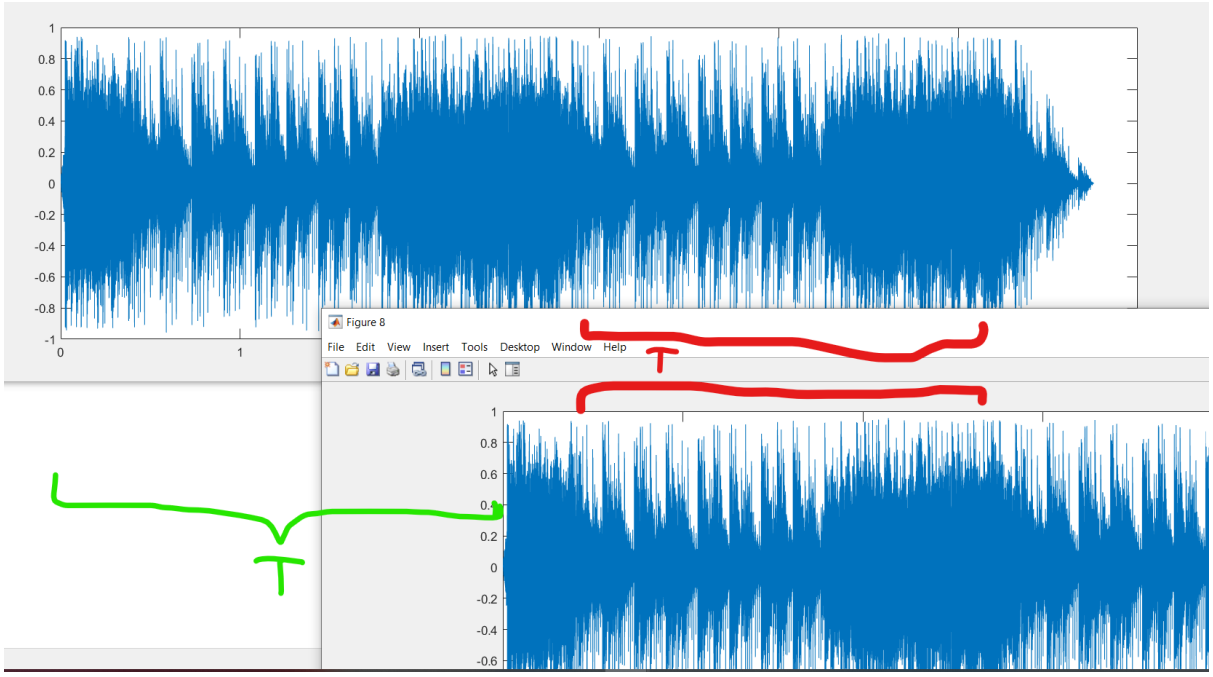



(When we apply inner product between f1 and f2, we are going to get the maximal correlation value between f1 and f2 at n=0 because we did not offset f2.)

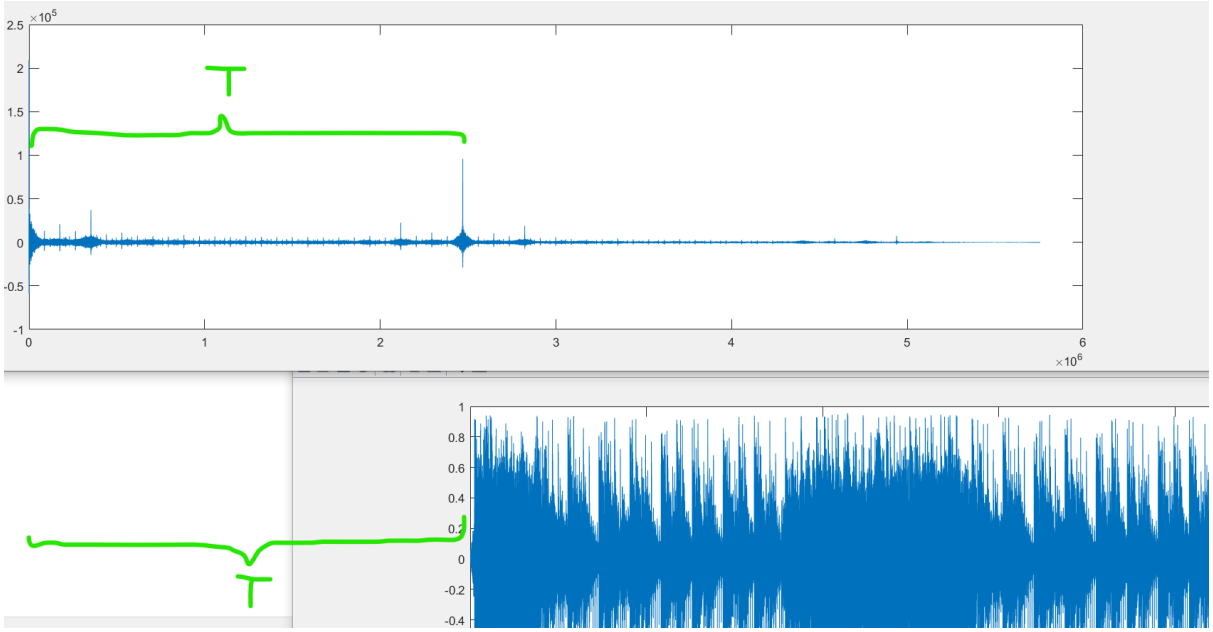
The cross correlation at index n is the inner product between f1 and f2 delayed by n:



Note than when n=T, the second loop of f1 will overlap with the first loop of f2, so the inner product will give us an exceptionally high value!:



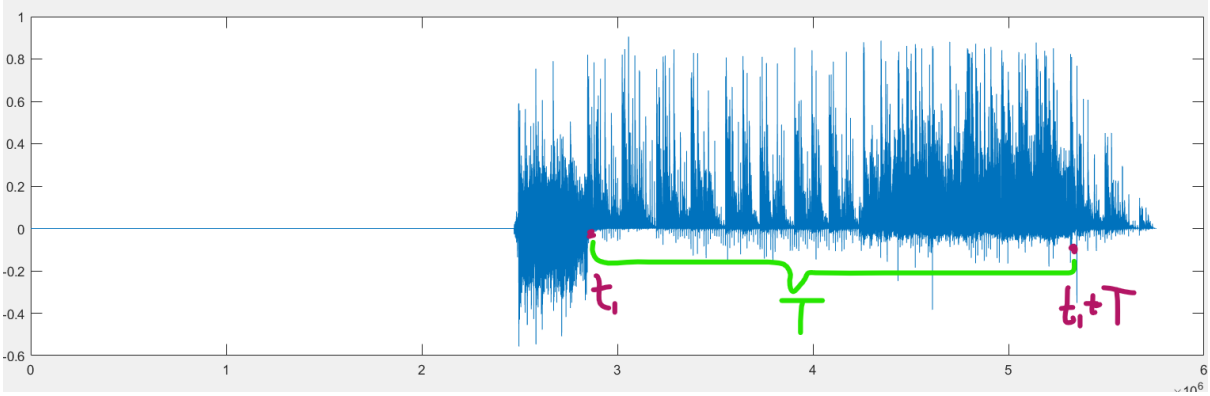

The result of the autocorrelation:




Since we don’t want the first peak (n=0 trivial case), we can look for the maximum point in the interval n>Threshold where Threshold is a lower bound on the loop length (typically 1-5 seconds are enough). Its x-component will be our loop length as seen in the picture above.

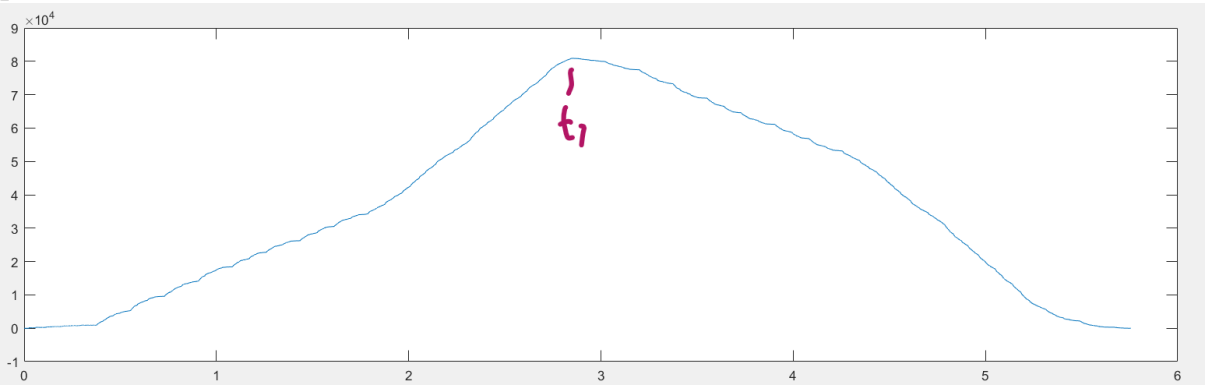
So we found the loop length T, great! Now how do we find the beginning and the end of one of the loops? (Since they are identical it doesn't matter which)

Let's look at the signal f[n]f[n-T] where f is our original audio:

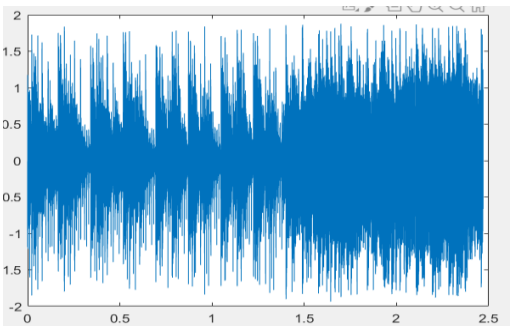



Recall that this is the product between f1 and f2 when the first loop of f2 overlapped with the second loop of f1. As expected, when the two loops overlap their product will yield exceptionally high positive values in the interval where they overlap. Since we fixed f1 and moved f2, this interval will be the location of the second loop in our audio file.

What we can do now is find t1 that maximizes the amount of energy in the interval [t1,t1+T] of the signal f[n]f[n-T]. In other words, find the maximum point of the correlation between f[n]f[n-T] and a unit window of size T:




And we have found our loop interval. we can finally crop the audio signal into the interval [t1,t1+T], export back to MP3 and we're done.



For summary, all we did is:

T=argmax{corr(f[n],f[n])} under the constraint T>Threshold
t1=argmax{corr(f[n]f[n-T],W[n])} where W[n] is unit window of size T.
return the interval [t1,t1+T]

Cross-correlation's time complexity is O(NlogN) because corr(x,y)=conv(x,-y), (convolution takes O(NlogN) time by applying FFT, multiplying in frequency domain, apply IFFT). Finding the maximum is scanning the array in O(N).

All in all we have O(NlogN)+O(N)+ O(NlogN)+O(N)=O(NlogN).

The music used in this example can be found here:
https://downloads.khinsider.com/game-soundtracks/album/mega-man-x6-psx/10_Blizzard%2520Wolfang%2520Stage.mp3

Notes:

//I saw no advantage trying to solve the problem in frequency domain.

//In the matlab file I used 2 algorithms to determine t1, the first is thoroughly explained here. I'll quickly explain the second one: use a mean filter on f[n]f[n-T] and use a step window that acts as an edge detector to find the biggest jump in energy, that gives us t1. Still O(NlogN).

//The matlab files exports 2 files for each algorithm: the loop, and the loop doubled, so you can check how the transition between 2 loops sounds at the middle.