Für die Implementierung habe ich mich für drei Programmiersprachen entschieden: Python, Java und Javascript (in einer React App).
In keiner Implementierung konnte ich bei der Analyse in Blöcken eine Shift Size von 1 nehmen, da die Ausführung dann nicht funktioniert hat (zu lange, zu viel Speicher). Für Vergleichbarkeit habe ich daher in jeder Implementierung eine Blockgröße von 2048 und einen shift von 1024 gewählt.
In Python gebe ich die Hauptfrequenzen zusätzlich aus, was eine Frequenz von 43.07 zurückgibt. ChatGPT hat dazu folgendes zu sagen:

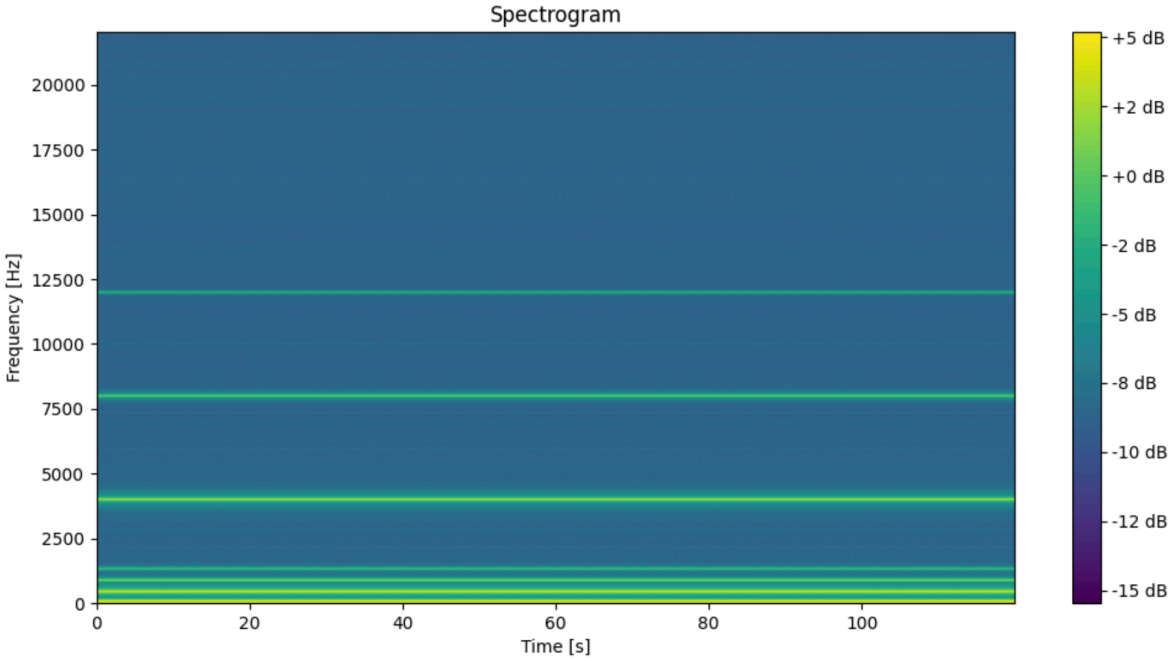
“A sound with a frequency of 43.07 Hz is special because it is associated with an interesting phenomenon called the "Schumann resonance." Schumann resonances are global electromagnetic resonances, generated and excited by lightning discharges in the Earth's atmosphere. They occur in the cavity formed between the Earth's surface and the ionosphere. The fundamental frequency of Schumann resonance is approximately 7.83 Hz, and 43.07 Hz is one of its higher harmonics.

These resonances are named after physicist Winfried Otto Schumann, who predicted them mathematically in 1952. They are significant in various scientific fields, including geophysics, atmospheric science, and even in the study of human health and well-being, as some theories suggest they might influence biological rhythms.”

Die weiteren in der Datei vorhanden Frequenzen scheinen vielfache dieser Hauptfrequenz zu sein.

1. Python

Die Implementierung in Python war die einfachste (für die generative KI). Mit dem in Python standardmäßig enthaltenem Package ‘tracemalloc’ habe ich die Speicherallokation vor und nach der Ausführung der FFT Funktion gemessen und die Differenz bestimmt. Die Erstellung eines Spektograms habe ich für die Analyse auskommentiert, da hier auch viel Speicher alloziert wird. Das Spectrogramm sieht folgendermaßen aus:



Mit den oben genannten Shift und Blockgrößen und ohne Spectrogram benötigt die Fourier Analyse **420413 Byte = 420,413 MB**

2. Java

In Java habe ich die Runtime Klasse verwendet um die Speicherdifferenz zu messen, aber hier kommt vor und nach FFT eine Differenz von 0 Bytes raus. Ich erkläre mit das mit (1) der Tatsache, dass der Aufruf Runtime.getRuntime().freeMemory() eine *Schätzung* des freien Speichers ist (mit Fehler ± 2.5MB) und, dass der Garbage Collector den Speicher nach den Methodenaufrufen (die ja wegen der Blockweisen Analyse oft stattfinden) wieder freigibt. Für ein genaueres Bild habe ich gezählt, wie viele Variablen über alle Aufrufe (für beide Kanäle) erstellt werden (und somit alloziert werden müssen).
Über die FFT Aufrufe mit Blockgröße 2048 und Shift 1024 werden
- 179864622 doubles
- 126969948 integer

Zugewiesen.
Ein primitiver Double ist 8 Bytes und ein primitiver integer 4 Bytes, also entspricht dies
- 179864622*8 = 1438916976 Bytes für die Doubles (= 1438,916976 MB)
- 126969948*4 = 507879792 Bytes für die Integer (= 507,879792 MB)

Dazu kommen pro Kanal zwei float arrays deren länge der Anzahl der Samples in der Audio Datei entspricht:
Das sind 44100 samples/s für 120s = 5292000 floats, für zwei arrays also 10584000 Floats
Ein Float is 4 Bytes also sind dies zusätzlich:
10584000 * 4Byte = 42336000 Bytes = 42,336 MB für einen Kanal
In Summe ergibt dies
1438,916976 MB + 507,879792 MB + 42,336 MB + 42,336 MB **= 2031,468768 MB ≈ 2.03 GB**

3. Javascript

Also dritte Implementierung habe ich eine React App erstellt (einfach weil ich damit vertraut bin und mir debugging so leichter fällt). Die App kann gestartet werden, in dem man in der Console zum Rootverzeichnis (fft_js) und den development server mit “npm start” startet. Dies öffnet ein Browserfenster, indem man eine .wav Datei hochladen und den FFT darauf anwenden kann. In der Browserkonsole wird danach der Speicherverbrauch ausgegeben. Die Analyse verwendet die JS-Native “performance”-klasse mit dem Aufruf p*erformance.memory.usedJSHeapSize*, dies funktioniert aber nur in Google Chrome.
Die Werte, die dabei herauskommen, variieren stark (ich vermute wieder wegen dem JS Garbage Collector, der zu unvorhersehbaren Zeiten Speicher freiräumt), aber bei mehreren Ausführungen der Analyse ist ein oft wiederkehrender Wert **~7000000 Bytes, also ≈ 7 GB** über alle Ausführungen der FFT Funktion.