MDFourier

Artemio Urbina

June 21, 2019

Abstract

MDFourier is an *open source* software solution created to compare *audio signatures* and generate a series of graphs that show how they differ. The software consists of two separate programs, one that generates the signal for recording from the console and the other that analyses and displays the audio comparisons.

The information gathered from the comparison results can be used in a variety of ways: to identify how *audio signatures* vary between *systems*, to detect if the *audio signals* are modified by *audio equipment*, detect if *modifications* resulted in audible changes, to help tune *emulators*, *FPGA* implementations or *mods*, etc.

This document serves as an introduction, a manual and a description of the methods used and how they work. Its intent is to guide new users in making simple comparisons while still providing advanced users with information to perform a more detailed analysis of audio signatures.

Contents

Chapter 1

Introduction

MDFourier is a free and open source[1] software suite for analyzing and comparing *audio signatures*.

It can be used to identify the differences between two *audio recordings*[2]. For instance, a *Sega Genesis Model 1 VA3* and a *Sega Genesis Model 2 VA 1.8*[3] can be compared in order to verify how different they are across the human hearing frequency spectrum using objective and repeatable results[4].

The process is not limited to one specific system[5], and the software can be used to compare any other platform with new configuration files[6].

The intention is not to disparage any particular system, but to help understand and improve them whenever possible by identifying their differences.

A secondary but not less important goal is to create a community driven catalog of audio signatures from these systems, in order to help preservation efforts achieve their objectives.

I believe it would be very useful to have configurations tuned to replicate the audio signatures of vintage retail hardware variations in emulators and *FPGA*[7] implementations. But that doesn't mean there isn't room for improved interpretations; reducing noise while keeping the reference sound signature is one such scenario.

[1] Licensed under *GNU GPL* see appendix P.

[2] Ideally, these recordings are of a set of tests signals generated by software that runs on the targeted system. For more details see section 2

[3] These two models are generally listed as having notorious audio differences [2]

[4] The reference is selected by the user in the form of a control file against which to make comparisons.

[5] The term *system* will be used to cover vintage retail hardware, emulators, FPGA implementations and any other possible variant that executes binaries for the target hardware

[6] At the moment a profile for *Mega Drive/Genesis* is functional and implemented with more to follow.

[7] *Field-programmable gate array*

Although powerful industrial solutions for this kind of analysis must exist, they do not seem to have reached the enthusiast community for this purpose. I'm not aware of any other similar effort to create public software analysis tools[8] that are aimed at vintage console hardware.

1.1 Possible applications

- Figure out if *vintage hardware* retail variations really have different audio signatures when compared against each other.
- Help emulator and FPGA implementation authors with tools to tune *filter profiles* that match *vintage retail* hardware audio signatures.
- Help the modding community to compare how audio signatures change for each *synthesizer* within the targeted system while developing audio modifications.
- Determine if there were changes in the audio *spectrum* after modifications to a console, like *recapping* or changing the audio circuitry.
- Evaluation of equipment, such as *switchers* and *upscalers* with audio *passthrough*. It can help figure out if they have any effect on the signal, by comparing a recording with and without the device connected in the AV[9] chain.
- Help to recreate specific audio signatures.
- Help to tune a system to a particular taste

[8] It must be noted that there have been detailed comparisons made in particular for the *MegaDrive/Sega Genesis*, see post at [2]

[9] Audio/Video

1.2 Disclaimer

MDFourier and this documentation are work in progress. Although I tried to adhere to the best practices known to me, my experience in *digital signal processing* was almost non-existent before this project.

Please contact me if you have corrections, improvements, suggestions or comments. These are encouraged and welcome. Contact information is available in appendix Q.

This project was born out of my curiosity to compare the audio signatures of different revisions of the *Mega Drive/Genesis*, and verify them with a tool assisted analysis. Due to my interest in *game preservation* it naturally grew into its current form, and will hopefully continue to change.

Chapter 2

How *MDFourier* works

The primary thing to keep in mind is what *MDFourier* does. It takes two signals, the first one is designated as the *Reference* file. The second one is the file under scrutiny, and is referred to as *Comparison* file.

The *Reference* file is used as a control. This means that its characteristics are considered the true values to be expected and against which the *Comparison* file will be evaluated. In consequence, all results are relative between the signals.

Having these files, the software analyzes and compares their audio characteristics and generates plots that visually show how they differ.

These files are audio recordings from the desired hardware, preferably captured with a *flat frequency*[1] audio *capture card*[2] and generated by a *custom binary* that runs on the target platform. This *custom binary* is specifically crafted for the particular hardware capabilities and frequency range of the target. Whenever possible, this binary will be part of the *240p Test Suite*[3].

Although the analysis software is *command line* based - in order to be multiplatform and offer it on every operating system that has an *C compiler*[4] - a *GUI*[5] front end for *Microsoft Windows* is provided for simplicity and accessibility[6]. Full Source code can be downloaded from *github* [7].

[1] Flat frequency response refers to the capability of capturing the whole audible spectrum with little or no variation, regardless of frequency. A non flat frequency card can be used to gather relative information, but the captures won't match the ongoing catalogue in order to make further comparisons.

[2] See *Audio Cards* in appendix M

[3] A homebrew software suite for video game consoles developed to help in the evaluation of TVs, upscalers, upscan converters, line doublers and video processing in general.[4]

[4] *ANSI C99 compiler*

[5] *Graphical User Interface*

[6] See appendix B for a guide on how to use the software

[7] See download link [3]

2.1 Workflow

The first step is loading the custom binary to your console. This varies for each platform, from placing the binary in a *flash cart*, burning a *CD-ROM* or using a *custom loader*.

The next step is setting up your *audio capture card* and computer, in order to record a *44.1/48kHz*[8] *16 bit stereo*[9] *WAV*[11] file.

Once the *capture card* is ready and the cables are hooked up from the console to the *capture card*, start recording on the computer and execute the *MDFourier* test from the console.

Wait for the console to show a message indicating you can stop recording, this typically takes less than 1 minute. After you have at least two files: a *reference* - which can be one of the provided[12] files - and a *comparison* file, you are ready to go.

2.2 File alignment

MDFourier takes both files and auto detects the starting and ending point of the recording. These are identified by a series of *8820Hz*[13] pulses in the current *Mega Drive/Genesis* implementation. From these, a *frame rate*[14] is calculated in order to trim each file[15] into the segments that are defined in the *configuration file*[16] for further comparison.

Most importantly, it guarantees that the *Reference* and *Comparison* files are logically aligned, and that each note - or segment - is compared to its corresponding one, with no overlap and without any audio editing or trimming skills required from the user. Current pulse detection accuracy is around 1/4 of a millisecond.

After alignment is accomplished, the software reports the starting and end points of both signals, in seconds and bytes.

[8] *Kilohertz, refers to 1000 Hertz*

[9] At the moment only 16 bit WAV files with *PCM*[10] encoding and at *44.1 or 48kHz* are supported. In the future *FLAC* support could be added to save space and bandwidth. MP3 is not supported since it alters the results in this type of analysis, as shown in section 3.7

[11] *Waveform Audio File Format, also referred to as WAVE*

[12] Available at the web page of the project A

[13] *Hertz is the derived unit of frequency in the International System of Units (SI). It is defined as one cycle per second*

[14] The *frame rate* is the ratio at which the console sends video frames to a display. For more details see appendix F

[15] Since audio cards have their own *sample rate* clocks [12] [13] [14], and some implementations have different *frame rates*, frames are used as the base unit instead of time codes

[16] This file specifies the operating parameters for each hardware configuration. It is described in appendix C

2.3 The heart of the process

A process called *Discrete Fourier Transform*[17] is used in order to analyze and compare the *amplitudes*[18] from each of the corresponding fundamental *frequencies* and *harmonics* - the *spectrum* - that compose the audio signal. The software uses the *FFTW*[19] library in order to accomplish this.

In order to compare the frequencies and amplitudes of each file against each other, a reference point must be set. This is achieved via a relative *normalization*[20] based on the *maximum amplitude* of the *Reference* file. A *local* maximum search is done at the same frame in the *Comparison* signal, and that amplitude is then used to normalize both files.

This is done in the *frequency domain*[21], in order to reduce amplitude imprecision when comparing recordings with different *frame rates*[22]. The software does have the option to do this in the *time domain*[23], but quantization and amplitude imprecision is to be expected if used.

After normalization is finished, the frequencies of each *block*[24] are sorted out by amplitude, from highest to lowest. *MDfourier* can be configured to compare a range of these frequencies - by default 2000 of them are compared for each *block* defined in the *mfn* configuration file[25]. I've found such number to be more than enough, and usually the *minimum significant volume*[26] limits this parameter to an even lower number. In case a more detailed comparison of frequencies is needed, the amount can be incremented via the command line[27].

Sometimes it is helpful to listen to the results of these limiting filters, in order to evaluate if - for instance - 2000 frequencies are either enough or too little for the current application. For this and other purposes I made an extra tool named *MDWave*[28], which creates segmented audio files as internally processed by *MDFourier*, even including the effects of *window*[29] filters and amplitude limits.

[17] Discrete Fourier Transform (DFT) is the variant of *Fourier Transform* applied to discrete values, such as the ones we have in the audio file [1]

[18] The terms *volume* and *amplitude* are used interchangeably during the document, however they are technically not the same thing. *Volume* refers to the perception of *loudness*, and *amplitude* is the *quantifiable* quality of the signal related to its power. As a result, the software can only use amplitude for its calculations.

[19] *The Fastest Fourier Transform in the West* [5]

[20] This process is detailed in appendix E

[21] Basically all analysis from this point and forward takes place after the signal has been decomposed into its fundamental and harmonic sine waves by the Fourier Transform

[22] For more details see appendix F

[23] In other words, normalize based on teh samples and their values. This is detailed in appendix E

[24] A *block* is the basic segment of audio to be compared, such a specific note by a synthesizer as generated by the custom binary.

[25] Described in appendix C)

[26] See section 2.4

[27] See appendix B.1.5

[28] Described in appendix D

[29] Window functions are described in section B.1.1)

After comparing these frequencies between both files, matches are made and the differences in volume are plotted to a graph. Please read chapter 3 for a detailed description of what the various plots created by the program represent.

If you are interested in learning what the *Fourier Transform* does and how its *"magic"* works, there are several resources online. Here are a few of them:

- But what is the Fourier Transform? A visual introduction.
  `https://www.youtube.com/watch?v=spUNpyF58BY`
- An Interactive Introduction to Fourier Transforms.
  `http://www.jezzamon.com/fourier/`
- The Uncertainty Principle and Waves.
  `https://www.youtube.com/watch?v=VwGyqJMPmvE`

2.4 Minimal significant volume

Although the whole frequency spectrum can be compared, there is little practical use in doing so - due to execution time and extra noise from low amplitude harmonics. As a result, rule of thumb defaults are set in order to minimize these issues.

One of these values is a *minimum volume* at which to stop comparing the fundamental frequencies that are found after decomposing the signal with the *Fourier Transform*. This is the *minimal significant volume* and it is the cutoff at which comparisons made by the software are stopped.

This volume is the *amplitude*, and it is measured in *dBFS*[30]. In the *dBFS* scale, the value of *0* is assigned to the maximum possible digital level and it can go down to $-\infty dBFS$.[31]

Currently *MDFourier* can recognize three scenarios to define the *minimum significant volume* to compare the signals, and the three are derived from the first *silence block* in the file.

The first scenario is the *electrical grid* frequency noise, which is searched for at either *60Hz* for *NTSC*[32] or *50Hz* for *PAL*[3334]. This is automatically selected from the previously calculated *frame rate*, if available.

The second one is *refresh rate noise*, again derived from the frame rate, in NTSC it is between *15697-15698Hz*.

If neither is found, which would be surprising for a file generated by recording

[30] *Decibels relative to full scale*
[31] The minimum volume when using *16 bit* samples is *-96dBFS*.
[32] *National Television System Committee, a colour encoding system for analogue television*
[33] *Phase Alternating Line, a colour encoding system for analogue television*
[34] PAL needs to be tested yet, since I don't have console for that video system

from a vintage console via analogue means, the frequency with the highest volume within the *silence block* is used.

Finally, in case none of the previous scenarios is met - or if those values are lower than *-60dBFS* - a default level of *-60dBFS*[35] is used.

[35] This can be changed via *command line* options if needed. See appendix B.1.5

Chapter 3

How to interpret the plots

The main output of the program is a set of different plots that vary in quantity based on the definitions made in the *mfn* file detailed in appendix C, and the selected options[1].

The plot files are saved under the folder *MDFourier* and a sub-folder named after the input WAV file names. They are stored in *PNG*[2] format, currently *1600x800* plots are used, although this can be changed via options.

For the current document *800x400* plots were used in order to fit within a *PDF* or *HTML* presentation. The output plots that are created by the software are listed and described in appendix B.1.4.

In our examples for the *Mega Drive/Genesis*, there are three *active blocks*[3]: *FM*, *PSG* and *Noise*. These will result in a plot of each type, plus a general one named *ALL*.

We'll follow a series of results from different input files to *MDFourier*, starting with cases that have either none or a few differences and build on top of each one, so you can familiarize with what to expect as output.

3.1 Plot Elements

All plots use the horizontal axis for frequency, starting from *1Hz* on the left up to *20kHz* on the right in order to cover the human hearing spectrum[4].

[1] See appendix B

[2] *Portable Network Graphic*

[3] *The Mega Drive/Sega Genesis* has two audio synthesizers: a *Yamaha 2612* (YM2612) for *Frequency Modulated* (FM) audio, and a *Texas Instruments SN76489 Programmable Sound Generator* (PSG) - or equivalent in an *ASIC* - for *Sega Master System* compatibility and extra audio channels.

[4] Although the plotted range can be altered to cover only a fraction of the spectrum, the plot will hold the same scale.

3.2 Scenario 1: Comparing the same file against itself

The first scenario we'll cover is the basic one, the same file against itself. Let's keep in mind that *MDFourier* is designed to show the relative differences between two audio files.

So, what is the expected result of comparing a file to itself? No differences at all. An empty plot file as shown in figure 3.1.

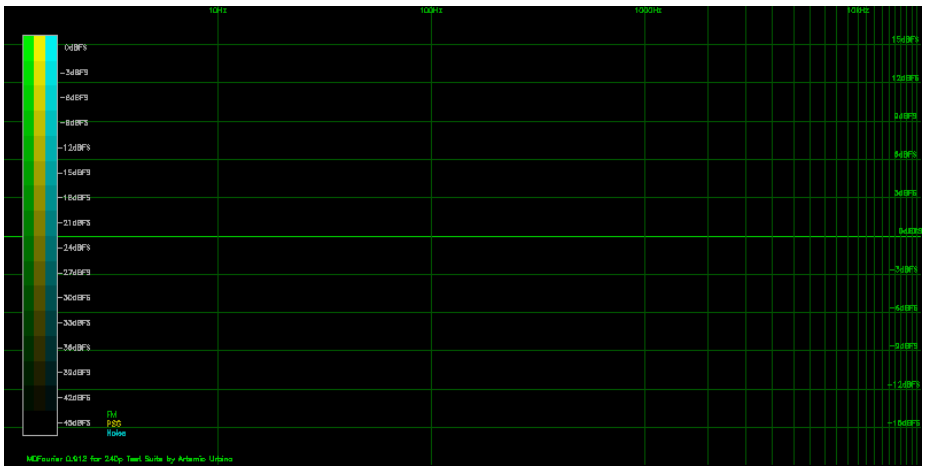

Figure 3.1: Different Amplitudes result file when comparing the same file against itself

Of course all of the *Differences* and *Missing* plots will only have the grid and reference bars, with no plotted information since both input files are identical.

However there will be two sets of *Spectrograms*, one for the *Reference* file and one for the *Comparison* file, with one plot for each defined *type* plus the general one.

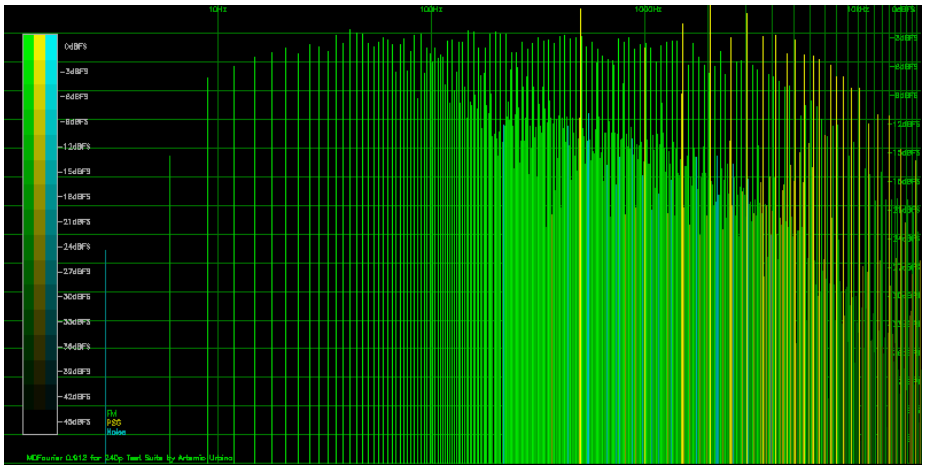

Figure 3.2: The Spectrogram for a Genesis 1 VA3 via headphone out

The Amplitude, or volume, of each of the fundamental *sine waves* that compose the original signal is represented by vertical lines that reach from the bottom to the point that corresponds to the amplitude in dBFS. The line is also colored to represent that amplitude with the scale on the left showing the equivalence.

Three colors - as defined from the *mfn file*[5] - are used to plot the graph, with each one of them plotting the frequencies from each corresponding *type* from the WAV file.

The top of the plot corresponds to the maximum possible amplitude, which is *0dBFS*. The bottom of the plot corresponds to the *minimum significant volume*, as described in section 2.4.

As expected, both sets of *spectrograms* are identical in this case, since we used the same file compared against itself.

3.3 Scenario 2: Comparing two different recordings from the same console

This is another control case. What should we expect to see if we record two consecutive audio files from the same console using the same sound card?

[5]The configuration file is described in appendix C

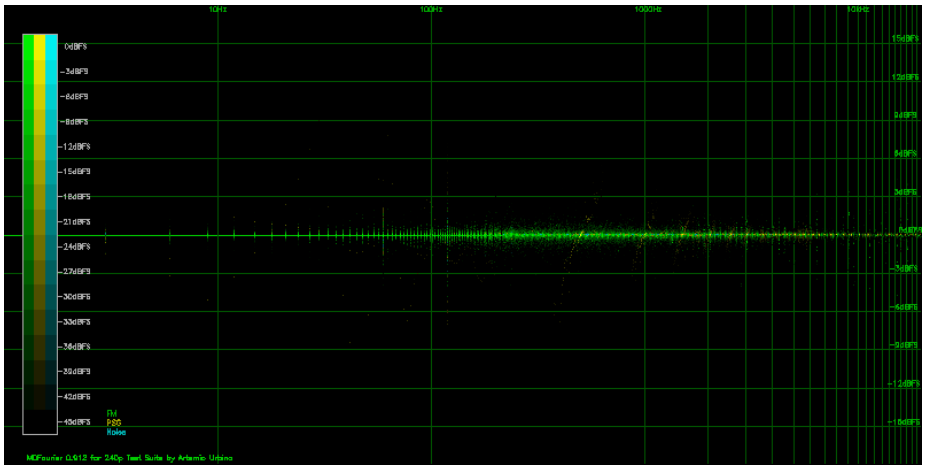

Figure 3.3: The same console, two different recordings

As you can see, we have basically a flat line around zero. This means that there were no meaningful differences found.

But wait, there *are* differences. Why is that? Due to many reasons: analogue recordings are not always identical. There are also variations from the analogue part of console itself, and probably from the internal states and clocks from the digital side of the process. It can also be noise generated by differences in frequency bins when performing the *DFT*[6] after calculating the frame rates - we have that $^1/_4$ alignment error after all.

We now know that there will be certain *fuzziness*, or variation, around each plot due to this subtle recording and performance nuances. It is a normal situation that is to be expected, and a baseline for further results.

3.4 Scenario 3: Comparing against a modified file

For demonstration purposes, the same *Reference* file was modified to add a *500Hz 6dBFS* parametric equalization across all the signal. This is an artificially modified and controlled scenario in order to demonstrate what the plots mean.

[6] *Discrete Fourier Transform*

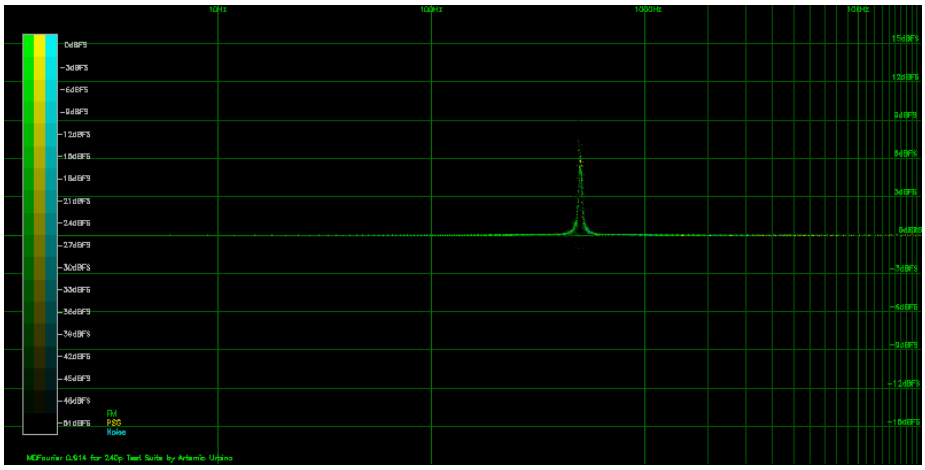

Figure 3.4: Compared against itself modified with a *500Hz 6 db* equalization

As expected all three *types* (*FM*, *PSG* and *Noise*) were affected and show a spike, exactly *6dBFS* tall and centered around *500HzHz*.

It is interesting to note both spectrograms (figures 3.5 and 3.6), since the *500Hz* spike is also shown there.

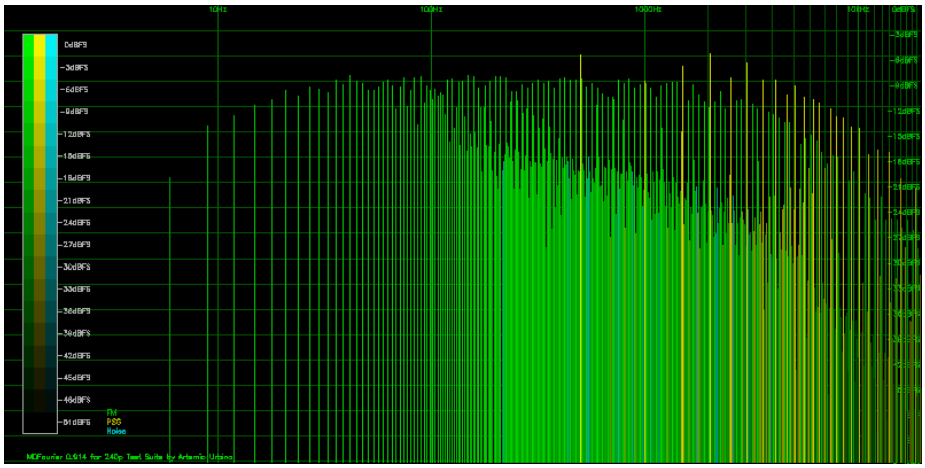

Figure 3.5: *Reference* File

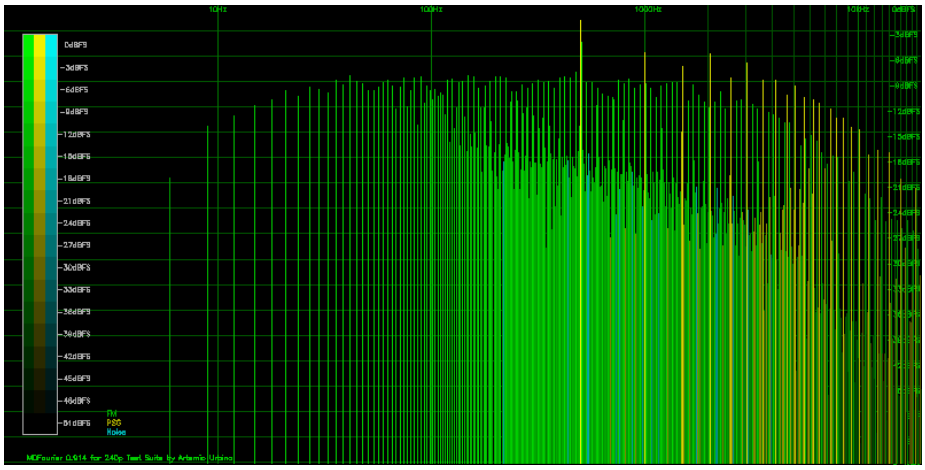

Figure 3.6: Reference file modified with *500Hz* peak

And the *Missing Frequencies* plots are basically empty, since no relevant frequencies are missing from the *Comparison* file.

3.5 Scenario 4: Comparing against digital low pass and high pass filters

We will use the same *Reference* file, and compare it to a file modified by several digital filters that were inserted via an audio editor:

- A *low pass filter*[7] to the *FM* section of the file
- A steeper *low pass filter* at a different cutoff frequency to the *PSG* section
- A *high pass filter*[8] to the *Noise* section at a different frequency

Figure 3.7 shows the general plot for the three types:

[7] A *low pass filter* allows only frequencies lower than a specified value pass through.
[8] A *high pass filter* allows only frequencies higher than a specified value pass through.

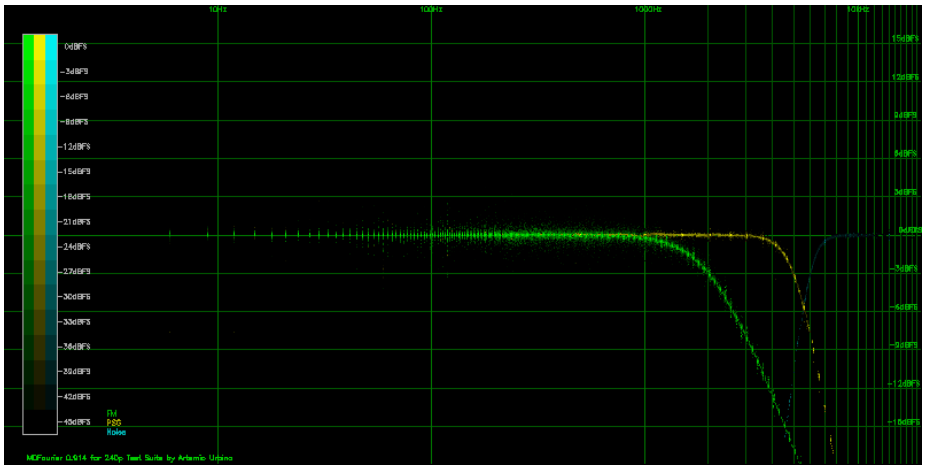

Figure 3.7: *FM* with a low pass filter, *PSG* with a low pass filter and *Noise* with a high pass filter

We can now see that the higher frequencies above *1kHz* in the *FM* plot quickly fall down towards $-\infty dBFS$, so the first low pass filter is there.

The second low pass filter for *PSG* is at *3kHz*, and it is steeper.

But we can barely see what is going on with the *Noise* part of the plot. We can see that there is some black dots on top of the *0dBFS* line.

In order to better see what is going on, we'll change the *color filter function* to $\sqrt{dBFS}$ so we can have higher contrast.[9]

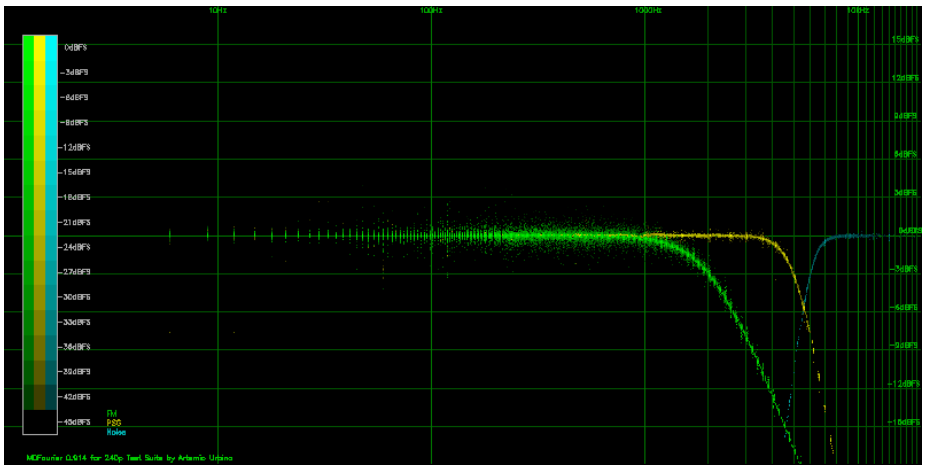

Figure 3.8: Using the $\sqrt{dBFS}$ color filter function

With the higher contrast, we can now make out the curve that raises from

[9]Described in section B.1.2)

$-\infty dBFS$ to *0dBFS*, and it aligns with *8kHz*.

We can still do better than that, by using the *Average Plot* option.[10]

Here is the resulting graph for only the *Noise* section of the signal, with average enabled:

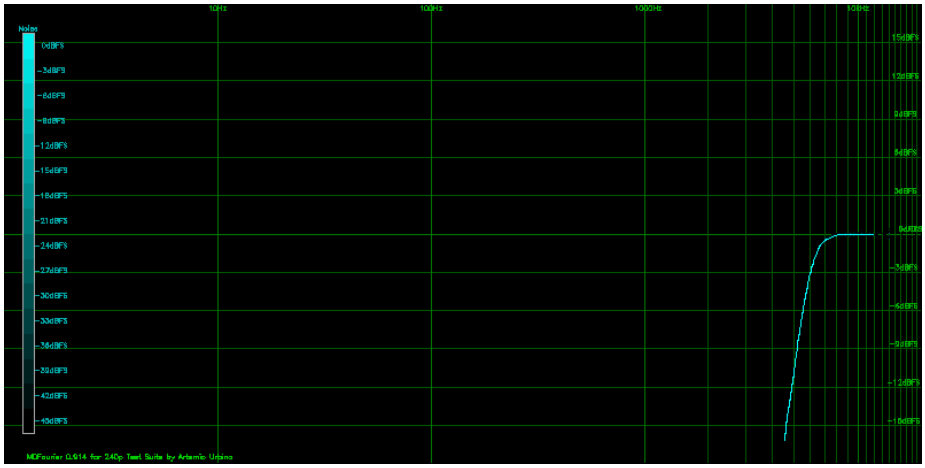

Figure 3.9: *Noise* plot with Average

There are some other interesting plots that result from this experiment. For example, the *Missing* plots now show all the frequencies the *low/high pass* filters cut off.

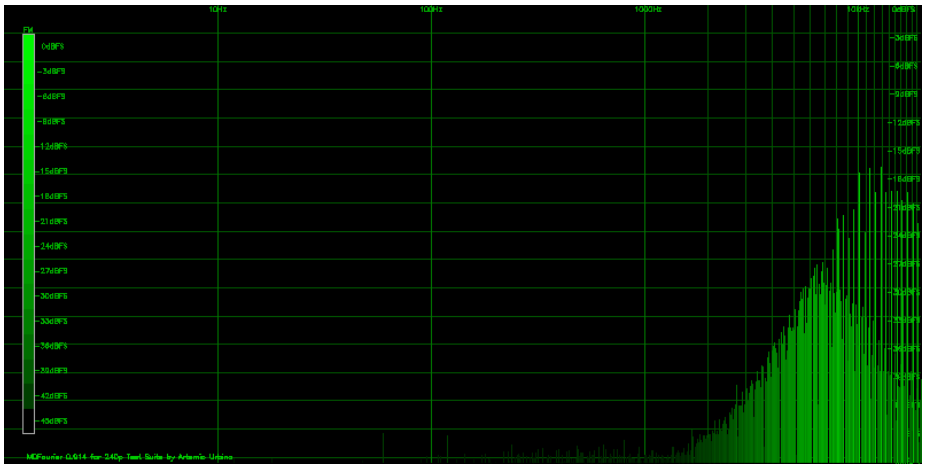

Figure 3.10: Missing frequencies in *FM* cutoff by low pass filter

As show in figure 3.10, there is a curve in the spectrogram and only frequencies above *1kHz* show up, slowly rising in amplitude.

[10] See section B for details

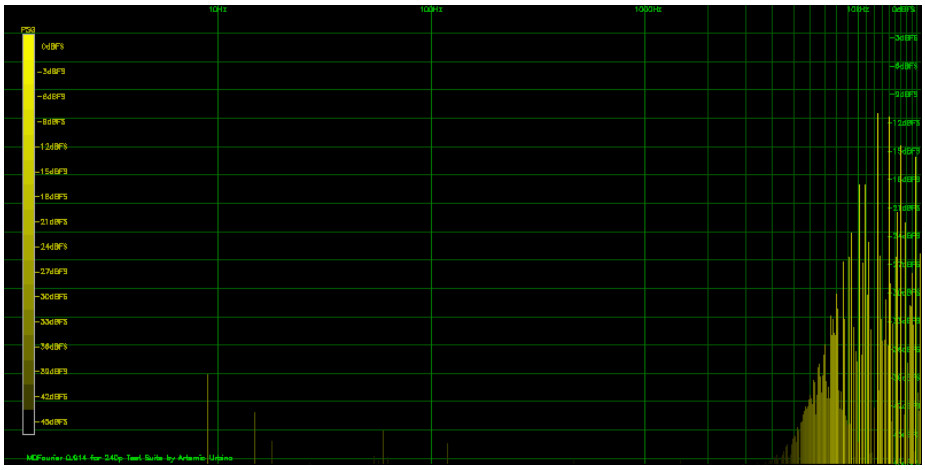

Figure 3.11: Missing frequencies in *PSG* cutoff by low pass filter

The same behavior can be observed in in figure 3.11. This is the *PSG spectrogram*, and it shows a different curve that starts at *4kHz* .

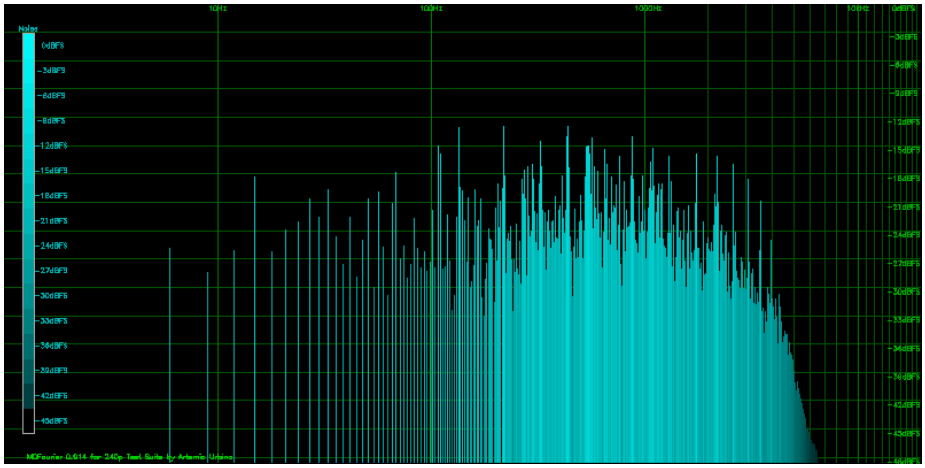

Figure 3.12: Missing frequencies in *Noise* cutoff by high pass filter

And finally, figure 3.12 shows the opposite kind of curve, the *high pass filter* that cuts off everything higher than *8kHz* in the *Noise* section.

It is a good moment to emphasize that these are relative plots. They show how different the *Comparison* signal is to the *Reference* signal. And so far we've compared the same signal to itself, although modified though very precise digital manipulations. An analog filter would look the same, but a bit fuzzier.

However, some interesting ideas arise. What would happen if we take this *low/high pass filter* signal and use it as *Reference* and the original one as *Comparison*?

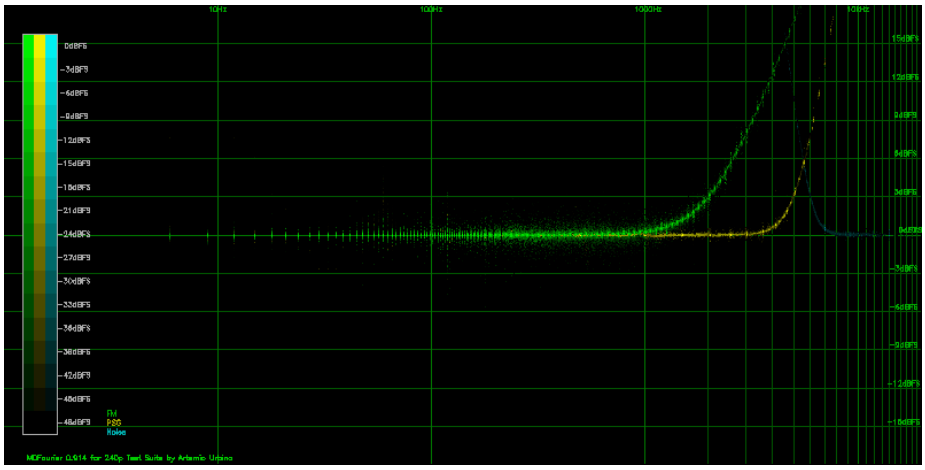

Figure 3.13: Results when using modified signal as *Reference*

Based on this, one could jump to the conclusion that everything will simply be inverted. After all, the original signal now rises to $+\infty dBFS$ at the same spots - and that makes complete sense - since those frequencies now have a higher amplitude.

Although the *Differences* plots will indeed be inverted under these controlled conditions, the *Missing* plots are different. Most of them are now empty:

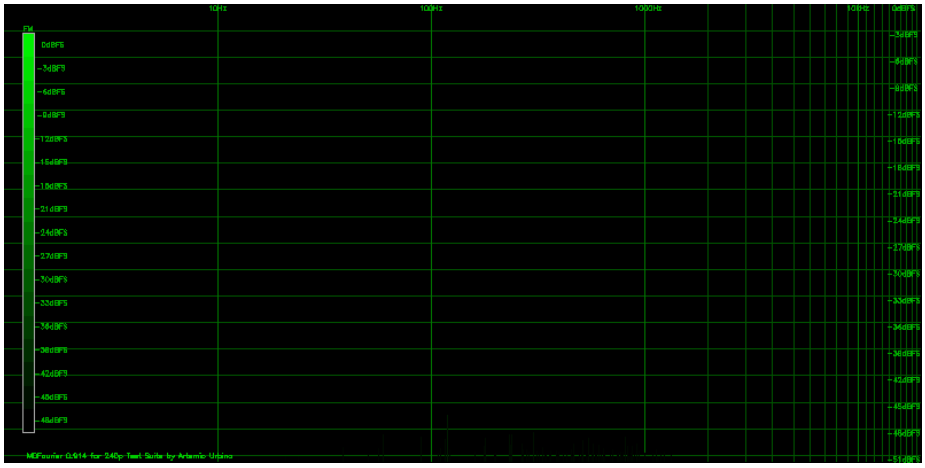

Figure 3.14: *Missing Frequencies* plot for *FM* is empty

This happens because although we cut a lot of frequencies with such steep *low* and *high pass filters*, all the frequency content from this modified signal is present in the original, but not the other way around as shown above.

3.6 Scenario 5: Comparing two recordings from the same console made with different Audio Cards

We'll now compare the same console using two different recordings, one made with an internal *PCI M-Audio 192* and the other with a *USB Lexicon Alpha*. Here are the results:

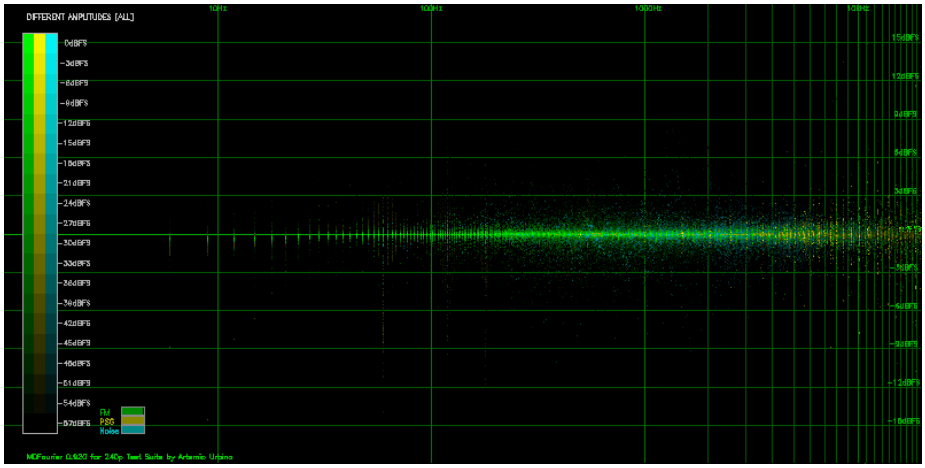

Figure 3.15: Differences using same hardware and cables, different capture cards

Well, I am guessing that was unexpected. We can tell a few things though. First, the frequency response is slightly different, since we now have that *scatter* at the higher end of the spectrum, in what is commonly referred as *treble*.

But we can still clearly tell that the scatter is centered around the *0dBFS* line, which means that even using different sound cards we can tell the differences between systems[11].

Also, there was a slight difference in the detected frame rates. This happens since the sampling clock in not exactly the same in both audio cards. Here is the output text from the analysis that shows the detected frame rates:

```
* Loading 'Reference' audio file A-MD1UTVA3_M-192.wav
- WAV file is PCM 44100hz 16bits and 63.51 seconds long
- Starting sync pulse train:  1.25499s [221380 bytes]
- Trailing sync pulse train:  59.0045s [10408396 bytes]
- Detected 59.914 hz video signal (16.6906ms per frame) from WAV file

* Loading 'Comparison' audio file A-MD1UTVA3-lexicon.wav
- WAV file is PCM 44100hz 16bits and 60.39 seconds long
```

[11] Under the assumption both cards have a relatively flat frequency response, like the ones used here. See the Audio Cards appendix for specifications M.1

```
- Starting sync pulse train:  1.30923s [230948 bytes]
- Trailing sync pulse train:  59.0523s [10416824 bytes]
- Detected 59.9208 hz video signal (16.6887ms per frame) from WAV file
```

The *USB Lexicon alpha* has a more accurate sampling clock, although the difference is minimal[12] and *MDFourier* compensates for such issues.[13]

Here is the graph with the *Average plot* option turned on.

3.7 Scenario 6: Comparing wav vs mp3

The audio file from *Scenario 1* is used as *Reference* and *Comparison* file, but this time encoded as MP3[14] and decompressed to WAV in order to use it with the software.

If the files were identical, an empty plot would result as shown in figure 3.1 from *Scenario 1*. However we get something slightly different:

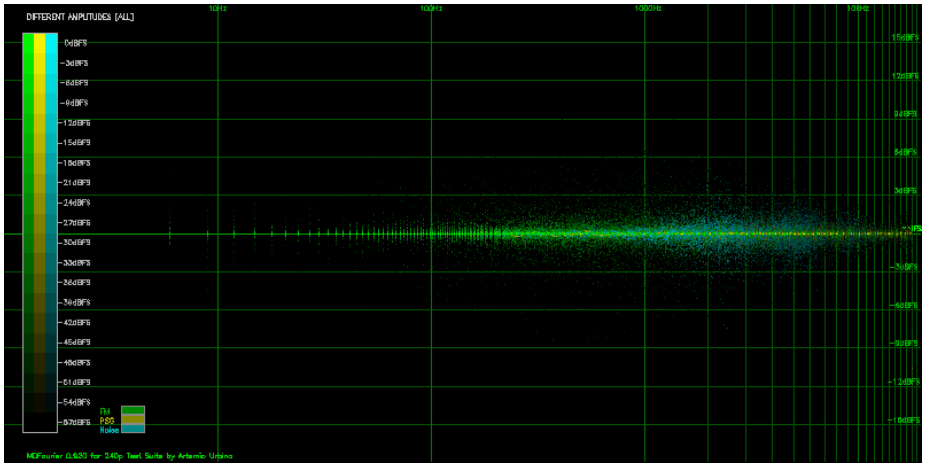

Figure 3.16: Differences between WAV as *reference* and MP3 as *Comparison*

We get everything centered around the *0dBFS* - as it should be - which indicates the *audio signatures* are very similar. Variations are correspondent to what we found in *Scenario 5* while we were comparing the same console with different audio cards. It must be noted that we used high quality compression for the test, and lower settings will provide different results. The reader is invited to experiment with the tools.

[12] We are talking *0.0019ms* in this case, that is 0.0000019 seconds!

[13] The expected frame rate from the vintage console analyzed here is *16.688ms* per frame, or *59.92Hz* as measured with a scope. See appendix F

[14] The file was created with the *MP3 LAME* encoder using *variable bit rate* in high quality mode, with slow encoding enabled and using real stereo.

However in this case some more differences show up is we enable the *Average Plot* option:

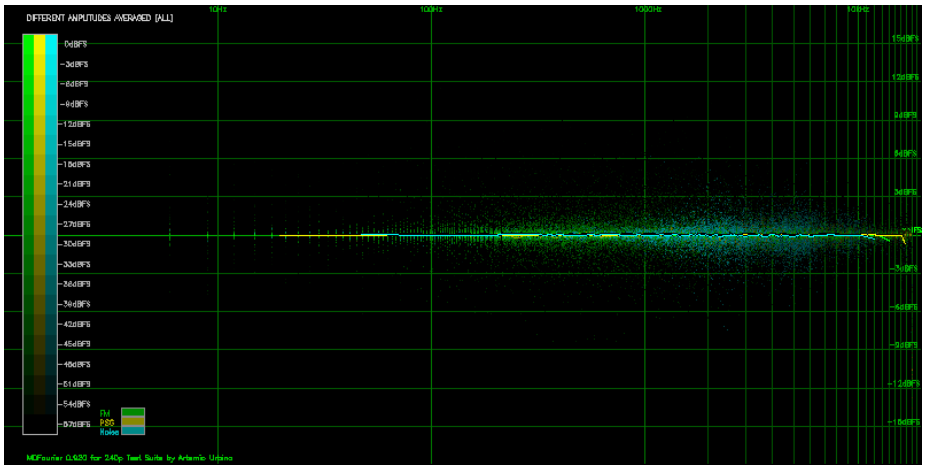

Figure 3.17: Differences between WAV as *reference* and MP3 as *Comparison*, averaged

As it can be appreciated in the rightmost part of the plot, there is a low pass filter around *18kHz*. These frequencies are at the border of acoustic human limits, and most adults won't even be able to hear them. In part this is what MP3 compression banks on for reducing the required storage size of each audio file: cutting off frequencies that human beings are unlikely to hear. This is how the *Fourier Transform* helps to create smaller audio - and video - files.

The other side effect of using *lossy*[15] compression - such as MP3 - is the fuzzy cloud of amplitude differences present in the *500Hz* to *8kHz* range. We are highly tuned to this part of the spectrum, since most of the frequencies that typically compose the human voice reside here[16]. This is most likely what some people perceive as lower quality audio when using these type of files.

Although impressive this is not good enough for pour purpose, since we are trying to measure minute differences between signals. This kind of noise would make it harder to identify the real source of the differences, specially with different encoding settings, audio cards and between consoles.

Another graph that shows the same differences is the *Missing* frequencies plot:

[15] This means it looses some information, as contrasted to lossless.
[16] See [28]

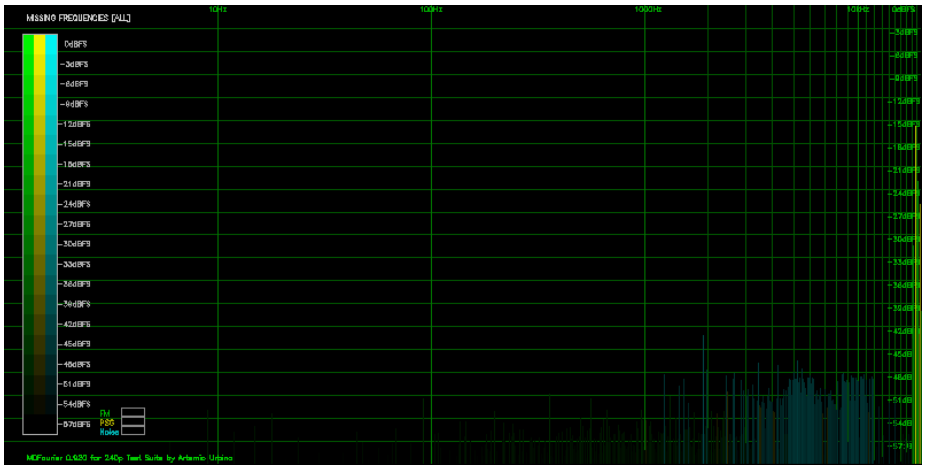

Figure 3.18: Missing frequencies from MP3

It can be appreciated that the missing frequencies were detected after the *18kHz* line, aside form some harmonics below *-45dBFS*.

Chapter 4

Results from vintage retail hardware

The following table lists all the hardware used to make the recordings for the following result plots for this chapter. All systems had stock parts at the time of recording, no modifications and used original power supplies.[1].

The *MDFourier ID* value in the following table refers to the file name in the catalog.

Type	Model	Revision	FCCID	Serial	Region	Made in	MDFourier ID	Recorded from
Model 1	HAA-2510	VA1		89N61751	Japan	Japan	A-MD1JJVA1	Headphone Out
Model 1	1601	VA3	FJ846EUSASEGA	30W59853	USA	Taiwan	A-MD1UTVA3	Headphone Out
Model 1	1601	VA6	FJ8USASEGA	B10120356	USA	Japan	A-MD1UJVA6	Headphone Out
Model 1	1601	VA6	FJ8USASEGA	59006160	USA	Taiwan	A-MD1UTVA6-1	Headphone Out
Model 1	1601	VA6	FJ8USASEGA	31X73999	USA	Taiwan	A-MD1UTVA6-2	Headphone Out
Model 1	HAA-2510	VA6		A10416197	Japan	Japan	A-MD1JJVA6	Headphone Out
Model 2	MK-1631	VA1.8	FJ8MD2SEGA	151014280	USA	China	A-MD2UCVA18	AV Out
Nomad	MK-6100		50059282		USA	Taiwan	A-NMUT	Headphone Out
CDX	MK-4121		Y40 014198		USA	Japan	A-CDXUJ-LO	Line Out
CDX	MK-4121		Y40 014198		USA	Japan	A-CDXUJ-HP	Headphone Out

For simplicity, the recordings shown were made using the *USB Lexicon Alpha* audio card[2]. The first system is the *Reference*, and the second one is the *Comparison* signal.

All the recordings are available from the *downloads page* from the *MDFourier* website[3].

[1] They were all connected to a 4" *CRT* via *RGB*, although the *CRT* was turned off while recording. Since *video refresh* noise is indeed detected by the software, but since it is such an isolated frequency range - at a sharp *20Hz* - it is not filtered out for comparisons.

[2] See appendix M.1

[3] See appendix A

4.1 Sega Genesis Model 1 VA3 (US) vs Mega Drive Model 1 VA1 (JP)

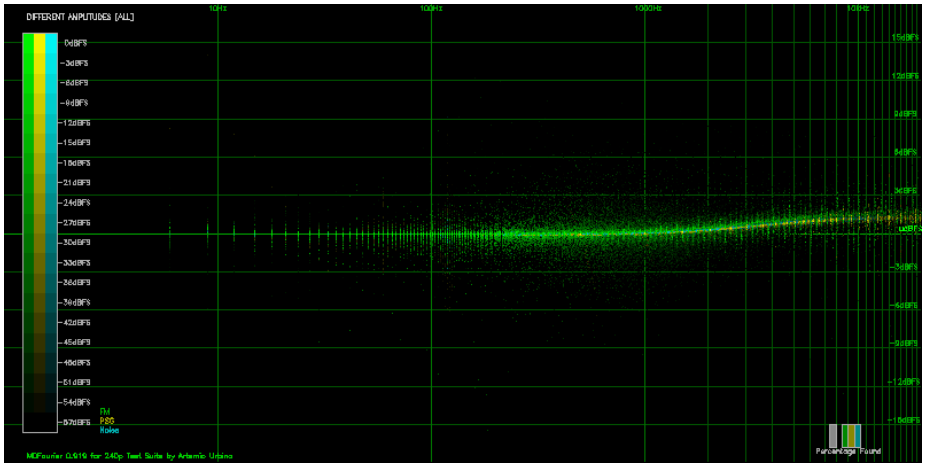

Figure 4.1: Differences between A-MD1UTVA3-LA and A-MD1JJVA1-LA

4.2 Sega Genesis Model 1 VA3 (US) vs Sega Genesis Model 1 VA6 (JP)

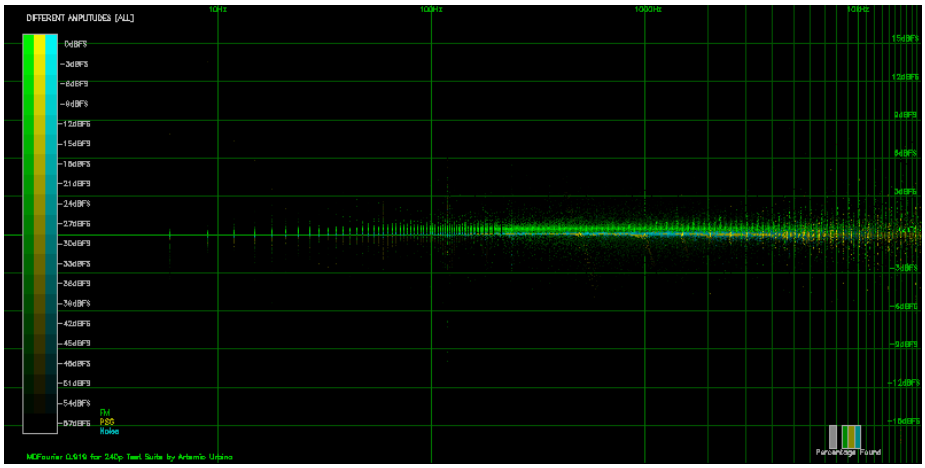

Figure 4.2: Differences between A-MD1UTVA3-LA and MD1JJVA6-LA

4.3 Sega Genesis Model 1 VA6 (JP) vs Mega Drive Model 1 VA6 (US)

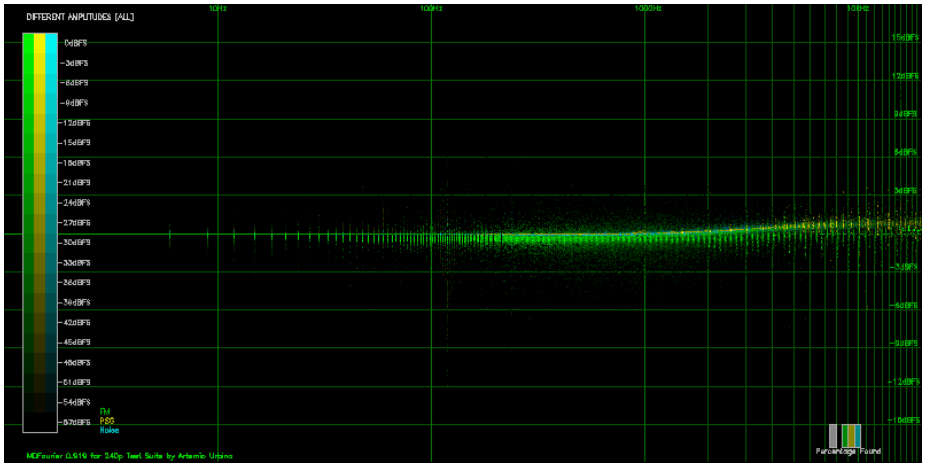

Figure 4.3: Differences between A-MD1JJVA6-LA and A-MD1UJVA6-LA

4.4 Sega Genesis Model 1 VA6 (US) vs Sega Genesis Model 1 VA6 (US)

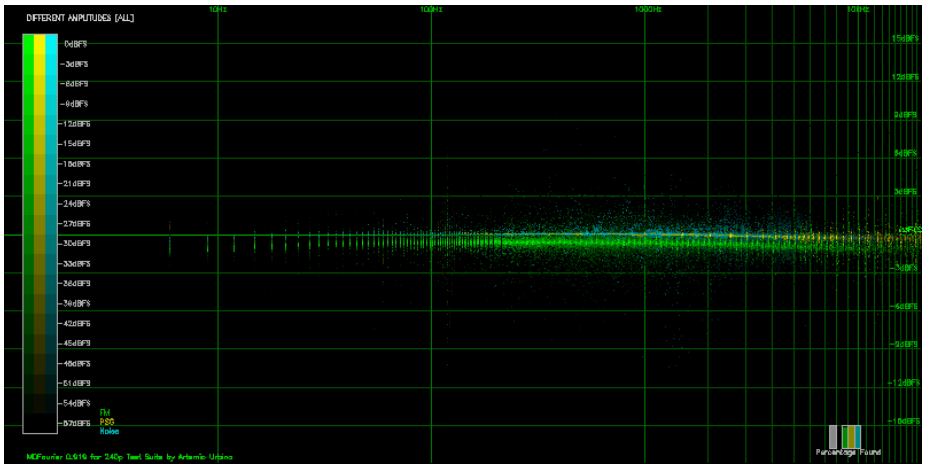

Figure 4.4: Differences between A-MD1UTVA6-1-LA and A-MD1UTVA6-2-LA

4.5 Sega Genesis Model 1 VA3 (US) vs Sega Genesis Model 2 VA1.8 (US)

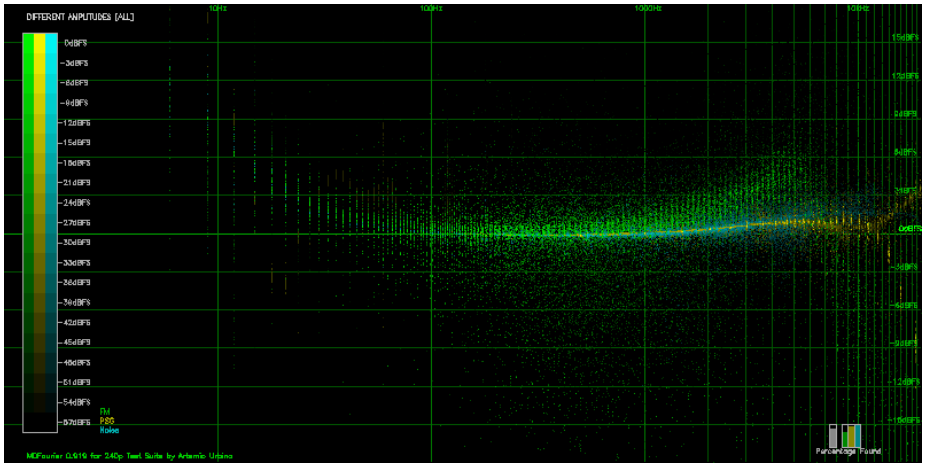

Figure 4.5: Differences between A-MD1UTVA3-LA and A-MD2UCVA18-LA

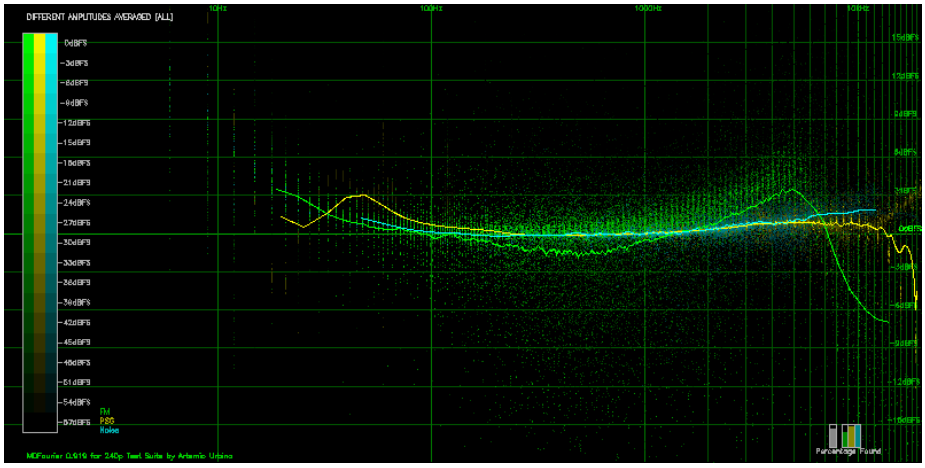

Figure 4.6: Averaged Differences between A-MD1UTVA3-LA and A-MD2UCVA18-LA

4.6 Sega Genesis Model 1 VA3 (US) vs Sega Nomad (US)

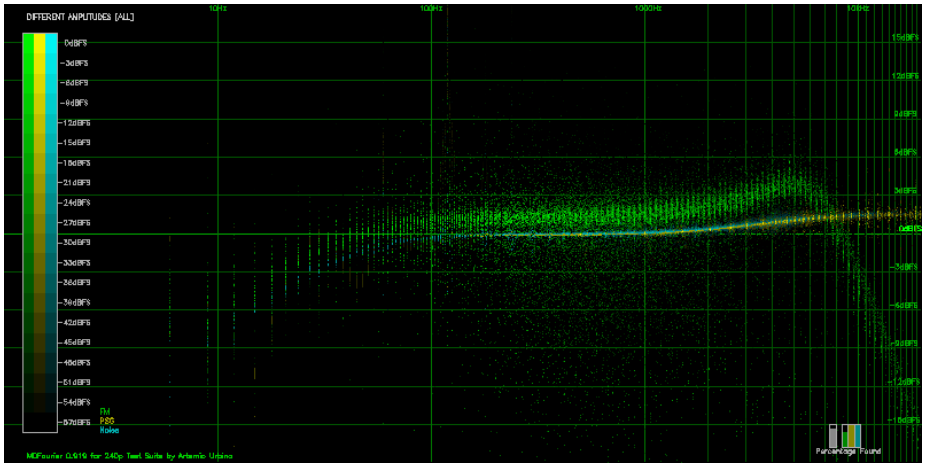

Figure 4.7: Differences between A-MD1UTVA3-LA and A-NMUT-LA

4.7 Sega Genesis Model 1 VA3 (US) vs Sega CDX (US)

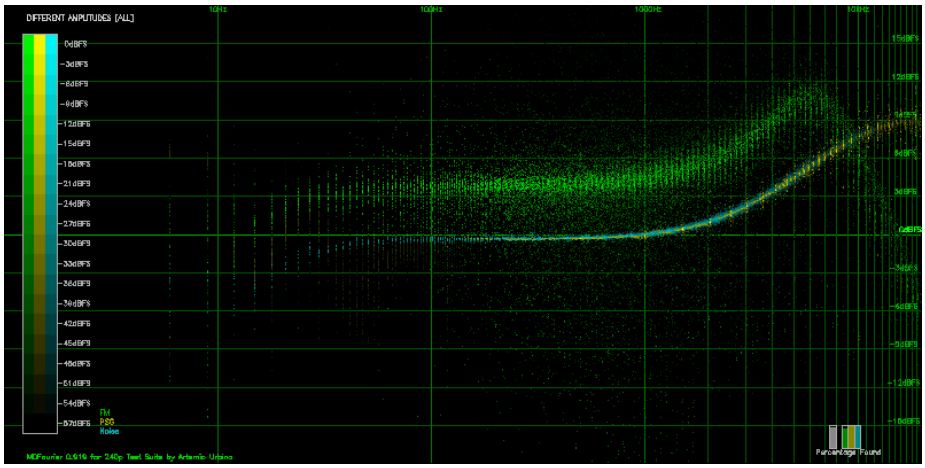

Figure 4.8: Differences between A-MD1UTVA3-LA and A-CDXUJ-LO-LA

4.8 Sega CDX (US) Line-out vs Sega CDX (US) Headphone port

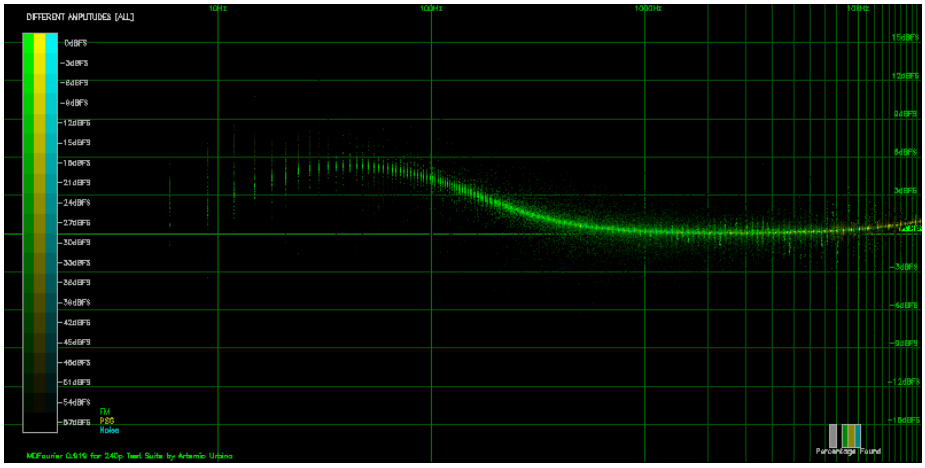

Figure 4.9: Differences between A-CDXUJ-LO-LA and A-CDXUJ-HP-LA

Chapter 5

Future developments

At the time of writing there are still several things to support, including playing *CDDA*[1] from the *Sega CD*.

PC Engine/Turbo Grafx-16 support would be the next target, as well as all systems that are currently supported by the *240p Test Suite*.

[1] *Compact Disc Digital Audio*

Chapter 6

Conclusions

So far *MDFourier* has shown results that are consistent with previous empirical evaluations, and can support many such claims with data. I hope it can provide the community with a process that is repeatable and objective that can be used in any project where someone wants to preserve, compare, replicate or play with the results.

I believe these cases support the idea that this process has value, and that it can be replicated in as many platforms as possible. Audio has been historically relegated to a secondary position, when the audio visual experience is one - as a whole. Games are not the same without their acoustic experience.

This could help to document the original audio signatures of old systems. And of course, to enable the community to play with variations and modifications that are as valid as the ones the systems originally had at launch.

Appendices

Appendix A

Downloads

The project web page is available at `http://junkerhq.net/MDFourier/`, where the latest version of this document is available in *PDF* and *HTML*.

The *Downloads* page contains the following items:

- A pre-compiled *Microsoft Windows* executable of the latest build with GUI front end.
- The custom binaries for supported platforms.
- Updated source code for all tools.
- Sample recordings used for the creation of this document as detailed in chapter 4.

Appendix B

How to use the Front End

The current version of the front end allows access to the main options of *MDFourier*. It is a *Windows* executable and all corresponding files must be placed in the same folder. *Uncompressing* the package to a folder should have all that is necessary to run the program.

After executing *MDFourierGUI.exe* you should be presented with the following interface:

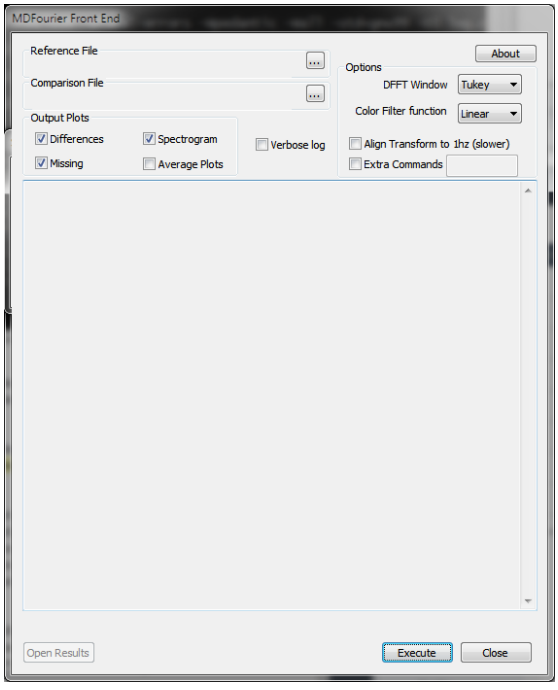

Figure B.1: MDFourier Windows Front End

In order to generate the output plots, two files must be selected to compare them. One as a *Reference* and the other as the *Comparison* file, as detailed in section 2.

The following sequence of steps indicates the typical work flow within the GUI:

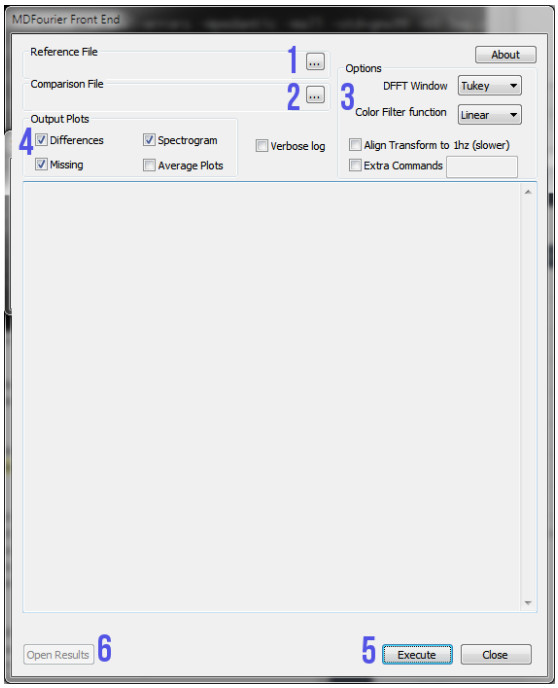

Figure B.2: Typical sequence of steps

1. Select a *Reference* file
2. Select a *Comparison* file
3. Change the default *options* if needed
4. Select the desired output *plots*
5. Execute *MDFourier*
6. When execution ends, open the *results folder*

The *Front End* will display the output text from the command line tool, including any errors or progress as it becomes available.

Keep in mind that the *Open Results* button will only be enabled after a successful comparison between files has finished, and it won't open a second instance of the window if you have one already present.

The default options will generate plots that work on most situations. In some cases fine tuning the results might highlight specific aspects. These options will be described in the following sections.

B.1 Front End Options

The currently available options in the *Front End* are:

B.1.1 Window Functions

In order to reduce *spectral leakage*[1] when applying the DFT, a *filtering window*[2] is applied to each element to be compared between both signals. Since we are generating the signal ourselves from the *custom binary*, the signal can be analyzed as *periodic*[3], and has a natural *attack* and *decay* rate if possible.

By default we use a custom *Tukey window*[4] with very steep slopes. *MD-Fourier* does offer alternate windows as options for further analysis.

B.1.2 Color Filter Functions

Each dot in the *Differences* graph uses the *X axis* for the frequency range and the *Y axis* for the amplitude difference between the *Comparison* and *Reference* signals.[5]

Color intensity of each dot is used to represent the amplitude for that frequency in the *Reference* signal. In other words, how relevant it was to create the original signal in that note. Please refer to chapter 3 in order to see examples of their use.

A color scale is presented in each graph, with the color graduation and the corresponding volume level.

[1] This term describes new frequency components created when applying a Fourier Transform to a segment of a signal. See [16]

[2] All details regarding the windows used, their formulas and graphs are in appendix K

[3] In theory this means we could get away without applying a window, but in practice the signal is not perfect and *windowing* the data helps to eliminate noise.

[4] This is not a common choice for window function, but was selected due to the above reasons after comparing results with the rest of the available options.

[5] See section B.1.4 for output file details

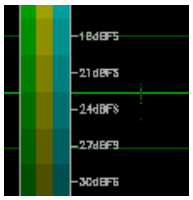

Figure B.3: Detail of color scale in plot

The options are useful to *highlight* or *attenuate* these differences by applying the range to one of the following functions.

They are sorted in descending order. The topmost option will highlight all differences; and the bottom one will attenuate most of them, and show just the ones with highest amplitudes in the *Reference* signal.

All filters, their graphs and effects are listed in appendix L.

B.1.3 Align Transform to 1Hz

When designing the *audio signal* for use during analysis, one consideration is how to balance gathering more information versus the duration of each recording.

For practical reasons, it is desirable to have a short test that will be recorded for analysis. This reduces the time it takes to digitize audio from several systems, the storage used, analysis time by the software, and makes distributing such files easier.

In contrast, when applying the *Fourier transform* a compromise is made between frequency detail and time accuracy, very similar to the Heisenberg's uncertainty principle. If we compare a longer signal for each element, we end up with more frequency information. In our case, we don't care much about time accuracy but we do care about the length of the test. Nobody wants to record a 5 or 10 minute signal for each test to be made.

The compromise made is to use *sub second signals* for each element to be compared. Since the time and sample rate determine how the DFT *frequency bins*[6] are spaced after analysis - and how much information we end up analyzing - the end result is a lower plot resolution.

Zero padding the input signal for Fourier analysis is a controversial subject[7], but for the present application no adverse effects have been found. This might be the case since we control how the source signal is generated, with a predetermined *attack* and *decay*. However, it is disabled by default so the results presented can be free from questioning regarding the effects, and available

[6] The size of each slice in Hertz is the derived unit of frequency in the International System of Units (SI). It is defined as one cycle per second for the frequency spectrum

[7] Zero-padding a signal does not reveal more information about the spectrum. See [17] [18]

for cases where more precise frequency information is needed with no spectral leakage to adjacent *frequency bins*.

B.1.4 Output Plots

Several plots will be generated as a result of the analysis. There will be several plots of each type in the *output folder*[8]. One for each type, and one that summarizes all types in a single plot.

All plots are saved in the *PNG*[9] format, which is lossless and open source. Please read section 3 for examples and a guide to interpret their meaning.

Different Amplitudes

Enabling this option creates the most relevant output plots from *MDFouriuer*. These contain the amplitude difference for the frequencies common to both files across the hearing spectrum using the *Reference* file as control.

If the files are identical, the plot will be a perfect line across the *0dBFS* line. In case the signals differ, a scatter plot will show how it behaves across the typical human hearing spectrum[10].

The file names for these plots have the format:
DA_Reference_vs_Comparison_block_options.png

Missing Frequencies

Plots the frequencies available in the *Reference* file but not found in the *Comparison* file within the significant amplitude range. This is in effect a *Spectrogram*, but limited to the frequencies that were expected to be in the intersection but that were not present in the *Comparison* file.

The file names for these plots have the format:
MIS_Reference_vs_Comparison_block_options.png

Average Plot

This option traces an *average line* on top of the *Difference* plots, making it easier to follow the trend when the output has severely scattered data. It is *off by default*.

[8] It is named *MDFourier* and a subfolder named as the compared files. It can be accessed from the *Front End* with the *Open Results* button.

[9] *Portable Network Graphic* support via *libpng* [6]

[10] Typically defined as *20-20000Hz*, but the higher end tends to be lost with age.

The curve is created by averaging time segments from the frequency data sorted in ascending order. A *Simple Moving Average*[11] is then calculated to smooth out the results.

The curve is weighted according to the *Color Filter Functions* described in section B.1.2, by a repeating each data point by the amount mapped in the *0-1* interval described by the function.

As a result, the average will follow the relative amplitudes from the *Reference* signal proportional to the selected *filter function*.

If there is need for a plot without weighting, please disable the *Color Filtering function*.

The file names for these plots have the format:
DA_AVG_Reference_vs_Comparison_block_options.png

Spectrograms

Plots all the frequencies available in each file[12]. Two sets of spectrograms are generated, one for the *Reference* file and one for the *Comparison* file.

The file names for these plots have the format:
SP_AVG_Reference_vs_Comparison_block_options.png

B.1.5 Extra Command

This checkbox enables the text field to send any extra commands that are not available via the GUI to *MDFourier*.

Current options as of version *0.918* are:

```
MDFourier 0.918 [240p Test Suite Fourier Audio compare tool]
Artemio Urbina 2019 free software under GPL

usage: mdfourier -r reference.wav -c compare.wav
FFT and Analysis options:
-a: select <a>udio channel to compare. 's', 'l' or 'r'
-w: enable <w>indowing. Default is a custom Tukey window.
'n' none, 't' Tukey, 'h' Hann, 'f' FlatTop & 'm' Hamming
-f: Change the number of analyzed frequencies to use from FFTW
-s: Defines <s>tart of the frequency range to compare with FFT
-e: Defines <e>nd of the frequency range to compare with FFT
-i: <i>gnores the silence block noise floor if present
```

[11] This is a type of average that is intended to smooth out sudden peaks in the data. Often used in exchange rate plots. See [27].

[12] As limited by the analysis parameters, such as amplitude and frequency count per block

```
-t: Defines the <t>olerance when comparing amplitudes in dBFS
-z: Uses <z>ero Padding to equal 1 hz FFT bins
-n: <N>ormalize: 't' Time Domain Max, 'f' Frequency Domain Max or 'a' Average
-B: Do not do stereo channel audio <B>alancing
Output options:
-l: <l>og output to file [reference]_vs_[compare].txt
-v: Enable <v>erbose mode, spits all the FFTW results
-g: Create avera<g>e points over the plotted graphs
-A: Do not weight values in <A>veraged Plot (implies -g)
-L: Create 800x400 plots as shown in the manual
-H: Create 1920x1080 plots
-D: Don't create <D>ifferences Plots
-M: Don't create <M>issing Plots
-S: Don't create <S>pectrogram Plots
-d: Max <d>BFS for plots vertically
-k: cloc<k> FFTW operations
-j: (text) Cuts per block information and shows <j>ust total results
-x: (text) Enables e<x>tended log results. Shows a table with all matches
-m: (text) Enables Show all blocks compared with <m>atched frequencies
-h: Shows command line help
```

Sending *-h* in this field will enlist all the currently supported options for your version.

B.1.6 Verbose Log

A log is always created by default when using the *Font End*, however this option enables a verbose version with the whole frequency analysis and many other details dumped to the file.

Useful for reporting errors or unexpected behavior. *Please send the audio files if possible as well!*[13]

[13] Contact details are available in appendix Q.

Appendix C

Configuration file

All these parameters are defined in the file *mdfblocks.mfn*[1]. Here is the current file we are using to compare *Mega Drive/Genesis* audio characteristics:

```
MDFourierAudioBlockFile 1.0
MegaDriveAudio
16.688
8820 -25 25 14 18 10
7
Sync s 1 20 red s
Silence n 1 20 red s
FM 1 96 20 green s
PSG 2 60 20 yellow m
Noise 3 16 20 aqua m
Silence n 1 20 red s
Sync s 1 20 red s
```

This file defines what *MDFourier* must do and how to interpret the WAV files. For now it can read *44.1kHz* and *48kHz* files, in *Stereo PCM* format[2].

The first line is just a header, so that the program knows it is a valid file and in the current format.

The second line is the name of the current configuration, since I plan to add support for different consoles or arcade hardware in the future. This would imply creating a new *mfn* file for each configuration, and a specific binary to be run on the hardware.

The third line is the expected *frame rate*[3]. This is only used as a reference

[1] This is the default file name, a different file can be used and selected via the command line. In future *Front End* revisions it might be selected via a combo box. One for each supported hardware profile

[2] This is more than enough for the human hearing spectrum [15]

[3] For more details regarding frame rate, see appendix F

to estimate the placement for the blocks within the file before calculating the precise *frame rate* of the recording as captured by the *sound card*. After that is calculated, each signal uses its own timing in order to be fully aligned. *Frame rate* variations in the order of *0.001ms*[4] are natural, since we have an error of 1/4 of a millisecond[5] during alignment, and differences also occur by the deviations in *sample rates* and audio card limitations[6].

The fourth line defines the characteristics of the pulse tone used to identify the starting and end points of the signal within the WAV file. Its frequency, relative *amplitude difference* to the *background noise* (silence), and length *intervals* that will be better explained in future revisions of this document.

The fifth line defines how many different blocks are to be identified within the files. There are seven blocks in this case.

Each block is composed of six characteristics: A *Name*, a *type*, the *total number of elements* that compose it, the *duration* for each element specified in frames, the *color*[7] to be used for identifying it when plotting the results and a letter marking if it is a *stereo* or *mono* element[8]. Each block must correspond to a line with these parameters.

For example, FM audio has been named *"FM"*, type *1*, *96* elements of *20* frames each and will be colored in *green*. Definition is in frames since emulators and FPGA implementations tend to run at different frame rates than the vintage retail platform, which result in different durations. The only way to align them, is by respecting the driving force that tied this up in the old days: the video signal.

There are currently two special types, identified by the letters *'s'* and *'n'*. The first one defines a *sync pulse*, which is used to automatically recognize the starting and ending points of the signal inside the WAV file.

The second one is for null audio, or silence. This *silence* is used to measure the *background noise*[9] as recorded by the audio card.

[4] *Millisecond (1/1000 of a second=*

[5] For reference 1/4 of a millisecond corresponds to 0.00025 seconds. Modern systems have different frame rates adapted for modern displays, small differences are more likely caused by the audio capture hardware [12]

[6] See appendix M.1

[7] Available colors are listed in Appendix O

[8] This is important for stereo audio balancing, see appendix G

[9] How analysis is affected by this is described in section 2.4

Appendix D

MDWave

MDWave is a companion command line tool to *MDFourier*. During development and while learning about *DSP*, I needed to check what I was doing in a more tangible way. So, in order to visualize the files in an audio editor and listen to the results *MDWave* was born.

It takes a single WAV file as argument, and loads all the parameters defined in the configuration file in order to verify the same environment[1].

The output is stored under the folder *MDWave*, and a subfolder with the name of the input audio file. The default output is a WAV file named *Used* which has the reconstructed signal from the original file after removing all frequencies that were discarded by the parameters used.

This means that it does a *Fourier Transform*, applies the selected *window*[2] and estimates the noise floor. The highest amplitude frequencies are identifies and limited by range for each element defined in the configuration file, and rest are discarded. An *Inverse Fourier Transform* is applied in order to reconstruct the WAV file and the results are saved.

The opposite can be done as well by specifying the *-x* option, and the result is a *Discarded* WAV file, that has all the audio information that was deemed irrelevant and discarded by the specified options. With this you can listen to these and determine if a more severe comparison is needed.

In addition, the *-c* option creates a WAV file with the chunk that corresponds to each element from the *Reference* file being used, trimmed using the detected frame rate. Two chunks are created for each element, the *Source* WAV chunk has the element trimmed without modification and the *Processed* WAV chunk has the same element but with the windows and frequency trimming applied.

It has a few more command line options, which I'll detail in later revisions

[1] See appendix C
[2] See appendix B.1.1

of the document. You can type *mdwave -h* in your *mdfourier* folder for details.

```
MDWave 0.920 (MDFourier Companion)
[240p Test Suite Fourier Audio compare tool]
Artemio Urbina 2019 free software under GPL

usage: mdwave -r reference.wav
FFT and Analysis options:
-a: select <a>udio channel to compare. 's', 'l' or 'r'
-c: Enable Audio <c>hunk creation, an individual WAV for each block
-w: enable <w>indowing. Default is a custom Tukey window.
'n' none, 't' Tukey, 'h' Hann, 'f' FlatTop & 'm' Hamming
-x: e<x>cludes results to generate discarded frequencies wav file
-i: <i>gnores the silence block noise floor if present
-f: Change the number of <f>requencies to use from FFTW
-s: Defines <s>tart of the frequency range to compare with FFT
-e: Defines <e>nd of the frequency range to compare with FFT
-t: Defines the <t>olerance when comparing amplitudes in dBFS
-z: Uses Zero Padding to equal 1 hz FFT bins
-B: Do not do stereo channel audio <B>alancing
-C: Use <C>omparison framerate profile in 'No-Sync' compare mode
Output options:
-v: Enable <v>erbose mode, spits all the FFTW results
-l: <l>og output to file [reference]_vs_[compare].txt
-k: cloc<k> FFTW operations
-h: Shows command line help
```

Appendix E

Normalization and amplitude matching

Since each signal is probably at its own distinct volume, we need to perform a a *normalization* process in order to have a common compare point between them.

Each type of *normalization* has its own strengths and weaknesses, but the *frequency domain* normalization designed for this process is always accurate with respect o the *Reference* signal, but might be confusing in some corner cases[1] if the underlying causes are not well understood when interpreting the results.

Since silence can't be used as reference[2], the only other option is fixing points from within the signal. Every normalization process used follows a different logic for fixing one such point for comparison.

The default is to do this in the *frequency domain*[3], but there are two other options available via command line[4]. All three methods are described in the following sections.

E.1 Frequency domain normalization

This is the default option used my the software. It involves finding the *highest magnitude*[5] from the *Fourier Transform* of the *Reference* signal before ampli-

[1] One such example is available in appendix I

[2] Noise floor will vary by console and by audio card

[3] This means it is performed after the Discrete Fourier Transform and using the data generated by it.

[4] These can be enabled via the *extra command* option from the GUI as explained in section B.1.5

[5] At this point in the analysis there are no amplitudes defined, since we have no reference points. Hence we don't use amplitude, and raw magnitude values from each transform are

tudes are calculated. Then a corresponding match in the *Comparison* signal's frequency spectrum for the same block is searched for.

This means that the exact same fundamental frequency with the highest magnitude value is searched for, occurring at the same position in time - which corresponds to the block. Having both points, a meaningful *reference point* is set for the comparison, and the relative amplitudes between the signals can be calculated.

In order to calculate the amplitudes, *0dBFS* is matched against the absolute highest magnitude from the *Comparison* file[6], after both signals have been relatively normalized in amplitude against the reference point.

This method has shown to be always accurate within the tests. However, results can be unexpected in certain corner cases as the one shown in appendix I.

E.2 Time domain normalization

The process is very similar to the *frequency domain* variant but it is done using the raw audio samples directly from the WAV file[7].

The highest amplitude is searched for within the samples of the meaningful audio signal, as dictated by the configuration file[8]. The corresponding segment in time is then located in the *Comparison* file, and a pre-defined duration[9] is searched for the sample with the *maximum local* value.

The *Comparison* signal is then absolutely normalized to *0dBFS*[10], and the *Reference* signal is then relatively normalized using the adjusted *local maximum* value. This follows the same rationale described for the *frequency domain* equivalent.

Although the results can sometimes be deceivingly familiar, they are not correctly referenced in corner cases[11], and they do not represent the real relation between the signals. However, they can be useful for analysis while in the process of understanding how to interpret the plots.

---

used.

[6] Since the highest point of the *Reference* signal was matched to one other point in the *Comparison* file, there are only two options: either both peaks are in the exact same spot, or there is a higher peak in the *Comparison* file

[7] Values are changed only in RAM during execution, input files are never modified

[8] Described in section C

[9] One frame in the current implementation

[10] Relative to *0xFFFF* - the top value in *16 bit* samples in the WAV file

[11] See appendix I for an example

E.3 Highest fundamental average normalization

This normalization option also takes place in the *frequency domain*. The idea is to average the highest magnitudes - the fundamentals - from all segments and use the resulting ratio calculated between both signals to normalize them.

The results are always centered around the *0dBFS* line in the *Differences plot*, allowing a globalized view. However, the amplitude differences are not to be relied upon for calibration, since they are not relative to a fixed point from the *Reference* signal. This option is just available for cases when both signals present a high difference rate between them, which is unexpected but possible.

Appendix F

Frame rates and their importance

The *frame rate* is the amount of frames per second a system sends to the display. Since the *custom binary* runs from within the target system, we are subject to the internal timing. Every time a frame starts the process for being sent to the display, an event called *vertical sync* occurs. This is the driving clock for the whole console[1].

It is vital for the process to have a basic unit of time, in order to segment the file in the chunks needed for analysis and to send the correct values to the *Discrete Fourier Transform*.

Signals generated from different sources can have dissimilar *frame rates*, even when running the same programs and representing the same platform. This can be due to several reasons: having a different display technology as target, hardware inconsistencies between revisions, etc.

Another source for variation is the *audio capture device*, since these can vary slightly in their sample rate clocks. But usually these are lower variations that are reported and compensated for internally. It has also been found that some audio cards use a more inaccurate clock when sampling at *44.1kHz* and a very accurate one when using *48kHz*[2].

The frame duration is the basic unit of measurement, since we are dealing with signals that are well below one second. Because of this, the configuration file[3] defines the expected frame duration in milliseconds as measured from a vintage console. One such example is the frame duration as measured from an NTSC *Sega Genesis* using an *oscilloscope*[4] as shown in figure F.1.

[1] As a matter of fact, almost all the code for running games - or the custom binary in out case - is executed during a segment called *vertical blank*.

[2] The internal *M-Audio 192* exhibits this behavior.

[3] See appendix C

[4] Electronic tool for measuring electric signals.

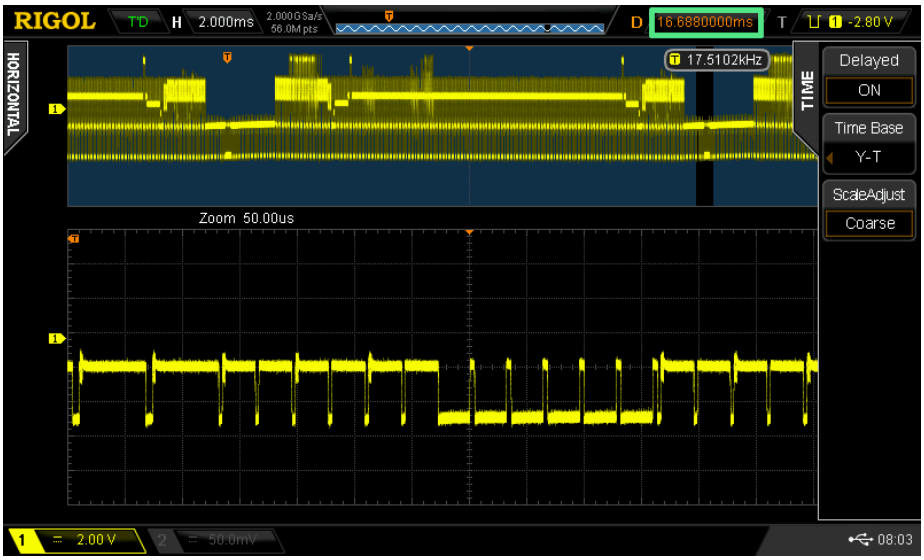

Figure F.1: Frame duration as measured with a scope from a *Model 1 VA3 system*

The oscilloscope shows a frame duration of *16.888ms*, and this is the value used in the configuration file[5] for the initial frame rate calculations.

After the *sync pulses* are detected[6], a new frame rate value is calculated from the audio file. This new frame rate is used across the whole process, and can vary from the vintage console due to a combination of different reasons.

Every audio card can have a slightly different *sample clock*[7], and this variation will affect the starting and ending points within the recording. Such variation can be detected and estimated from the audio signal, and it is reported if the *verbose log*[8] option is enabled.

The selected *sample rate* for recording can also affect the *sample clock* precision even while using the same card[9].

The most important case, and the one for which the *sync pulse* solution was implemented, is when comparing a modern system such as an *emulator* or a FPGA implementation. These usually run at modern *refresh rate*[10] for better compatibility with current display technology, such as *HDTVs*, since the vintage hardware has *refresh rates* that are slightly off-spec[11].

Since *frame rates* differ and the system uses the frame rate as a master clock

[5] Described in appendix C
[6] Described in appendix C
[7] See appendix M.1
[8] See section B.1.6
[9] This is the case when using the *M-Audio 192* [22] in *44100Hz*, but when using *48000Hz* the sample clock is spot on
[10] The refresh rate is the inverse of the frame rate. For example with *16.888* it would be 1/16.888 which is *59.92 frames per second*.
[11] NTSC is *59.97 frames per second*

for program execution, the resulting audio recordings also have a difference in length. For example when using an implementation adjusted for NTSC, there is a *0.001* second difference in each block of 20 frames. Although this seems to be negligible, it adds. In a one minute recording, such as the ones used for *MDFourier*, every note was misaligned due to this difference.

The software discards the extra duration in these cases, adjusting for fractions of a second every time they reach the next sample. Since the notes are not played during the whole 20 frames and the window filters help to compensate the signals, the comparison is perfectly aligned for the tests. The reader can verify these results via the *MDWave*[12] tool.

[12]See appendix D

Appendix G

Stereo Audio Balancing

Sometimes the audio recording has different volume levels between the right and left audio channels, this is referred to as *stereo imbalance*. Whenever possible, *MDFourier* detects and auto compensates this imbalance in order to give a more uniform plot result. The default behavior can be disabled if desired with option *-B*[1].

This should only be a concern if an imbalance in the source system is suspected.

G.1 Possible causes

Two causes for imbalance have been identified. At the time of analysis *MDFourier* cannot possibly discern between them, and both can be present at the same time.

- Amplification deficiencies: The system being recorded has an imbalance due to aging capacitors or other issues in its internal amplification circuitry.
- Sound card balance: Another common source is caused by imbalance while capturing the audio data. This is regularly present in USB audio cards that have individual knobs to adjust gain per channel as shown in figure G.1.

It must be noted that the internal *PCI audio card*[2] used during testing has minimal audio imbalance[3], and this allows detecting *system amplification deficiencies* with consistency.

[1] See appendix B.1.5 for details on how to use *extra commands*.
[2] See appendix M.1 for information regarding *audio cards*
[3] In this case the detected unbalance is 0.0018% in average

Figure G.1: Lexicon Alpha volume knobs

The sound card balance issue can be minimized by adjusting the knobs carefully. *MDFourier* can be useful for compensating this via repeated testing, since any imbalance is reported in the output. It has also been found that leaving gain at zero in some cards, like the *M-Track*, produces more pronounced imbalance than at *50%*. However this requires recording from a source that is known to be balanced.

Figure G.2: M-Track volume knobs at 50%

In order to detect and compensate audio imbalance, at least one *monaural* block must exist within the *mfn*[4] configuration profile being used.

The software assumes that the monaural signal must be the exact same volume between left and right audio channels, and compensates by normalizing the *samples* from one unbalanced channel by adjusting them via the proportional ratio.

[4]See appendix C for details.

Appendix H

Differences due to temperature

During the initial development of the software there was a single instance where unexpected variations were present between different recordings of the same hardware using the same audio capture card as shown in figure H.1.

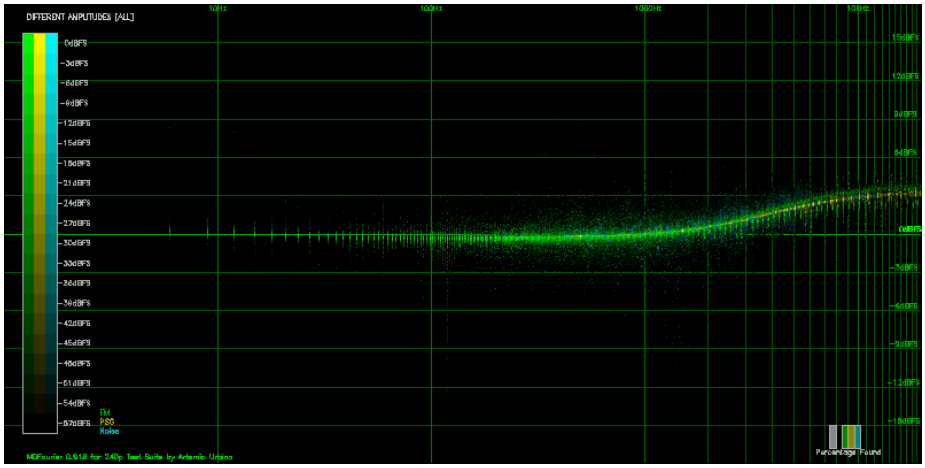

Figure H.1: Audio difference caused by heat in *Model 1 VA3 system*

After analysis, it was found that the difference was present only in the left channel and was caused by a severe difference in temperature while recording. The room was at 29°C[1] as shown in figure H.2, and the *Sega Genesis Model 1 VA3* under testing had been powered on for around 30 minutes.

[1] 84.2°F

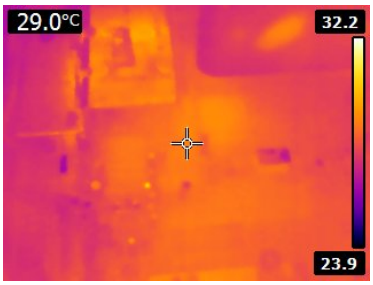

Figure H.2: Console at room temperature *Model 1 VA3 system*

At first warm up differences were suspected, but after some cooling, *cold boot*[2] and measurements using a thermal camera it was determine that the room temperature on top of the warm up was the culprit, since the results could not be replicated when the room had cooled down to 22°C[3].

In order to replicate the results, its vents were obstructed to prevent heat dissipation[4] for a brief period. The results could be replicated at will under these conditions after just a few minutes of use.

It must be noted that the temperature of the headphone amplifier[5] of the console reaches 82.9°C[6] with cover removed as shown in figure H.3, so higher temperatures are expected with it closed. This is important since some of the stock capacitors around the headphone amplifier are radiated due to proximity, and can probably reach temperatures beyond their intended rating in warm climates.

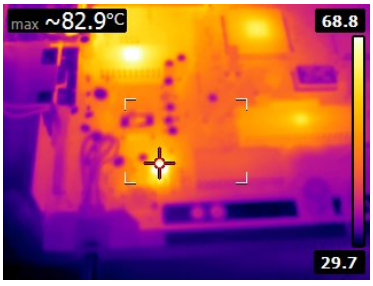

Figure H.3: *Model 1 VA3 system* with amp at 80°C

[2]Powering on the console after it had cool down, and immediately making the recording.
[3]71.6°F
[4]Ventilation were covered with electrical tape
[5]CXA1034
[6]181.22°F

Appendix I

Corner cases

While testing the default *frequency domain* normalization[1], I found that one artificially generated case would produce results that were unexpected and surprising. After analysis, I believe them to be the correct handling of the data, and the appropriate comparison results.

For this example we'll modify a file as detailed in *Scenario 3*[2], but this time a *1zkHz 6dBFS* tone will be added instead of the *500Hz* from that previous case.

I was expecting the result detailed in *Scenario 3*, but was greeted with the plot shown in figure I.1:

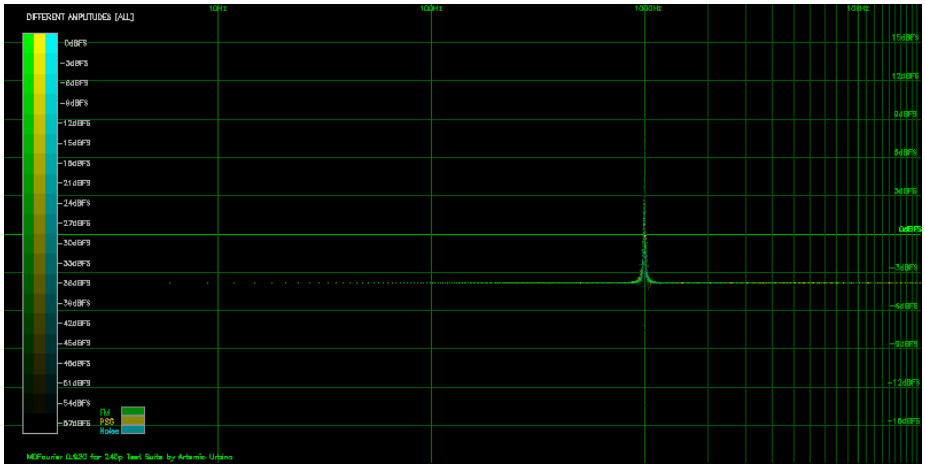

Figure I.1: Signal compared to itself with *1kHz 6dBFS* artificially inserted

I thought it was a bug in the processing of *frequency domain normalization*.

[1] See appendix E
[2] See section 3.4

But after careful analysis, this is the correct result from such process.

It is all due to a big coincidence: the maximum global *amplitude* of the original signal used for this scenario is located at *1005.99Hz*, at just *5.99Hz* from the *1kHz* peak that was artificially inserted. As you might sharply point out, figure I.1 doesn't show the peak *6dBFS* below zero. But that is because it is not centered at *1005.99Hz*.

Here is the result when centering the peak at *1005.99Hz*:

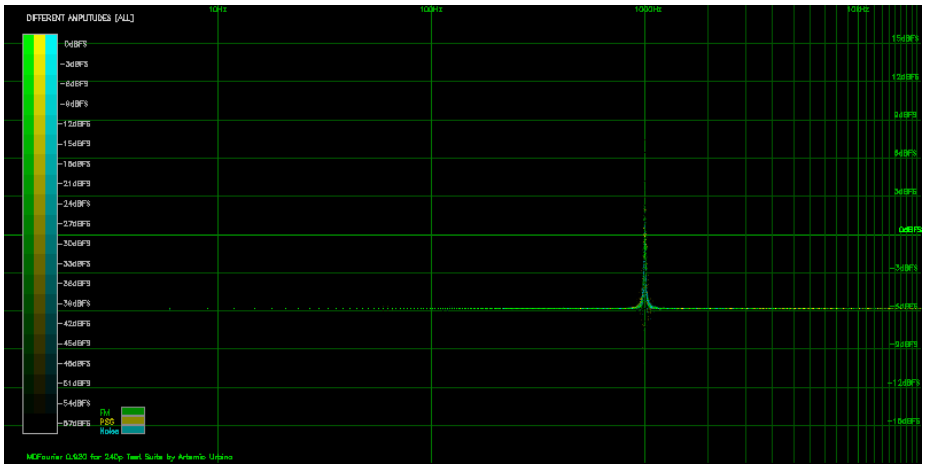

Figure I.2: Signal compared to itself with *1005.99kHz 6dBFS* artificially inserted

Now the majority of the curve is *6dBFS* below zero, and the peak is at zero. But why?

As explained in appendix E, the process of frequency normalization consists in finding the highest amplitude in the signal and use that as a reference point for matching both signals. In this case, both signals have their maximum at *1005.99Hz*, but we also modified the *comparison* file to have a *6dBFS 1005.99Hz* peak. As a result when using the original *reference* file as a model and judging the *comparison* file against it, the rest of the signal has been lowered from this perspective.

This would mean that if two systems had these audio signatures, the resulting plot would tell us that the *comparison* console is - in general - producing the frequencies at lower amplitudes. If the *comparison* system had to be modified to match the reference system, the rest of the frequencies in the signal should be raised by *6dBFS*, since the single point at *1005.99Hz* already matches the reference signal. It is just a matter of perspective. It helps keeping in mind that *MDFourier* does relative comparisons, and in this framework the *reference* file is the absolute model.

In case we'd like to see a plot that would probably be easier to digest - but which I now believe is not as accurate in representing the relative differences - one could enable the *time domain normalization*. The resulting plot would be

the familiar now result from figure I.3:

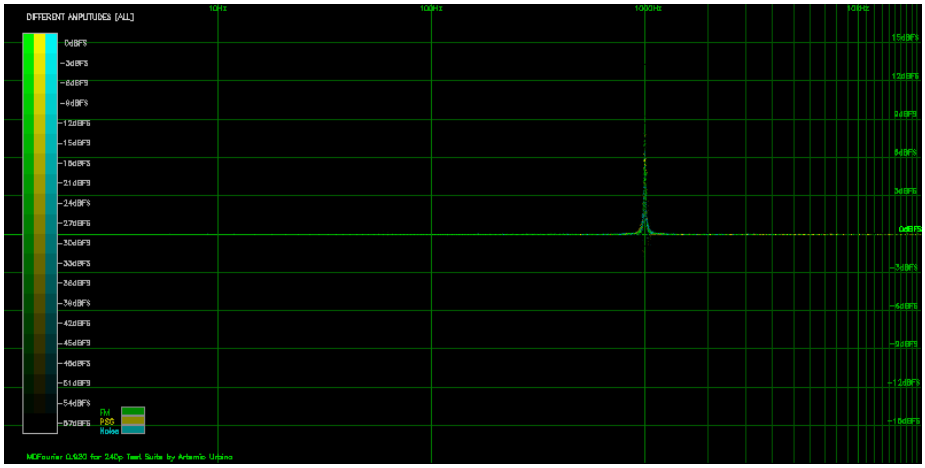

Figure I.3: Signal compared to itself with *1005.99Hz 6dBFS* artificially inserted, but normalized in the time domain.

This plot tells us the exact same thing: if the *comparison* signal is to be modified to match the *reference* signal, the peak at *1005.99Hz* must be lowered by *6dBFS*. Since the results in both cases would be the same, both interpretations are equivalent.

However, I am inclined to adopt the first position, since it has an absolute reference point. Whereas the second plot and interpretation must be modified depending on the input. For this second scenario to work, our framework needs to be modified for special cases - that are highly unlikely to be found in the wild. And since that creates unnecessary inconsistencies and results in imprecision, the *time domain normalization* is not used by default.

Appendix J

Known Issues

I'm positive several deficiencies in my implementation still escape me. However here are a few I'm aware of and have not addressed yet.

- PCM generated by the *FM* chip is not covered in the current *Mega Drive/Sega Genesis* implementation. It is assumed the response will follow the *YM2612* curve.
- A better signal could be generated to cover more or specific cases

Please contact[1] me if you are aware of any details that have eluded me.

[1] See appendix Q

Appendix K

Window Function equations and plots

This appendix lists the equation and curve of each *Window Function* used to limit *spectral leakage* as described in section B.1.1.

K.1 Tukey

The default is a *Tukey* window selected for this purpose. It uses $\alpha = 0.6$, zeroing just a few samples, with minimal *spectral leakage* and good amplitude response.

The following equation is used to create the slopes:

$$tukey(x) = \frac{1}{2}(1 + \cos(\frac{\pi(|x - \frac{N-1}{2}| - \alpha\frac{N-1}{2})}{(1-\alpha)\frac{N-1}{2}})) \tag{K.1}$$

And this is the resulting plot of the *Tukey* window, ranges are 0.0 to 1.0 horizontally and -0.1 to 1.1 vertically.

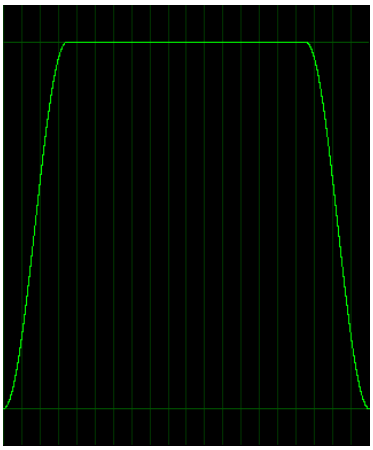

Figure K.1: Tukey window used by *MDFourier*, vertical lines are frames on a 20 frame signal

Detailed information can be found in the reference webpage [21].

K.2 Hann

When selected, a typical *Hann* window is used. This should be used to get the last *spectral leakage*, with a very small trade off in amplitude accuracy.

$$hann[x] = \frac{1}{2}(1 - \cos(\frac{2\pi(x+1)}{n+1})) \tag{K.2}$$

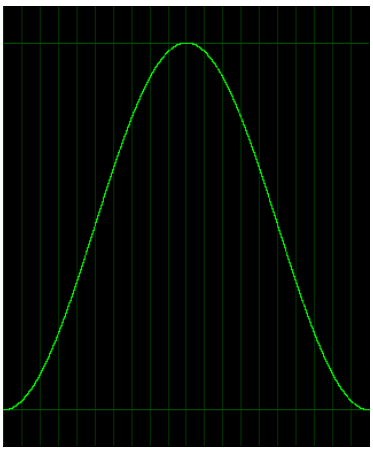

Figure K.2: Hann Window

K.3 Flattop

A typical *Flat top* window is used, selecting this will target amplitude accuracy against frequency bin precision.

$$flattop(x) = 0.21557895 - 0.41663158\cos(2\pi\frac{x}{n-1}) + 0.277263158\cos(4\pi\frac{x}{n-1}) \\ -0.083578947\cos(6\pi\frac{x}{n-1}) + 0.006947368\cos(8\pi\frac{x}{n-1})$$

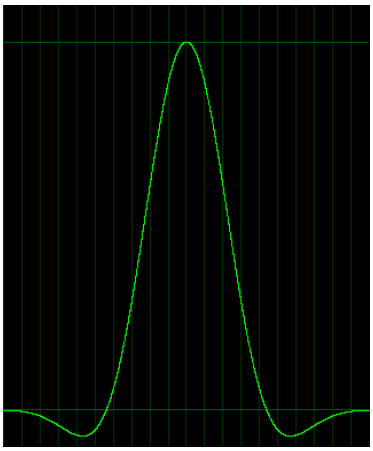

Figure K.3: Flat Top window

K.4 Hamming

A typical *Hamming* window is used, presented for completeness and reference since the samples are never zeroed out.

$$hamming[x] = 0.54 - 0.46\cos(\frac{2\pi x}{n-1}) \tag{K.3}$$

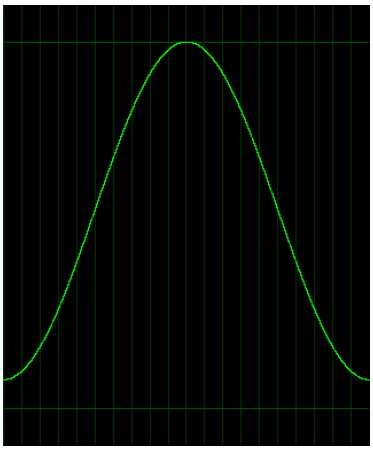

Figure K.4: Hamming window

K.5 No Window

No window is applied to the signal before applying the DFT, equivalent to a *rectangular window*. This leaves the signal unprocessed and any uncontrolled decay and audio card noise will be factored in as part of the periodic signal.

There is more information on windows and their usage in the reference webpage [10].

Appendix L

Color Filter Function details

This appendix contains a description, plot and example of each *Color Filter Function* from section B.1.2.

L.1 None

No filtering is applied, as a result all differences are plotted with the brightest color.

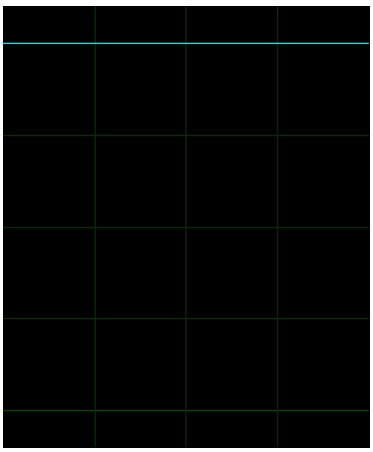

Figure L.1: No Filter

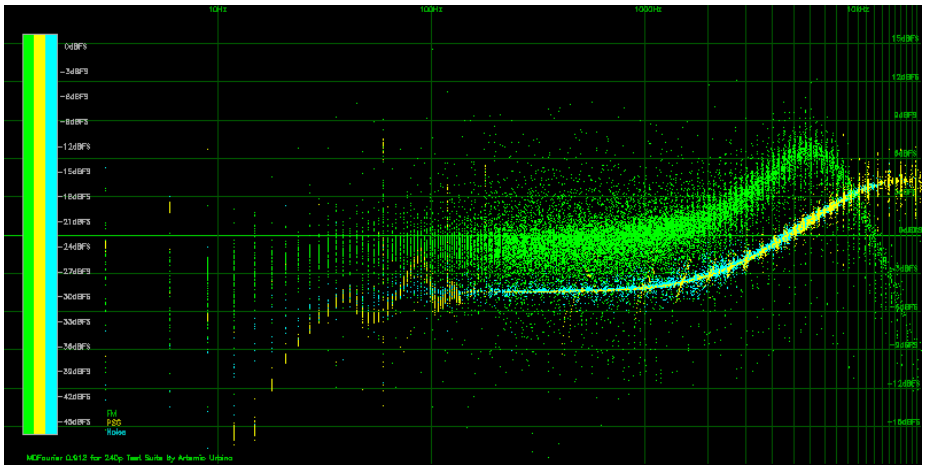

Figure L.2: No Filter Applied

L.2 $\sqrt{dBFS}$

A square root function will only attenuate the lowest amplitude differences.

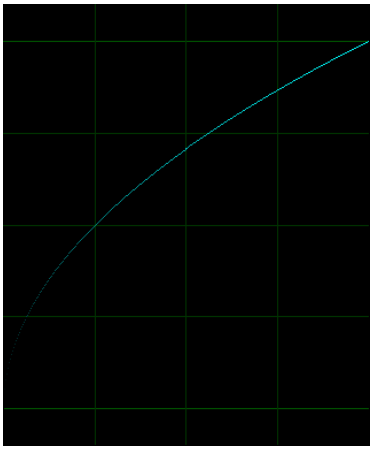

Figure L.3: Square Root filter

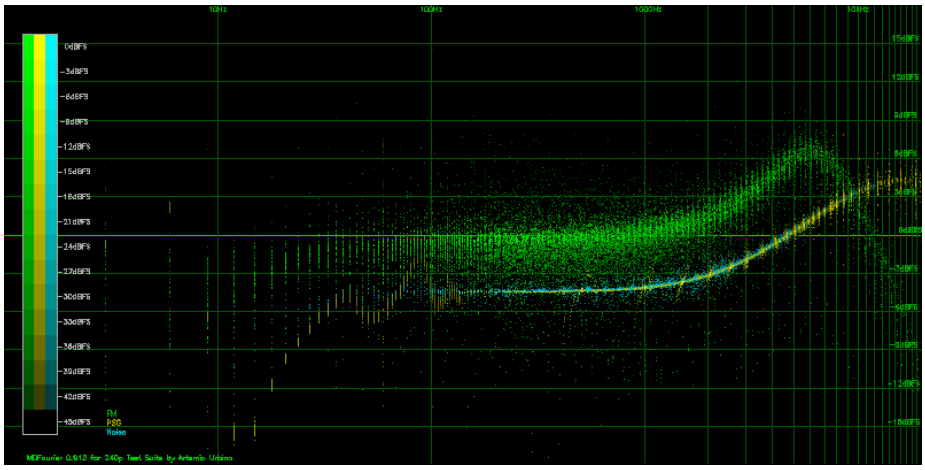

Figure L.4: Square Root filter Applied

L.3 $\beta(3,3)$

A Beta Function filter with parameters (3, 3) will attenuate a bit more from the lower range, still showing most of the differences.

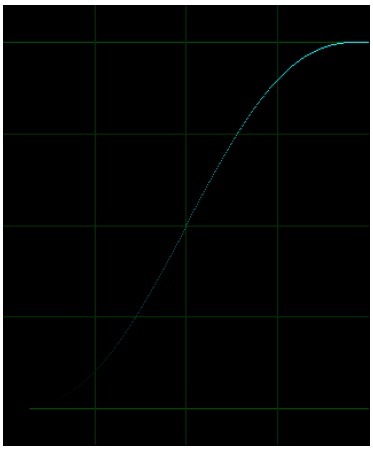

Figure L.5: Beta Function(3,3)

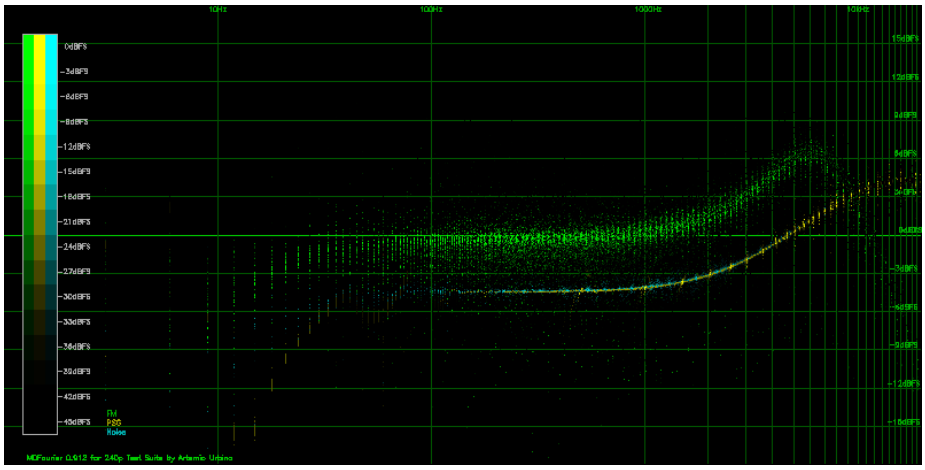

Figure L.6: Beta Function(3,3) Applied

L.4 *Linear*

The linear function is the default, and has no bias. Half the dinamic range corresponds to half the color rage.

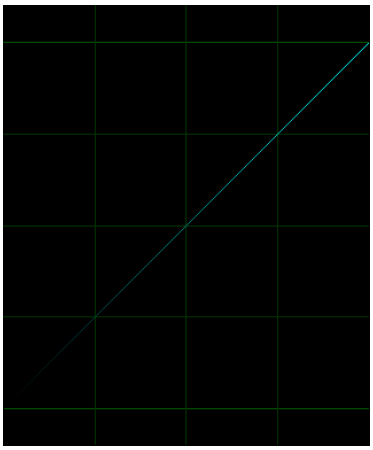

Figure L.7: Linear Function

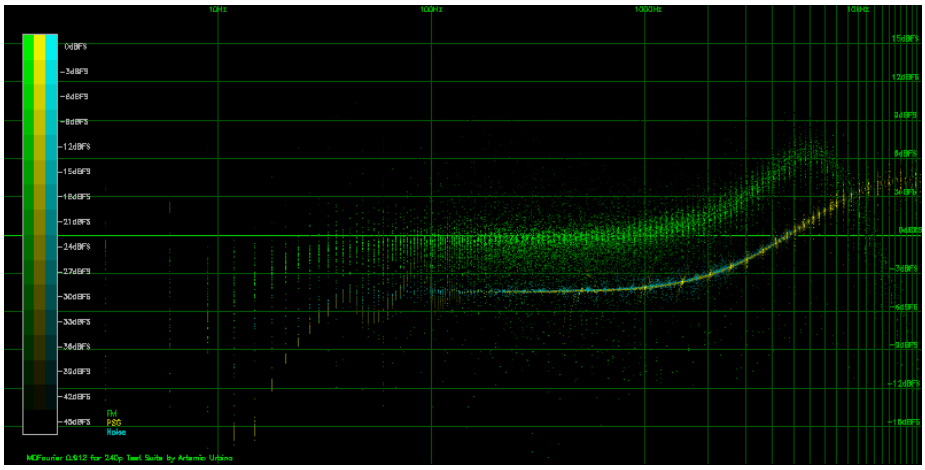

Figure L.8: Linear Function Applied

L.5 $dBFS^2$

A squared function will attenuate a lot more differences, as a result frequencies with the highest amplitude in the reference signal will be brighter.

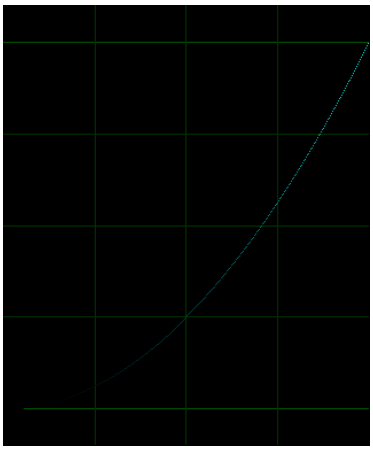

Figure L.9: Linear Function

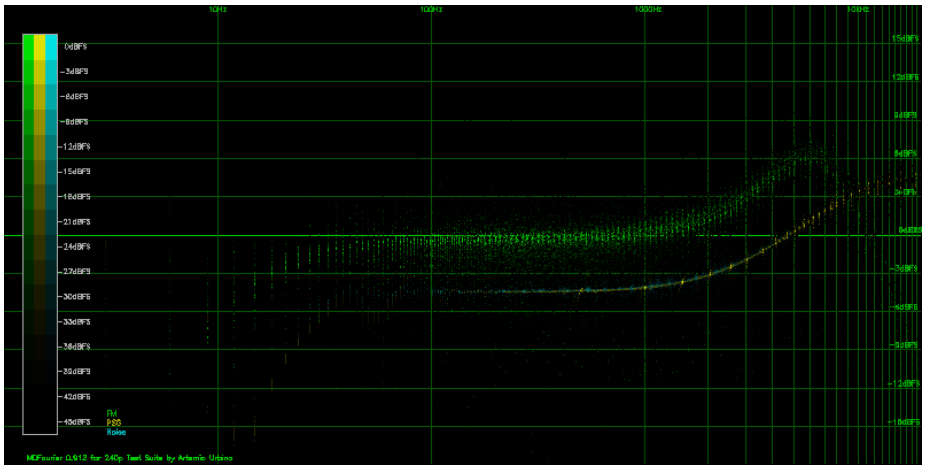

Figure L.10: Linear Function Applied

L.6 $\beta(16, 2)$

A Beta Function filter with parameters (16,2) will attenuate almost all the differences, and only the frequencies with the highest amplitude in the reference signal will be brighter.

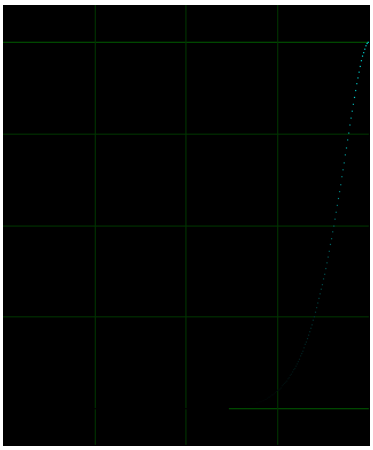

Figure L.11: Beta Function(16,2)

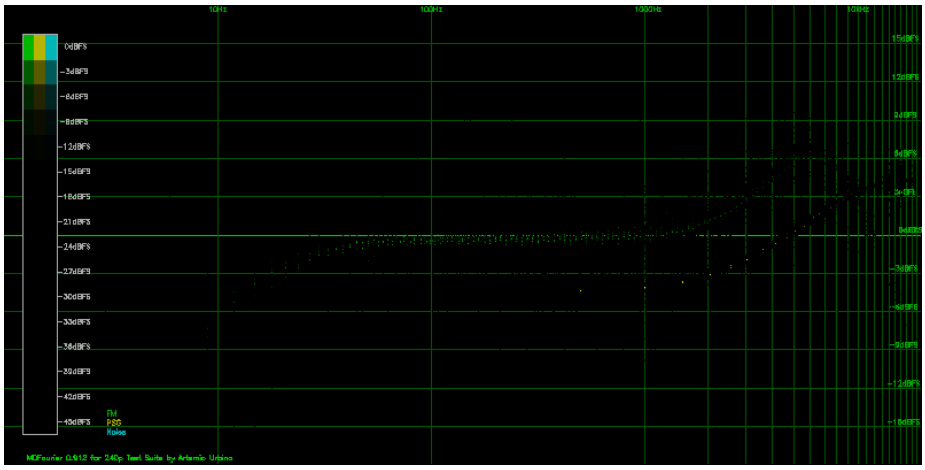

Figure L.12: Beta Function(16,2) Applied

Appendix M

Requirements

M.1 Audio capture device

For capturing the audio files, an *audio capture device* is needed. It is recommended to use a musical grade *audio card* in order to get a *flat frequency* response across the human hearing spectrum. Fortunately, they are not that expensive as they were a few years ago.

So far three audio cards have been used in my personal setup. The reference recordings available for download[1] were made with all three cards, a complete set with each one of them.

I have no association or business relationship with these products, they are just presented as the working examples. As more people use the software and we - as a community - compare files, this list can be expanded with recommendations.

- **M-Audio Audiophile 192:** An internal audio card that is no longer available in the market [22]
- **M-Audio M-Track:** A USB audio card. [24]
- **Lexicon Alpha:** An affordable USB audio card. [23]

We have tested some cards that don't have a flat frequency response, you should try to use a sound card that is aimed to musicians or instrument recording and not gear marketing for podcasts or audio cassette digitization[2].

Sound cards have their own sampling internal clock[3], which tends to deviate

[1] See appendix A

[2] Gear of this type could exist with flat frequency response, current technology should allow any product to offer this basic requirement. But alas, the ones we've tested failed to cover it.

[3] For more details, see [12] [13] [14]

enough that frame rate differences can be detected by *MDFourier*. From my tests, this is more prominent when using the *44.1kHz* sample rate, and minimized when using *48kHz*.

The effect is compensated for while loading the file and during trimming, so it shouldn't be a problem in the frequency domain due to the small variations.

M.2 Computer

Any computer and operation system can be used if you are compiling the source code from scratch[4], but a statically linked *Microsoft Windows executable* is provided for convenience, alongside a front end. See chapter B for instructions on using the GUI.

M.3 Game Consoles or emulators

You'll need either the provided example audio files[5] or to create your own, by recording from your own source. This is probably the desired route, since you will want to compare against your files.

M.4 Flash cart, or means to run the binary

The console needs to run a *custom binary* that is provided with the rest of *MDFourier*. If possible this will be integrated into each version of the *240 test suite* [4].

In order to run these, you'll either need a *flash cart* or a custom loading solution compatible with the target platform.

M.5 Cables and adapters

You'll need cables - and maybe some adapters - to connect the audio output from the console to the input of your *audio capture card*.

[4]See appendix A for download links
[5]See appendix A for download urls.

M.6 Audio capture software

Your *audio capture card* will probably be bundled with some audio editing software, or you can use *Audacity* [25] or *Goldwave* [26] depending on your operating system.

Appendix N

Compiling from source code

N.1 Dependencies

MDFourier needs a few libraries to be compiled. In *Linux*, *UN*X* based systems and *Cygwin* [20]; you can link it against the latest versions of the libraries.

- Fastest Fourier Transform in the West (fftw) [5]
- The GNU plotutils package [7]
- PNG Reference Library: libpng [6]

The following implementations are also used and included with the source files:

- sort.h for tim sort [8]
- Incomplete Beta Function [9]

The pre-compiled binary for *Windows* is created with *MinGW* [19] and statically linked for distribution against these libraries:

- fftw-3.3.8 [5]
- plotutils-2.6 [7]
- libpng-1.5.30 [1]

The *makefiles* to compile either version are provided with the source code [3].

[1] This older version was used to simplify the build process in this statically linked executable. Sources at `https://sourceforge.net/projects/libpng/files/libpng15/1.5.30/`

Appendix O

Colors available for plots

This is the list of colors that can be used in the *MFN configuration file* described in appendix C:

- red
- green
- blue
- yellow
- magenta
- aqua
- orange
- purple

Appendix P

Licensing

Appendix Q

Contact the author

You can contact me via twitter `http://twitter.com/Artemio` or e-mail me at *aurbina@junkerhq.net*.

Appendix R

Acknowledgments

I'd like to thank the following people for helping me so these tools could be completed. First and foremost to my family, who helped me find the time to learn and make progress.

Bibliography

[1] Bracewell, Ronald N. *The Fourier Transform and Its Applications (2 ed.).* McGraw-Hill. ISBN 978-0-07303938-1.

[2] Ace, *GUIDE: A complete overview of the many Sega Genesis/MegaDrive models* http://www.sega-16.com/forum/showthread.php?7796-GUIDE-Telling-apart-good-Genesis-1s-and-Genesis-2s-from-bad-ones

[3] Github, *MDFourier source code C99*, https://github.com/ArtemioUrbina/MDFourier/.

[4] 240p test Suite, *Wiki web page*, http://junkerhq.net/240p/.

[5] Fastest Fourier Transform in the West., *web page*, http://fftw.org/.

[6] libPNG, *web page*, https://libpng.sourceforge.io/

[7] GNU Plot Utils, *web page*, https://www.gnu.org/software/plotutils/

[8] An implementation of a ton of sorting algorithms in C with a user-defined type, *web page*, https://github.com/swenson/sort/

[9] incbeta, *Incomplete Beta Function in C*, https://codeplea.com/incomplete-beta-function-c/.

[10] Window Types: Hanning, Flattop, Uniform, Tukey, and Exponential , *web page*, https://community.plm.automation.siemens.com/t5/Testing-Knowledge-Base/Window-Types-Hanning-Flattop-Uniform-Tukey-and-Exponential/ta-p/445063/.

[11] dB Full Scale, https://www.sweetwater.com/insync/dbfs/

[12] Sound Card Sampling clock variation http://www.stu2.net/wiki/index.php/Calibrate_Sound_Card/

[13] So how accurate is a typical PC sound card? How stable is the output? How would one measure this? Experiments with a PC sound card by leapsecond http://www.leapsecond.com/pages/sound-1pps/

[14] Each sound card has it's own sampling clock, Goldwave support forums http://www.goldwave.ca/forums/viewtopic.php?p=17470

[15] D/A and A/D | Digital Show and Tell (Monty Montgomery @ xiph.org) https://www.youtube.com/watch?v=cIQ9IXSUzuM

[16] Harris, Fredric j. (Jan 1978). *"On the use of Windows for Harmonic Analysis with the Discrete Fourier Transform"*. Proceedings of the IEEE. 66 (1): 51–83. https://web.mit.edu/xiphmont/Public/windows.pdf

[17] Spectral Leakage and Zero-Padding of the Discrete Fourier Transform https://dspillustrations.com/pages/posts/misc/spectral-leakage-zero-padding-and-frequency-resolution.html

[18] Why is it a bad idea to filter by zeroing out FFT bins? https://dsp.stackexchange.com/questions/6220/why-is-it-a-bad-idea-to-filter-by-zeroing-out-fft-bins

[19] MinGW, *Minimalist GNU for Windows*, http://mingw.org/.

[20] Cygwin, *a large collection of GNU and Open Source tools which provide functionality similar to a Linux distribution on Windows.* https://www.cygwin.com/

[21] Tukey window, Recording Blogs https://www.recordingblogs.com/wiki/tukey-window

[22] M-Audio Audiophile 192, *Specifications*, https://www.soundonsound.com/reviews/m-audio-audiophile-192/.

[23] Lexicon Alpha USB card, *Product web page*, https://lexiconpro.com/en/products/alpha/.

[24] M-Audio M-Track USB card, *Product web page*, https://www.m-audio.com/products/view/m-track.

[25] Audacity *web page*, https://www.audacityteam.org/.

[26] Goldwave *product web page*, https://www.goldwave.com/.

[27] Adam Hayes, Simple Moving Average - SMA Definition https://www.investopedia.com/terms/s/sma.asp

[28] Human Speech Spectrum, Frequency Range, Formants, *web site*, http://www.bnoack.com/index.html?http&&&www.bnoack.com/audio/speech-level.html.

Glossary

CDDA Compact Disc Digital Audio. 33

dBFS Decibels relative to full scale. 3, 4, 11, 12, 15–17, 19, 20, 22–24, 26, 41, 49, 50, 58–60, 67, 70

DFT Discrete Fourier Transform. 10, 16, 39, 40, 65

FPGA Field-programmable gate array. 5, 6, 45, 52

GUI Graphical User Interface. 8, 36, 38, 42, 48, 74

Hz Hertz is the derived unit of frequency in the International System of Units (SI). It is defined as one cycle per second. 9, 11, 13, 16–18, 24, 25, 27, 40, 41, 52, 58–60

kHz Kilohertz, refers to 1000 Hertz. 9, 13, 19–21, 25, 26, 44, 51, 58, 59, 74

MP3 MPEG-2 Audio Layer III, a popular coding format for digital audio. 9, 24–26

ms Millisecond (1⁄1000 of a second=. 24, 45, 52

NTSC National Television System Committee, a colour encoding system for analogue television. 11, 51–53

PAL Phase Alternating Line, a colour encoding system for analogue television. 11

PCM Pulse-code modulation, a method used to digitally represent sampled analog signals. 9, 44, 61

PNG Portable Network Graphic. 13, 41, 76

WAV Waveform Audio File Format, also referred to as WAVE. 9, 13, 15, 24, 25, 44–46, 49