MDFourier

Artemio Urbina

May 30, 2019

Contents

Chapter 1

MDFourier Objective

Provide a *Fourier* based analysis framework to compare audio generated by a targeted video game system and its variants, clones and *FPGA* implementations. This is of course not limited to one specific system, and the software can be used to compare any other platform with new configuration files. At the moment a profile for *Mega Drive/Genesis* is functional and implemented with more to follow.

The intention is not to disparage any particular implementation; but to help understand and improve them whenever possible, by identifying the differences. This can help emulators, *FPGA* implementations and even hardware modifications to better match the desired reference profile.

1.1 What is it for?

MDFourier can be used to identify the differences between two audio signatures. These can come from iterations of the console family, like a *Sega Genesis Model 1 VA3* and a *Sega Model 2 VA 1.8*, in order to verify how different they are across the spectrum.

It can also be used to compare a vintage version of the console against any other variation like an emulator or an *FPGA* implementation. It can also be used to compare how the audio changes after modifications, like recapping or changing the audio circuitry.

These results can be used to determine if the signals are indeed different, and how they differ across the human hearing frequency spectrum. Such information can then be used for different means, such as recreating specific audio signatures, tuning to a different taste, etc; based on an objective, repeatable and measurable data set and framework.

It can also be used to evaluate audio equipment, such as switchers and up-

scalers that have passthrough audio, by comparing a recording with and without the equipment in the audio chain.

Although I believe any present and future implementation of a gaming platform should offer a configuration based on vintage retail hardware, that doesn't mean there isn't room for improvement upon those configurations. Reducing noise while keeping the original sound signature is one such case.

This project was born from my curiosity to compare the audio signatures of different revisions of the *Mega Drive/Genesis*, and verify them myself with a tool assisted analysis. I was also curious about *FPGA* and software implementations, and how similar they were to vintage console audio signatures.

1.2 Warning

MDFourier is a work in progress, just as this document. It still has a few rough edges, and although I tried to adhere to the best practices known to me, my expertise in *digital signal processing* was almost non-existent before this project.

If you have suggestions, please contact me 9.3. Any corrections and improvements are encouraged and welcome.

1.3 Licensing

Chapter 2

Requirements

2.1 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.

So far two audio cards have been used in my personal setup. The reference recordings available for download were made with both cards, a set with each one of them.

I have no association or business relationship with these products, they are just presented as the references used. 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 [3]
- **Lexicon Alpha:** An affordable USB audio card. [4]

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.

Sound cards have their own sampling internal clock, which tends to deviate enough that frame rate differences can be detected by *MDFourier*. This is compensated for in the time domain while trimming, and it shouldn't be a problem in the frequency domain due to the small variation. [15]

2.2 Computer

Any computer can be used if you are compiling the source code from scratch [2], but a statically linked *Microsoft Windows executable* is provided for convinience, alongside a front end. See chapter 4 for instructions on using the GUI.

2.3 Game Consoles or emulators

You'll need either the provided example audio files or create your own by recording from the desired source, which makes more sense since you probably want to compare how these behave.

2.4 Flash cart, or means to run the binary

The console needs to run a custom built binary, a ROM. I will build this functionality into each version of the *240 test suite*[5] as possible.

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

2.5 Cables and adapters

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

2.6 Audio capture software

Your capture card will probably be bundled with some audio editing software, or you can use Audacity[6] or Goldwave[7] options depending on your operating system.

Chapter 3

How MDFourier works

The first thing to keep in mind is what *MDFourier* does. It takes two signals, the first one is the *Reference* file and the second one is the *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.

These files are audio recordings from the desired hardware, captured with a flat frequency audio capture card and generated by a custom binary. The program is targeted for the particular hardware capabilities and frequency range. Whenever possible, this binary will be part of the *240p Test Suite*[5].

The analysis software is itself command line based, in order to be multi-platform and offer it on every operating system that has an *ANSI C99 compiler*. However a *GUI* front end for *Microsoft Windows* is provided for simplicity and accessibility. Not all options from the command line tool are currently available via the *GUI*, but the most relevant ones are readily available. Full Source code can be downloaded under the *GNU GPL* from *Github* [2].

3.1 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* pulses in the current *Mega Drive/Genesis* implementation. From these, a frame rate is calculated in order to trim each file into the segments that are defined in the configuration file for further comparison. (see section 3.2)

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

After alignment is accomplished, the software reports both starting and end points of both signals, in seconds and bytes. A specialized tool called *MDWave* is included, that can trim each file and segment each block for acoustical and visual verification if required. At the moment *MDWave* has no GUI Front end available. (see section 7 for more *MDWave* information)

3.2 Configuration file

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

```
MDFourierAudioBlockFile 1.0
MegaDriveAudio
16.6905
8820 -25 25 14 18 10
7
Sync s 1 20 red
Silence n 1 20 red
FM 1 96 20 green
PSG 2 60 20 yellow
Noise 3 14 20 aqua
Silence n 1 20 red
Sync s 1 20 red
```

This file defines what *MDFourier* must do and how to interpret the WAV files. For now it can read *44khz* and *48khz* files, in *Stereo PCM* format.

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 any console 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. This is only used as a reference to estimate the placement for the blocks within the file before calculating the frame rate as captured by the audio capture card. After that is calculated, each file uses its own definition in order to be fully aligned. Variations in the detected frame rate are natural, since we have an error of 1/4 of a millisecond, dictated by the sample rates and audio card limitations. For reference 1/4 of a millisecond corresponds to 0.000125 seconds.

The fourth line defines the characteristics of the pulse tone used to identify the starting and end points of the signal within the wave file. Its frequency, relative amplitude difference to the background noise (silence), and length intervals that will be better explained on a future revision 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 five characteristics: A Name, a *type*, the *total number of elements* that compose it, each element *duration* specified in frames and the *color* to be used for identifying it when plotting the results. Each block mus 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 original platform, which result in different durations. The only way to align them, is by respecting the driving force in hardware like these: 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 start and ending points of the signal within the wave file.

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

3.3 The heart of the process

In order to compare the signals, audio levels are checked and a relative normalization takes place, based on the maximum amplitude of the *Reference* file. A local maximum search is done in the same frame on the *Comparison* signal, and that amplitude is then used to normalize both files. This is done in the frequency domain, in order to reduce amplitude imprecision caused when comparing recordings with different frame rates. The software does have the option to do this in the time domain, but quantization and amplitude imprecision is to be expected in such case.

Then a *Discrete Fourier Transform*[1] is used in order to analyze the frequency content of the signal, as well as the amplitudes from each of the corresponding fundamental frequencies that compose the signal.

The software uses the *FFTW*[8] library in order to accomplish this, and then proceeds to sort out the frequencies of each block by amplitude. It can be configured to compare a range of these frequencies, but by default it compares 2000 of them for each element defined in the *mfn* configuration file (see section 3.2).

I've found such comparison to be more than enough, and the *minimun significant volume* (section 3.4) even limits these 2000 to a lower number, based on significant amplitude. In case more frequencies are needed, this can be changed via the command line parameters.

After comparing these frequencies between both files, matches are made and the differences in volume are plotted to a graph. Please read the corresponding

chapter to help you understand the results. (see chapter 5)

Sometimes it is helpful to listen to the results of these filters in order to evaluate if 2000 frequencies are enough or too little for the current application. For this and other purposes I made an extra tool named *MDWave* (chapter 7), which creates the segmented wave files as processed with the *Fourier Transform*, *window filters* included (section 4.2).

If you are interested in learning what the *Fourier Transform* does and how it works it's magic, there are several resources online to help you out. Here are a few:

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

3.4 Minimal significant volume

Currently *MDFourier* can recognize three scenarios used as a 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 grid power frequency noise, which currently searches for *60/50hz* noise (*PAL* needs to be tested yet, since I don't have a PAL console and its use is triggered based on the identified frame rate). The second one is refresh rate noise, again derived from the frame rate, and around *15697-15698hz*.

In case neither is found, which would be surprising for a file generated by recording from a real console via analogue means, the frequency with the highest volume within the silence block is used.

In case neither of the three scenarios is met, a default $-96bDBFs$ level is used.

3.5 Workflow

The first step is loading the custom binary to your console, this varies from a flash cart, burning a CD or using a custom loader.

The next step is seting up your audio card and computer, in order to record a *44khz 16 bit stereo* wav 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 start 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 files, and a comparison file you are ready to go.

Chapter 4

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 programs.

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

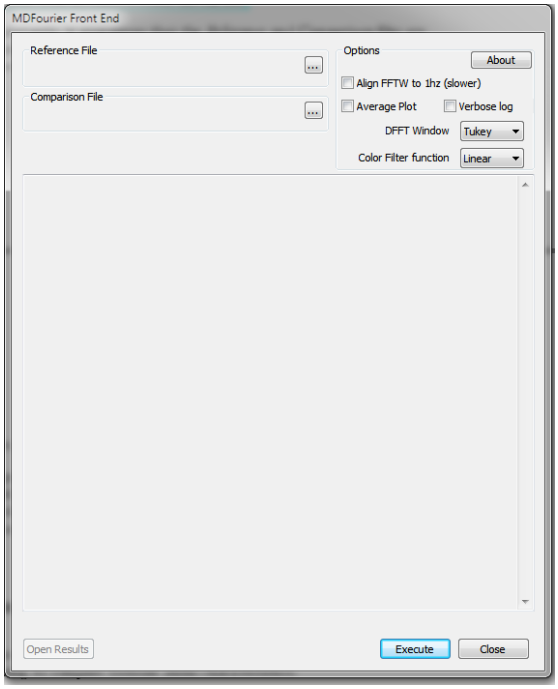

Figure 4.1: MDFourier Windows Front End

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 3.

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

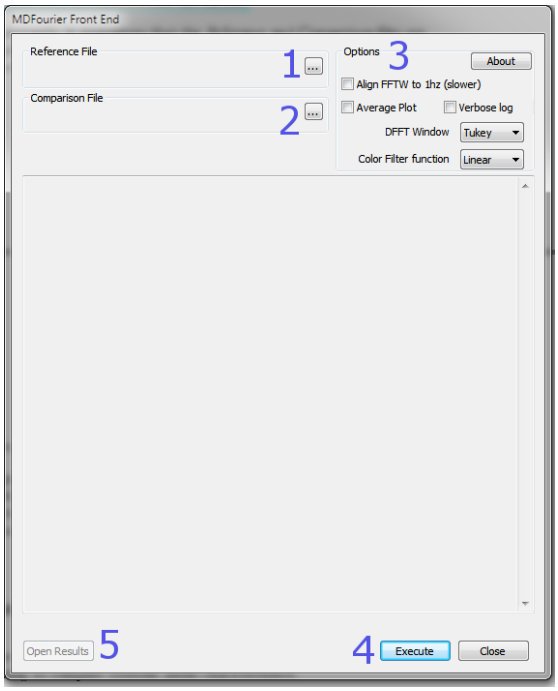

Figure 4.2: Typical sequence of steps

1. Select a *Reference* file
2. Select a *Comparison* file
3. Change the default *options* if needed
4. Execute *MDFourier*
5. 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 available.

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

4.1 Front End Options

The currently available options in the Front End are:

- **DFFT Window:** In order to reduce spectral leakage a filtering window is applied to each element compared between both signals. Please consult section 4.2 for details.
- **Color Filter Function:** This is a *filter function* applied to the results in order to highlight or attenuate the differences between the files. Please consult section 4.3 for details.
- **Align FFTW to 1hz:** Creates a version of each trimmed note, zero padded to match the sample rate in order to align the FFT bins to 1hz, as a result there will be more dots plotted. Off by default.
- **Average Plot:** This traces an average line on top of the plots, making it easier to follow the data trend when the comparison file has severley scattered data. Off by default.
- **Verbose Log:** This option creates a detailed log in the oputput folder, useful for reporting errors or unexpected behaviour. (*Please send the wav files if possible as well!*)

4.2 Window Functions

In order to reduce *spectral leakage* a filtering window is applied to each element compared between both signals. Since we are generating the signal ourselves from the custom binary for each hardware platform, the signal can be analyzed as periodic and has a natural attack and decay rate.

By default we use a custom *Tukey window* with very steep slopes. *MDFourier* does offer alternate windows as options for further analysis. All details regarding the windows used, their formulas and graphs are in section 9.1.

4.3 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. (See section 5.1 for output file details)

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 5 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.

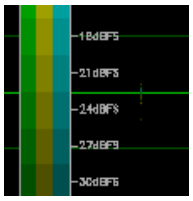

Figure 4.3: Detail of color scale in plot

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 under section 9.2.

Chapter 5

How to interpret the plots

The main output of the program is a set of different graphics that vary in quantity based on the definitions made in the *mfn* file detailed in section 3.2.

In its current form for the *Mega Drive/Genesis*, there are three *active blocks*: *FM*, *PSG* and *Noise*. These will result in a plot of each type, and a general plot, being generated as output.

The files are saved under the folder *MDFourier* and a sub-folder named after the input *WAV* file names. They are stored in *PNG*[10] format, currently *1600x800* plots are used, although this can be dynamic.

For the current document *800x400* plots were used in order to fit within a *PDF* or *HTML* presentation.

5.1 Output Files

There are common features here that we'll describe. First the type of output files:

- **DifferentAmplitude:** Plots the amplitude difference for the frequencies common to both files
- **MissingFrequencies:** Plots the frequencies available in the *Reference* file but not found in the *Comparison* file within the significant volume range.
- **Spectrogram:** Plots all the frequencies available in each file. Two sets of spectrograms are generated, one for the *Reference* file and one for the *Comparison* file.

As mentioned above, there will be several plots of each type in the output folder. One for each type, and one that summarizes all types in a single plot.

In our current *Mega Drive/Genesis* scenario, we'll get four plots of each type: *FM*, *PSG*, *Noise* and a general one, named *ALL*.

We'll follow a series of results from differen 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.

5.2 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 below.

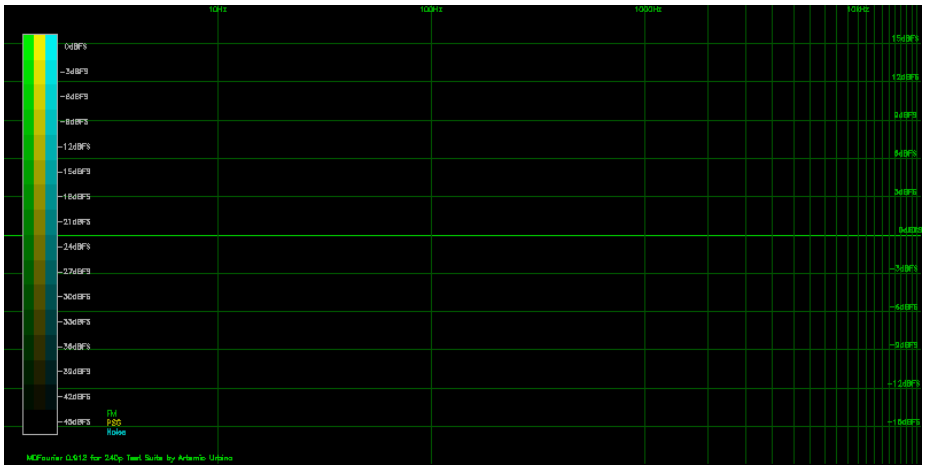

Figure 5.1: Different Amplitudes result file when comparing the same file against itself.

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 type plus the general one.

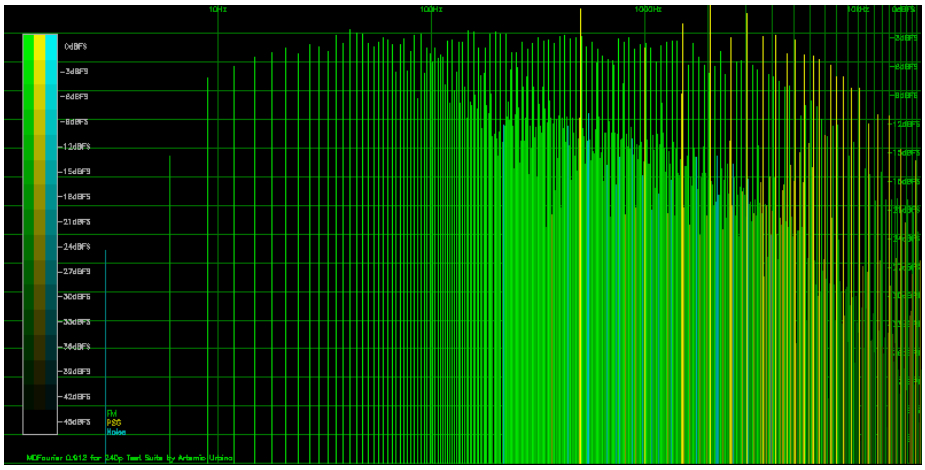

Figure 5.2: The Spectrogram for a Genesis 1 VA3 via hedphone out

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 represents the amplitude in *dBFS*[14]. 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* are used to plot the graph, with each one of them plotting the frequencies from each corresponding block from the *WAV* file.

The top of the plot corresponds to the maximum possible amplitude, which is *0 dBFS*. the bottom of the plot corresponds to the *minimum significant volume*, as described in section 3.4.

Both sets of spectrograms in this case are identical as expected.

5.3 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 exact game console using the same sound card?

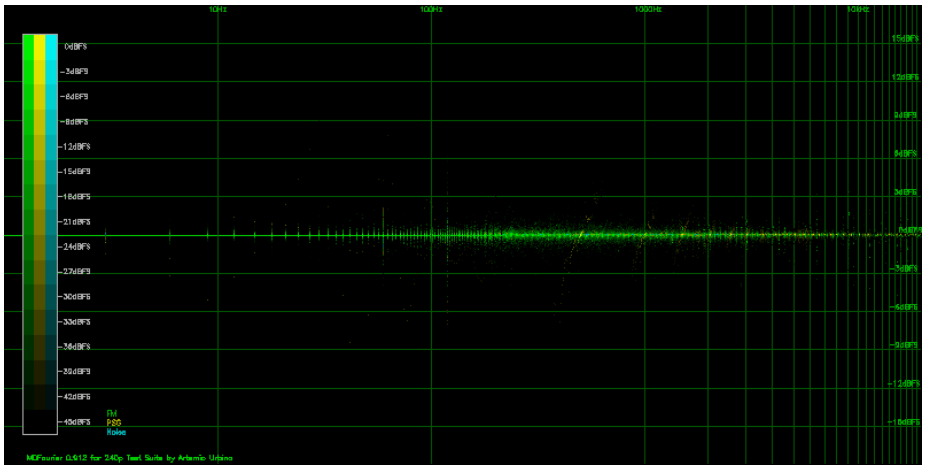

Figure 5.3:

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 the same for one. Then we have variations from the analogue part of console itself, and probably from the internal states and clocks from the digital side. It can also be noise generated by differences in frequency bins when performing the *FFTW* after calculating the frame rates, we have that $^1/_4$ 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 future results.

5.4 Scenario 3: Comparing against a modified file

For demonstration purposes, the same *Reference* file was modified to add a *1khz 6 dbs* parametric equalization across all the analyzed signal. This is a controlled scenario to demonstrate what the plots mean.

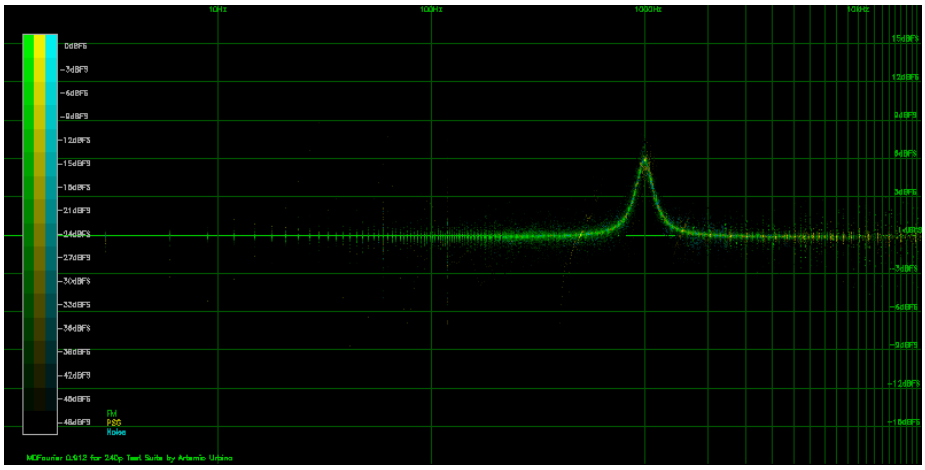

Figure 5.4: Compared against itself modified with a 1khz 6 db equalization

As expected all three blocks (*FM*, *PSG* and *Noise*) were affected and show a spike, exactly *6 dBFS* tall and centered around *1khz*.

It is interesting to note both spectrograms, since the *1khz* spike is also shown there.

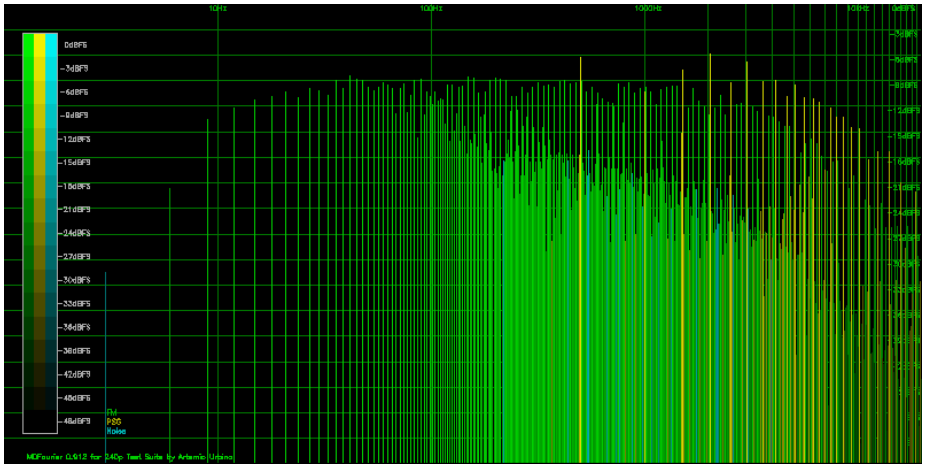

Figure 5.5: *Reference* File

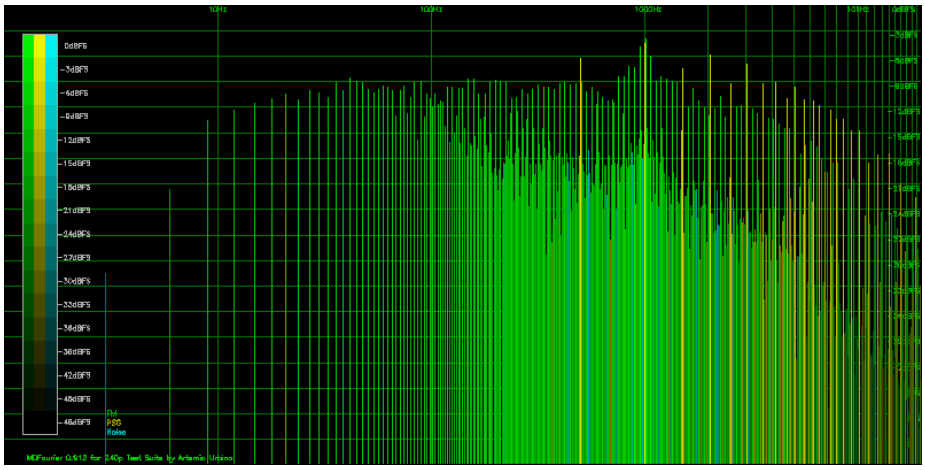

Figure 5.6: *Reference modified with 1khz* File

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

5.5 Scenario 4: Comparing against a digital low pass and high pass filter

We'll use the same *Reference* file, and compare it to a file with a several filters:

- A *low pass filter* 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* to the Noise section at a different frequency

This is the general plot with the three sections:

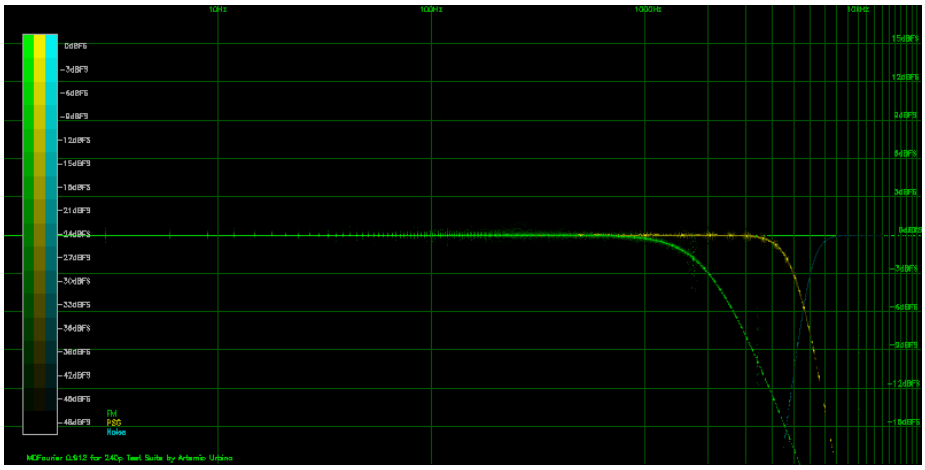

Figure 5.7: *FM*, *PSG* and *Noise* with low pass, low pass and high pass filters.

We can now see that the higher frequencies above *1khz* in the FM plot steeply go to $-\infty$ *dBFS*, so the first low pass filter is there.

The second low pass filter for *PSG* is at *3khz*, and 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 a higher contrast. (see section 4.3)

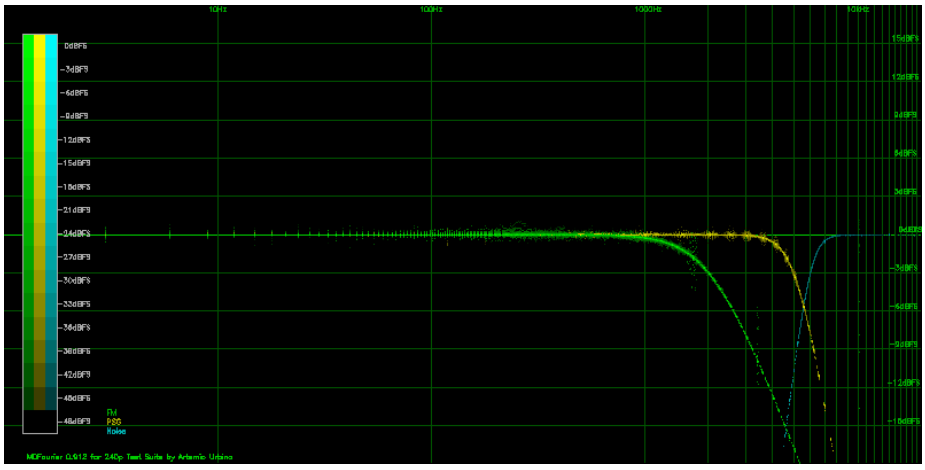

Figure 5.8: Using the $\sqrt{dbFS}$ color filter function

With the higher contrast, we can now make out the curve that raises from $-\infty$ *dBFS* to *0 dBFS*, and it aligns with *8khz*.

We can still do a little bit better, by using the *Average Plot* option. (see section 4)

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

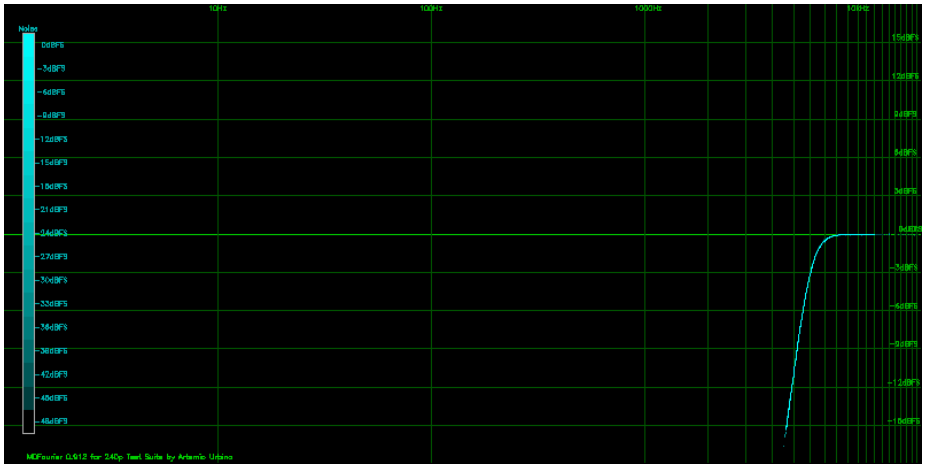

Figure 5.9: *Noise* plot with Average

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

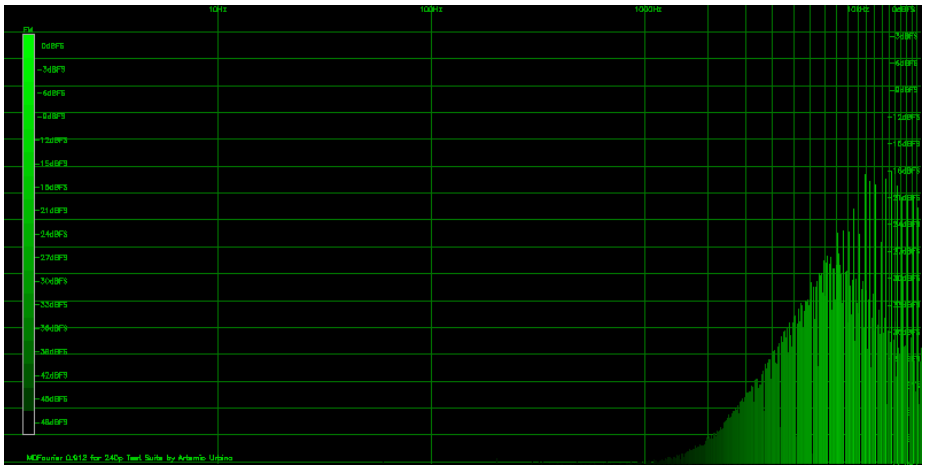

Figure 5.10: Missing frequencies in *FM* cutoff by low pass filter

As show in figure 5.10, there is a curve in the spectrogram and only frequencies above *1khz* show up, slowly rising in amplitude.

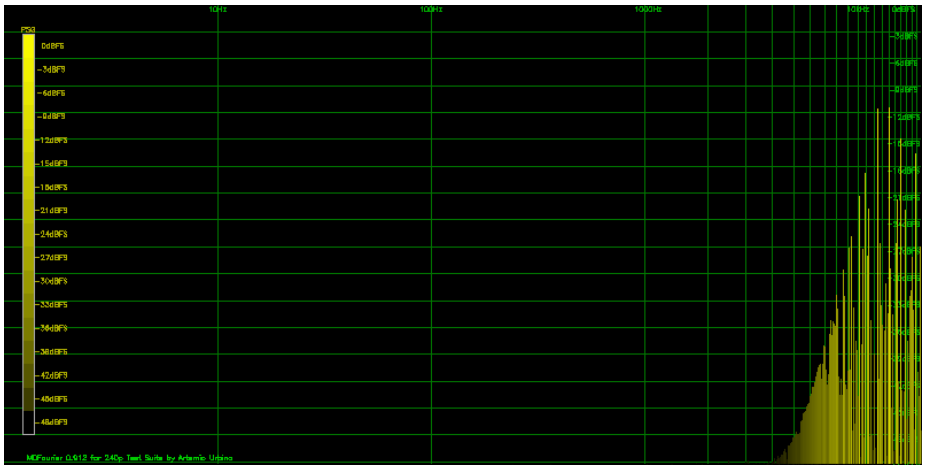

Figure 5.11: Missing frequencies in *PSG* cutoff by low pass filter

The same behavior can be observed in the *PSG spectrogram*, but with a different curve that starts at *4khz* in figure 5.11.

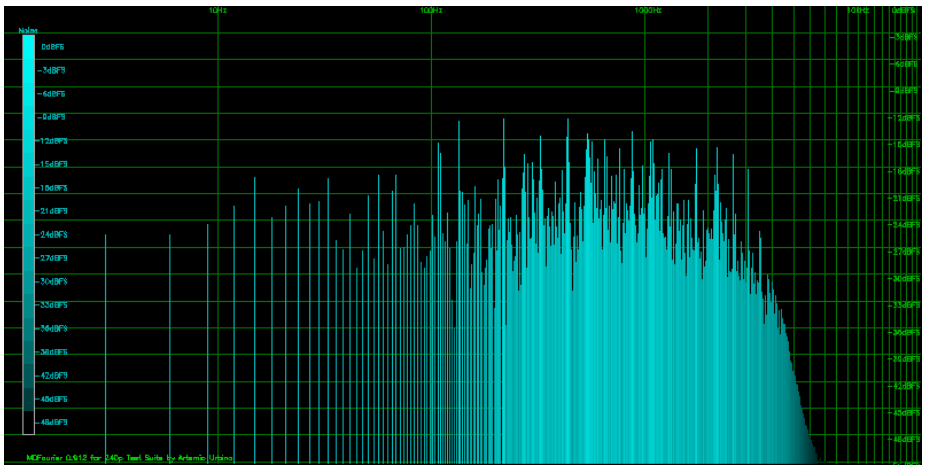

Figure 5.12: Missing frequencies in *Noise* cutoff by high pass filter

And finally, figure 5.12 shows the opposite kind of curve, the high pass filter 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 with 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*?

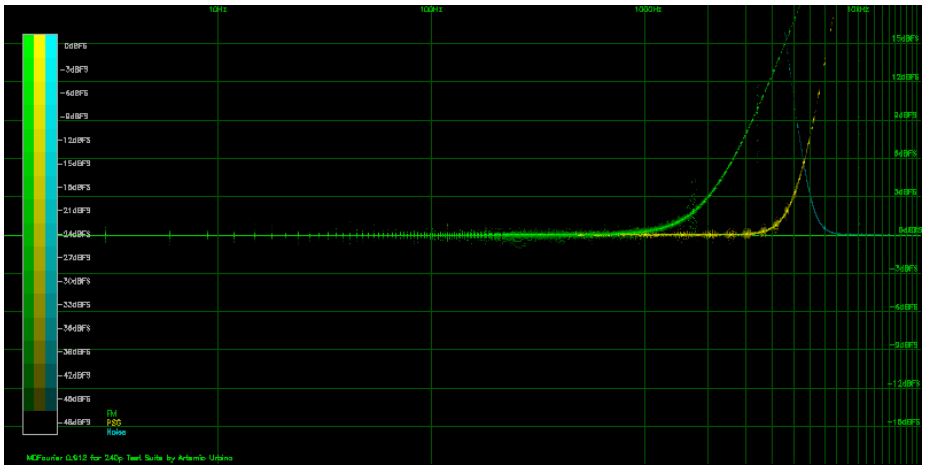

Figure 5.13: Results when using modified signal as *Reference*

Based on this, one could jump to the conclusion that everything will simply be inverted. After all, the original signal not rises to $+\infty$ *dBFS* at the same spots, and that makes complete sense since those frequencies have now a higher amplitude. And although the *Differences* and *Spectrogram* plots will indeed be inverted under these controlled conditions, the *Missing* plots are different.

Most of them are now empty:

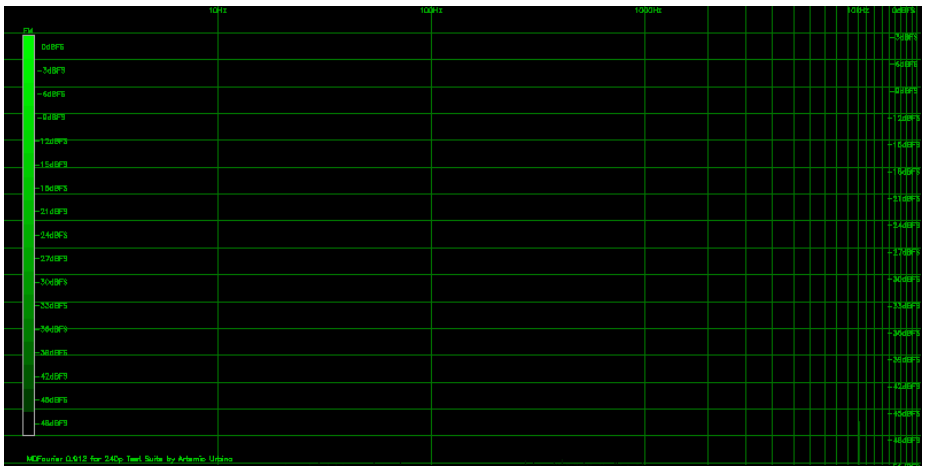

Figure 5.14: *Missing Frequencies* plot for *FM* is empty

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

5.6 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:

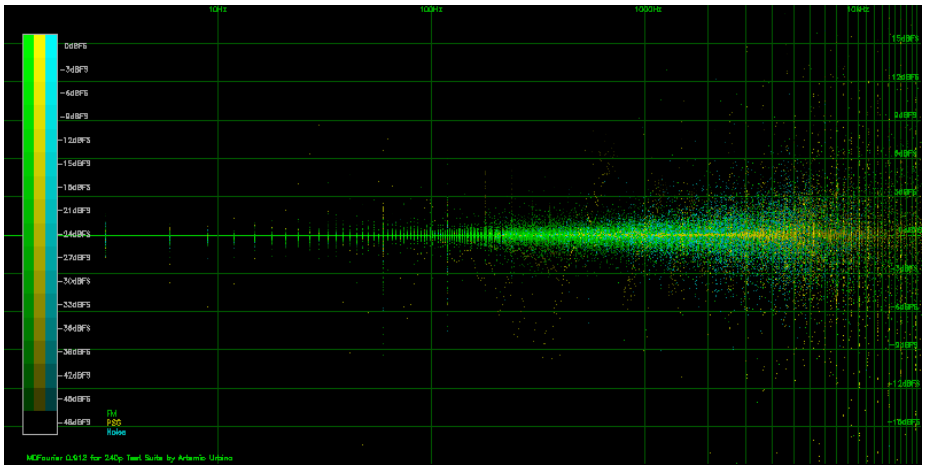

Figure 5.15: Differences using same hardware and cables, different capture cards

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 huge scatter at the higher end of the spectrum, the treble.

But we can still make out the scatter is centered around the *0 dBFS* line, which means that even using different sound cards we can probably make out differences. Also, the sampling was not exactly the same with both cards, there was a slight difference in the detected frame rates:

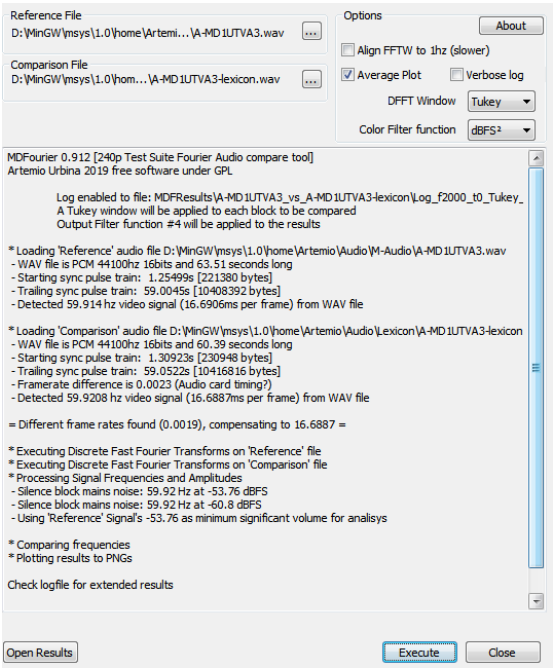

Figure 5.16: Frame rate difference

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

Chapter 6

Results from vintage retail hardware

The following table lists all the hardware used to make the recordings for the plots that will be shown. All had stock parts at the time of the recording, and used original power supplies. They were all connected to a 4" *CRT* via *RGB*, although the *CRT* was turned off while recording. Recordings were made on *5/25/2019*.

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

Chapter 7

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 wave file as argument, and loads all the parameters defined in the configuration file in order to verify the same environment. (see section 3.2).

The output is stored under the folder *MDWave*, and a subfolder with the name of the input *WAV* file. The default output is a *Wave* 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* (section 4.2) 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 wave file and the results are saved.

The opposite can be done as well by specifying the *-x* option, and the result is a *Discarded* wave 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 wave 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 versions of the document. You can type *mdwave -h* in your *mdfourier* folder for details.

Chapter 8

Compiling from source code

8.1 Dependencies

MDFourier needs a few libraries to be compiled. The pre-compiled binary created with *MinGW*[12] is statically linked against:

- fftw-3.3.8[8]
- libpng-1.5.30[10]
- plotutils-2.6[11]
- incbeta[13] (included with source code)

In *Linux* or *UN*X* based systems, you can link it against the latest versions or the libraries.

The *makefiles* to compile either version are provided with the source code[2].

Chapter 9

Appendix

9.1 Window Function details

9.1.1 Tukey

The default is a Tukey window specifically designed for this purpose. It uses a 2.5% slope on each side of the signal, zeroing just a few samples and with minimal amplitude and frequency distortion.

The following equation is used to create the slopes:

$$tukey(x) = 85(1 + \cos(\frac{2\pi}{n-1}\frac{x-(n-1)}{2})) \tag{9.1}$$

And this is the resulting plot of the Tukey window, tanges are 0-1 horizontally and -0.1 to 1.1 vertically.

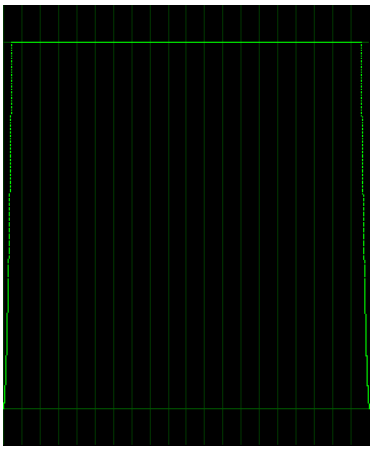

Figure 9.1: This is the custom Tukey window used by MDFourier

9.1.2 Flattop

A typical Flat top window is used.

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

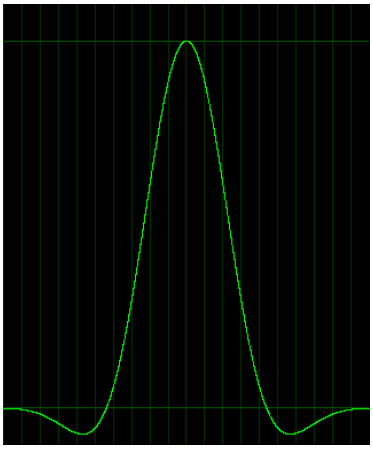

Figure 9.2: Flat Top window

9.1.3 Hann

A typical Hann window is used.

$$hann[x] = \frac{1}{2}(1 - \cos(\frac{2\pi(x+1)}{n+1})) \tag{9.2}$$

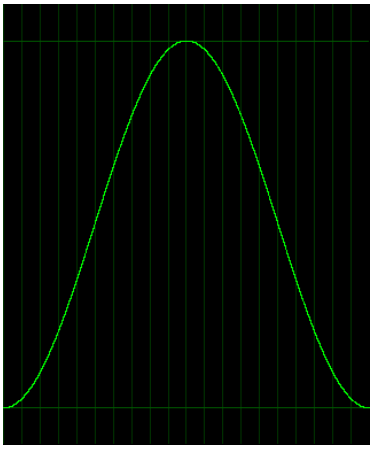

Figure 9.3: Hann Window

9.1.4 Hamming

A typical Hamming window is used.

$$hamming[x] = 0.54 - 0.46\cos(\frac{2\pi x}{n-1}) \tag{9.3}$$

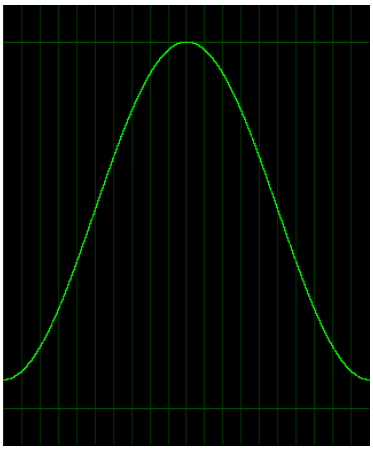

Figure 9.4: Hamming window

9.1.5 No Window

No window is applied, 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 [9].

9.2 Color Filter Function details

9.2.1 None

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

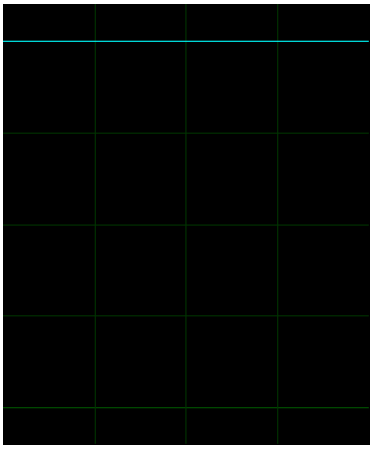

Figure 9.5: No Filter

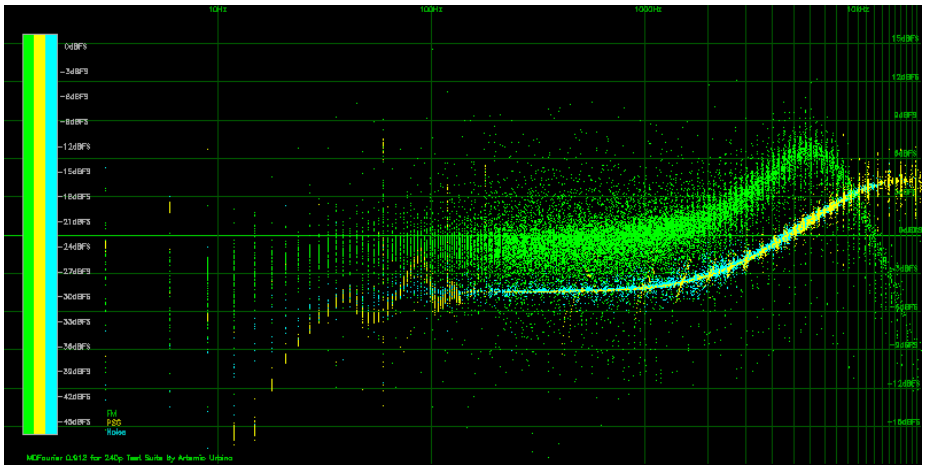

Figure 9.6: No Filter Applied

9.2.2 $\sqrt{dbFS}$

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

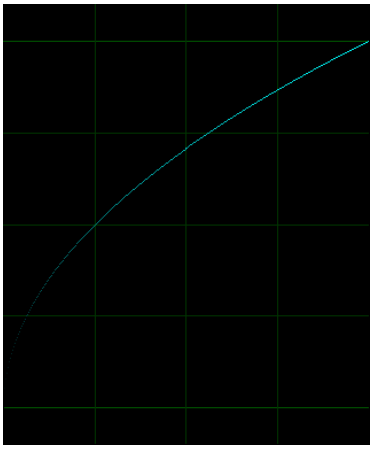

Figure 9.7: Square Root filter

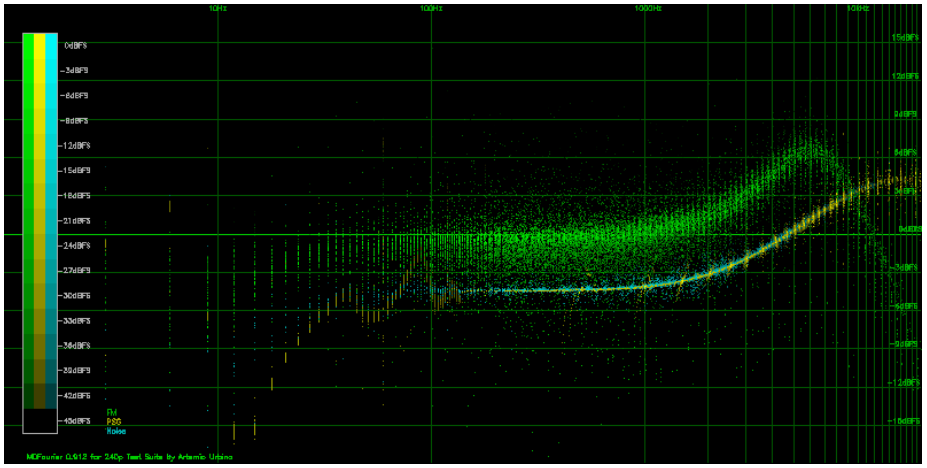

Figure 9.8: Square Root filter Applied

9.2.3 $\beta(3,3)$

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

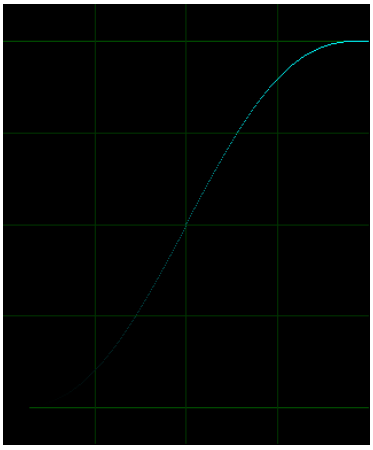

Figure 9.9: Beta Function(3,3)

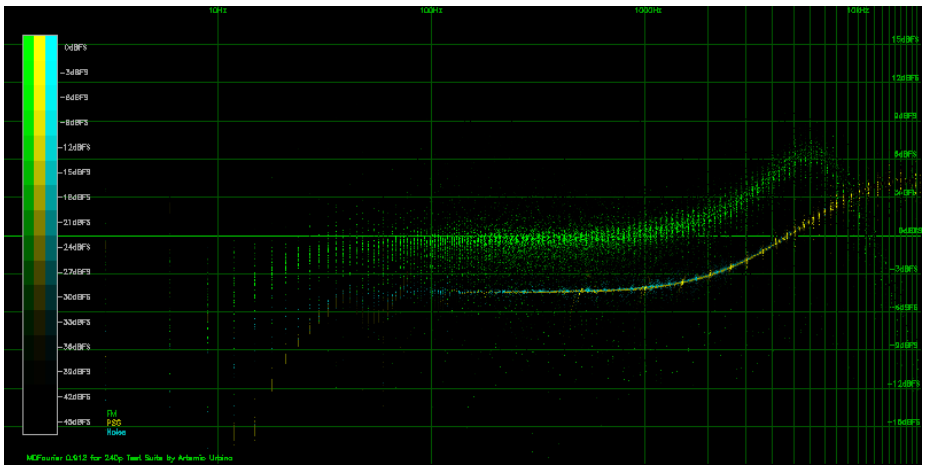

Figure 9.10: Beta Function(3,3) Applied

9.2.4 *Linear*

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

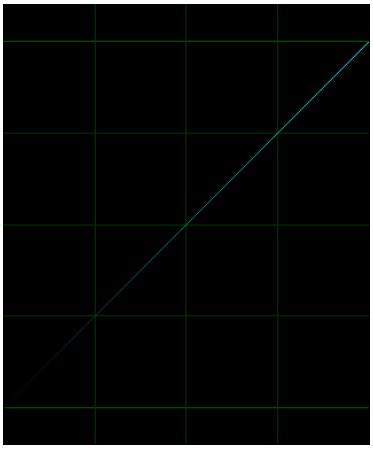

Figure 9.11: Linear Function

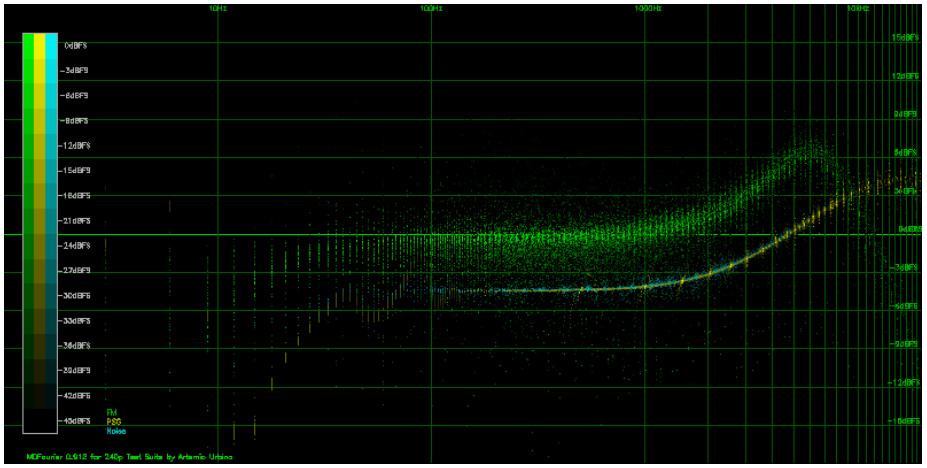

Figure 9.12: Linear Function Applied

9.2.5 $dBFS^2$

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

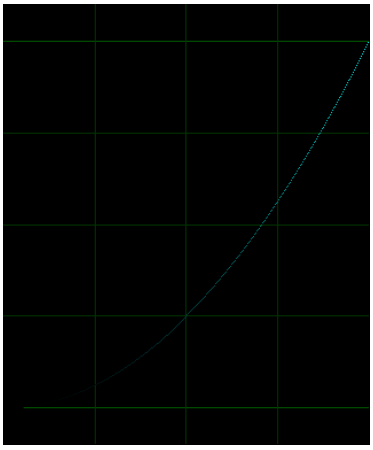

Figure 9.13: Linear Function

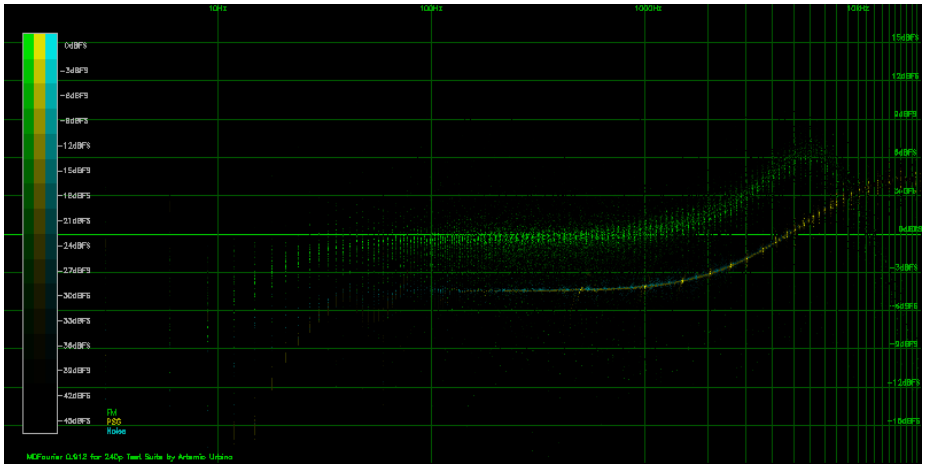

Figure 9.14: Linear Function Applied

9.2.6 $\beta(16, 2)$

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.

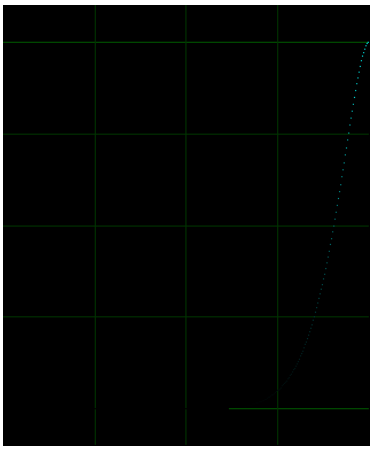

Figure 9.15: Beta Function(16,2)

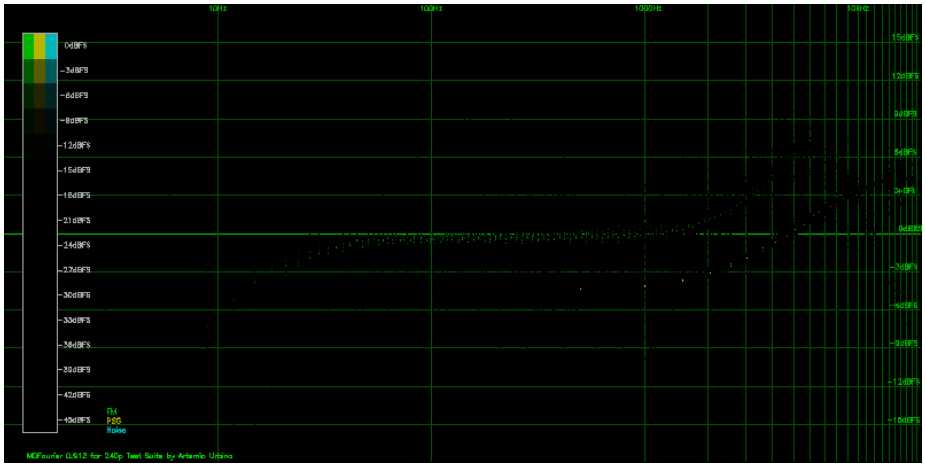

Figure 9.16: Beta Function(16,2) Applied

9.3 Contact the author

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

9.4 Acknowledgements

I'd like to thank the following people for helping me so this tool could be completed. First and foremost to my family, that although never understood what I was doing, they 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] Github, *MDFourier source code C99*, `https://github.com/ArtemioUrbina/MDFourier`.

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

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

[5] 240p test Suite, *Wiki web page*, `http://junkerhq.net/240p`.

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

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

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

[9] 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`.

[10] libPNG *web page*, `https://sourceforge.net/projects/libpng/files/libpng15/1.5.30/`

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

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

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

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

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