CLIO 13/14 BINARY FILE STRUCTURE WITH IMPORT EXAMPLES IN SCILAB

by Daniele Ponteggia – dp@audiomatica.com

INTRODUCTION

In this document we describe the binary file structure of CLIO 13/14 MLS&LogChirp .mls and FFT .fft files and report data import example scripts in Scilab[1].

The file structure of CLIO 13/14 Sinusoidal .sin files is already covered by our Application Note AN-025 "CLIO 13/14 SINUSOIDAL FILE STRUCTURE WITH IMPORT EXAMPLES IN SCILAB".

MLS&LOGCHIRP FILE STRUCTURE

The MLS&LogChirp file structure is composed of an header section of a fixed length followed by a data section with length dependent on MLSsize.

Position (Bytes)	Field	Type	Length (Bytes)	Notes
0	Undocumented Space	char	28	
28	RelBackComp	unsigned int	4	Sets the lowest compatible release (This file structure applies to RelBackComp=627)
32	Undocumented Space	char	765	
797	TimeW	char	1	Time window (0=rectangular, 1=HalfHann, 2=Hann, 3=HalfBH, 4=BH)
798	Undocumented Space	char	2	
800	TimeWb	unsigned int	4	First sample of selected impulse
804	TimeWe	unsigned int	4	Last sample of selected impulse
808	MLSSize	unsigned int	4	MLS length in samples
812	Undocumented Space	char	3	
815	ScaleType	char	1	Y Unit, see TLevUnit definition
816	Fcamp	unsigned int	4	Sampling frequency
820	Undocumented Space	char	13	

1In this document we will show data import examples using Scilab open source software www.scilab.org. Translation of the scripts to Matlab/Octave or Python should be straightforward.

Position (Bytes)	Field	Type	Length (Bytes)	Notes
833	MlsStimuli	char	1	Mls Stimulus Signal (0=MLS, 1=LogChirp)
834	Undocumented Space	char	122	
956	MLSPRe	Array [1..MLSSize] of float	4*MLSSize	Impulse data, real part
+4*MLS Size	MLSPIm	Array [1..MLSSize] of float	4*MLSSize	Impulse data, imaginary part
+4*MLS Size	MLSXRe	Array [1..MLSSize] of float	4*MLSSize	Frequency response data, real part
+4*MLS Size	MLSXIm	Array [1..MLSSize] of float	4*MLSSize	Frequency response data, imaginary part

The TlevUnit is of a set type:

```
TLevUnit=(Vrms,dBV,dBu,dBSpl,dBRel,Ohm,Deg,ms,dB,Perc,dBmet,dBms2,dBPa
,dBPaV,dBms,dBamp,dBSplWm,tCels,Watt);
```

Value	Shown Unit in CLIO	Saved Data Unit
0	Vrms	V
1	dBV	V
2	dBu	V
3	dBspl	Pa
4	dBrel	V
5	Ohm	Ohm
6	Deg	
7	ms	
8	dB	
9	%	
10	dBmet	m
11	dBm/s2	m/s2
12	dBPa	
13	dBPa/V	
14	dBm/s	m/s
15	dBamp	
16	dBSplWm	
17	tCels	° C

Value	Shown Unit in CLIO	Saved Data Unit
18	Watt	W

MLS&LOGCHIRP SCILAB IMPORT EXAMPLE

Here is a simple example of Scilab script to import an .mls measurement with the above file structure.

```
function [RelBackComp, MLSsize, Fcamp, TimeW, TimeWb, TimeWe, frq,
tim, frqdata, timdata]=readMLS(filename);
    [fd]=mopen(filename,'rb');
    skip=mget(28,'uc',fd);
    RelBackComp=mget(1,'ui',fd);
    if RelBackComp<627 then
        mclose(fd);
        error('File not compatible');
    end;
    skip=mget(765,'c',fd);
    //time window (0=no, 1=HalfHann, 2=Hann, 3=HalfBH, 4=BH)
    TimeW=mget(1,'c',fd);
    skip=mget(2,'c',fd);
    TimeWb=mget(1,'ui',fd)       //first sample of selected impulse
    TimeWe=mget(1,'ui',fd)       //last sample of selected impulse
    MLSsize=mget(1,'ui',fd)      //MLS size
    skip=mget(3,'c',fd);
    //Y scale unit (3=Pascal, 5=Ohm, 0,1,2,4=Volts)
    ScaleType=mget(1,'c',fd)
    Fcamp=mget(1,'ui',fd);       //sampling frequency
    skip=mget(136,'c',fd);
    MLSPRe=mget(MLSsize,'f',fd); //impulse data, real part
    MLSPIm=mget(MLSsize,'f',fd); //impulse data, imaginary part
    MLSXRe=mget(MLSsize,'f',fd); //frequency response data, real part
    MLSXIm=mget(MLSsize,'f',fd); //frequency response data, imag. part
    mclose(fd);
    frqdata=complex(MLSXRe,MLSXIm);
    timdata=complex(MLSPRe,MLSPIm);
    frq=0:Fcamp/MLSsize:(Fcamp/MLSsize)*(MLSsize-1);
    tim=0:1/Fcamp:(MLSsize-1)/Fcamp;
endfunction
```

The **readMLS** function returns the measured data in form of vectors alongside with the measurement settings.

frqdata is a (1,MLSsize) complex vector with the frequency response data calculated from the measured impulse response.

frq is a (1,MLSsize) real vector with the linearly spaced analysis frequency points.

timdata is a (1,MLSsize) complex vector with the measured impulse response data.

tim is a (1,MLSsize) real vector with the time of the impulse response measured data.

MLSsize is the measurement size in samples

Fcamp is the sampling frequency

TimeW is the time window type: 0=rectangular, 1=Half Hanning, 2=Hanning, 3=Half

Blackman-Harris, 4=Blackman-Harris[2]

TimeWb, TimeWe are the indexes of the **tim** time data vector with the time window begin and end.

When retrieving data from a .mls binary file we get two different but equally important sets of data:

- **timdata** is the non windowed measured impulse response
- **frqdata** is a complex vector with linearly spaced frequency points, with the result of the following process:
  - Time window applied to measured impulse response
  - Complex frequency response calculated from FFT of windowed impulse response
  - Post-processing applied to complex frequency response

Please note also that the curve shown in CLIO Frequency Response magnitude and phase plots is the result of a data reduction/extrapolation from above linearly spaced frequency data to a 2048 points logarithmic spaced frequency data (which can also be smoothed).

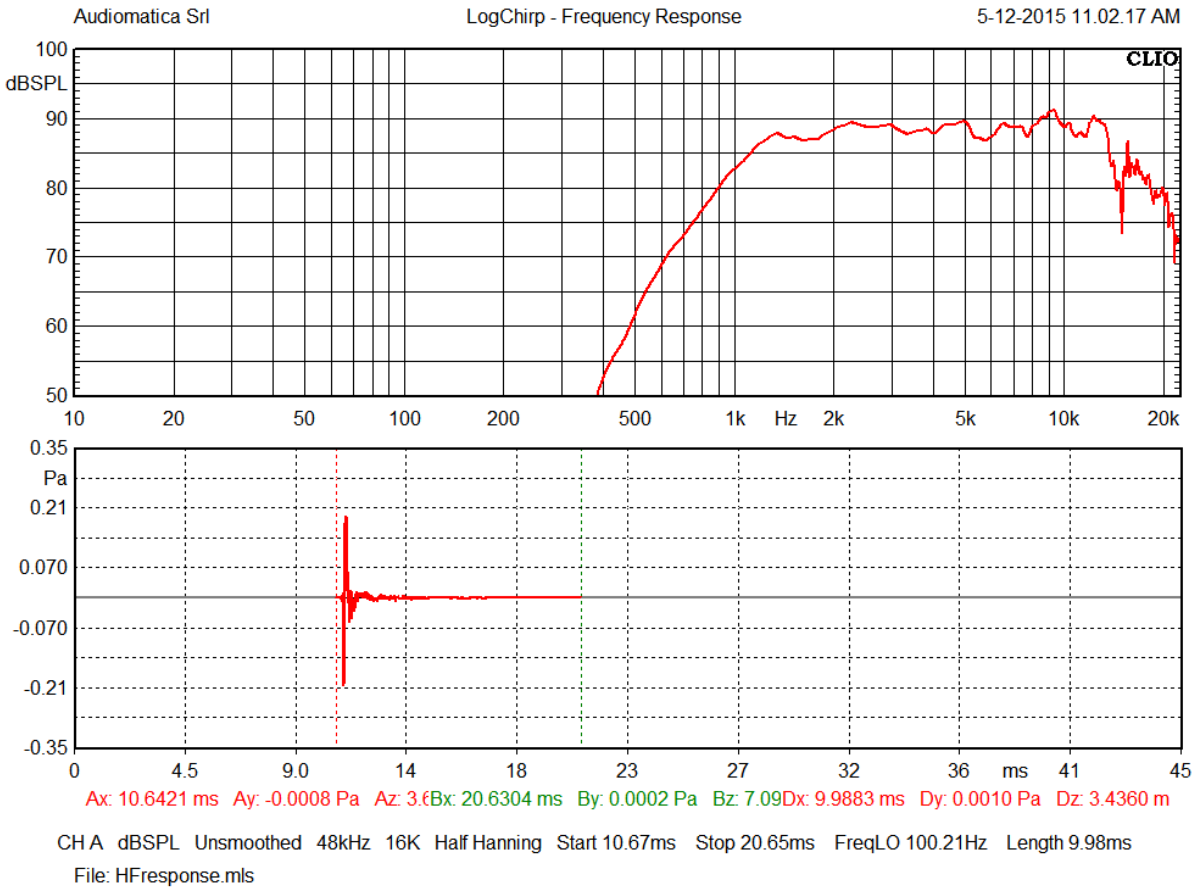
The `readMLS` function can be called from the Scilab console:

```
--> [RelBackComp,MLSsize,Fcamp,TimeW,TimeWb,TimeWe,frq,tim,frqdata,
timdata]=readMLS(filename);
```

and returns a series of vectors and matrices of the data available into the file.

Let's see an example measurement in CLIO, this is the response of a high frequency horn measured on-axis in an anechoic chamber at about 3.5 meter distance.

2For a comprehensive description of window types and effects on data we suggest to consult Fredric J. Harris "On the Use of Windows for Harmonic Analysis With the Discrete Fourier Transform", Proc. IEEE, 1978, pag. 51-83.



As can be seen in the measurement graph comments, the measurement is carried out at 48 kHz sampling rate and measurement size is 16 ksamples (16384 points). Time window type is "Half-Hanning" and window size with start and stop time is also shown in graph comments.

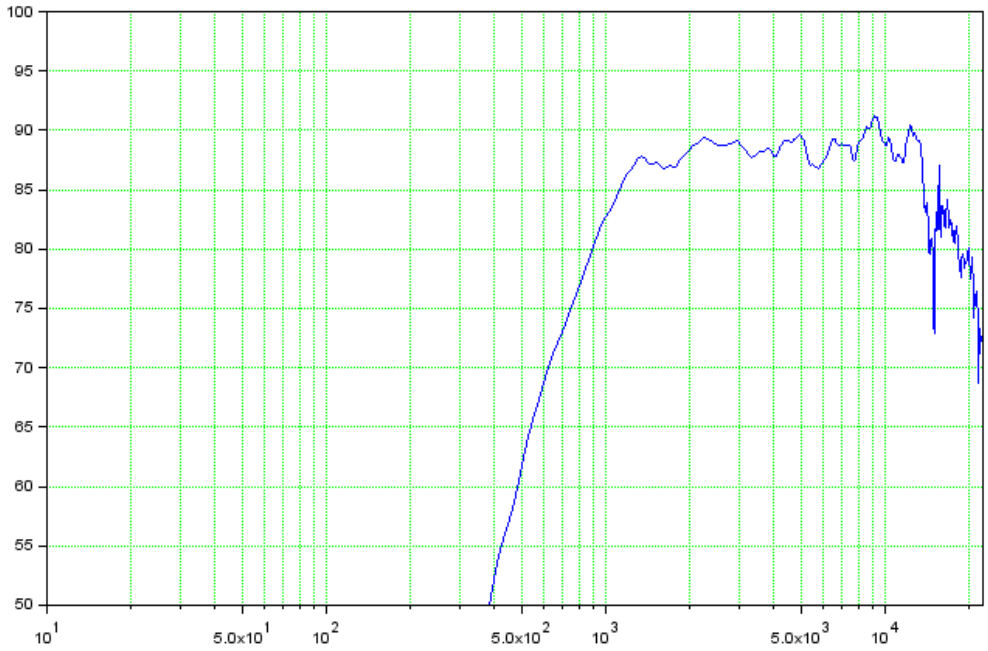
We can load the data from the binary file:

```
--> [RelBackComp,MLSsize,Fcamp,TimeW,TimeWb,TimeWe,frq,tim,frqdata,
timdata]=readMLS('HFresponse.mls');
```

The magnitude frequency response can be easily plot using the following command:

```
--> plot2d(frq(2:MLSsize/2),20*log10(abs(frqdata(2:MLSsize/2)))
+94,logflag="ln",style=2)
```

which after editing axes proprieties will look like:

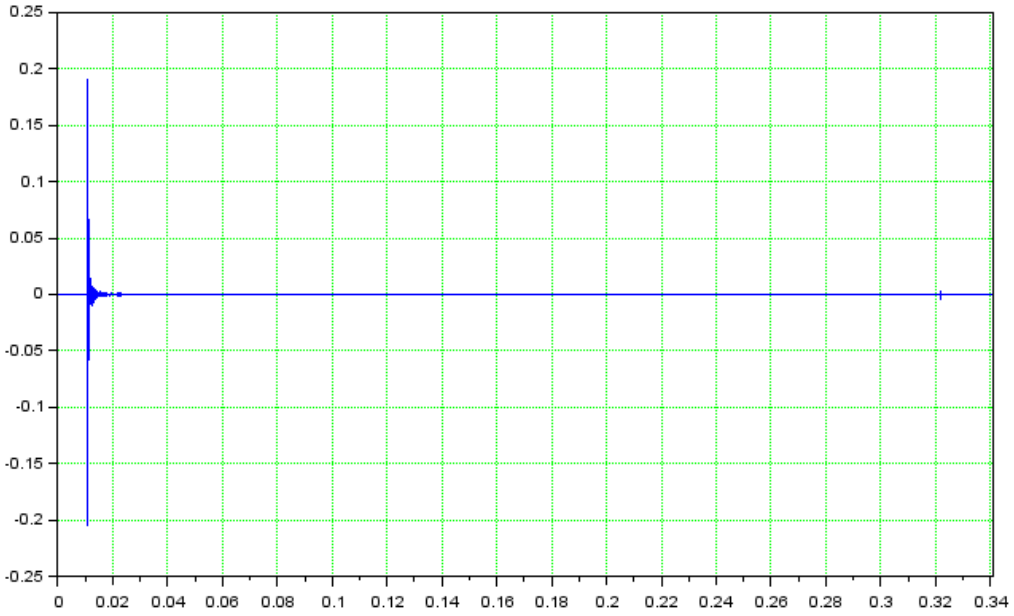


A note on the **`frq`** vector should be made. It can be seen in the above plotting command that we used only a the first half of the vector `frq(2:MLSsize/2)`.

This is due to the fact that the **`frq`** vector has content ranging from 0 Hz up to the sampling frequency, and we are interested only on the frequency range up to the Nyquist frequency. It should also be noted that we started from index 2, this is to avoid the 0 Hz frequency point which cannot be represented in x-axis logarithmic scale.

If we would like to recalculate frequency domain data from impulse response we must start from the time data available in the `timdata` vector.

```
--> plot2d(tim,timdata,2)
```



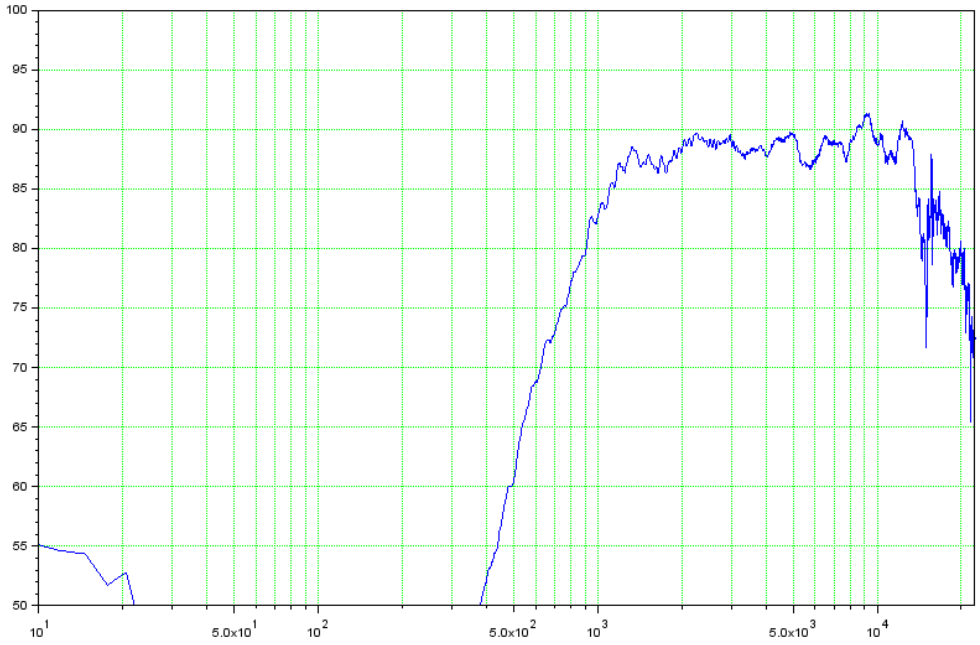
As an example we calculate the response by applying a different time window.

Let's start by applying frequency transformation to the non windowed data, this can be achieved by simply calling the Scilab FFT function:

```
--> Y=fft(timdata);
```

Y will be a complex vector with the frequency response. The magnitude frequency response can be plot using the following code:

```
--> plot2d(frq(2:MLSsize/2),20*log10(abs(Y(2:MLSsize/2)))
+94,logflag="ln",style=2)
```



It can be seen that while this measurement is carried out in an anechoic chamber there are some reflections which are causing the magnitude frequency response to show some ripple which was not present in the previously shown magnitude frequency response calculated using a time windowed impulse.

Suppose here we would like to use a different time window shape in respect to the Half-Hanning originally used, as an example we would like to use here a window not available in CLIO, an asymmetric window which is rectangular in the left half with start from the first sample up to the peak of the IR and then half-Hanning in the right half, with a duration from the peak of 10 milliseconds, i.e. 480 samples at 48 kHz sampling rate.

First of all we should find where the IR peak lies in time domain:

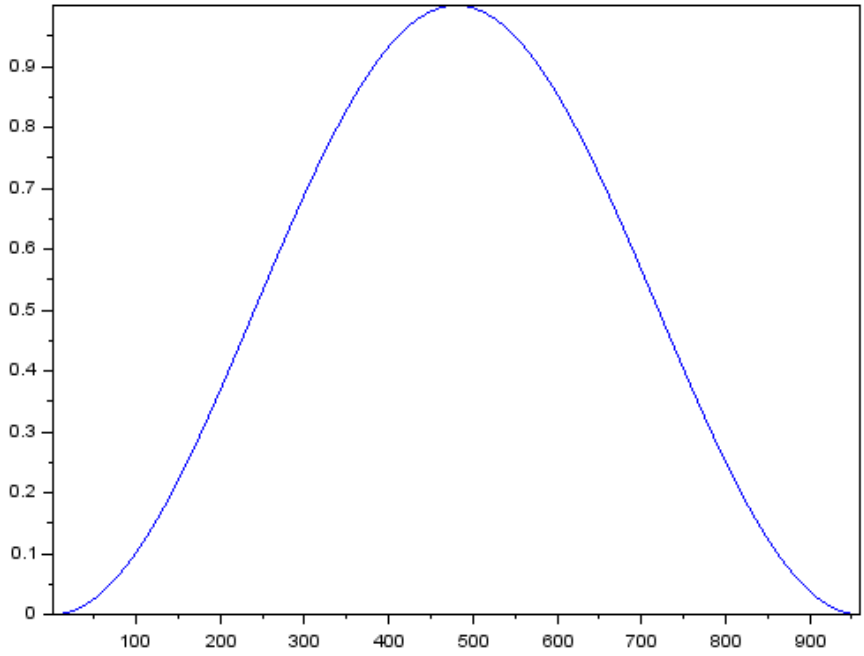
```
--> [pval,pind]=max(abs(timdata))
 pval  =

   0.2045097
 pind  =

   526.
```

Where `pind` is the index of the time data absolute peak value.

To create such a window we can use the Scilab `window` function with the following parameters:

```
--> hwn=window('hn',2*480);
```

This will create a full Hanning window with 2*480=960 points size, we will use only the decaying part of this window.



Our complete time window will be instead:

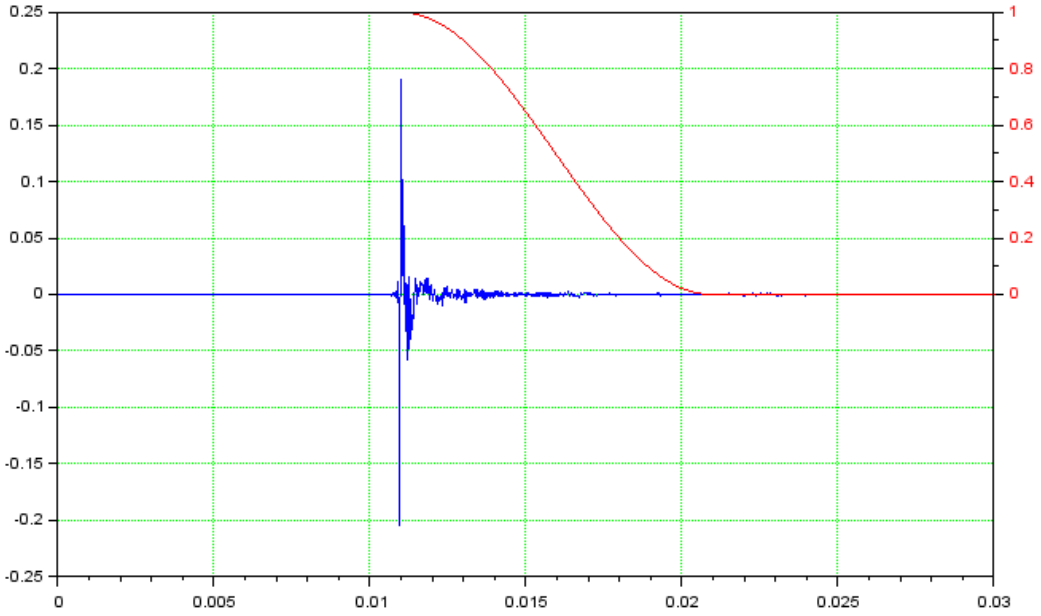
```
--> twn=[ones(1,pind) hwn(1,481:960) zeros(1,MLSsize-480-pind)];
```

Which should have the same size as **`timdata`**:

```
--> size(twn)
 ans  =
   1. 16384.
```

The time data can be plotted alongside the impulse response using the following code:

```
plot2d(tim,timdata,2)
h1=gca();
h2=newaxes();
plot2d(tim,twn,5)
h2.filled="off";
h2.axes_visible(1)="off";
h2.font_color=color("red");
h2.y_location="right";
h1.box="on";
h1.tight_limits="on";
h2.tight_limits="on";
h1.data_bounds=[0 0.05, -0.25 0.25];
h2.data_bounds=[0 0.05, -1 1];
h1.grid=[3 3];
```



The windowed impulse response **wtimdata** will be the product of the measured impulse response with the window function:

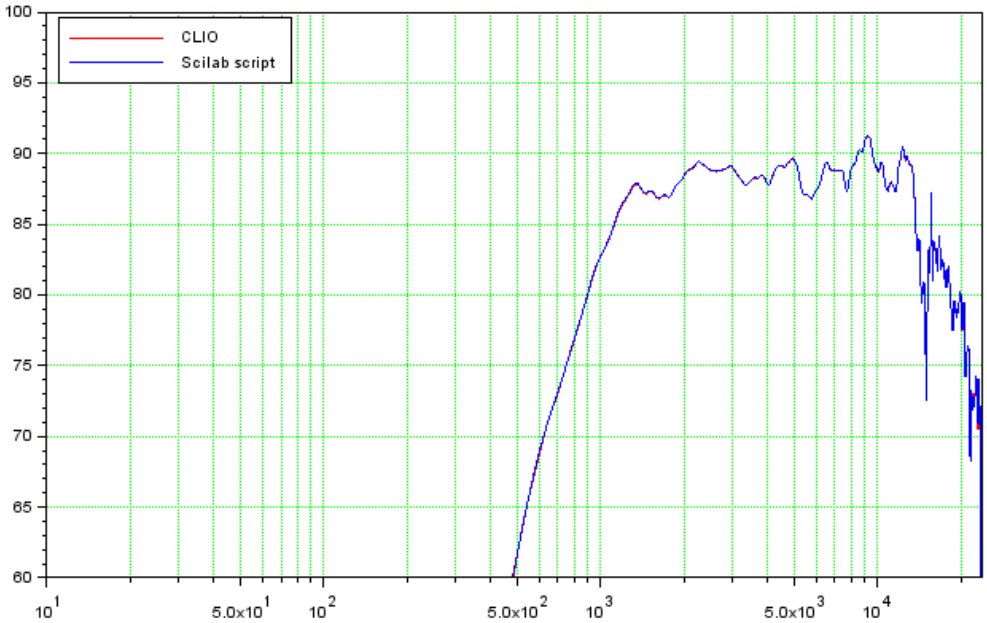
```
--> wtimdata=timdata.*twn;
```

The complex frequency response can be then calculated:

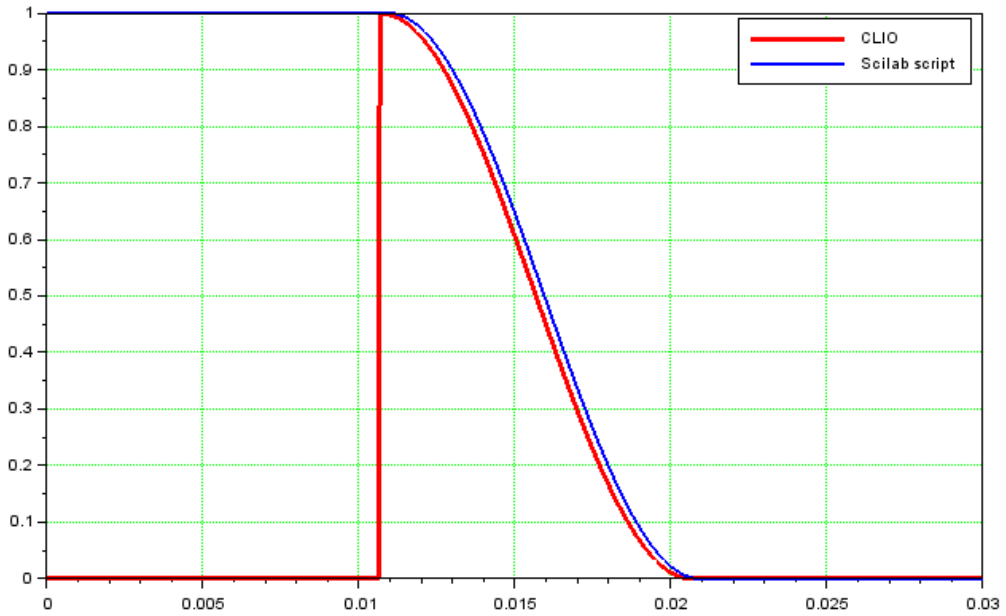
```
--> YW=fft(wtimdata);
```

And plotted:

```
--> plot2d(frq(2:MLSsize/2),20*log10(abs(YW(2:MLSsize/2)))
+94,logflag="ln",style=2)
```



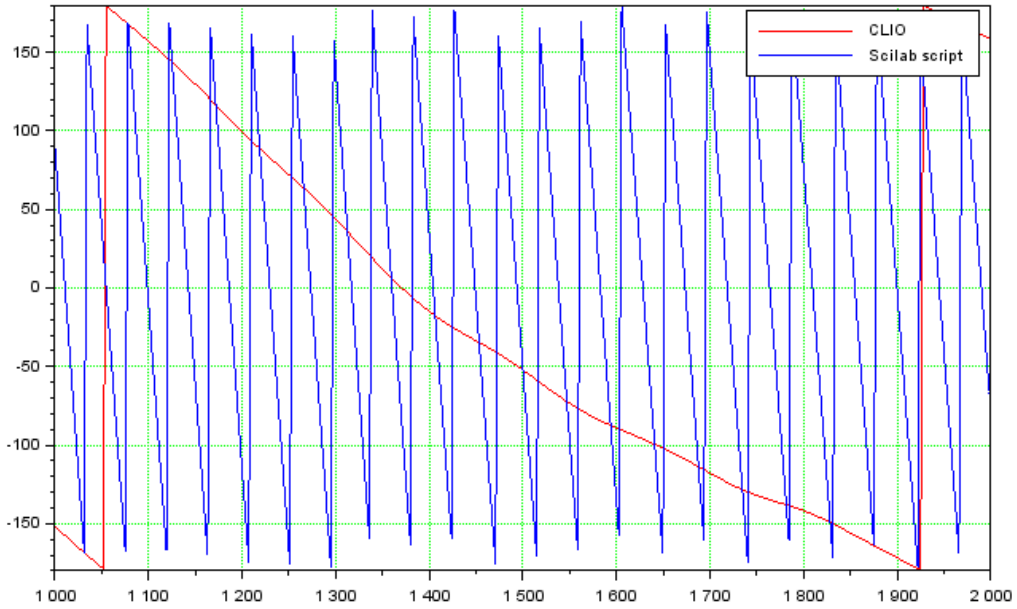
The calculated magnitude response looks very similar to the original measurement frequency domain magnitude response, in fact the time window shape is similar in the two cases, except for the initial part. Let's compare the two time windows:



We can plot the phase response to see the difference between the two complex frequency responses:

```
-->
plot2d(frq(2:MLSsize/2),(360./%pi).*atan(imag(YW(2:MLSsize/2))./real(Y
W(2:MLSsize/2))),style=2)

-->
plot2d(frq(2:MLSsize/2),(360./%pi).*atan(imag(frqdata(2:MLSsize/2))./
real(frqdata(2:MLSsize/2))),style=5)
```

We can zoom in a narrow frequency range and see that the response calculated with the Scilab script has a delay component which is not removed.


CLIO
Scilab script
150
100
50
0
-50
-100
-150
1 000
1 100
1 200
1 300
1 400
1 500
1 600
1 700
1 800
1 900
2 000

FFT FILE STRUCTURE

The FFT file structure is composed of an header section of a fixed length followed by a data section with length dependent on FFTSize.

Position (Bytes)	Field	Type	Length (Bytes)	Notes
0	Undocumented Space	char	788	
788	FFTSize	unsigned int	4	FFT length in samples
792	Undocumented Space	char	40	
832	Fcamp	unsigned int	4	Sampling frequency
836	Undocumented Space	char	192	
1028	AFFT	Array [1..FFTSize] of float	4*FFTSize	Squared FFT data CHA*
+4*FFT Size	BFFT	Array [1..FFTSize] of float	4*FFTSize	Squared FFT data CHB*
+4*FFT Size	ATime	Array [1..FFTSize] of float	4*FFTSize	Time data CHA*
+4*FFT Size	BTime	Array [1..FFTSize] of float	4*FFTSize	Time data CHB*

*If the FFT saved file is a Transfer Function then:
AFFT -> GAA
BFFT -> GBB
ATime -> GAB Real
BTime -> GAB Imag

Where GAA and GBB are auto-spectrum of CHA and CHB data. GAB Real and GAB Imag are real and imaginary parts of cross-spectrum between CHA and CHB.

FFT SCILAB IMPORT EXAMPLE

Here is a simple example of Scilab script to import an .fft measurement with the above file structure.

```
function [FFTsize,Fs,AFFT,BFFT,ATime,BTime]=readFFT(filename)
    [fd]=mopen(filename);
    skip=mget(788,'uc',fd);
    FFTsize=mget(1,'ui',fd);
    skip=mget(40,'uc',fd);
    Fs=mget(1,'ui',fd);
    skip=mget(192,'uc',fd);
    AFFT=mget(FFTsize,'f',fd);
    BFFT=mget(FFTsize,'f',fd);
    ATime=mget(FFTsize,'f',fd);
    BTime=mget(FFTsize,'f',fd);
    mclose(fd);
endfunction
```

The **readFFT** function returns the measured data in form of vectors alongside with the measurement settings.
AFFT is a (1,MLSsize) real vector with the squared FFT data of CHA (GAA auto-spectrum of CHA data if the measurement is a live transfer function)

BFFT is a (1,MLSsize) real vector with the squared FFT data of CHB (GBB auto-spectrum of CHB data if the measurement is a live transfer function)

ATime is a (1,MLSsize) real vector with the latest acquired CHA time data (GAB real part of cross-spectrum of CHA and CHB data)

BTime is a (1,MLSsize) real vector with the latest acquired CHB time data (GAB imaginary part of cross-spectrum of CHA and CHB data)

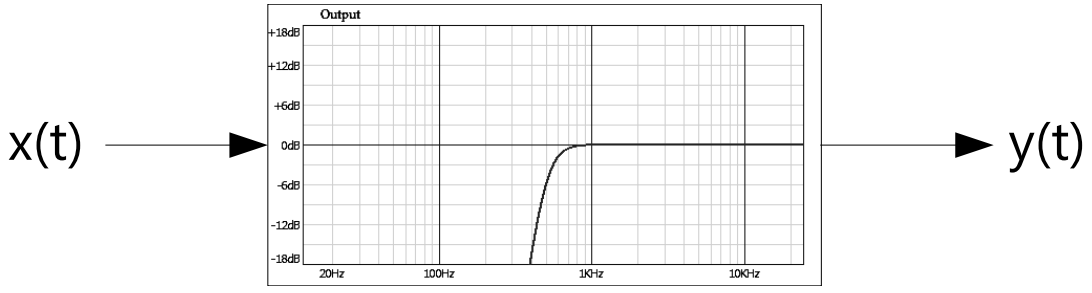
FFTsize is the measurement size in samples

Fs is the sampling frequency

Let's start with an example of "standard" FFT measurement, by standard we mean a measurement taken with FFT – Narrowband Analyzer or RTA – Octave Band Analyzer mode.

In both cases only the narrowband data is saved in the binary file. In case of an RTA analysis the narrowband data should be recombined in octave fraction bands by the Scilab script.

In our example we will measure the response of a DSP with a 500 Hz Linkwitz-Riley 48 dB/oct high pass filter:

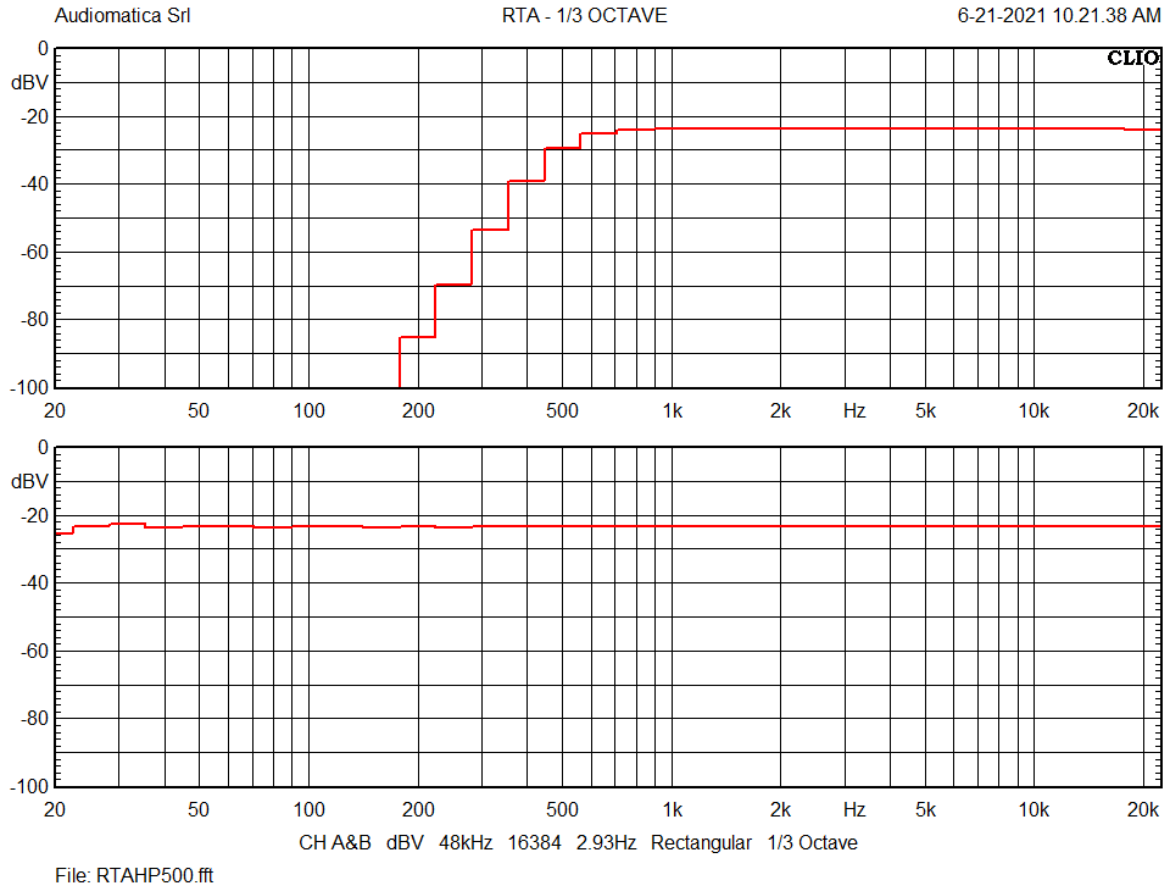


We will use a pink noise as stimulus x(t) and evaluate the response using an RTA – Octave Band Analyzer with CLIO FFT.

CLIO connection should be as follows:

- CLIO CHA output on DUT input
- CLIO CHA input on DUT output
- CLIO CHB in loop

In this way we will have the stimulus signal on CLIO CHB which will be used later as a reference channel in Live Transfer Function FFT mode.



The measurement is saved with "RTAHP500.fft" filename.

We now try to use `readFFT` Scilab function to read the file data:

```
--> [FFTsize,Fs,AFFT,BFFT,ATime,BTime]=readFFT('RTAHP500.fft');
```

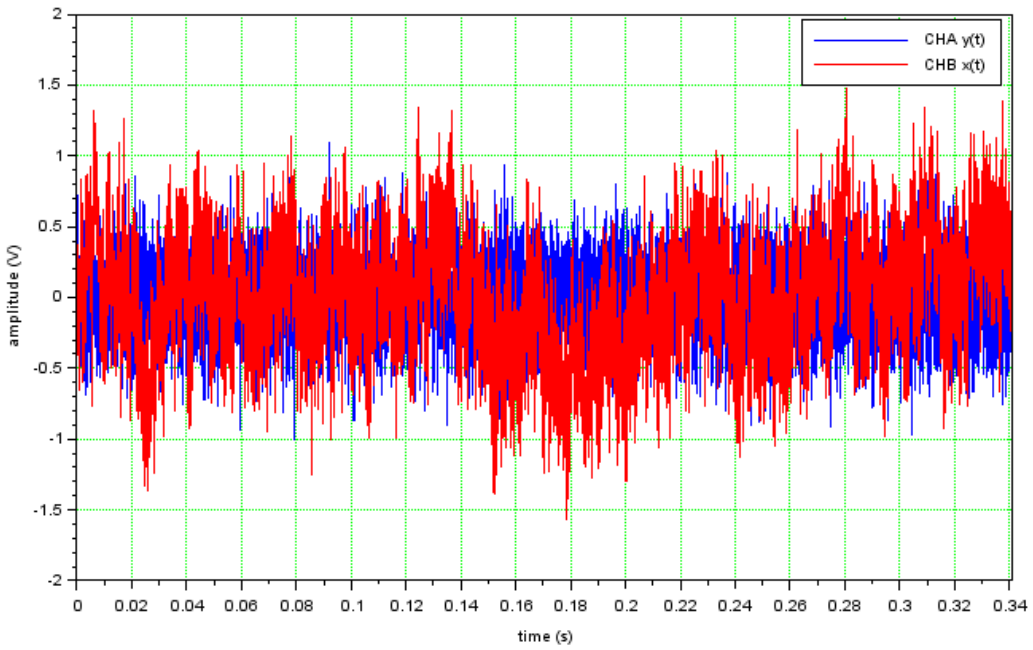
We need to create time and frequency vectors as follows to manipulate and plot the data in Scilab:

```
--> tim=0:1/Fs:(FFTsize-1)./Fs;
--> frq=0:Fs./FFTsize:(FFTsize-1).*(Fs./FFTsize);
```

The time data can be plot with the commands:

```
--> plot2d(tim,ATime,2);
--> plot2d(tim,BTime,5);
```

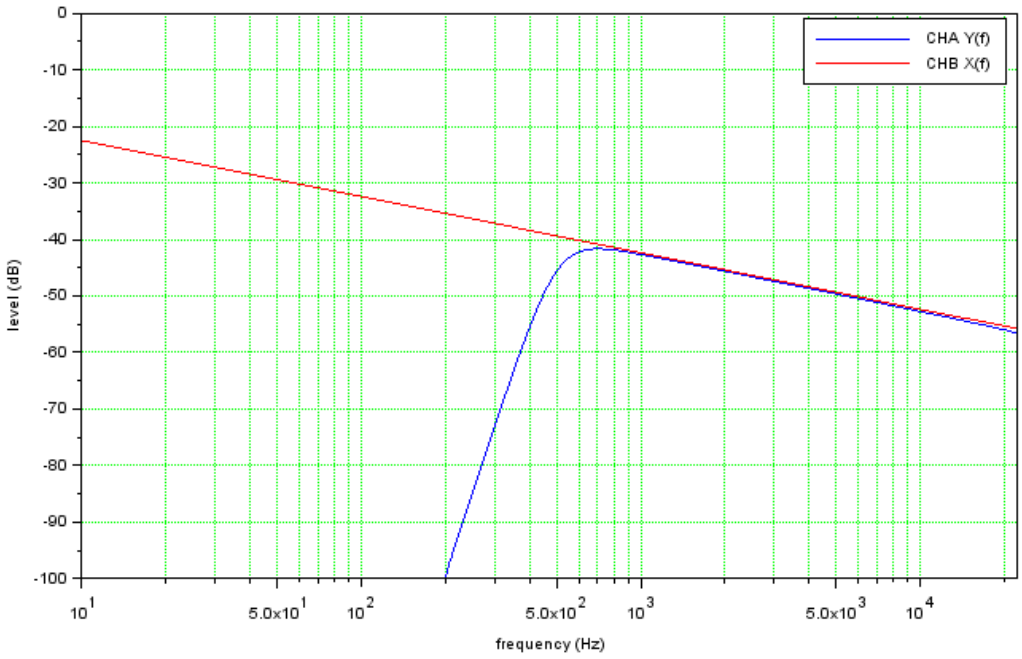
Which after some beautification should look like:



Please note that the time data is relative only the last FFT time frame acquisition, averages do not affect this time data.

Let's see the FFT frequency spectrum:

```
--> plot2d(frq(2:FFTsize/2),10*log10(abs(AFFT(2:FFTsize/2))),2);
--> plot2d(frq(2:FFTsize/2),10*log10(abs(BFFT(2:FFTsize/2))),5);
```



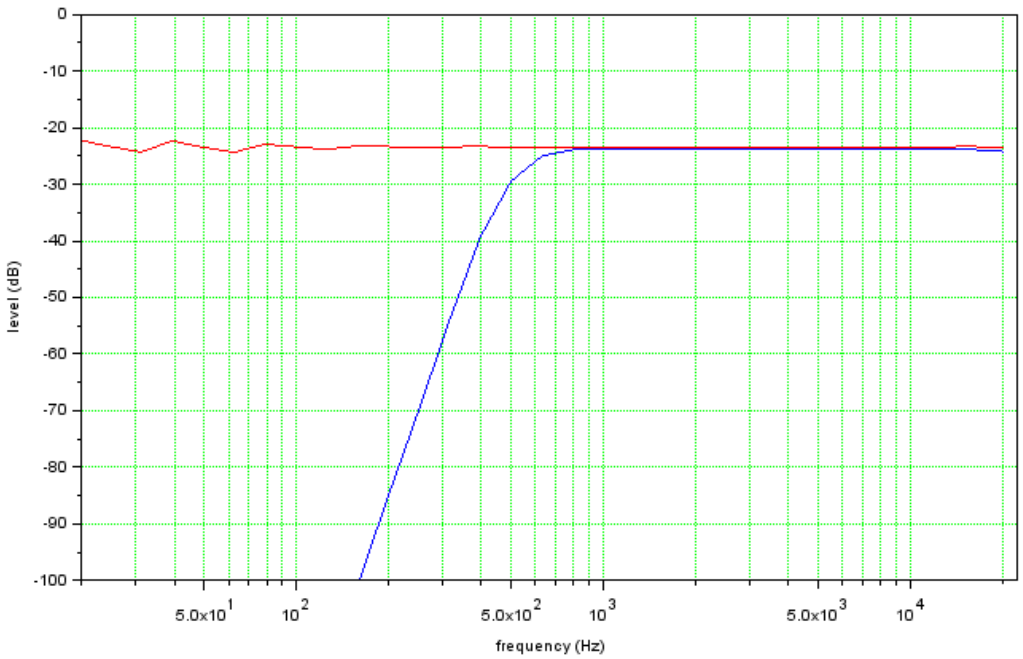
It can be seen that, as expected, the narrowband data shows the typical 3 dB/oct decay for the pink noise input.

It is possible to recalculate the third octave band data from the FFT narrowband data using the following Scilab script:

```
fcen=10^3*(2.^([-18:13]/3));
fhi=fcen*2^(1/6);
flo=fcen/2^(1/6);
for i=1:size(fcen,2) do
    fmask=(frq<=fhi(i))&(frq>flo(i));
    fin=find(fmask);
    RTAA(i)=10*log10(sum(AFFT(fin)));
    RTAB(i)=10*log10(sum(BFFT(fin)));
end
```

Followed by these plotting commands:

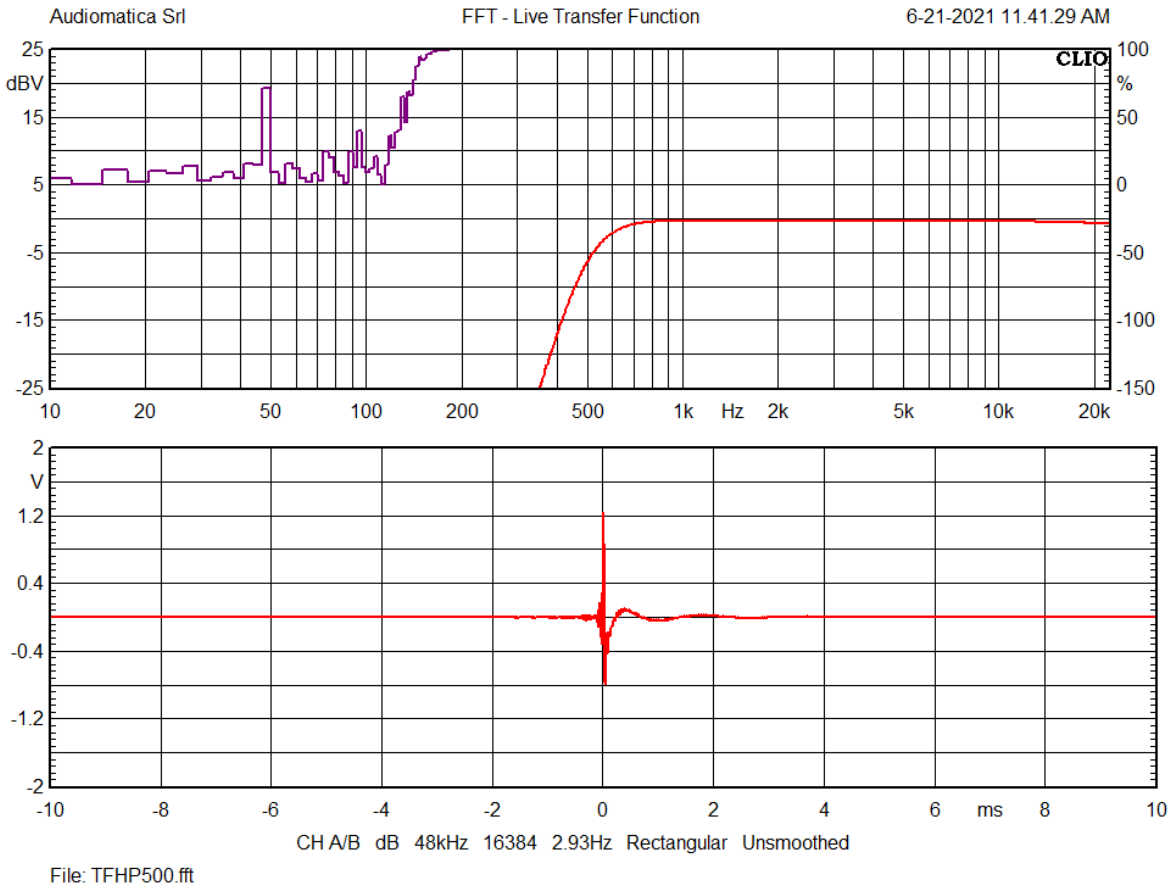
```
--> plot2d(fcen,RTAA,2);
--> plot2d(fcen,RTAB,5);
```



We leave to the reader the task to convert this plot to a bar plot data with bars centered at `fcen` values using the Scilab `plot2d2` function.

We now carry out the measurement on the DUT using Live Transfer Function FFT mode.

CLIO CHB acquires the reference signal, which is in loop with the CLIO generator. In order to correctly estimate the coherence the number of averages must be greater than 1, we select 100. It should be also considered that, since we are using a periodic stimulus signal such as is the CLIO pink 16k noise, coherence is NOT an estimate for DUT distortion.



We can save the measurement with name "TFHP500.fft".

Let's try to import the measurement in Scilab.

```
--> [FFTsize,Fs,GAA,GBB,GABRe,GABIm]=readFFT('TFHP500.fft');
```

The transfer function can be calculated from the GAA and GBB auto-spectrum quotient and plot using the following Scilab commands:

```
--> H=GAA./GBB;
--> HdB=10*log10(abs(H));
```

Coherence can be calculated using the well known formula:

$$\gamma^2 = \frac{\left|G_{AB}(f)\right|^2}{G_{AA}(f)\cdot G_{BB}(f)}$$

which translates in the following Scilab statement:

```
--> COH=(GABRe.^2+GABIm.^2)./(GAA.*GBB+%eps);
```

Which can be plotted alongside magnitude response using the following code:

```
f1=scf();
f1.figure_size=[800 600];
plot2d(frq(2:FFTsize/2),HdB(2:FFTsize/2),2);
h1=gca();
h1.box="on";
h1.tight_limits="on";
h1.log_flags="lnn";
h1.data_bounds=[10 22388 -50 25];
h1.x_label.text="frequency (Hz)";
```

```
h1.y_label.text="level (dB)";
h1.grid=[3 3];
h2=newaxes();
plot2d(frq(2:FFTsize/2),COH(2:FFTsize/2),5);
h2.filled="off";
h2.axes_visible(1)="off";
h2.font_color=color("red");
h2.y_location="right";
h2.tight_limits="on";
h2.log_flags="lnn";
h2.data_bounds=[10 22388 0 1];
h2.y_label.text="squared coherence";
h2.y_label.font_color=5
```