CLIO POCKET FILE STRUCTURES WITH IMPORT EXAMPLES IN SCILAB

by D. Ponteggia – dp@audiomatica.com

INTRODUCTION

We report here briefly the file structure of the LogChirp .crp and FFT .ffp CLIO pocket 1.50 measurement files. Scilab scripts and examples are included.

FILES STRUCTURES

CLIO pocket LogChirp Impulse Measurement file .crp:

Position (Bytes)	Field	Type	Length (Bytes)	Notes
0	Undocumented space	char	820	
820	Undocumented field	unsigned int	4	
824	Channel number	unsigned int	4	
828	Chirp size	unsigned int	4	
832	Sampling frequency	unsigned int	4	
836	Time Window	char	1	(RectW, HalfHann, Hann, HalfBH, BH)
837	Time Wb	unsigned int	4	Time window begin index
841	Time We	unsigned int	4	Time window end index
845	Measurement Unit	char	1	(Vrms, dBV, dBu, dBSpl, dBRel, Ohm, Deg, ms, dB, Perc, dBmet, dBms2, dBPa, dBPaV, dBms, dBamp, dBSplWm, tCels, Watt)
846	Smoothing	char	1	(SmNone, Sm48, Sm24, Sm12, Sm6, Sm3, Sm1)
847	Undocumented space	char	263	
1110	Chirp Impulse Real	float	4*Chirp Size	
1110+4* ChirpSize	Chirp Impulse Imag	float	4*Chirp Size	

CLIO pocket FFT measurement files .ffp:

Position (Bytes)	Field	Type	Length (Bytes)	Notes
0	Undocumented space	char	860	
860	FFT Size	unsigned int	4	
864	Sampling frequency	unsigned int	4	
868	FFT Window	char	1	(None, Hanning, Hamming, Blackman, Bartlett, FlatTop)
869	Undocumented space	char	8	
877	Measurement Unit	char	1	(Vrms, dBV, dBu, dBSpl, dBRel, Ohm, Deg, ms, dB, Perc, dBmet, dBms2, dBPa, dBPaV, dBms, dBamp, dBSplWm, tCels, Watt)
878	Undocumented space	char	10	
888	Smoothing	char	1	(SmNone, Sm48, Sm24, Sm12, Sm6, Sm3, Sm1)
889	Undocumented space	char	336	
1225	FFT Data	float	4*FFT size	Magnitude data
1225+4* FFT size	Time Data	float	4*FFT size	Time Data

SCILAB IMPORT SCRIPTS

```
function [NumCh,ChirpSize,FCamp,TimeW,TimeWb,TimeWe,MisUnit,Smooth,
         ChirpImpRe,ChirpImpIm]=loadcrp(filename)
    // Load a ClioPkt .crp Chirp Response File
    [fd]=mopen(filename,'rb');
    skip=mget(820,'uc',fd);
    skip=mget(1,'ui',fd);  //NextOfs
    NumCh=mget(1,'ui',fd);
    ChirpSize=mget(1,'ui',fd);
    FCamp=mget(1,'ui',fd);
    TimeW=mget(1,'uc',fd);  //(RectW,HalfHann,Hann,HalfBH,BH)
    TimeWb=mget(1,'ui',fd);
    TimeWe=mget(1,'ui',fd);
    MisUnit=mget(1,'uc',fd); //(Vrms,dBV,dBu,dBSpl,dBRel,Ohm,
Deg,ms,dB,Perc,dBmet,dBms2,dBPa,dBPaV,dBms,dBamp,dBSplWm,tCels,Watt)
    Smooth=mget(1,'uc',fd); //(SmNone,Sm48,Sm24,Sm12,Sm6,Sm3,Sm1)
    skip=mget(263,'uc',fd);
    ChirpImpRe=mget(ChirpSize,'f',fd);  //impulse data, real part
    ChirpImpIm=mget(ChirpSize,'f',fd);  //impulse data, imaginary part
    mclose(fd);
endfunction
```

```
function [FFTSize,FCamp,FFTWindow,MisUnit,ATime,
          AFFT]=loadffp(filename)
    // Load a ClioPkt .ffp FFT Response File
    [fd]=mopen(filename,'rb');
    skip=mget(860,'uc',fd);
    FFTSize=mget(1,'ui',fd);
    FCamp=mget(1,'ui',fd);
    FFTWindow=mget(1,'uc',fd);
    skip=mget(8,'uc',fd);
    MisUnit=mget(1,'uc',fd); //(Vrms,dBV,dBu,dBSpl,dBRel,Ohm,
Deg,ms,dB,Perc,dBmet,dBms2,dBPa,dBPaV,dBms,dBamp,dBSplWm,tCels,Watt)
    skip=mget(10,'uc',fd);
    Smooth=mget(1,'uc',fd); //(SmNone,Sm48,Sm24,Sm12,Sm6,Sm3,Sm1)
    skip=mget(336,'uc',fd);
    AFFT=mget(FFTSize,'f',fd);
    ATime=mget(FFTSize,'f',fd);
    mclose(fd);
endfunction
```

IMPORT EXAMPLES

Loudspeaker response

In this first example we show the usage of the `loadcrp` Scilab function. We load the .crp file of a measured impulse response of a loudspeaker.

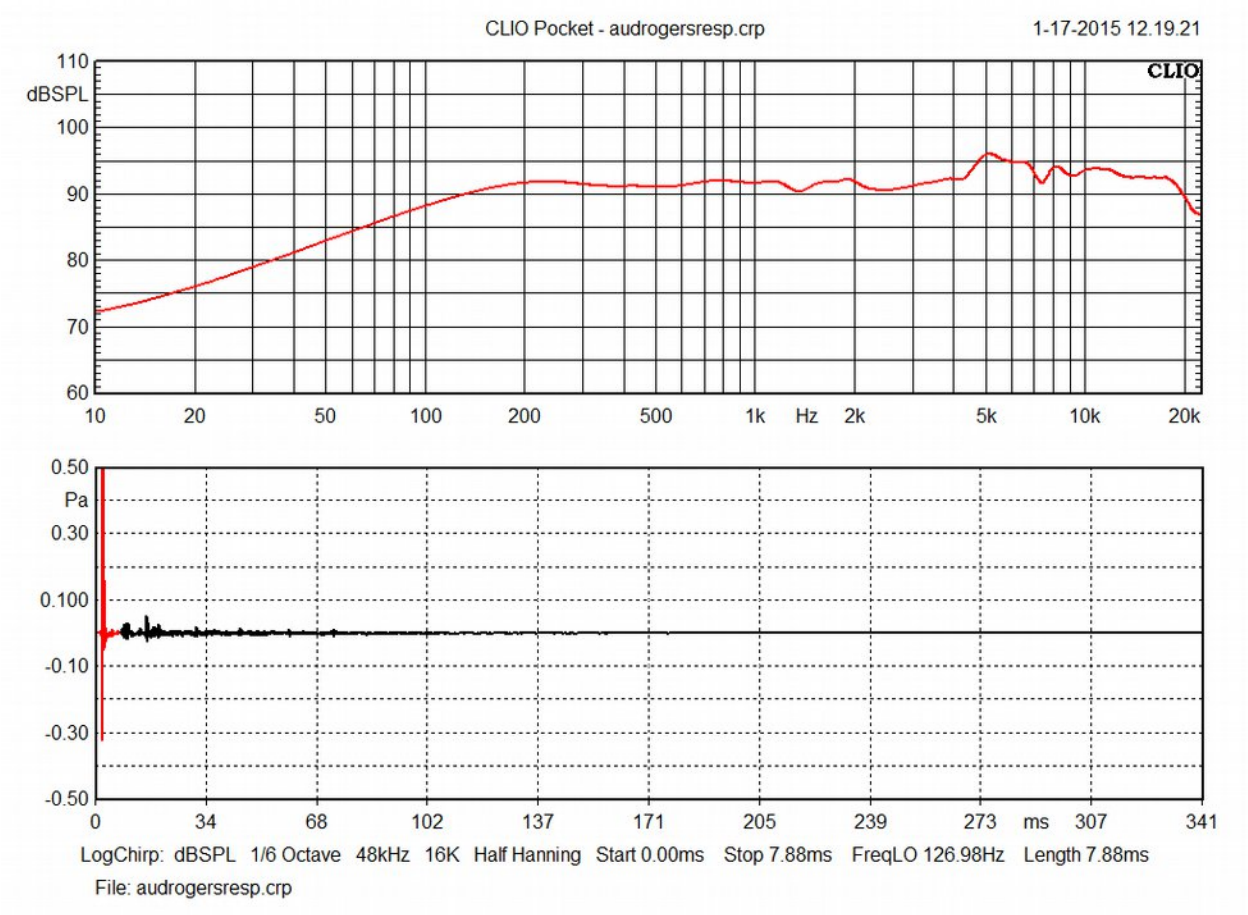


Figure 1: CLIO pocket LogChirp measurement

The `loadcrp` function should be already loaded in Scilab memory.

The command can be invoked from command line or inside an .sci script:

```
--> [NumCh,ChirpSize,FCamp,TimeW,TimeWb,TimeWe,MisUnit,Smooth,
ChirpImpRe,ChirpImpIm]=loadcrp('audrogersresp.crp');
```

As a result the `ChirpImpRe` and `ChirpImpIm` vectors are the Real and Imaginary part of the impulse response. Unlike CLIO 11, **CLIO pocket does not store the complex frequency response inside the .crp files**. The response is calculated each time it is needed starting from the impulse response, the selected window function and then applying other frequency related operations such as smoothing.

The vector with the temporal data of the samples can be easily created with the command:

```
--> tim=0:1/FCamp:(ChirpSize-1)*(1/FCamp);
```

We are interested in the Real part only since we would like to process again the response via FFT in order to get the complex frequency response.

To plot the impulse response graph we can use the command:

```
--> plot2d(tim,ChirpImpRe,2);
```

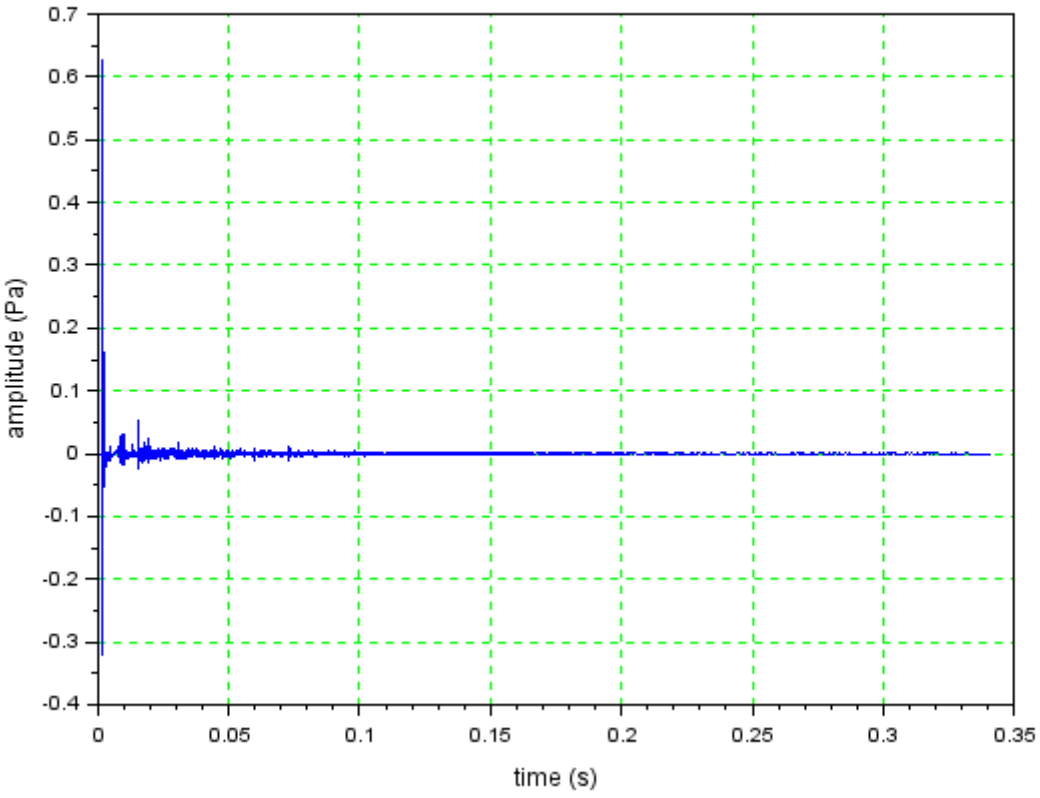


Figure 2: Imported impulse response

CLIO pocket 1.50 features two kind of time windows: **Rectangular** and **Auto Half Hann**. While the Rectangular window has a simple interpretation the Auto Half Hann needs some explanation. Auto Half Hann is a non symmetric window where the first part is rectangular and the second part is Half Hann. The transition point between the two is the peak of the real part of the impulse.

The start and stop indexes of the time window are available in the `TimeWb` and

`TimeWe` variables.

The window type is available in the `TimeW` variable, in case of measurements made with CLIO pocket 1.50 can have only two values:

0 – rectangular window,

1 – Auto Half Hann window.

The imported measurement shown has `TimeW=1`, thus the selected window is Auto Half Hann.

To recreate the same window in Scilab we should find the impulse peak, this can be done with the Scilab command:

```
--> [peak,TimeWp]=max(abs(ChirpImpRe))
 TimeWp  =

   94.

 peak  =

   0.6288845
```

Now it is possible to create the Auto Half Hann window in Scilab, this is a rather simple command thanks to Scilab powerful syntax:

```
--> win_l=window('hn',2*(TimeWe-TimeWp));

--> win_ahh=[ones(1,TimeWp-TimeWb-1) win_l(1,(TimeWe-TimeWp):$)];
```

The time window can be plotted over the impulse response:

```
--> plot2d(tim(1:(TimeWe-TimeWb)),win_ahh,5)
```

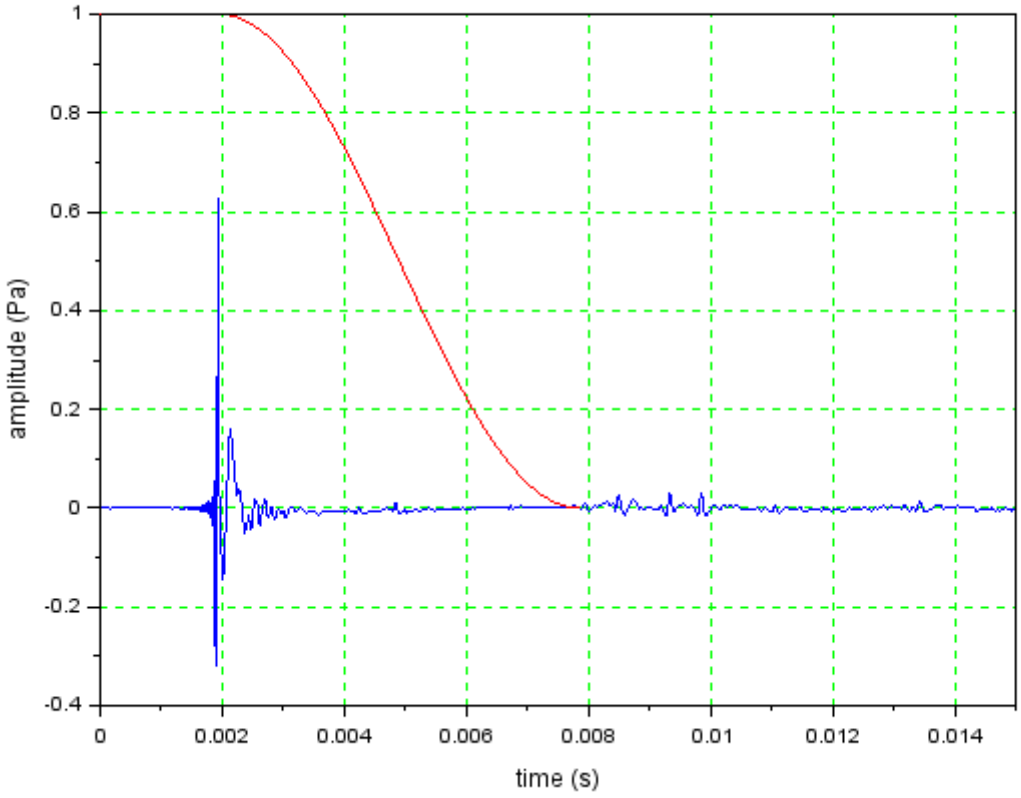


Figure 3: Impulse response (blue) with Auto Half-Hann time window (red)

Now a version of the impulse response with the windowing applied can be created with the command:

```
--> ChirpImpReWin=[ChirpImpRe(TimeWb+1:TimeWe).*win_ahh
zeros(1,ChirpSize-(TimeWe-TimeWb))];
```

Finally this windowed version of the impulse can be transformed via FFT:

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

The frequency points can be calculated with:

```
--> frq=0:(FCamp/ChirpSize):(ChirpSize-1)*(FCamp/ChirpSize);
```

And the complex frequency response can be shown with the command:

```
--> plot2d(frq(1:ChirpSize/2),20*log10(abs(Y(1:ChirpSize/2)))+94,2);
```

After selecting the proper logarithmic scale for the frequency axis the response will look like:

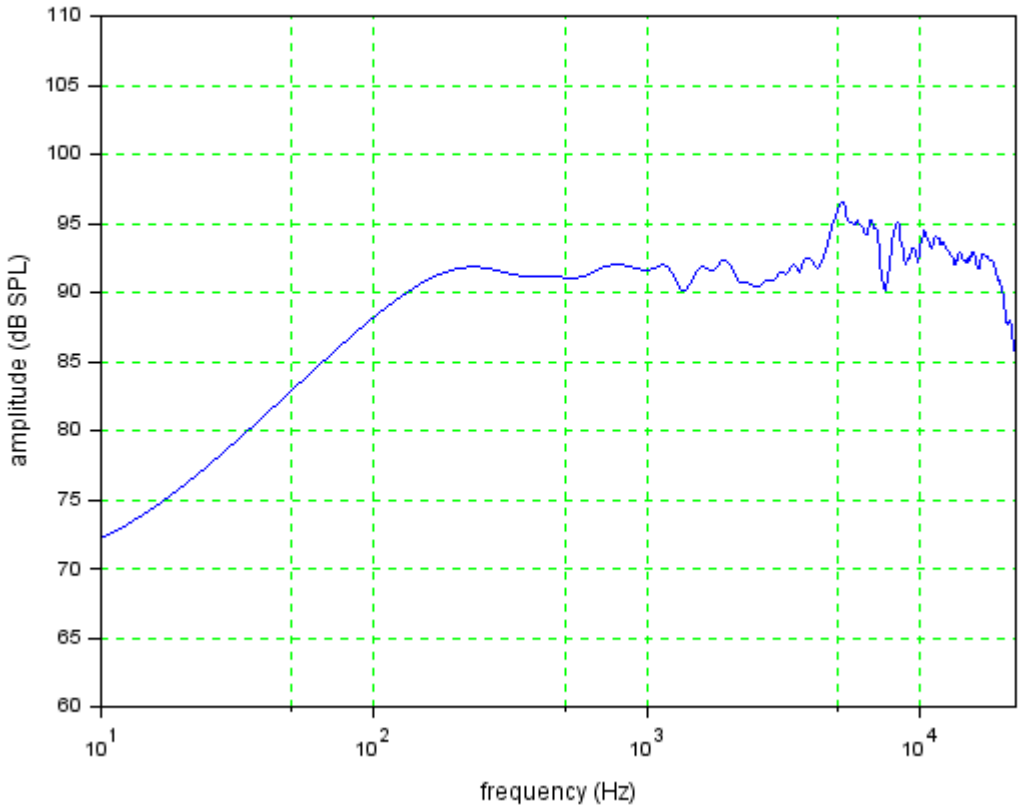


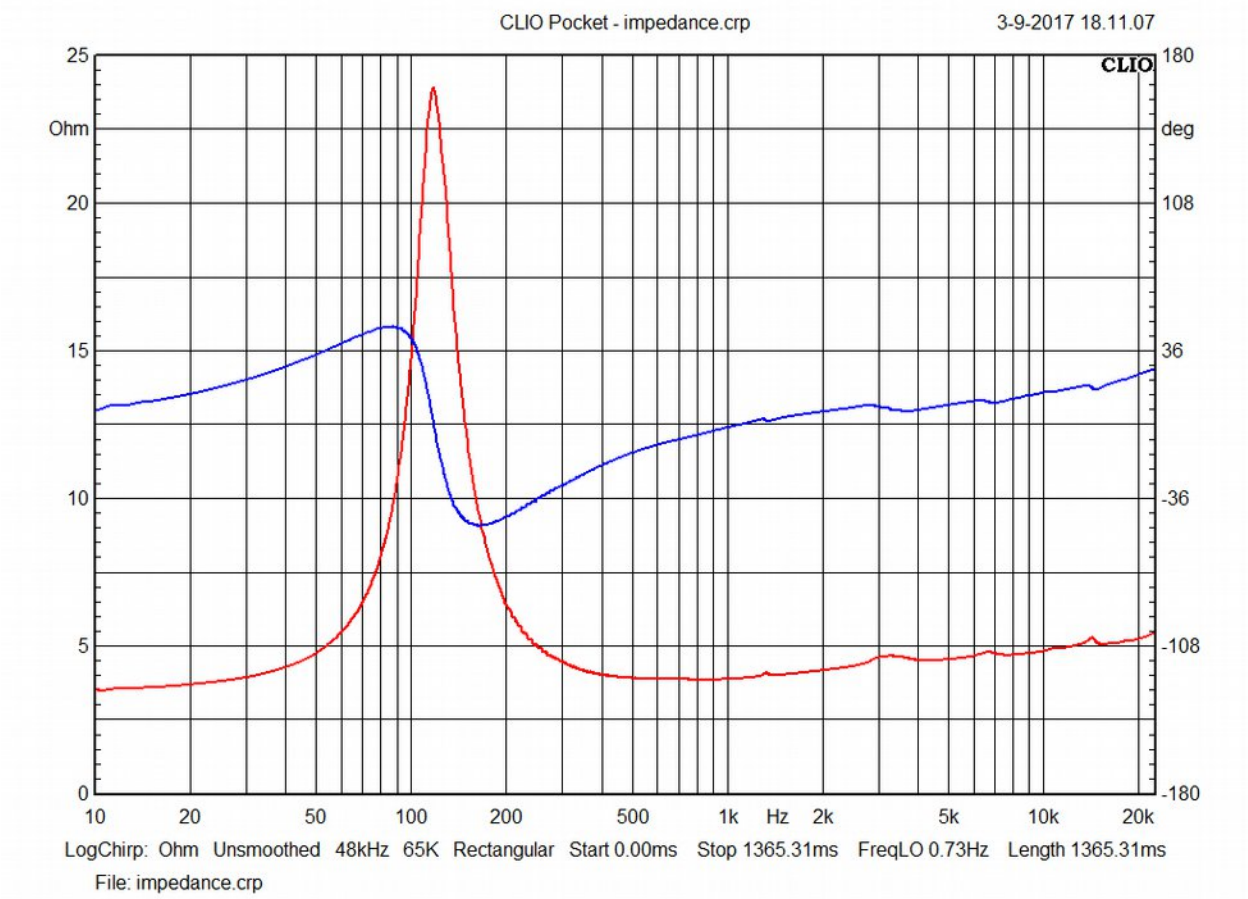
Figure 4: Magnitude of frequency response - unsmoothed

The only difference from the response shown by CLIO pocket now resides in the smoothing applied to the frequency response magnitude data, which is `smooth=4` which corresponds to a sixth of octave.

The frequency smoothing can also be coded into Scilab, but it is omitted here because it will go outside the scope of the present document.

Loudspeaker impedance

In this second example we will load a .crp file of an impedance measurement.




```
--> [NumCh,ChirpSize,FCamp,TimeW,TimeWb,TimeWe,MisUnit,Smooth,
ChirpImpRe,ChirpImpIm]=loadcrp('impedance.crp');
```

In impedance measurements there should be no time windowing applied, thus the FFT can be calculated directly on the `ChirpImpRe` impulse.

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

--> frq=0:(FCamp/ChirpSize):(ChirpSize-1)*(FCamp/ChirpSize);
```

The magnitude and phase can be plotted with the commands:

```
--> plot2d(frq(1:ChirpSize/2),(abs(Y(1:ChirpSize/2))),2);

--> plot2d(frq(1:ChirpSize/2),atan(imag(Y(1:ChirpSize/2))./
real(Y(1:ChirpSize/2)))*(180/%pi),2)
```

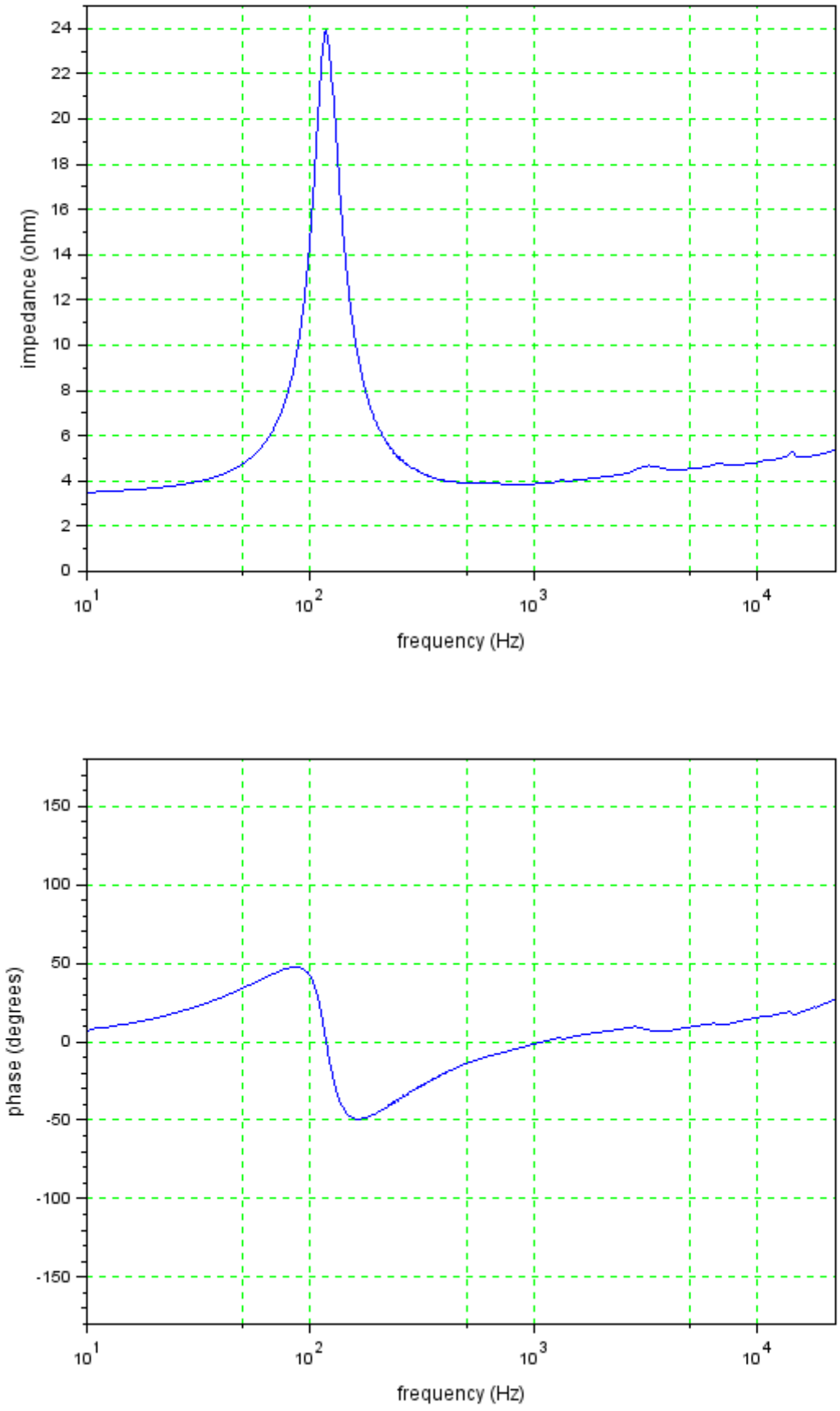


Figure 5: Magnitude and phase impedance plots

FFT of a burst signal

In this third example the acquisition of a burst signal with an .ffp FFT file is shown.

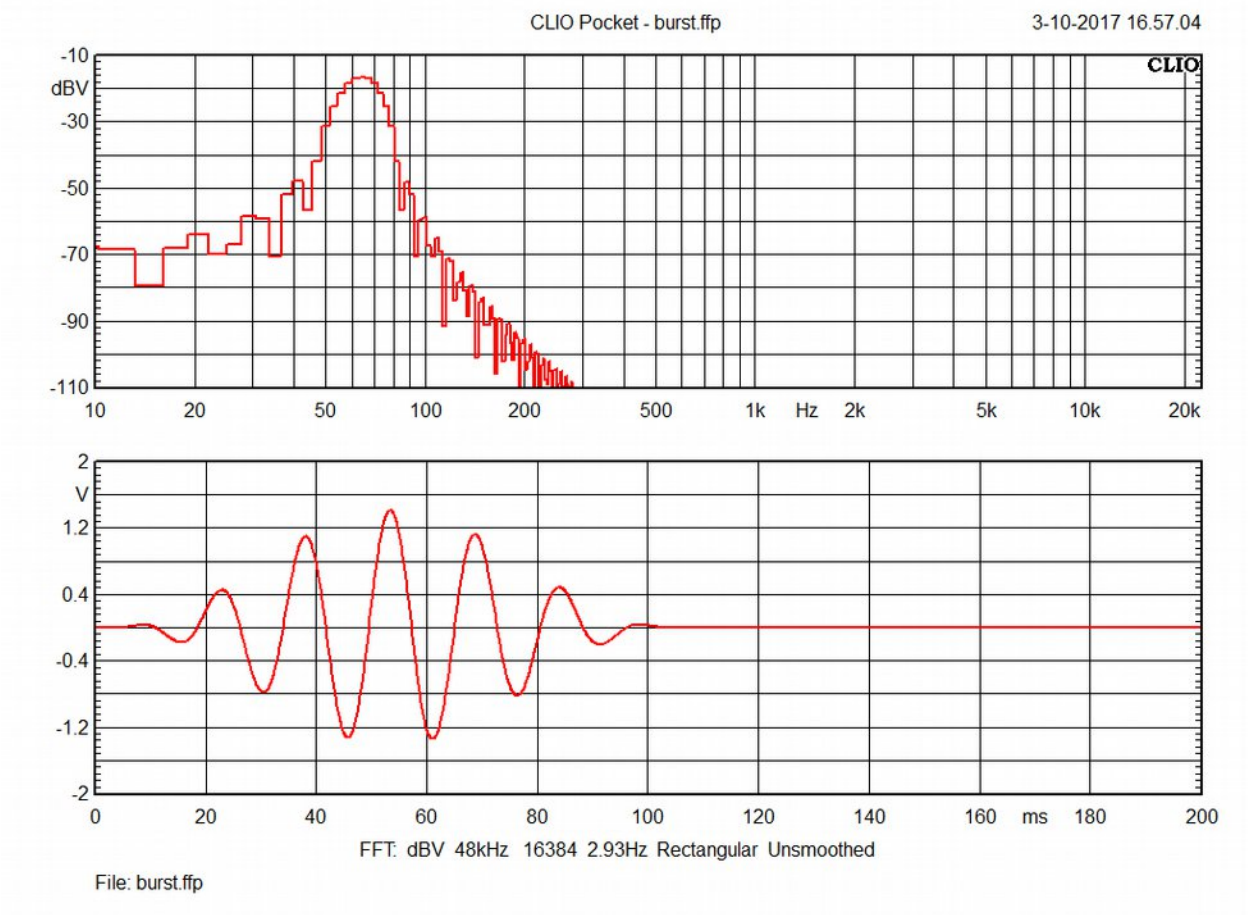


Figure 6: Burst signal FFT acquisition

The file can be loaded into Scilab using the loadffp function:

```
--> [FFTSize,FCamp,FFTWindow,MisUnit,ATime,AFFT]=loadfft('burst.ffp');
```

The data available in a .ffp file are both the time data vector `ATime` and the FFT data vector `AFFT`.

The time data can be plotted using the following commands:

```
--> tim=0:1/FCamp:(FFTSize-1)*(1/FCamp);

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

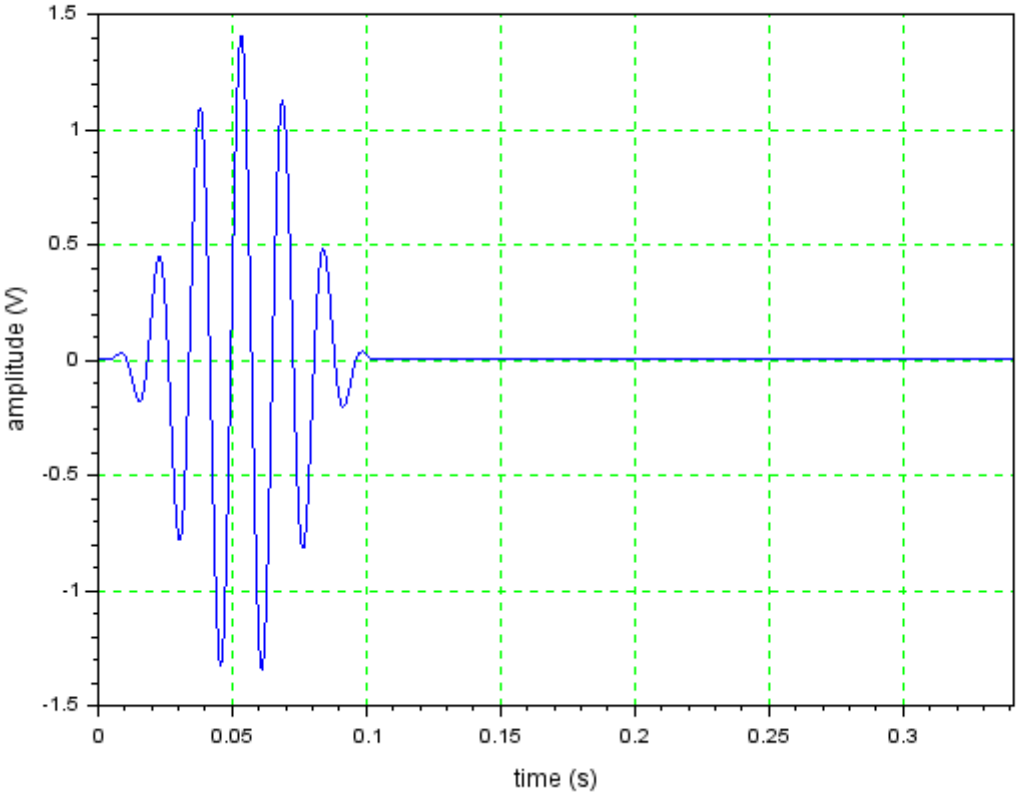


Figure 7: .ffp file time data

The FFT data can be calculated from the above time data, by applying the proper windowing as in FFTWindow, or as an example it is possible to plot the data already calculated by CLIO pocket software:

```
--> frq=0:FCamp/FFTSize:(FCamp/FFTSize)*(FFTSize-1);

--> plot2d(frq(1:FFTSize/2),10*log10(AFFT(1:FFTSize/2)),2);
```

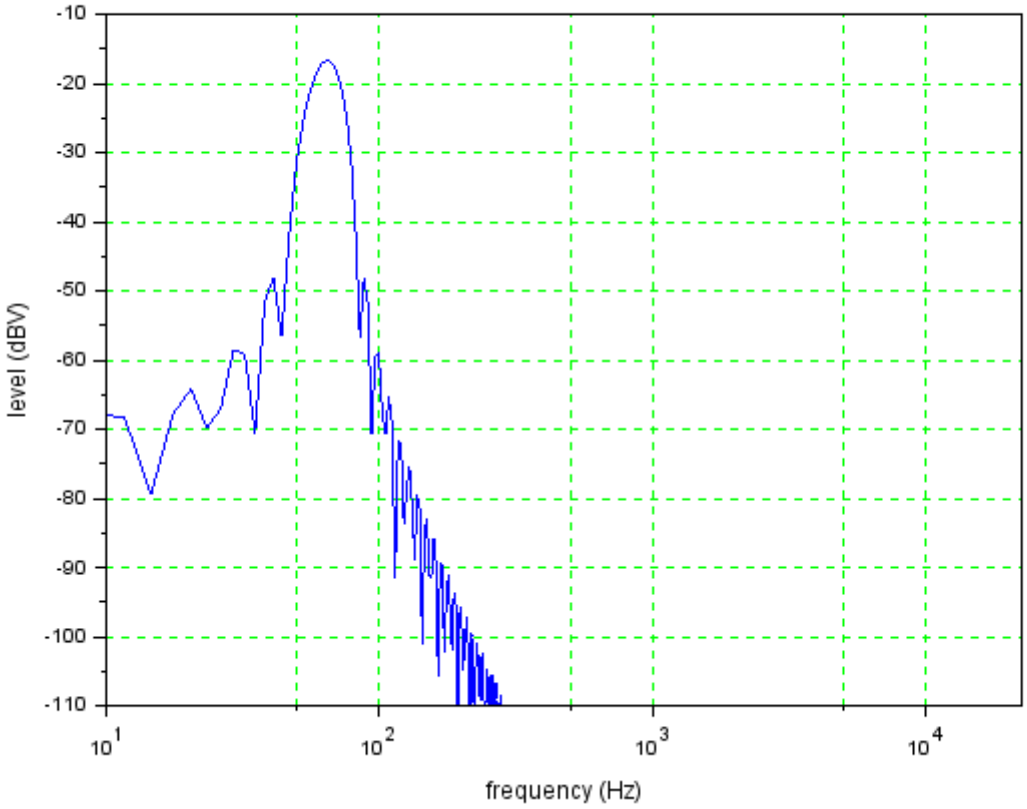


Figure 8: .ffp file FFT frequency data