CREATING SPINORAMA PLOTS FROM POLAR MEASUREMENTS SETS

by Daniele Ponteggia – dp@audiomatica.com

Introduction

The anechoic polar response data of a loudspeaker is a helpful tool to understand the device characteristics and carry out predictions on speaker/room interactions. The polar response is usually measured by sampling the spherical surface around the speaker along specific scan-lines.

The radiation pattern in a given frequency band is usually represented as a 2D polar plot, or as a 3D balloon plot. The data can also be included in specific simulation models such as CLF or EASE GLL and used to predict the loudspeaker/room interaction. Both CLIO and CLIO Pocket are featuring specific tools to carry out polar measurements, produce polar and balloon plots and export data[1] to simulation models.

In this application note we will show a method to import a polar measurement set taken with CLIO or CLIO Pocket in Scilab, and how to create the ANSI/CEA 2034A "Spinorama" plot. This set of frequency responses plot has been developed by Dr. Floyd Toole[2] to rate loudspeaker preference.

ANSI/CEA 2034A

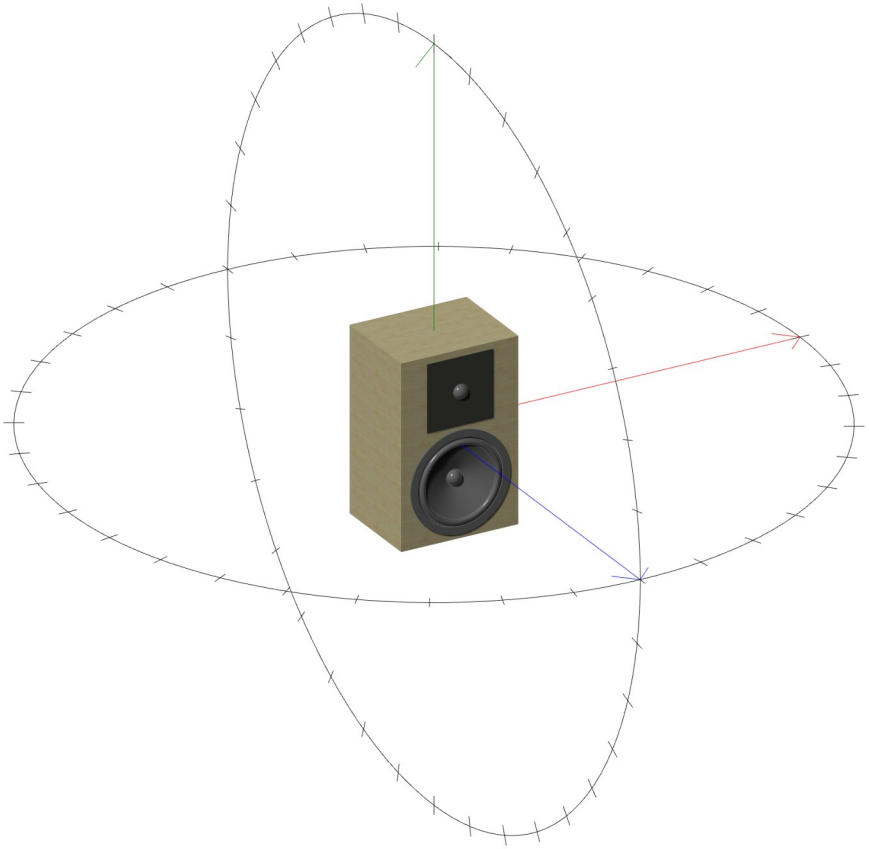
The "Spinorama" plot is a set of frequency responses averaged from specific subsets of polar response data, each subset focusing on a particular loudspeaker/room interaction mechanism. This representation has been included into the ANSI/CEA 2034A "Standard Method of Measurement for In-Home Loudspeakers, where description of the test is clearly stated:

"The objective of these measurements is to be able to anticipate how a loudspeaker might sound in a normally reflective room. No single curve can sufficiently describe how a loudspeaker might sound in a normally reflective room. To better predict its sound, many measurements shall be made at multiple positions surrounding the loudspeaker. These data shall then be processed to estimate the direct, early-reflected and late-reflected sound in a typical small listening room.".

The response shall be measured at 70 positions around the loudspeaker, basically this means gathering the horizontal and vertical polar scan-lines at 10 degrees angular resolution.

1 CLIO QC software version is required to create 3D balloon plots and export data compatible with CLF and EASE formats.

2 A recommended reading is Dr. Floyd Toole's book "Sound Reproduction: loudspeaker and rooms", Focal Press, 2008, but also his JAES papers "Loudspeakers and Rooms for Sound Reproduction-A Scientific Review" appeared in J. Audio Eng. Soc., Vol. 54, No. 6, 2006 June and ""The Detection of Reflections in Typical Rooms," J. Audio Eng. Soc., vol. 37, 1989 July/Aug coauthored by Dr. Sean Olive.



The measurements shall be then averaged as:

1) **Listening Window** (average)

• on-axis, ± 10º vertical, ± 10º, ± 20º, ± 30º horizontal

2) **Early Reflections** (average):

• Floor Bounce: 20º, 30º, 40º down

• Ceiling Bounce: 40º, 50º, 60º up

• Front Wall Bounce: 0º, ± 10º, ± 20º, ± 30º horizontal

• Side Wall Bounces: ± 40º, ± 50º, ± 60º, ± 70º, ± 80º horizontal

• Rear Wall Bounces: 180º, ± 90º horizontal

3) **Vertical Reflections** (average):

• Floor Reflection: - 20°, - 30°, - 40° vertical

• Ceiling Reflection: + 40°, + 50°, + 60° vertical

4) **Horizontal Reflections** (average);

Front: 0°, ± 10º, ± 20º, ± 30º horizontal

• Side: ± 40°, ± 50°, ± 60°, ± 70°, ± 80° horizontal

• Rear: ± 90°, ± 100°, ± 110°, ± 120°, ± 130°, ± 140°, ± 150°, ± 160°, ± 170°, 180° horizontal

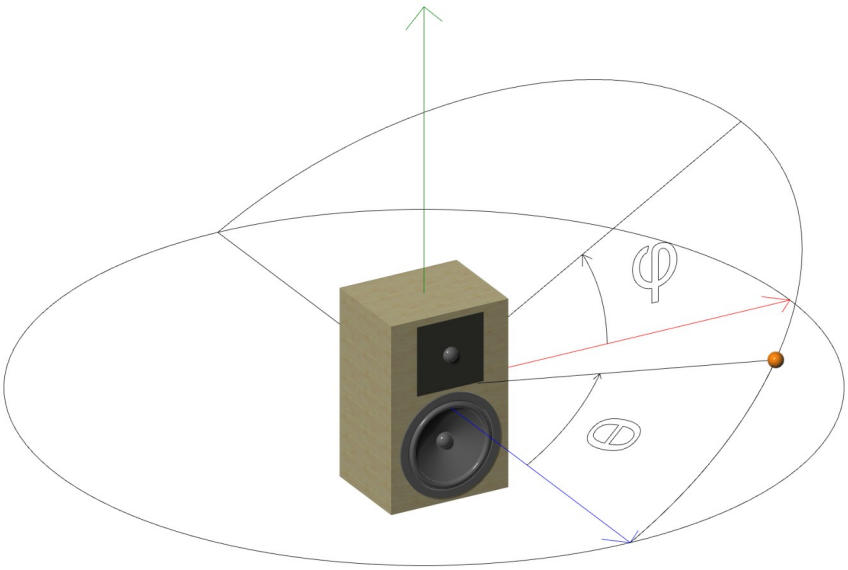
5) **Sound Power**, average of all 70 measurements weighted by solid angle associated to each spherical slice.

The **directivity index of Sound Power and Early Reflections** should also be calculated as the difference with the on-axis curve.

Scilab procedure

We suppose here that the polar measurements have already been carried out with CLIO or CLIO Pocket. With both systems it is possible to manage computer controlled turntables and automatically perform polar measurement sets. Instruction on how to connect turntables and carry out automatic sets of polar measurements are shown in system's manuals.

At the end of the measurement procedure you should have a folder with 70 measurement files, .mls in case of CLIO and .crp in case of CLIO Pocket, named with the CLIO polar naming convention. Let's start with an example featuring a set of CLIO .mls measurements[3], the CLIO naming convention uses two numbers in the file name which are phi (elevation) and theta (rotation) angles each multiplied by 100 to allow decimal degrees without the need of points in the file name string.



filename <phi angle*100> <theta angle*100>.mls

The horizontal polar will be:

Filename 0 -18000.mls

Filename 0 -17500.mls

[…]

Filename 0 0.mls

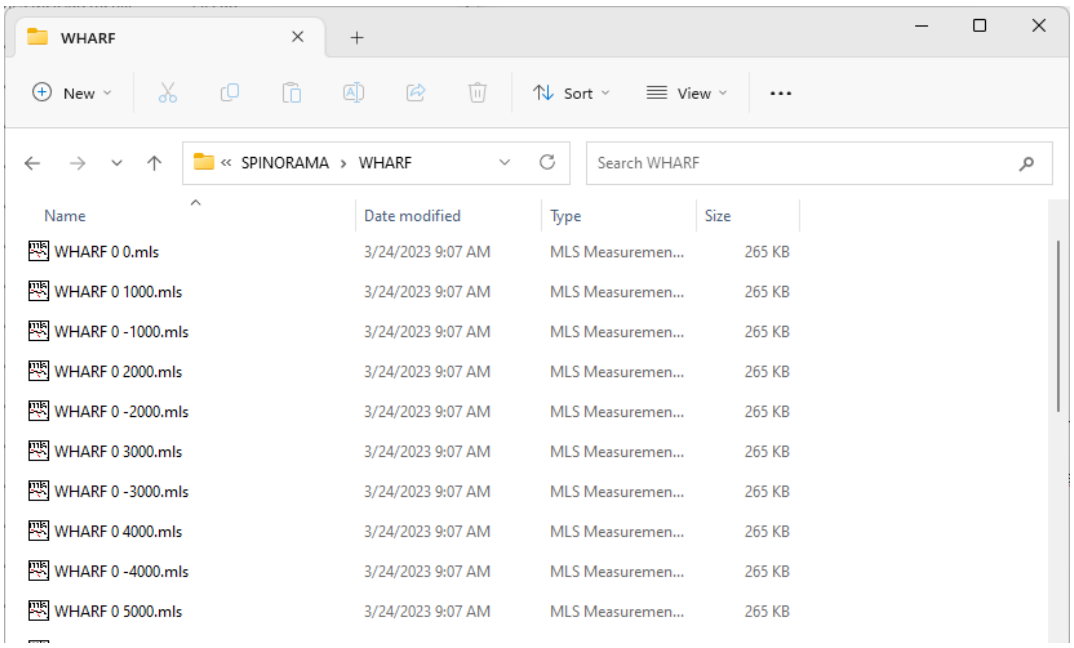
Filename 0 500.mls

[…]

Filename 0 18000.mls

The vertical polar will be similar, with 9000 instead 0 as first index.

3 An example featuring CLIO Pocket .crp files will be shown in appendix B.



Importing CLIO data in Scilab

We will treat here the case of .mls measured data collected with CLIO 12.

A function to import .mls files in Scilab is already available on our application note 21 "Import CLIO 12 Binary Files with SCILAB"[4]:

```
function [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
```

4 https://www.audiomatica.com/wp/wp-content/uploads/Appnote_021.pdf

The function opens the .mls binary file and returns:

- **`MLSsize`** size of the analysis
- **`Fcamp`** sampling frequency
- **`TimeW`** time window shape (0=rectangular, 1=Half Hanning, 2=Hanning, 3=Half Blackmann-Harris, 4=Blackmann-Harris)
- **`TimeWb, TimeWe`** indexes of the first and last sample of the time window
- **`frq`** vector of size (1,MLSsize) with frequency value of FFT samples
- **`tim`** vector of size (1,MLSsize) with time value of Impulse Response samples
- **`frqdata`** vector of size (1,MLSsize) of complex values with the FFT data, computed with time window applied
- **`timdata`** vector of size (1,MLSsize) of complex values with IR data, non windowed

What we retrieve from an .mls files is the raw measured data. The **`frqdata`** is the FFT output with time windowing applied, whilst the **`timdata`** vector is stored in the .mls file without any windowing or other processing.

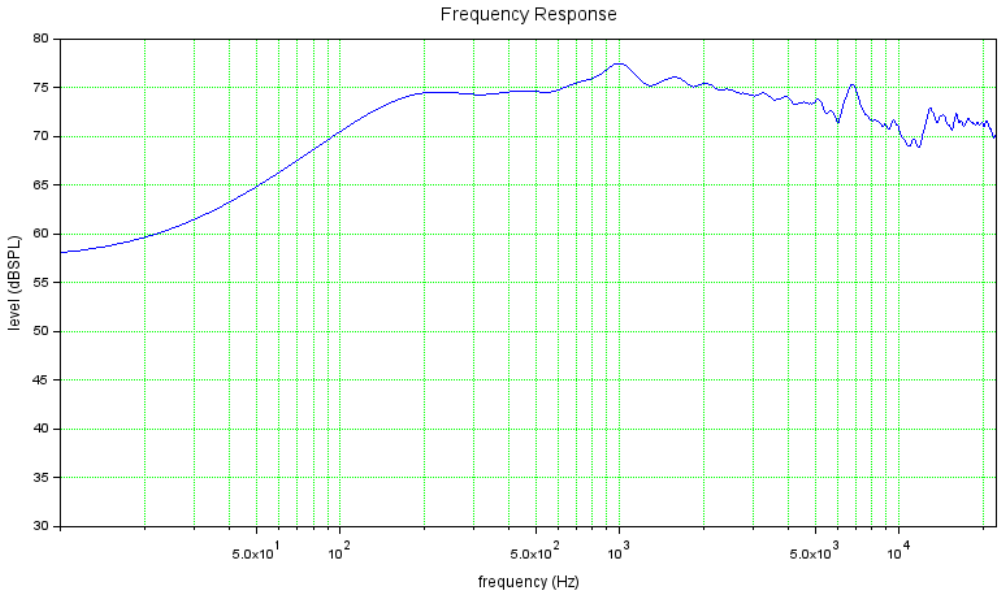
We could have used the .txt export from CLIO instead, which features data points conveniently spaced on logarithmic scale, and this is also a viable solution. But in our case we would like to have more flexibility and thus use linearly spaced data from the **`frqdata`** vector.

As an example we might import the on-axis response, in this example we are using a polar file-set of a small 2-way loudspeaker, with file named “WHARF 0 0.mls”:

```
-->[MLSsize, Fcamp, TimeW, TimeWb, TimeWe, frq, tim, frqdata, timdata]=
readMLS('WHARF 0 0.mls');
```

The frequency response can be plotted using the commands:

```
-->plot2d(frq(2:MLSsize/2),20*log10(abs(frqdata(2:MLSsize/2)))+94,2);
a=gca();
a.log_flags="lnn";
a.box="on";
a.tight_limits="on";
a.grid=[3 3];
a.data_bounds=[10 30; 22400 80];
title('Frequency Response');
xlabel('frequency (Hz)');
ylabel('level (dBSPL)');
```



Please note that we are limiting the plot of our vectors in the range from index 2 to MLSsize/2 since we are interested only on the FFT positive frequency axis and we would avoid DC (zero frequency) to plot on a log-scale frequency graph. We also applied a +94 dB factor because measurements are in dBSPL referenced to 20 uPa.

Using commands as those shown before it is possible to import ad process measured data sets at our wish.

In this case we want to load and average data to create the Spinorama curves. Before going into this we might add two more functions to manage and present the data in a more convenient fashion: resample from linear to log-scale and smoothing.

Resampling FFT data from linear to log-scale

Frequency data points from an FFT are linearly spaced with a frequency step which is equal to the sampling frequency divided by the FFT length in samples. As an example if sampling frequency **`Fcamp`** is 48000 and **`MLSsize`** is 16384 we would have data points linearly spaced with 48000/16384 which is about 2.93 Hz. This is unnecessarily dense at high frequencies and coarse at low frequency when represented on log-scale frequency axis. For the sake of the average curve calculation and representation we might want to reduce the amount of data by interpolation of response curves on log-scale frequency axis ranging from 20 Hz to 20 kHz as an instance.

This can be conveniently done in Scilab using Spline interpolation with the **`splin`** and **`interp`** functions. A simple function to achieve this can be written as:

```
function [outfrq, outresponse]=freqlintolog(frq, inresponse, fstart, fstop, npoints)
    fmult=(fstop/fstart)^(1/(npoints-1));
    outfrq=fmult.^(0:(npoints-1)).*fstart;
    d=splin(frq,inresponse);
    outresponse=interp(outfrq,frq,inresponse,d);
endfunction
```

This function accepts as an input a `frq` vector with the entire axis of linearly spaced frequency points and the level in dB of the response at these points

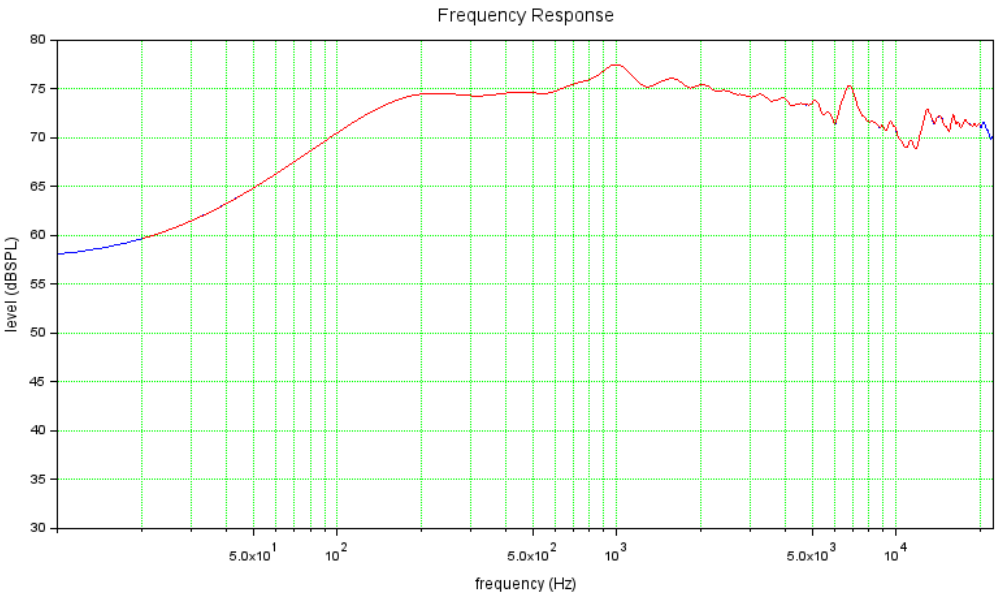
20*log10(abs(frqdata)), log-scale limits **fstart** and **fstop**, number of log-scaled points **npts** and returns two vectors **outfrq** and **outresponse** with the log-spaced frequency points and the value of the level in dB.

Using our previously loaded data we can resample data with 1024 log-spaced points from 20 Hz up to 20 kHz:

```
--> [outfrq,outresponse]=freqlintolog(frq,20*log10(abs(frqdata)),20,20000
,1024);
```

The downsampled response data can be plotted alongside the original data:

```
--> plot2d(outfrq,outresponse+94,5);
```



Smoothing frequency response in Scilab

We might also want to apply smoothing to the frequency response, once the data is in form of log-spaced frequencies this process is a simple moving average. However, it is necessary to find out the width of the interval of points on which to perform the average in order to obtain a smoothing with a given bandwidth.

If we suppose to apply the smoothing within a range spanning a fraction of octave **octfrac**, we can use the following Scilab function to calculate the averaged points on the same log-spaced points. This function takes advantage of the matrix computation capabilities of Scilab, calculating the **outresponse** vector points as an average of the **inresponse** points around an **octfrac** range:

```
function [outresponse]=smoothresponse(frq,inresponse,octfrac);
    frq1=frq.*2^(-1/octfrac/2);
    frq2=frq.*2^(1/octfrac/2);
    FRQ=repmat(frq,size(frq,2),1);
    FRQ1=repmat(frq1',1,size(frq1,2));
    FRQ2=repmat(frq2',1,size(frq2,2));
    COND=((FRQ<FRQ2) & (FRQ>=FRQ1));
    RESPONSE=repmat(inresponse,size(inresponse,2),1);
    PRESPONSE=10.^(RESPONSE/20);
    poutresponse=sum(PRESPONSE.*COND,2)./sum(COND,2);
    outresponse=20.*log10(poutresponse);
endfunction
```

Using the above smoothing function it is possible as an example to smooth at 1/3rd

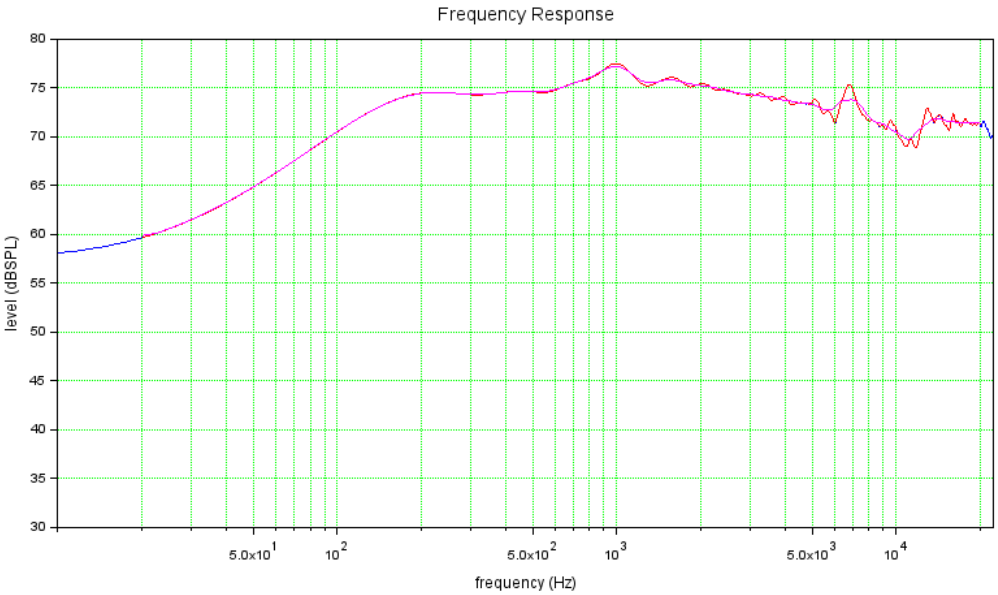
of octave our previously loaded data:

```
--> [outresponsesmooth]=smoothresponse(outfrq,outresponse,3);
```

followed by a:

```
--> plot2d(outfrq,outresponsesmooth+94,6);
```

will add the smoothed log-spaced data to our plot as the purple curve:



Spinorama Scilab procedure

Now that we have all the basic functions, it is possible to load all the 70 polar responses, reduce each response data to log-scale, smooth and finally calculate the averages and plot the results. Please see Appendix A for the complete listing of the Scilab procedure, here we will briefly describe how the script works.

At the beginning of the script few variables are defined, and these are the one which are meant to be modified by the user:

- **`rootfilename`** is the file name root which is common to all the measurement files
- **`octfrac`** is the fraction of octave of the applied smoothing
- **`npts`** is the number of points which are resampled in the 20 Hz to 20 kHz range

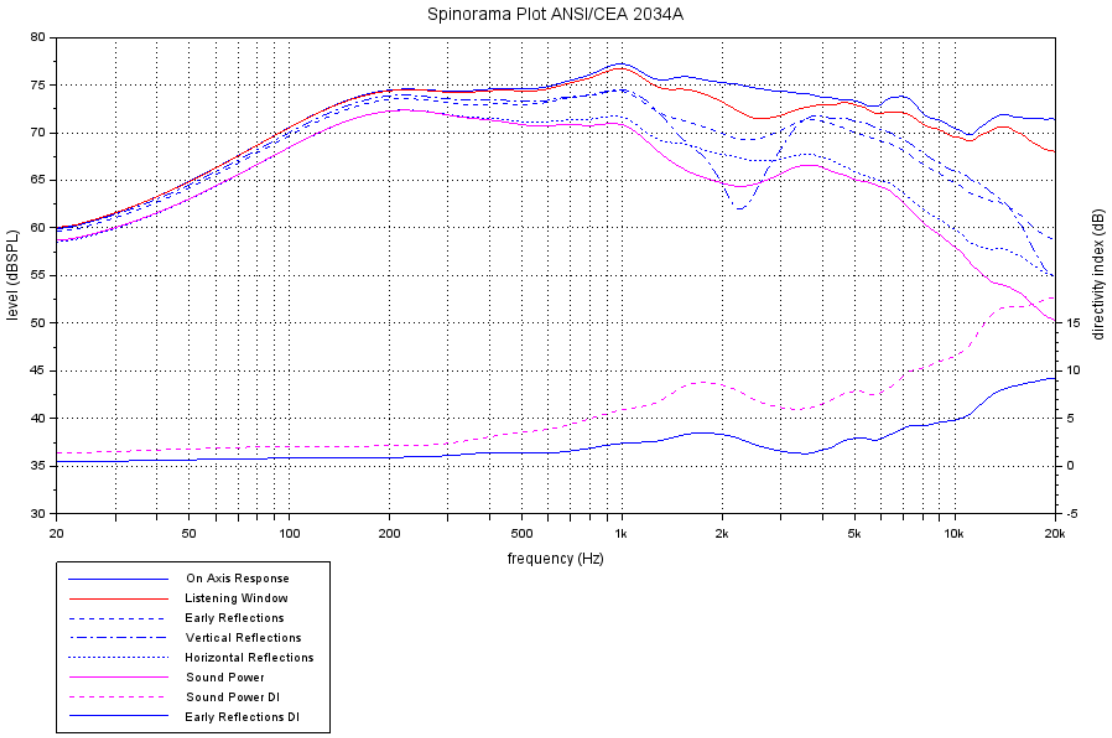
Then the scripts proceed defining the functions described earlier, the sound power weighting values as reported in the ANSI/CEA 2034A standard are also set in a weighting vector **`wval`**.

Horizontal and vertical polar data is then loaded through two separate for statements[5]. Each measurement is loaded using the **`readMLS`** function, then the data is resampled at **`npts`** log-spaced frequency points using **`freqlintolog`** function and finally smoothing is applied by **`smoothresponse`** function.

5 The script does not exploit any possible symmetry property of the device under test, which might reduce the amount of data to be imported. A possible modification of the script to manage symmetry should be introduced at this stage.

Data is stored in two matrices **`horfrqdata`** and **`verfrqdata`** both with size (37:npts) which are then used to calculate averaged curves and sound power. Each row of the matrices holds the frequency response at a given angle, as an example **`horfrqdata(19,:)`** represents the vector with the on-axis response.

The script then calculates the average and the DI responses as requested by the standard, and finally creates a plot with all the calculated curves.



The script should be easily modified in case of a customization of data calculation or graphical presentation, if needed.

Conclusions

This application note shows a simple method to create ANSI/CEA 2034A compliant Spinorama plots starting from a polar measurement set carried out with CLIO or CLIO Pocket.

The Scilab code introduced can also be used for other similar applications where CLIO measured responses need to be averaged and/or combined.

Appendix A

Scilab procedure to create Spinorama plots from sets of CLIO .mls measurements

```
// ANSI/CEA 2034A SPINORAMA PLOT
// from a set of 70 CLIO .mls polar measurements
// ---------------------------------------------
//
// v1.00 - 12/04/2023
// dp@audiomatica.com
//

clear
funcprot(0);

// script variables
rootfilename='WHARF';
octfrac=3;              // fraction of octave smoothing
npts=1024;              // log spaced frequency points between 20 Hz and 20
kHz to be calculated

// define functions
function [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

function [outfrq,outresponse]=freqlintolog(frq,inresponse,fstart,fstop,
npoints)
    fmult=(fstop/fstart)^(1/(npoints-1));
    outfrq=fmult.^(0:(npoints-1)).*fstart;
    d=splin(frq,inresponse);
    outresponse=interp(outfrq,frq,inresponse,d);
```

```
endfunction

function [outresponse]=smoothresponse(frq,inresponse,octfrac);
    frq1=frq.*2^(-1/octfrac/2);
    frq2=frq.*2^(1/octfrac/2);
    FRQ=repmat(frq,size(frq,2),1);
    FRQ1=repmat(frq1',1,size(frq1,2));
    FRQ2=repmat(frq2',1,size(frq2,2));
    COND=((FRQ<FRQ2) & (FRQ>=FRQ1));
    RESPONSE=repmat(inresponse,size(inresponse,2),1);
    PRESPONSE=10.^(RESPONSE/20);
    poutresponse=sum(PRESPONSE.*COND,2)./sum(COND,2);
    outresponse=20.*log10(poutresponse);
endfunction

// Sound Power Weighting Values from ANSI/CEA 2034A appendix C
// with 10 degrees resolution
wval=[
0.000604486
0.004730189
0.008955027
0.012387354
0.014989611
0.016868154
0.018165962
0.019006744
0.019477787
0.019629373
0.019477787
0.019006744
0.018165962
0.016868154
0.014989611
0.012387354
0.008955027
0.004730189
0.000604486
0.004730189
0.008955027
0.012387354
0.014989611
0.016868154
0.018165962
0.019006744
0.019477787
0.019629373
0.019477787
0.019006744
0.018165962
0.016868154
0.014989611
0.012387354
0.008955027
0.004730189
0.000604486
]'

// end functions ----------------------

// LOAD DATA
disp("Load data...");

// load H and V data 72 points at 10 degree resolution
// i=0 theta=-180, i=19 theta=0, i=37 theta=180
```

```
// load horizontal data (no symmetry)
for i=1:37
    filename=rootfilename+' 0 '+string(100*(i-19)*10)+'.mls';
    disp("Load "+filename);
    [MLSsize, Fcamp, TimeW, TimeWb, TimeWe, frq, tim, frqdata,
timdata]=readMLS(filename);
    frqns=frq(2:MLSsize/2);
    respns=20*log10(abs(frqdata(2:MLSsize/2)));
    [outfrq,outresponse]=freqlintolog(frq,20*log10(abs(frqdata)),20,20000,
npts);
    [outresponsesmooth]=smoothresponse(outfrq,outresponse,octfrac);
    horfrqdata(i,:)=outresponsesmooth;
end

// load vertical data (no symmetry)
for i=1:37
    filename=rootfilename+' 9000 '+string(100*(i-19)*10)+'.mls';
    disp("Load "+filename);
    [MLSsize, Fcamp, TimeW, TimeWb, TimeWe, frq, tim, frqdata,
timdata]=readMLS(filename);
    frqns=frq(2:MLSsize/2);
    respns=20*log10(abs(frqdata(2:MLSsize/2)));
    [outfrq,outresponse]=freqlintolog(frq,20*log10(abs(frqdata)),20,20000,
npts);
    [outresponsesmooth]=smoothresponse(outfrq,outresponse,octfrac);
    verfrqdata(i,:)=outresponsesmooth;
end

// ON AXIS response
disp("Calculate ON AXIS response...");

responseOA=horfrqdata(19,:);

// calculate LISTENING WINDOW curve
disp("Calculate LISTENING WINDOW response...");

// V 0, +10, -10
// H 0, +10, +20, +30, -10, -20, -30
hind=[18 19 20];
vind=[16 17 18 19 20 21 22];

appo=zeros(1,npts);
for i=1:size(hind,2)
    pappo=10.^(horfrqdata(hind(i),:)/20);
    appo=appo+pappo;
end
for i=1:size(vind,2)
    pappo=10.^(verfrqdata(vind(i),:)/20);
    appo=appo+pappo;
end
appo=appo./(size(hind,2)+size(vind,2));
responseLW=20.*log10(appo);

// calculate EARLY REFLECTIONS curve
disp("Calculate EARLY REFLECTIONS response...");

// V -20, -30, -40, 40, 50, 60
// H 0, +10, +20, +30, +40, +50, +60, +70, +80, +90, -10, -20, -30, -40, -50,
-60, -70, -80, -90, 180
hind=[10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 37];
vind=[15 16 17 23 24 25];

appo=zeros(1,npts);
for i=1:size(hind,2)
```

```
    pappo=10.^(horfrqdata(hind(i),:)/20);
    appo=appo+pappo;
end
for i=1:size(vind,2)
    pappo=10.^(verfrqdata(vind(i),:)/20);
    appo=appo+pappo;
end
appo=appo./(size(hind,2)+size(vind,2));
responseER=20.*log10(appo);

// calculate VERTICAL REFLECTIONS curve
disp("Calculate VERTICAL REFLECTIONS response...");

// V -20, -30, -40, +40, +50, +60
vind=[15 16 17 23 24 25];

appo=zeros(1,npts);
for i=1:size(vind,2)
    pappo=10.^(verfrqdata(vind(i),:)/20);
    appo=appo+pappo;
end
appo=appo./(size(vind,2));
responseVR=20*log10(appo);

// calculate HORIZONTAL REFLECTIONS curve
disp("Calculate HORIZONTAL REFLECTIONS response...");

// H use all horizontal data in 10 degree steps
hind=2:37;

appo=zeros(1,npts);
for i=1:size(hind,2)
    pappo=10.^(horfrqdata(hind(i),:)/20);
    appo=appo+pappo;
end
appo=appo./(size(hind,2));
responseHR=20*log10(appo);

// calculate SOUND POWER curve
disp("Calculate SOUND POWER response...");

// using weigthed values according to table
hind=2:37;
vind=[2:18 20:36];

appo=zeros(1,npts);
for i=1:size(hind,2)
    pappo=wval(hind(i)).*10.^(horfrqdata(hind(i),:)/20);
    appo=appo+pappo;
end
for i=1:size(vind,2)
    pappo=wval(vind(i)).*10.^(verfrqdata(vind(i),:)/20);
    appo=appo+pappo;
end
//appo=appo./(size(hind,2)+size(vind,2));
responseSP=20*log10(appo);

// calculate SOUND POWER DIRECTIVITY INDEX (SPDI)
disp("Calculate DIRECTIVITY INDEXES...");

dirindSP=responseLW-responseSP;

// calculate EARLY REFLECTIONS DIRECTIVITY INDEX (ERDI)
dirindER=responseLW-responseER;
```

```
// plot all response curves
f1=scf()
f1.figure_size=[1024 768];
plot2d(outfrq,responseOA+94,2)
plot2d(outfrq,responseLW+94,3)
plot2d(outfrq,responseER+94,4)
plot2d(outfrq,responseVR+94,5)
plot2d(outfrq,responseHR+94,6)
plot2d(outfrq,responseSP+94,7)
a1=gca()
a1.log_flags="lnn"
a1.tight_limits="on";
a1.box="on";
a1.grid=[1 1];
a1.grid_style=[9 9]
ymax=ceil((max([responseLW responseER responseVR responseHR responseSP])
+94)/10)*10;
a1.data_bounds=[20 ymax-50; 20000 ymax];
title("Spinorama Plot ANSI/CEA 2034A");
xlabel("frequency (Hz)")
ylabel("level (dBSPL)")
afrq=[20 30 40 50 60 70 80 90 100 200 300 400 500 600 700 800 900 1000 2000
3000 4000 5000 6000 7000 8000 9000 10000 20000]';
astr=["20" "" "" "50" "" "" "" "" "100" "200" "" "" "500" "" "" "" "" "1k"
"2k" "" "" "5k" "" "" "" "" "10k" "20k"]';
a1.x_ticks = tlist(["ticks", "locations", "labels"], afrq, astr);
a1.sub_tics=[0 1];

// plot DI curves on secondary vertical axis
a2=newaxes()
// faux curves labels
plot2d(1:2,1:2,2);
plot2d(1:2,1:2,3);
plot2d(1:2,1:2,4);
plot2d(1:2,1:2,5);
plot2d(1:2,1:2,6);
plot2d(1:2,1:2,7);
plot2d(outfrq,dirindSP,2)
plot2d(outfrq,dirindER,5)
a2.filled = 'off';
a2.tight_limits = 'on';
a2.y_location="right"
a2.log_flags="lnn";
a2.data_bounds=[20 -5; 20000 45];
ylabel("directivity index (dB)")
a2.x_ticks = tlist(["ticks", "locations", "labels"], [ ]', [ ]');
a2.y_ticks = tlist(["ticks", "locations", "labels"], [-5 0 5 10 15]', ["-5"
"0" "5" "10" "15"]');

// set curves colors and styles
cOA=a1.children(6).children(1); // ON AXIS RESPONSE
cLW=a1.children(5).children(1); // LISTENING WINDOW
cER=a1.children(4).children(1); // EARLY REFLECTIONS
cVR=a1.children(3).children(1); // VERTICAL REFLECTIONS
cHR=a1.children(2).children(1); // HORIZONTAL REFLECTIONS
cSP=a1.children(1).children(1); // SOUND POWER
cSPDI=a2.children(2).children(1); // SOUND POWER DI
cERDI=a2.children(1).children(1); // EARLY REFLECTIONS DI
cOA.foreground=2;
cOA.line_style=1;
cLW.foreground=5;
cLW.line_style=1;
cER.foreground=2;
```

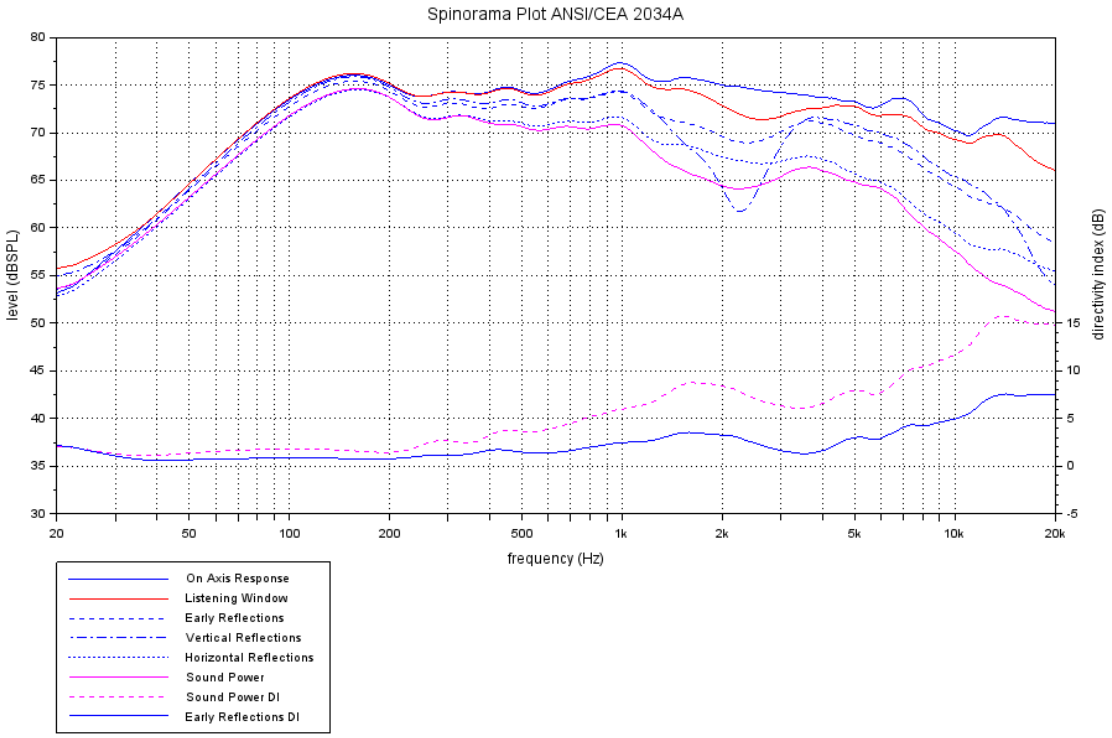
```
cER.line_style=3;
cVR.foreground=2;
cVR.line_style=6;
cHR.foreground=2;
cHR.line_style=8;
cSP.foreground=6;
cSP.line_style=1;
cSPDI.foreground=6;
cSPDI.line_style=3;
cERDI.foreground=2;
cERUDI.line_style=4;

// set also for the legend
// this is an hack because apparently there is no other
// way to manage a legend with multiple axes
c2OA=a2.children(8).children(1); // ON AXIS RESPONSE
c2LW=a2.children(7).children(1); // LISTENING WINDOW
c2ER=a2.children(6).children(1); // EARLY REFLECTIONS
c2VR=a2.children(5).children(1); // VERTICAL REFLECTIONS
c2HR=a2.children(4).children(1); // HORIZONTAL REFLECTIONS
c2SP=a2.children(3).children(1); // SOUND POWER
c2OA.foreground=cOA.foreground;
c2OA.line_style=cOA.line_style;
c2LW.foreground=cLW.foreground;
c2LW.line_style=cLW.line_style;
c2ER.foreground=cER.foreground;
c2ER.line_style=cER.line_style;
c2VR.foreground=cVR.foreground;
c2VR.line_style=cVR.line_style;
c2HR.foreground=cHR.foreground;
c2HR.line_style=cHR.line_style;
c2SP.foreground=cSP.foreground;
c2SP.line_style=cSP.line_style;

// create legend
legend(["On Axis Response" "Listening Window" "Early Reflections" "Vertical
Reflections" "Horizontal Reflections" "Sound Power" "Sound Power DI" "Early
Reflections DI"]);
a2.children(1).legend_location="lower_caption";

// resize axes to the same dimensions to fit legend
a2.margins=[0.1 0.1 0.1 0.3];
a1.margins=a2.margins;
```

Appendix B

Scilab procedure to create Spinorama plots from sets of CLIO Pocket .crp measurements

This procedure slightly differs from the previous one, as CLIO Pocket binary format differs from the one of CLIO 12.

The CLIO Pocket binary data format is well documented in application note 15 "CLIO Pocket File Structures with Import Examples in SCILAB". The main difference is that the CLIO Pocket binary file stores only the original time data of the Impulse Response and not the calculated FFT data. We then added a function respfromcrp which calculates the frequency response using the windowed IR as saved in the CLIO Pocket binary file.




```
// ANSI/CEA 2034A SPINORAMA PLOT
// from a set of 70 CLIO Pocket .crp polar measurements
// -----------------------------------------------------
//
// v1.00 - 12/04/2023
// dp@audiomatica.com
//

clear
funcprot(0);

// script variables
rootfilename='WHARF';
octfrac=3;              // fraction of octave smoothing
npts=1024;              // log spaced frequency points between 20 Hz and 20
kHz to be calculated

// define functions
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,dB
amp,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 [ChirpFrq,ChirpResponse]=respfromcrp(ChirpImpRe,ChirpSize,
Fcamp,TimeW,TimeWb,TimeWe)
    ChirpFrq=0:FCamp/ChirpSize:(FCamp/ChirpSize).*(ChirpSize-1);
    if TimeW==0 then
        win_rec=[zeros(1,TimeWb) ones(1,TimeWe-TimeWb) zeros(1,ChirpSize-
TimeWe)];
        ChirpImpRe=ChirpImpRe.*win_rec;
    end
    if TimeW==1 then
        [peak,TimeWp]=max(abs(ChirpImpRe));
        win_l=window('hn',2*(TimeWe-TimeWp));
        win_ahh=[ones(1,TimeWp-TimeWb-1) win_l(1,(TimeWe-TimeWp):$)
zeros(1,ChirpSize-TimeWe)];
        ChirpImpRe=ChirpImpRe.*win_ahh;
    end
    Y=fft(ChirpImpRe);
    ChirpResponse=20*log10(abs(Y));
endfunction

function [outfrq,outresponse]=freqlintolog(frq,inresponse,fstart,
fstop,npoints)
    fmult=(fstop/fstart)^(1/(npoints-1));
    outfrq=fmult.^(0:(npoints-1)).*fstart;
    d=splin(frq,inresponse);
    outresponse=interp(outfrq,frq,inresponse,d);
endfunction

function [outresponse]=smoothresponse(frq,inresponse,octfrac);
    frq1=frq.*2^(-1/octfrac/2);
    frq2=frq.*2^(1/octfrac/2);
    FRQ=repmat(frq,size(frq,2),1);
    FRQ1=repmat(frq1',1,size(frq1,2));
    FRQ2=repmat(frq2',1,size(frq2,2));
    COND=((FRQ<FRQ2) & (FRQ>=FRQ1));
    RESPONSE=repmat(inresponse,size(inresponse,2),1);
    PRESPONSE=10.^(RESPONSE/20);
    poutresponse=sum(PRESPONSE.*COND,2)./sum(COND,2);
    outresponse=20.*log10(poutresponse);
endfunction

// Sound Power Weighting Values from ANSI/CEA 2034A appendix C
// with 10 degrees resolution
wval=[
```

```
0.000604486
0.004730189
0.008955027
0.012387354
0.014989611
0.016868154
0.018165962
0.019006744
0.019477787
0.019629373
0.019477787
0.019006744
0.018165962
0.016868154
0.014989611
0.012387354
0.008955027
0.004730189
0.000604486
0.004730189
0.008955027
0.012387354
0.014989611
0.016868154
0.018165962
0.019006744
0.019477787
0.019629373
0.019477787
0.019006744
0.018165962
0.016868154
0.014989611
0.012387354
0.008955027
0.004730189
0.000604486
]'

// end functions ----------------------

// LOAD DATA
disp("Load data...");

// load H and V data 72 points at 10 degree resolution
// i=0 theta=-180, i=19 theta=0, i=37 theta=180
// load horizontal data (no symmetry)
for i=1:37
    filename=rootfilename+' 0 '+string(100*(i-19)*10)+'.crp';
    disp("Load "+filename);

[NumCh,ChirpSize,FCamp,TimeW,TimeWb,TimeWe,MisUnit,Smooth,ChirpImpRe,ChirpImp
Im]=loadcrp(filename);

[ChirpFrq,ChirpResponse]=respfromcrp(ChirpImpRe,ChirpSize,FCamp,TimeW,TimeWb,
TimeWe);
    [outfrq,outresponse]=freqlintolog(ChirpFrq,ChirpResponse,20,20000,npts);
    [outresponsesmooth]=smoothresponse(outfrq,outresponse,octfrac);
    horfrqdata(i,:)=outresponsesmooth;
end

// load vertical data (no symmetry)
for i=1:37
    filename=rootfilename+' 9000 '+string(100*(i-19)*10)+'.crp';
```

```
    disp("Load "+filename);

[NumCh,ChirpSize,FCamp,TimeW,TimeWb,TimeWe,MisUnit,Smooth,ChirpImpRe,ChirpImp
Im]=loadcrp(filename);

[ChirpFrq,ChirpResponse]=respfromcrp(ChirpImpRe,ChirpSize,FCamp,TimeW,TimeWb,
TimeWe);
    [outfrq,outresponse]=freqlintolog(ChirpFrq,ChirpResponse,20,20000,npts);
    [outresponsesmooth]=smoothresponse(outfrq,outresponse,octfrac);
    verfrqdata(i,:)=outresponsesmooth;
end

// ON AXIS response
disp("Calculate ON AXIS response...");

responseOA=horfrqdata(19,:);

// calculate LISTENING WINDOW curve
disp("Calculate LISTENING WINDOW response...");

// V 0, +10, -10
// H 0, +10, +20, +30, -10, -20, -30
hind=[18 19 20];
vind=[16 17 18 19 20 21 22];

appo=zeros(1,npts);
for i=1:size(hind,2)
    pappo=10.^(horfrqdata(hind(i),:)/20);
    appo=appo+pappo;
end
for i=1:size(vind,2)
    pappo=10.^(verfrqdata(vind(i),:)/20);
    appo=appo+pappo;
end
appo=appo./(size(hind,2)+size(vind,2));
responseLW=20.*log10(appo);

// calculate EARLY REFLECTIONS curve
disp("Calculate EARLY REFLECTIONS response...");

// V -20, -30, -40, 40, 50, 60
// H 0, +10, +20, +30, +40, +50, +60, +70, +80, +90, -10, -20, -30, -40, -50,
-60, -70, -80, -90, 180
hind=[10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 37];
vind=[15 16 17 23 24 25];

appo=zeros(1,npts);
for i=1:size(hind,2)
    pappo=10.^(horfrqdata(hind(i),:)/20);
    appo=appo+pappo;
end
for i=1:size(vind,2)
    pappo=10.^(verfrqdata(vind(i),:)/20);
    appo=appo+pappo;
end
appo=appo./(size(hind,2)+size(vind,2));
responseER=20.*log10(appo);

// calculate VERTICAL REFLECTIONS curve
disp("Calculate VERTICAL REFLECTIONS response...");

// V -20, -30, -40, +40, +50, +60
vind=[15 16 17 23 24 25];
```

```
appo=zeros(1,npts);
for i=1:size(vind,2)
    pappo=10.^(verfrqdata(vind(i),:)/20);
    appo=appo+pappo;
end
appo=appo./(size(vind,2));
responseVR=20*log10(appo);

// calculate HORIZONTAL REFLECTIONS curve
disp("Calculate HORIZONTAL REFLECTIONS response...");

// H use all horizontal data in 10 degree steps
hind=2:37;

appo=zeros(1,npts);
for i=1:size(hind,2)
    pappo=10.^(horfrqdata(hind(i),:)/20);
    appo=appo+pappo;
end
appo=appo./(size(hind,2));
responseHR=20*log10(appo);

// calculate SOUND POWER curve
disp("Calculate SOUND POWER response...");

// using weigthed values according to table
hind=2:37;
vind=[2:18 20:36];

appo=zeros(1,npts);
for i=1:size(hind,2)
    pappo=wval(hind(i)).*10.^(horfrqdata(hind(i),:)/20);
    appo=appo+pappo;
end
for i=1:size(vind,2)
    pappo=wval(vind(i)).*10.^(verfrqdata(vind(i),:)/20);
    appo=appo+pappo;
end
//appo=appo./(size(hind,2)+size(vind,2));
responseSP=20*log10(appo);

// calculate SOUND POWER DIRECTIVITY INDEX (SPDI)
disp("Calculate DIRECTIVITY INDEXES...");

dirindSP=responseLW-responseSP;

// calculate EARLY REFLECTIONS DIRECTIVITY INDEX (ERDI)
dirindER=responseLW-responseER;

// plot all response curves
f1=scf()
f1.figure_size=[1024 768];
plot2d(outfrq,responseOA+94,2)
plot2d(outfrq,responseLW+94,3)
plot2d(outfrq,responseER+94,4)
plot2d(outfrq,responseVR+94,5)
plot2d(outfrq,responseHR+94,6)
plot2d(outfrq,responseSP+94,7)
a1=gca()
a1.log_flags="lnn"
a1.tight_limits="on";
a1.box="on";
a1.grid=[1 1];
a1.grid_style=[9 9]
```

```
ymax=ceil((max([responseLW responseER responseVR responseHR responseSP])
+94)/10)*10;
a1.data_bounds=[20 ymax-50; 20000 ymax];
title("Spinorama Plot ANSI/CEA 2034A");
xlabel("frequency (Hz)")
ylabel("level (dBSPL)")
afrq=[20 30 40 50 60 70 80 90 100 200 300 400 500 600 700 800 900 1000 2000
3000 4000 5000 6000 7000 8000 9000 10000 20000]';
astr=["20" "" "" "50" "" "" "" "" "100" "200" "" "" "500" "" "" "" "" "1k"
"2k" "" "" "5k" "" "" "" "" "10k" "20k"]';
a1.x_ticks = tlist(["ticks", "locations", "labels"], afrq, astr);
a1.sub_tics=[0 1];

// plot DI curves on secondary vertical axis
a2=newaxes()
// faux curves labels
plot2d(1:2,1:2,2);
plot2d(1:2,1:2,3);
plot2d(1:2,1:2,4);
plot2d(1:2,1:2,5);
plot2d(1:2,1:2,6);
plot2d(1:2,1:2,7);
plot2d(outfrq,dirindSP,2)
plot2d(outfrq,dirindER,5)
a2.filled = 'off';
a2.tight_limits = 'on';
a2.y_location="right"
a2.log_flags="lnn";
a2.data_bounds=[20 -5; 20000 45];
ylabel("directivity index (dB)")
a2.x_ticks = tlist(["ticks", "locations", "labels"], [ ]', [ ]');
a2.y_ticks = tlist(["ticks", "locations", "labels"], [-5 0 5 10 15]', ["-5"
"0" "5" "10" "15"]');

// set curves colors and styles
cOA=a1.children(6).children(1); // ON AXIS RESPONSE
cLW=a1.children(5).children(1); // LISTENING WINDOW
cER=a1.children(4).children(1); // EARLY REFLECTIONS
cVR=a1.children(3).children(1); // VERTICAL REFLECTIONS
cHR=a1.children(2).children(1); // HORIZONTAL REFLECTIONS
cSP=a1.children(1).children(1); // SOUND POWER
cSPDI=a2.children(2).children(1); // SOUND POWER DI
cERDI=a2.children(1).children(1); // EARLY REFLECTIONS DI
cOA.foreground=2;
cOA.line_style=1;
cLW.foreground=5;
cLW.line_style=1;
cER.foreground=2;
cER.line_style=3;
cVR.foreground=2;
cVR.line_style=6;
cHR.foreground=2;
cHR.line_style=8;
cSP.foreground=6;
cSP.line_style=1;
cSPDI.foreground=6;
cSPDI.line_style=3;
cERDI.foreground=2;
cERUDI.line_style=4;

// set also for the legend
// this is an hack because apparently there is no other
// way to manage a legend with multiple axes
c2OA=a2.children(8).children(1); // ON AXIS RESPONSE
```

```
c2LW=a2.children(7).children(1); // LISTENING WINDOW
c2ER=a2.children(6).children(1); // EARLY REFLECTIONS
c2VR=a2.children(5).children(1); // VERTICAL REFLECTIONS
c2HR=a2.children(4).children(1); // HORIZONTAL REFLECTIONS
c2SP=a2.children(3).children(1); // SOUND POWER
c2OA.foreground=cOA.foreground;
c2OA.line_style=cOA.line_style;
c2LW.foreground=cLW.foreground;
c2LW.line_style=cLW.line_style;
c2ER.foreground=cER.foreground;
c2ER.line_style=cER.line_style;
c2VR.foreground=cVR.foreground;
c2VR.line_style=cVR.line_style;
c2HR.foreground=cHR.foreground;
c2HR.line_style=cHR.line_style;
c2SP.foreground=cSP.foreground;
c2SP.line_style=cSP.line_style;

// create legend
legend(["On Axis Response" "Listening Window" "Early Reflections" "Vertical
Reflections" "Horizontal Reflections" "Sound Power" "Sound Power DI" "Early
Reflections DI"]);
a2.children(1).legend_location="lower_caption";

// resize axes to the same dimensions to fit legend
a2.margins=[0.1 0.1 0.1 0.3];
a1.margins=a2.margins;
```