DSD

Design of Digital Filters

© Robert Ripio, 2012-2013

**Introduction**

DSD (for example) means Digital Speaker Design. It is a set of GNU-Octave scripts for the calculation of filters, FIR Equalization, “multi-way” speaker enclosures, and FIR filters for room correction. It contains specific functions for the design of digital filters, and the design scripts that use them.

GNU Octave is a programming environment interpretive program lanuage for mathematical calculations that can be used interactively or through scripts. DSD is executed within that environment, launching the scripts of spreadsheet with the name of the project as an argument.

**Installation of Octave**

For the installation of Octave and the necessary functions packages, see the document “octave-install.pdf” included in the distribution.

**DSD package installation**

It should be copied to a directory, for example:

*C:\Users\[usuario]\Octave*

To permanently save the search path, run the script:

*AddPath (genpath('C:\Users\Roberto\octave'))) savepath()*

In this script there are some things we need to know. First of all, it's a feature whose argument, the name of the directory to be added, is in parentheses. This argument is a data type string, a text string, and must be in quotation marks, double or single. If quotes are used simple escape sequences will not be expanded.

**Preparation of the Filters**

The procedure is as follows:

• Preparation of the files of the different box speaker’s frequency response acoustic.

• Choice of frequency cutoff and the type of crossover in each cutoff.

• Write the filter settings in the corresponding script (.xof extension) files, one for each track.

• Program execution of the DSD, with the project as a parameter file, and optionally, adding the sampling frequency and filter class as parameters.

• Check the levels of attenuation caused by the filter to take into account in the filtering and gain structure.

These steps are detailed below.

**Development of frequency response files**

The filter is made based upon certain speaker frequency response (or the multi-way, in case the item has two or more speakers) in format .FRD, whose development is the responsibility of the Designer, with appropriate measurement tools.

It is designed to have the maximum flexibility for averaging, smoothing, or whatever the designer intents to best represent the behavior of the speaker that is going to be filtered and equalize.

It is important to keep in mind that if the relative levels between the different speakers are respected in these curves, the attenuation levels t obtained with the DSD tool may also be used to level tracks in the program running convolution filters. It should be noted that the file name used is the same one used to compose the DSD scripts, and together with the kind of filter, the name of the output filters.

**Choice of cutoff frequencies and the type of crossover**

There are two main types of filter: (LP) linear phase and (MP) minimum phase. Inside we can choose the minimum phase and any order of Butterworth ot Linkwitz-Riley pair order.

It must be taken into account that providing linear phase filters to signal a delay that is intrinsic filter (ie. does not depend on the used hardware), so LP and MP filters should not be mixed in different cut-offs of the same acoustic box.

MP filters might indicate several HPF crossover frequencies, in order to add to a path the delays of the lower tract. Adding the same HPF with which these are cut-off, we reproduce the effect of an active filter cascade.

**Preparation of project files**

The package folder contains a file example \_”*ejemplo.xof* “ that you can copy (as mentioned above) to a working folder, which convenience of editing is preferably the one where you have measurement files.

You have to rename these files to other significant, and have been editing for assign the appropriate parameters.

As for the names, the author tends to call them for example “t.xof”. That is, with a prefix that refers to the driver (T, M, W tweeter, woofer, mid), but nothing of this is mandatory, except for the extension archive (.xof).

**Default settings**

Directory of the DSD is the file “Rrxof.ini”, with the default settings. The files of specific script take precedence over these options. A suitable operating mode is to make a script files that contains only specific parameters that vary regarding options by default.

Command line parameters have precedence over the above, so that sampling frequency and filter type may be omitted in the scripts in order to use only a single script for different filters with the same cutoffs.

**Parameters**

Below are the various parameters and their meaning, grouped by area of operation, which is expressed in the prefix of the name of each parameter:

**GS, General Settings**

***GSLExp***

It is the power of two which expresses the final length of the filter. It is known that the FFT operation (Fast Fourier Transforms) requires operands whose length is a power of two. Values usual will be 15 or 16, resulting in filters of 2 ^ 15 or 2 ^ 16 coefficients, respectively.

***GSFs***

It is the sampling frequency for which the filter is designed. It can be omitted if it is given as a second parameter when you call the program.

**FS File Settings**

***FSInputFile***

File name .frd of measurement the way that corresponds.

***FSNormFile***

File values for normalization of levels, for the subsequent application of values via the default "niveles.txt".

**CF Crossover Filter**

It should be noted that, while the woofer or the tweeter have no traditionally cutoff in the end of the band. Digital filtering enables or recommends cutoffs in those areas, for the Designer, and some strategies of equalization phase, or to improve the filters of reconstruction of converters (apodizing filters).

***CFClass***

Filter (LP or MP) class. It can be omitted if it is given as a third parameter when you call the program.

***CFLowType***

For MP, the high-pass filter type filters (which will filter the left end of the via ˗low˗). It can be "Butterworth" or "LinkwitzRiley". It has no effect if CFClass is LP. But It should be enclosed in brackets, to be a cell array.

It may be given as a cell array of various values will give several HPF cutoffs to simulate a filter cascade.

***CFLowOrder***

It is the order of the MP high-pass filter. It has no effect if CFClass is LP. It may be given as a vector of several values will give several HPF cutoffs to simulate a filter cascade.

***CFLowF***

Corresponding cut-off frequency. A value of 0 indicates that there is no cutoff on this side (e.g. woofers).

It may be given as a vector of several values will give several HPF cutoffs to simulate a filter cascade. If the script file is also used for linear phase filters, use only the first value of the vector as cutoff frequency.

***CFLowAsMP***

To implement a high-pass Butterworth in the woofer indicates if you must follow the same curve cutoff in the filters of linear phase, so that the response curve is identical for MP and LP. It must be a Boolean value.

***CFHighType***

For MP, the low-pass filter type filters (which will filter the right end of the˗high˗). It can be "Butterworth" or "LinkwitzRiley". It has no effect if CFClass is LP.

***CFHighOrder***

It is the order of the filter LPF MP. It has no effect if CFClass is LP.

***CFHighF***

A value of 0 indicates that there is no cut on this side (e.g. tweeters).

**TW Transition Window**

The general strategy of Equalization and filtering set which carries out the program can involve equalizations with very high values if you try to equalize out the band-pass. To set limits on this equalization is the following set of parameters.

***TWFlatInterval***

It is the range, expressed in octaves, ranging to equalize flat before filtering, beyond the cutoff frequency. I.e., a value 2 in this parameter for a low pass to 3000 Hz implies that prior to the Cutoff is equalized the curve up to 12000 Hz, two octaves further than the Cutoff.

***TWTransitionInterval***

Passed the threshold that defines TWFlatInterval, equalization is going to phase out during this another interval in octaves.

***TWLimitLowF1***

The same limit defined with TWTransitionInterval is defined here as absolute limit of frequency, which takes precedence over the result of applying TWTransitionInterval. If there is no crossover on the far left, this limit is applied directly.

***TWLimitLowF2***

The same limit defined with TWFlatInterval is defined here as absolute frequency limit, that it takes precedence over the result of applying TWFlatInterval. If there is no crossover in the left end, this limit is applied directly.

***TWLimitHighF1***

The same limit defined with TWFlatInterval is defined here as absolute frequency limit, that it takes precedence over the result of applying TWFlatInterval. If there is no crossover in the right end, this limit is applied directly.

***TWLimitHighF2***

The same limit defined with TWTransitionInterval is defined here as absolute limit of frequency, which takes precedence over the result of applying TWTransitionInterval. If there is no crossover on the far right, this limit is applied directly. You might want to be lower to higher frequency data “.frd” file, where filters are generated for one rate higher than the one used to make the measurement.

**PS Plot Settings**

These values define some aspects of graphics output.

***PSFLow***

Left end of frequency.

***PSFHigh***

Right end of frequency.

***PSVStep***

Steps of sound pressure (dB) graphics.

***PSVRange***

Full range of sound pressures of output (dB).

***DSD program execution***

After launching Octave, is changed from your prompt to the working folder (cd [working folder]), or well the parameter in the script file is given with its full path, without extension.

When you run RRxof [archivo\_de\_guion] filter will be generated pcm and explanatory graphics via corresponding.

If the script file does not include GSFs (sampling frequency) and CFClass (type of filter, mp or lp) they should be given as additional parameters. The sampling frequency is given as its value numeric.

For example, a possible call to the program could be:

Rrxof t at 44100 lp

That would calculate a filter as described in the file t.xof in the current directory, with filters for FS = 44100 linear phase.

Although GSFs and CFClass are specified in the script file may occur as parameters, It will have precedence over the script. The parameters are positional, so that they cannot skip intermediate parameters, but do not have to specify all.

It is right: "Rrxof t 44100" and "Rrxof t" but not "Rrxof t lp".

The generated filters will be < filter class > - < name of measuring .frd > .pcm, within a subdirectory (with respect to the one that is the script file) whose name is the frequency of sampling.

**Multiple calculations from a script of octave**

To be able to specify as parameters the sampling frequency and type of filter (LP or MP) it is possible to make script to calculate one-time LP and MP to several Fs, filters with a single file in which you specify the basic parameters (cuts, etc.) that is always repeat.

So the values that vary in the desired sequence of calculation are given as vectors, in the case of numeric values, or as cell arrays, in the case of values string. For example:

*% user data  
inputfiles = {"t", "m"};  
FS = [44100,48000];  
filtertypes = {'lp', 'mp'};  
% end data*

*for i2 = 1:length (fs)  
 unlink ([num2str (fs (i2)) "/ niveles.txt"]);  
 for i1 = 1:length (inputfiles)  
 for i3 = 1:length (filtertypes)  
 RRxof (inputfiles num2str (fs (i2)), filtertypes {i3}, {i1});  
 end  
 end  
end*

Is in this case were estimated for tweeter filters and media (files t.xof and m.xof) to the sampling frequency of 44100 and 48000, of the types LP and MP, for a total of eight output files.

**The DSD package Functions**

***audioplot***

Draw a graph of response in frequency with a conventional format.

*audioplot(F, dBmag, bottom, top, step, plotitle)*

*F = Vector of frequencies.  
dBmag = Vector of magnitudes in DB.  
bottom = minimum of magnitude (dB).  
top = maximum magnitude (dB).  
step = steps of dB grid and lettering.  
plotitle = title of the graph.*

***biqshelving***

Gets the coefficients of the IIR filter associated with a shelving filter as defined in [www.linkwitzlab.com](http://www.linkwitzlab.com). descent is limited by the absence of overshoot, a maximum of 6 dB/Oct. Earnings in the pass-band are always positive.

*[b, a] = biqshelving (fs, f1, f2, type)*

*[b, a] = coefficients of the IIR filter.  
FS = sampling frequency.  
F1 = start of descent rate.  
F2 = final frequency of the slope.  
type = string between value: lowShelf or highShelf.*

***biquad***

Gets the associated a biquad IIR filter coefficients. Earnings in the pass-band are always positive.

*[b, a] = biquad (Fs, f0, Q, type, dBgain)*

*[b, a] = coefficients of the IIR filter.  
FS = sampling frequency.  
F0 = central frequency of the filter.  
Q = defined in "DSP EQ cookbook". In "peakingEQ" the width of band is between gain half points.  
type = string between value: LPF, HPF, notch, peakingEQ, lowShelf and highShelf.  
dBgain = only for peakingEQ, lowShelf and highShelf.*

***buttwindow***

Generates a standard window of averaged power with 6th order Butterworth filter.

*x = buttwindow (m, ppo, ppoSm)*

*x = window.  
m = length of the logarithmic spectrum to average.  
PPO = fraction of octave of the frequency range.   
ppoSm = fraction of octave smoothing.*

***centerimp***

It increases the length of a pulse centering it. The original impulse should have odd length.

*IMP = centerimp (imporig, m)*

*IMP = FIR filter coefficients.  
imporig = pulse to focus. It must be odd length.  
m = final impulse length.*

***crossButterworth***

Gets the FIR filter of order n. Butterworth filter

*IMP = crossButterworth (Fs, m, nl, fl, nh, fh).*

*IMP = FIR filter coefficients.  
FS = sampling frequency.  
m = number of samples.  
NL = order of the high-pass filter.  
FL = lower frequency (high-pass). 0 for low pass.  
NH = order of the low pass filter.  
FH = frequency (lowpass) superior cutoff. HPF 0.*

***crossLinear***

Gets the FIR filter windowed sinc pending high and linear phase, Blackman-Harris window. Generates a filter effective length equal to a certain number of cycles of signal (frequency of cutting), focused on a vector of zeros of length m. m is always pair, and the filter as always It is odd, so the centering has a displacement of a sample into the past.

*IMP = crossLinear (Fs, mm, nc, fl, fh).*

*IMP = FIR filter coefficients.  
FS = sampling frequency.  
m = number of samples.  
NC = number of cycles of the impulse.  
FL = lower frequency (high-pass). 0 for low pass.  
FH = frequency (lowpass) superior cutoff. HPF 0.*

***crossLinkwitzRiley***

Gets a nl, nl couple order Linkwitz-Riley filter FIR filter.

*IMP = crossLinkwitzRiley (Fs, m, nl, fl, nh, fh).*

*IMP = FIR filter coefficients.  
FS = sampling frequency.  
m = number of samples.  
NL = order of the high-pass filter.  
FL = lower frequency (high-pass). 0 for low pass.*NH = order of the low pass filter.  
FH = frequency (lowpass) superior cutoff. HPF 0.

***crossLRmag***

Gets the size of the Linkwitz-Riley filters low-pass and high-pass slope given above a semi-spectrum. Frequency spacing is arbitrary.

*[magL, magH] = crossLRmag(F,fc,slope)*

*magL = Vector column with the magnitude of the LPF.  
magH = Vector column with the magnitude of the HPF.  
F = column Vector with the frequencies of the semiespectro.  
FC = cutoff frequency.  
slope = slope in dB/Oct.*

***dB2mag, dB2pow***

It becomes a decibel vector magnitude or power.

*b = dB2mag (a)*

*b = magnitude.  
a = decibels.*

*b = dB2pow (a)*

*b = power.  
a = decibels.*

***delta***

Gets a boost of length m with value one in his first exhibition.

*IMP = delta (m)*

*IMP = FIR filter coefficients.  
m = number of samples.*

***deltacentered***

Gets a boost of length m with value one in its central exhibition.

*IMP = deltacentered (m)*

*IMP = FIR filter coefficients.  
m = number of samples. It must be odd.*

***frdinterp***

Gets the scale in decibels on the semi-spectrum from a file. frd.

*magdB = frdinterp (filename, m, fs)*

*magdB = magnitude in dB on the semiespectro.  
filename = file name. frd.  
m = length of the spectrum full (must be twisted pair).  
FS = sampling frequency.*

***frjoin***

It unites two responses in magnitude over the semi-spectrum, mixing them in a range of indices given.

*SSP = frjoin (ssp1, ssp2, k1, k2)*

*SSP = Vector column with the magnitude of the mixture.  
ssp1 = Vector column response to mix on the left.  
ssp2 = Vector column response to mix right.  
K1 = first index of the range.  
K2 = second index of the range.*

***frjoinlog***

It unites two responses in magnitude over the semi-spectrum, mixing them in a range of indices given. It assumes that the answers to join are on a logarithmic scale of frequencies.

*SSP = frjoinlog (ssp1, ssp2, k1, k2)*

*SSP = Vector column with the magnitude of the mixture.  
ssp1 = Vector column response to mix on the left.  
ssp2 = Vector column response to mix right.  
K1 = first index of the range.  
K2 = second index of the range.*

***gainpcm***

It applies a certain gain in dB to a PCMfile and saves it.

*gainpcm (filename, gaindB)*

*filename = pcm file names.  
gaindB = apply gain (dB).*

***HouseCurve***

Gets the values of the Equalization Curve House on a vector of frequencies f.

*[mag, pha] = HouseCurve (F, f\_corner, house\_atten, fs)*

*MAG = Vector of magnitudes (dB).  
PHA = Vector of phases (deg).  
F = Vector of frequencies.  
f\_corner = frequency in which begins down the curve.  
house\_atten = attenuation at 20 kHz.  
FS = sampling frequency.*

***lininterp***

Gets the scale in decibels on the semi-spectrum from a file. frd.

*Maglin = lininterp (F, mag, m, fs)*

*Maglin = interpolated magnitude.  
MAG = magnitude to interpolate.  
F = Vector of frequencies.  
m = length of the spectrum full (must be twisted pair).  
FS = sampling frequency.*

***loadpcm***

(Taken from DRC-fir) Read PCM files.

*PCM = loadpcm (fname)*

*PCM = impulse Vector column.  
fname = file name. pcm.*

***loadpcms***

Load impulse file that verify a mask in an array of vector column.

*[imps, n] = loadpcms (filemask, kinit, kend)*

*IMPS = array of vectors charged pulse column.  
n = number of impulses.  
filemask = mask file (string) names.  
kinit = index for the start of the cut.  
kend = index for the start of the cut.*

***loadpir***

Read file .pir of ARTA.

*PCM = loadpir (fname)*

*PCM = impulse Vector column.  
fname = file name. pir.*

***logfreq***

Generates a vector of frequency spacing logarithmically, among the non-zero lower bin of the fft and FS/2.

*logf = logfreq (m, fs, ppo)*

*m = length of the original fft.  
FS = sampling frequency.  
PPO = fraction of octave of the frequency range.*

***mag2dB, pow2dB***

It becomes a vector of magnitude or power decibels.

*b = mag2dB (a)*

*b = decibels.  
a = magnitude.*

*b = pow2dB (a)*

*b = decibels.  
a = power.*

***minexcphsp***

Gets the minimum phase spectrum and the all-pass with the excess of phase from a spectrum full.

*[minph, excph] = minexcphsp (sp)*

*minph = full spectrum of minimum phase with the same magnitude of spectrum as IMP.  
excph = full spectrum pasatodo of excess phase.  
SP = full spectrum. Par length.*

***minphsp***

Gets the range of minimum phase from a full spectrum.

*minph = minphsp (sp)*

*minph = full spectrum of minimum phase with the same magnitude as SP.  
SP = full spectrum. Par length.*

***RoomGain***

Gets the values of the Room Gain equalization on a vector of frequencies f.

*[mag, pha] = RoomGain (F, gain\_dBS, fs)*

*MAG = Vector of magnitudes (dB).  
PHA = Vector of phases (deg).  
F = Vector of frequencies.  
gain\_dBS = total profit to DC on the flat response.  
FS = sampling frequency.*

***savepcm***

(Taken from DRC-fir) He writes pcm files.

*savepcm (pcm, fname)*

*PCM = Vector of the momentum.  
fname = filename. pcm.*

***semiblackman***

Gets the right half of a Blackman window of length m.

*w = semiblackman (m)*

*w = window.  
m = number of samples.*

***semisp***

Gets the positive spectrum from a full spectrum.

*SSP = semisp (wsp)*

*SSP = Semiespectro between 0 and m/2.  
WSP = full spectrum between 0 and m-1 (par m).*

***smooth***

It softens a real semi-spectrum with a width given in fraction of eighth.

*xsss = smooth (xws, ppo)*

*xsss = Vector column with anti-aliasing semi-spectrum.  
xws = column Vector of real values (magnitude or phase) of the semi-spectrum.  
PPO = fraction of octave smoothing.*

***smoothpw***

It softens in power a real semi-spectrum with a width given in fraction of eighth.

*xsss = smoothpw (xws, ppo)*

*xsss = Vector column with anti-aliasing semi-spectrum.  
xws = column Vector of real values (magnitude or phase) of the semi-spectrum.  
PPO = fraction of octave smoothing.*

***smoothlog***

It softens a logarithmic spectrum with a width given in fraction of eighth.

*xs = smoothlog (x, ppo, ppoSm)*

*xs = Vector column with anti-aliasing logarithmic spectrum.  
x = column Vector of actual values with the spectrum logarithmic.  
PPO = fraction of octave of the frequency range.  
ppoSm = fraction of octave smoothing.*

***smoothlogpw***

Smooth power a logarithmic spectrum with a width given in fraction of eighth.

*xs = smoothlogpw (x, ppo, ppoSm)*

*xs = Vector column with anti-aliasing logarithmic spectrum.  
x = column Vector of actual values with the spectrum logarithmic.  
PPO = fraction of octave of the frequency range.  
ppoSm = fraction of octave smoothing.*

***trwcos***

Generates additional transition windows in a given interval, based on a function ' raised cosine' on a logarithmic frequency scale.

*[trw1, trw2] = trwcos (n1, n2)*

*trw1 = window of transition from left to right.  
trw2 = window of transition from right to left.  
N1 = index on the leftmost side of the window.  
N2 = index of the right end of the window.*

***trwcoslog***

Generates additional transition windows in a given interval, based on a function ' raised cosine'. The input data must be in logarithmic magnitude, both in frequency and in value. [review]

*[trw1, trw2] = trwcoslog (n1, n2)*

*trw1 = window of transition from left to right.  
trw2 = window of transition from right to left.  
N1 = index on the leftmost side of the window.  
N2 = index of the right end of the window.*

***wholesplp***

Gets the full symmetric spectrum from the positive frequencies spectrum.

*13wsp = wholesplp (ssp)*

*WSP = full spectrum between 0 and m-1 (par m).  
SSP = Semiespectro between 0 and m/2.*

***wholespmp***

Gets the full causal spectrum from the positive frequencies spectrum.

*WSP = wholespmp (ssp)*

*WSP = full spectrum between 0 and m-1 (par m).  
SSP = Semiespectro between 0 and m/2.*