CDP FILTER Functions

(with Command Line Usage)

Functions to FILTER soundfiles

(Names in brackets mean that these are separate programs. The others are sub-modules of FILTER.)

BANK
Bank of filters, with time-varying *Q*

BANKFRQS
Generate a bank of frequencies for use as a filterbank (add amplitudes to the textfile for use with FILTER USERBANK)

[FILTRAGE]
Generate randomised VARIBANK filterbank files

FIXED
Boost or Cut: above, below or around a given frequency

ITERATED
Iterate a sound, with cumulative filtering by a filterbank

LOHI
Fixed low pass or high pass filter

PHASING
Phase shift sound, or produce phasing effect

SWEEPING
Filter whose focus-frequency sweeps over a range of frequencies

USERBANK
User-defined filterbank, with time-varying *Q*

VARIABLE
Lo-pass, High-pass, Band-pass or Notch filter with time-varying frequency

VARIBANK
User-defined time-varying filterbank, with time-varying *Q*

VFILTERS
Make datafiles for fixed-pitch FILTER VARIBANK filters

ALSO SEE:

[FASTCONV]
Multi-channel fast convolution

General comment about FILTER

When filters of time-varying Q are used, the output level tends to drop as the Q increases. This can be compensated for later by using **MODIFY LOUDNESS**, Mode **5** (Balance sources), submitting the original file and the filtered file as the two sources. The resultant soundfile will have the sound of the filtered file with the amplitude contour of the original sound.

FILTER BANK – Bank of filters, with time-varying *Q*

Usage

filter bank 1–3 *infile outfile Q gain lofrq hifrq* [**-s***scat*] [**-d**]
OR:
filter bank 4–6 *infile outfile Q gain lofrq hifrq param* [**-s***scat*] [**-d**]

Modes

1 HARMONIC SERIES over *lofrq*
2 ALTERNATE HARMONICS over *lofrq*
3 SUBHARMONIC SERIES below *hifrq*
4 HARMONIC SERIES WITH LINEAR OFFSET: *param* = offset in Hz
5 EQUAL INTERVALS BETWEEN *lofrq* and *hifrq*: *param* = number of filters
6 EQUAL INTERVALS BETWEEN *lofrq* and *hifrq*: *param* = number of semitones in the interval

Parameters

infile – input soundfile to filter
outfile – output (filtered) soundfile
Q – 'tightness' of filters (Range: 0.00100 <= *Q* < 10000) – the higher the value the more tightly the filter is focused on the centre frequency of that filter.

Q may vary over time.

gain – overall gain (Range: 0.00100 to 10000.0)
lofrq – low frequency limit of filters (Range: 10 to sample_rate/3)
hifrq – high frequency limit of filters (Range: *lofrq*+ to sample_rate/3)
param –

- in Mode **4**, offset in Hz
- in Mode **5**, number of filters
- in Mode **6**, number of semitones in interval

-s*scat* – random scatter of filter frequencies (Range: 0 to 1; the Default is 0)
-d – double filtering

Understanding the FILTER BANK Process

It may be helpful to compare this function with **FILTER USERBANK** and **FILTER VARIBANK**, which allow the user to specify the frequency centres for the bank of filters. FILTER BANK handles the definition of the frequencies for you, by means of a variety of presets.

Each Mode operates a different preset. This is how they work:

- Mode **1** – The harmonic series is formed by multiplying a given frequency by an ascending series of integers: 1, 2, 3, 4, etc. The harmonic relationships produced are sonorous, but depart from the structure of triadic chords higher in the series. This Mode builds this type of harmonic relationship onto *lofrq*, which may be the actual pitch of a pitched tone or any arbitrary pitch. Because this is being done with a filter process, other sonic material is cleaned away, so a cleaner (and possibly quieter) sound may result, with more complex noise elements removed. Quite a bit of gain may be required – values of 40 or 50+ in this situation will not be unusual. Set *hifrq* as high as possible less too few harmonics are produced. This clean sound may provide a good input for spectral time-stretching, for example, to achieve a well-tuned sound with a minimum of artefacts.
- Mode **2** – This will be like Mode **1** but with every other harmonic omitted: the **odd** numbered partials are retained. The resulting sound will probably sound more 'hollow'.
- Mode **3** – The 'subharmonic series' is the intervallic inverse of the harmonic series. Thus the departure from triadic-type intervals increases as the series descends. The resulting sound has a deeper tone and is somewhat hollow and somewhat inharmonic.
- Mode **4** – The offset in Hz is added to each harmonic. This will displace them so that they will no longer be exact multiples of the fundamental. This means that the cycles of the waveforms no longer line up at nodes, which introduces an 'inharmonic' dimension into the sound, heard as an increase in timbral components.
- Mode **5** – Here we depart from the harmonic series and simply divide up the specified frequency range into an equal number filters: how many is specified by the user with *param*. The first effect this equal spacing will have is to create inharmonic (rather than integer multiple) relationships between the partials. A low number of filters will produce an 'open' sound, and a high number of filters will produce a denser, richer sound. These might be very interesting sounds to timestretch.
- Mode **6** – By specifying the interval (in semitones), we in effect repeat these intervals within the frequency space between *lofrq* and *hifrq*, thus producing complex chords composed of the same interval 'piled up'. This is a quick way to 'harmonise' a sound, with the resulting density dependent on the size of the interval. It is well worth re-running this function, entering intervals from 1 to 7, for example, to hear what kind of transformations will be produced. Some unexpected resonances may result; they could be filtered out later.

Musical Applications

The use of the presets provide quick ways to create a variety of harmonic and inharmonic transformations. The resulting sounds have a sonorous, chordal quality. One may need to push the gain control as far as it can go without overloading, less too much signal is lost through the filtering process.

The resulting sounds can be used for rich, sonorous textures or clean time-stretching, or other spectral transformations.

Having said that, the 'chordal quality' will be diminished if the *Q* is lower (i.e., less pitch focus). Also note that a lower *Q* 'lets through' more of the original signal, so the *gain* will also have to be reduced. For example, with a *Q* of 100, a *gain* of 100 worked fine; but with a *Q* of 100, a *gain* of 5 was required (for the source sound used).

End of FILTER BANK

FILTER BANKFRQS – Generate a bank of frequencies for use as a filterbank (add amplitudes to the textfile for use with FILTER USERBANK)

Usage

filter bankfrqs 1–3 *anysndfile outtextfile lofrq hifrq* [**-s***scat*]
OR:
filter bankfrqs 4–6 *anysndfile outtextfile lofrq hifrq param* [**-s***scat*]

Modes

1 HARMONIC SERIES over *lofrq*
2 ALTERNATE HARMONICS over *lofrq*
3 SUBHARMONIC SERIES below *hifrq*
4 HARMONIC SERIES WITH LINEAR OFFSET: *param* = offset in Hz
5 EQUAL INTERVALS BETWEEN *lofrq* and *hifrq*: *param* = number of filters
6 EQUAL INTERVALS BETWEEN *lofrq* and *hifrq*: *param* = number of semitones in the interval

Parameters

infile – input soundfile to filter
outtextfile – output textfile containing frequency data
lofrq – low frequency limit of filters (Range: 10 to sample_rate/3)
hifrq – high frequency limit of filters (Range: *lofrq*+ to sample_rate/3)

NB: The sample rate of the input soundfile determines the filter frequency range.

param –

- in Mode **4**, offset in Hz
- in Mode **5**, number of filters
- in Mode **6**, number of semitones in interval

-s*scat* – random scatter of filter frequencies (Range: 0 to 1; the Default is 0)

Understanding the FILTER BANKFRQS Function

This function carries out the same operations as FILTER BANK, but writes the resulting frequencies to a textfile rather than applying them to a soundfile.

NB: An existing *outtextfile* of the same name will be overwritten without checking with you first.

See **FILTER BANK** for the meaning of each Mode.

Musical Applications

This textfile can then be edited and is then available as an input to **FILTER USERBANK** and **FILTER VARIBANK**, the filter bank functions which take a user-defined set of filter centre-frequencies as an input.

End of FILTER BANKFRQS

FILTRAGE – Generate randomised VARIBANK filterbank files

Usage

filtrage filtrage 1 *outfiltdatafile dur cnt min max distrib rand ampmin amprand ampdistrib* [**-s***seed*]

filtrage filtrage 2 *outfiltdatafile dur cnt min max distrib rand ampmin amprand ampdistrib timestep timerand* [**-s***seed*]

Example command line to create a randomised filterbank:

```
filtrage filtrage 1 filtout.txt 10 4 36 72 1 0.75 0.5 0.5 0 2 0
```

Modes

1 Generate a fixed filter data file
2 Generate a time-varying filter data file

Parameters

outfiltdata – output text file containing the time-varying filterbank data for **FILTER VARIBANK**
dur – the duration spanned by the output filter file
cnt – the number of parallel filters
min – the minimum MIDI value for the filters
max – the maximum MIDI value for the filters
distrib – the distribution of pitch values:
- **1** = pitch-linear
- **> 1** squeezes the filter towards the low pitches
- **< 1** squeezes the filter towards the high pitches

rand – randomisation of the filter pitches
ampmin – the minimum filter amplitude. The maximum amplitude allowed here is = 1.0
amprand – randomisation of the filter amplitudes
ampdistrib – the distribution of the filter amplitudes:
- **1** = increasing with pitch
- **-1** = decreasing with pitch
- **0** = random
- Intermediate values give decreasing degrees of randomisation.

timestep – the amount of time in seconds between each specified set of filter-pitches
timerand – randomisation of the *timestep*
-s*seed* – non-zero values fix the randomisation so that, on repeating the process identical random values are produced.

Understanding the FILTRAGE Function

Filtrage produces a filter data file for **FILTER VARIBANK**, with random frequencies. These are distributed within the desired frequency range either evenly, or skewed toward the top or bottom of the range. The command line above (4 filters, MIDI range 36-72, and a set of values every 2") produced this file:

Time	Frequencies							
0.00	36.652402	0.962668	48.366554	0.835011	54.429201	0.673751	60.270501	0.514595
2.00	37.025983	0.993088	46.420885	0.844629	54.091361	0.700716	62.949427	0.506138
4.00	37.794362	0.842264	48.314848	0.675147	54.824309	0.972373	61.072253	0.506926
6.00	37.754604	0.961428	46.912400	0.697998	56.042592	0.852223	61.980918	0.539829
8.00	38.631340	0.983648	47.445116	0.874975	55.960192	0.708066	60.152669	0.525479
10.00	38.285569	0.868233	45.837905	0.707601	56.749790	0.988629	62.810274	0.511495

The randomisation is not as great as you might imagine, but permits a controllable degree of variation of amplitude, frequency and output times, which can be further adjusted manually as required.

End of FILTRAGE

FILTER FIXED – Boost or Cut: above, below or around a given frequency

Usage

filter fixed 1–2 *infile outfile boost/cut frequency* [**-s***prescale*]
filter fixed 3 *infile outfile bandwidth boost/cut frequency* [**-s***prescale*]

Modes

1 Boost or cut below a given frequency
2 Boost or cut above a given frequency
3 Boost or cut a band centered on a given frequency

Parameters

infile – input soundfile to filter
outfile – output (filtered) soundfile
bandwidth – filter bandwidth in Hz – Mode **3** only
boost/cut – amplitude boost or cut, in dB
frequency – the frequency of the filter in Hz
-s*prescale* – scales gain on the **input** to the filter

Understanding the FILTER FIXED Process

This filter is referred to as 'fixed' because a single frequency of fixed amplitude roll-off (*Q*), is preset within the function.

FILTER FIXED requires very precise frequency inputs. It can therefore be useful to analyse the sound and use **SPECINFO PEAK** get a profile of the frequency bands in which most energy occurs.

For example, a trombone sound playing an F below Middle-C (174.61 Hz) showed almost no change when this frequency was used as the input to FILTER FIXED. SPECINFO PEAK showed that the energy was concentrated between 640 Hz and 905 Hz. Mode **1** and Mode **2** inputs relating to these frequencies worked fine: e.g., filtering out below 905 Hz or above 640 Hz really made a difference; similarly, filtering out a 200 Hz band centered on 700 Hz.

Musical Applications

FILTER FIXED can be a quick way of achieving three common filtering operations: hi-pass (Mode **1**: cut below), lo-pass (Mode **2**: cut above), or notch (Mode **3**: cut around), given a precise knowledge of the frequency area to be filtered and an acceptance of a fixed, average amount of *Q*.

End of FILTER FIXED

FILTER ITERATED – Iterate a sound, with cumulative filtering by a filterbank

Usage

filter iterated mode *infile outfile datafile Q gain delay dur* [**-s***prescale*] [**-r***rand*] [**-p***pshift*] [**-a***ashift*] [**-d**] [**-i**] [**-e**] [**-n**]

Modes

1 Enter filter pitches as frequency, in Hz
2 Enter filter pitches as MIDI note values

Parameters

infile – input soundfile to filter
outfile – output (filtered) soundfile
datafile – contains the pitch and amplitude of the filters (paired, one pair on each line)

- **Pitch:** as Hz or MIDI note values (Range: 10Hz to sample_rate/3)
- **Amplitude:** amplitude level (Range: 0.00100 to 10000.0 or entered as dB values)
- Remember that it is possible to create a file of frequency data with FILTER BANKFRQS and then edit it to add the amplitude information. Such files can be useful because of their harmonic structure.

Q – 'tightness' of filters (Range: 0.00100 <= *Q* < 10000) – the higher the value the more tightly the filter is focused on the centre frequency of that filter.

> *Q* may vary over time.

gain – overall gain (Range: 0.00100 to 10000.0)
delay – average delay in seconds between iterations (Range: > 0 to 32767)
dur – (min) duration of output file (must be longer than *infile*
-s*prescale* – scales gain on the **input** to the filtering process (Range: 0.0 to 1.0; the Default is 1.0)

> If set to 0.0, FILTER ITERATE automatically divides the input level by the maximum number of sound overlays occurring in the iteration process.

-r*rand* – randomisation of the delay time (Range: 0 [none] to 1 [max])
-p*pshift* – maxmimum pitch shift of any segment in (fractions of) semitones (Range: >= 0)
-a*ashift* – maximum amplitude reduction of any segment (Range: 0.0 to 1.0)
-d – double filtering
-i – turn off interpolation during filtering (makes it fast, but dirty)
-e – add exponential decay: each segment gets quieter before the next segment enters
-n – turn off normalisation: segments may grow or fall in amplitude quickly; *gain* settings will have more impact when **-n** is used – but use this cautiously, as overload may occur, which FILTER ITERATED does not report.

Understanding the FILTER ITERATED Process

Note that the duration of the output soundfile is set with *dur*, it will usually be considerably longer than *infile* to provide space in which the iterations can occur. In effect, this determines the number of iterations there will be, given the value for *delay*, the time between iterations.

If you receive a message about 'insufficient memory', you can increase the buffer capacity by increasing the buffer size. This is done by setting the environment variable CDP_MEMORY_BBSIZE. This is 1 megabyte by default. The units are 1K each, so a buffer size of e.g., 6 megabytes can be set with the phrase: 'set CDP_MEMORY_BBSIZE=6000'. Note that there must NOT be spaces before and after the equals sign. This can be done in the existing DOS window, or in *autoexec.bat* so that it sets this size upon boot-up. If the latter, the new size will not come into force until you re-boot.

More iterations means that the sound will be more filtered by the time it reaches the end. This has an important relationship to *Q*, because the higher the *Q*, the more the original sound material disappears into a pure and clean resonance as the sound progresses. Similarly, this resonance can be made to modulate nicely by using some pitch transposition with *pshift*, producing a weaving of differently tuned strands.

Delay is perhaps the key parameter, because it determines the degree of overlap with each overlap – or no overlap if it is longer than *infile*. A long *dur* (many iterations) with a *delay* longer than *infile* means a series of differently filtered repeats, with gaps between them. Higher values of *rand*, will close some of these gaps by introducing random overlaps. However, with no or occasional overlap, there will be no (or very little) resonance effect.

NB: Each iteration begins at the beginning of *infile* and plays *delay* amount of *infile*, before the next iteration commences over the top of it. How your source sound begins will greatly affect the results, especially the sharpness of attack and initial amplitude. Normally, it should have a high amplitude in order to maximise signal as the filtering progresses.

A soundfile with a strong attack and a short *delay* (e.g., 0.2 or 0.1 sec) will produce a series of pulsations which gradually disappear into resonance, depending on *Q*. It takes quite a low *Q* (e.g., well under 10) to hold off the resonance effect for a bit. The timing of the transition from the source sound to a resonant effect is handled by balancing *dur*, *delay* and *Q*.

FILTER ITERATED is processor intensive. Very short values for *delay* will increase the processing time, though this does not appear to affect the demands on available RAM.

Musical Applications

The results achieved with FILTER ITERATED can vary widely. Here is a listing of some of the main effects:

- long *dur* – many iterations and plenty of time for the repeated filtering to alter the sound
- very short *delay* (< 0.05) – a granular sound which gradually becomes a rapidly pulsating resonance; add some *pshift* and it may become a soft, modulating wash
- short *delay* (e.g., ca 0.1 or 0.2) – rapidfire pulsations
- the use of exponential decay (**-e**) will increase the sense of pulsation by dipping the amplitude between iterations
- long *delay* – gross iterations of the beginning of the sound, becoming a bit softer as the filtering takes effect
- *delay* longer than *infile* – *delay* amounts of *infile* with gaps between each iteration; add some *rand* factor and there will be some random overlapping of segments
- high *Q* – the tendency to dissolve into resonance will be considerably greater
- high *Q* with *pshift* – the resonance produced as the sound progresses will tend to warble and weave tuned strands

End of FILTER ITERATED

FILTER LOHI – Fixed low pass or high pass filter

Usage

filter lohi mode *infile outfile attenuation pass-band stop-band* [**-s***prescale*]

Modes

1 *Pass-band* and *stop-band* are given as frequency in Hz
2 *Pass-band* and *stop-band* are given as (possibly fractional) MIDI note values

Parameters

infile – input soundfile to filter
outfile – output (filtered) soundfile
attenuation – gain reduction of the filter, in dB (Range: 0 to -96)

The greater the attenuation, the sharper the filter – but it takes longer to calculate.

pass-band – last pitch to be passed by the filter
stop-band – first pitch to be stopped by the filter

- If *stop-band* is above *pass-band*, this is a low pass filter.
- If *stop-band* is below *pass-band*, this is a high pass filter.

-s*prescale* – apply gain on the input to the filtering process – avoid overflows (Range: 0.005 to 200.0)

Understanding the FILTER LOHI Process

The lo-pass filter 'lets through' all of the sound below *pass-band* and attenuates (makes gradually softer) the frequencies higher than *pass-band*, ending at *stop-band*.

The high-pass filter 'lets through' all of the sound above *pass-band* and attenuates (makes gradually softer) the frequencies lower than *pass-band*, ending at *stop-band*.

The key is how the *pass-band* and *stop-band* are placed against the significant frequencies of the source sound. As mentioned elsewhere, it may be useful to perform a spectral analysis on *infile* and make a report with SPECINFO PEAK.

An interesting fact about the operation of the filters has recently been observed. With the hipass-lopass filter (FILTER LOHI – and possibly with other filters?), if your source file has LOTS of silence at the end, the filter suddenly slows down by a factor of 20 (or more!!) in the silent buffers. At the moment, we don't know why this happens – there's no known bug in the code. So you are recommended to 'topntail' (ENVEL DOVETAIL) any long sounds that fade away to nothing before running them through filters.

Musical Applications

These basic filter operations can be used to clean up a sound by removing excessive bass with a high-pass filter, or some hiss (if it is above significant frequencies in the source) with a low-pass filter.

Musically, a high-pass filter makes the sound 'thin' by removing the bottom, and a low-pass filter makes the sound deep and rumbling by removing the top.

End of FILTER LOHI

FILTER PHASING – Phase shift sound, or produce phasing effect

Usage

filter phasing mode *infile outfile gain delay* [**-s***prescale*] [**-l**]

Modes

1 Allpass filter (phase-shifted)
2 Phasing effect

Parameters

infile – input soundfile to phase shift
outfile – output (phase-shifted) soundfile
gain – amplitude adjustment (Range: -1.0 to 1.0)

In Mode **2** the phasing effect increases as *gain* increases from -1.0, but a *gain* of 1.0 will produce complete phase cancellation and the output signal will be 0.

delay – time in milliseconds between return of delayed material (Range: 0.045 to 1644.35 ms)

Delay may vary over time.

-s*prescale* – scale gain on the **input** to the filtering process (Range: 0.0 to 1.0; the Default is 1.0)
-l – linear interpolation of changing delay values (Default: logarithmic)

Understanding the FILTER PHASING Process

The 'phasing effect' is a kind of sweeping band passing through the sound, such as is sometimes heard when aeroplanes fly overhead, and is much used in popular music.

Increasing the *gain* factor has some effect on the reverberant quality of the sound. So does increasing the *delay* time. The sound will still sound quite dry with *delay* < 20 (ms). Between about 20 and 45 ms there is a touch of resonance in the sound. Around 50 ms there is significant echoey reverberation (though somewhat granulated), and after 100 ms we start to hear larger portions of the sound repeating.

Overall, the degree of reverberant effect is controlled by increasing both the *gain* and *delay* parameters in tandem.

Musical Applications

Used with care, this function can be used to produce a variety of reverberant effects. The mid-range results are similar to the sound of a loud clap in a stairwell or under the overhang of a cement building.

Longer *delay* times will produce echo effects, but also see **MODIFY REVECHO** for echoes proper, as well as the other **REVERB** options.

End of FILTER PHASING

FILTER SWEEPING – Filter whose focus-frequency sweeps over a range of frequencies

Usage

filter sweeping mode *infile outfile acuity gain lofrq hifrq sweepfrq* [**-p***phase*]

Modes

1 High-pass
2 Low-pass
3 Band-pass
4 Notch (band-reject)

Parameters

infile – input soundfile
outfile – output soundfile
acuity – tightness of the filter (Range: 0.000100 to 1.0)

Smaller values give a tighter filter.

gain – overall gain on output (Range: 0.001000 to 10000.0)

Rule of thumb: If *acuity* = (1/3)-to-power-n, *gain* = (2/3)-to-power-n.

lofrq – lowest frequency to sweep to (Range: 10.0 to sample_rate/2)
hifrq – highest frequency to sweep to (Range: 10.0 to sample_rate/2)
sweepfrq – frequency of the sweep itself (Range: 0.0 to 200; the Default is *infile*_duration/2)

For example, to sweep once over the time of the soundfile, set *sweepfrq* to *infile*_duration/2 and set *phase* to 0 (upsweep) or 0.5 (downsweep)

-p*phase* – start position of the sweep (Range: 0 to 1; the Default is 0.25 – midway along the rising curve)

Imagine a series of points along a rising and falling curve. The curve begins at 0 (= *lofrq*) rises to 0.5 (= *hifrq*), and moves on to 1, falling back to *lofrq*. The *phase* value identifies where along this curve the sweep shape begins.

Acuity, *lofrq*, *hifrq* and *sweepfrq* may all vary over time.

Understanding the FILTER SWEEPING Process

This filter is most effective in Band-pass or Low-pass mode, with low *acuity*. It is difficult to get a result in High-pass mode. A tight filter makes the sweep more audible, but too tight and it may appear as a thin sine wave (resonance) which doesn't really connect with the sound.

A fairly long soundfile and a fairly broad range of frequencies gives this function something to work with.

Watch for overflows and reduce the *gain* as necessary.

Musical Applications

With *acuity* less than 0.1 (tight) and a *sweepfrq* less than 0.5 (slow), one can produce wah-wah effects. Sometimes it may be useful to cut only a segment of these timbrally modulating glissandi for use as a separate soundfile.

With sweep frequencies of, for example, 20 or 100, and perhaps a wider *acuity*, one can produce fluttering effects, the wider the *acuity*, the 'looser' the flutter/flapping effect.

Sometimes, with tighter *acuity* settings, one can hear movement through the harmonic overtone series.

End of FILTER SWEEPING

FILTER USERBANK – User-defined filterbank, with time-varying *Q*

Usage

filter userbank mode *infile outfile datafile Q gain* [**-d**]

Modes

1 The pitches to filter are entered as frequency in Hz
2 The pitches to filter are entered as MIDI note values

Parameters

infile – input soundfile to filter
outfile – output (filtered) soundfile
datafile – contains the pitch and amplitude of the filters (paired, one pair on each line)

- Pitch: as Hz or MIDI note values (Range: 10Hz to sample_rate/3)
- Amplitude: amplitude level (Range: 0.00100 to 10000.0 or entered as dB values)
- Comment lines (starting with ';') may be used.

Q – tightness of filter (Range: 0.001000 to 10000.0) – the higher the value the more tightly the filter is focused on the centre frequency of that filter.

Q may vary over time.

gain – overall gain (Range: 0.001000 to 10000.0)
-d – double filtering

Understanding the FILTER USERBANK Process

Here *Q* may vary over time, but the frequency settings are fixed for the duration of the sound. The tightness of the *Q* determines how much of the original sound 'comes through', or, to put it another way, the degree to which the sound is tuned to the frequencies defined in the filter bank.

Much of the effectiveness of the function therefore depends on the design of the *datafile*. Normally, dB amplitude settings will fall within a 0dB to -96dB range, but the function does allow dB greater than 0 (i.e., applying gain to the sound). If you have existing filter bank files with dB greater than 0 and they overload, you can use *gain* with values below zero to scale the amplitudes back within range – i.e., without having to edit your file.

Musical Applications

As with any graphic equaliser, this function can be used to emphasize or de-emphasize different frequency regions of the source sound.

It can also be used to impart a harmonic sonority to a sound. The higher the *Q*, the clearer the pitched effect (e.g., values between 300 and 1000). Given a source with a rapidly changing amplitude envelope, this may come across as a series of harmonic scintillations, whereas a sound with a steadier amplitude envelope will become a chord.

End of FILTER USERBANK

FILTER VARIABLE – Lo-pass, High-pass, Band-pass or Notch filter with time-varying frequency

Usage

filter variable mode *infile outfile acuity gain frq*

Modes

1 High-pass
2 Low-pass
3 Band-pass
4 Notch (Band reject)

Parameters

infile – input soundfile to filter
outfile – output (filtered) soundfile
acuity – tightness of the filter (Range: 0.000100 to 1.0)

Smaller values give a tighter filter.

gain – overall gain on output (Range: 0.001000 to 10000.0)

Rule of thumb: If *acuity* = (1/3) $^{\text{-to-power-n}}$, *gain* = (2/3)$^{\text{-to-power-n}}$.

frq – frequency of the filter (Range: 10.0 to sample_rate/2)

Acuity and *frq* may vary over time.

Understanding the FILTER VARIABLE Process

The *acuity* parameter controls the rate at which the amplitude decreases as one moves away from *frq*. The faster the rate, or, to put it more technically, the steeper the slope of the roll-off, the more the resulting sound is focused on *frq* (Band-pass), or the area above (Hi-pass) or below (Lo-pass) *frq*, or omits the area around *frq* (Notch). It is best not to make the *acuity* too tight with this filter, as the resonance produced may become unpredictable.

Hi-pass and Lo-pass are not very effective in FILTER VARIABLE, which has been designed as a 'lightweight', fast filter. For more powerful Hi-pass and Lo-pass filtering, it is recommended that you use **FILTER LOHI**.

Because a filter often creates resonance (boosts the amplitude of frequencies in a certain frequency area), especially with a broader roll-off region, overload can easily occur. FILTER VARIABLE reports any overflows, in which case the *outfile* should be deleted *without being played* and the function run again with a reduction in the *gain*.

Musical Applications

FILTER VARIABLE provides a convenient environment for exploring the effect of a variety of filter types on a given sound.

End of FILTER VARIABLE

FILTER VARIBANK and VARIBANK2 – User-defined time-varying filterbank, with time-varying *Q*

Usage

filter varibank mode *infile outfile datafile Q gain* [**-h***hcount*] [**-r***rolloff*] [**-d**]

filter varibank2 mode *infile outfile datafile Q gain* [**-t***tail*] [**-d**]

Modes

1 The pitches to filter are entered as frequency in Hz
2 The pitches to filter are entered as MIDI note values

Parameters

infile – input soundfile to filter
outfile – output (filtered) soundfile
datafile – contains lines of data for filter bands at successive times

- Each line contains data in a certain format.
- This format is: *Time: Pitch1 Amp1* [*Pitch2 Amp2* etc...]
- The Pitch and Amplitude values must be paired, but any number of pairs can be used on one line.
- All the lines in *datafile* must have the **same number of pairs**.
- To eliminate a band in any line(s), set its amplitude to 0.0)
- Time values are in seconds and must be in ascending order and >= 0.0
- Pitch values may be entered as frequency in Hz (Mode **1**) or as MIDI note values (Mode **2**)
- Amplitude values may be numeric or expressed as dB values (e.g., -4.1dB)
- Comment lines may be used – they must begin with ';'
- In FILTER VARIBANK2, you can also specify partials and their relative weighting. To do this you add a '#' sign on the next line at the bottom of the *datafile*, followed by lines which specify the way the partials vary over time. The default (in **VFILTERS**) is *harmonics* at 2,3,4,5,etc. times the given frequency (as MIDI pitch or frequency). In FILTER VARIBANK2 you can specify the partials in this format to add changing timbral colouring (also see below):

```
#
Time Partials with Amplitude weighting (pairs)]
0    1 .5  2.2  .7   4.6532   .5
4.5  1 .25 2.4  .2   3        .14
```

- See a synopsis of the file format in **CDP Files & Codes**.

Q – tightness of filter (Range: 0.001000 to 10000.0) – the higher the value the more tightly the filter is focused on the centre frequency of that filter.

Q may vary over time.

gain – overall gain on output (Range: 0.001000 to 10000.0)
-h*hcount* – number of harmonics of each pitch to use (Default: 1)

The high harmonics of high pitches may go beyond the Nyquist frequency (sample_rate/2)

-r*rolloff* – decrease in amplitude level in dB from one harmonic to the next (Range: 0 to -96)
-d – double filtering

Understanding the FILTER VARIBANK Process

The frequencies specified in the *datafile* are in fact the centre-frequencies of a filter band. With a tight *Q*, these specific frequencies will be heard as pitches; with a relaxed *Q*, these specific frequencies will locate a relatively fuzzy pitch region. The pitch will glissando between different pitch levels when these differ between time points.

Note that if the *datafile* has, for example, 10 Pitch/Amplitude pairs on a line, each column of paired values relates to a given 'band'. Thus the change over time for each band will be specified by the sequence of paired values in a given column. The value for *Q* is not contained in the *datafile*, but the time points for changes in *Q* can be made to match or vary from those in the *datafile*.

In the additional lines supplied for the FILTER VARIBANK2 option, the **first value** in each line is a **time**. The **following values** are in pairs representing the **partials** (they can in fact be ANY number) and their **relative amplitude**. Note that as with the frequency (or MIDI) data already given, there must be the same number of entries in each line. If you want to omit a partial at given times, give it an amplitude of 0.

Thus you might omit the second partial during specified times:

```
Time Partials with Amplitude weighting (pairs)]
0    1 0.3  2.237 0.4  7.615 0.8
1    1 0.3  2.37  0    7.615 0.8  [partial 2.37 disappears
2    1 0.3  2.37  0    7.615 0.8  during these times]
3    1 0.3  2.37  0.4  7.615 0.8
```

The partials could also be changing:

```
0    1 0.3  2.237 0.4  7.615 0.8
1    1 0.3  3.34  0   14.22  0.8
2    1 0.3 16.3   0    8.322 0.8
3    1 0.3 22.6   0.4  7.615 0.8
```

Here's a complete example, with VARIBANK + VARIBANK2 data:

Example Data and Q settings for FILTER VARIBANK

Datafile									Q-file		
Time	**Pitch1**	**Amp1**	**Pitch2**	**Amp2**	**Pitch3**	**Amp3**	**Description**	**|**	**Time**	**Q**	**Description**
0.0	53	-3dB	53	-3dB	53	-3dB	F below Middle C	|	0.0	5000	Very tight focus
2.0	65	0dB	53	0dB	41	0dB	F in 3 octaves	|	2.0	70	Very fuzzy
3.0	57	-6dB	54	-6dB	50	-6dB	D-Major chord	|	3.0	2000	Mid-range pitch focus
4.5	67	-3dB	62	-3dB	59	-3dB	G-major, 1st inversion	|	4.5	2000	No change
7.0	64	-3dB	60	0dB	60	-2dB	2 Middle C's & 1 E	|	7.0	5000	Very tight pitch focus
#							'#' sign for VARIBANK2 information to come				
Time	**Par1**	**Amp2**	**Par2**	**Amp3**	**Par3**	**Amp1**	NB: Same number of pairs on each line				
0	1	0.5	2.2	0.7	4.6532	0.5	Time + partial - amplitude_weight pairs				
4.5	1	0.25	2.4	0.2	3.0	0.14	Put zero amp if you want to omit a given partial				

The whole file (version for FILTER VARIBANK2) therefore looks like this:

```
0.0 53 -3dB 53 -3dB 53 -3dB
2.0 65  0dB 53  0dB 41  0dB
3.0 57 -6dB 54 -6dB 50 -6dB
4.5 67 -3dB 62 -3dB 59 -3dB
7.0 64 -3dB 60  0dB 60 -2dB
#
0    1 0.5   2.2 0.7  4.6532 0.5
4.5  1 0.25  2.4 0.2  3.0    0.14
```

Higher values for *Q* filter the sound more, which reduces overall amplitude. Higher *Q* therefore allows for and sometimes requires higher *gain* levels. Similarly, lower values for *Q* 'let through' more of the original sound, so there is less amplitude reduction through the filtering process, and therefore more likelihood of overflows. Higher values for *hcount* also increase the tendency to overflow, because there are more harmonics in the filter effect. These harmonics enrich the sound.

The data in the two files used in the above example were used in a command line with Mode **2**, a *gain* of 20 and *hcount*'s of 6 and 12. Also try it without using *hcount* to hear the difference. This was the full command line:

```
filter varibank2 2 infile outfile datafile.txt qfile.txt 20 -h12
```

The input was a sustained trombone tone on F below Middle C, which is why the filter frequency began at MIDI note 53.

Mapping out the data with a little diagram may be helpful:

- Vertical lines to mark the time points
- A series of horizontal lines rising and falling towards each time point (or not changing) to indicate pitch levels (use MIDI note values & Mode **2** unless exact frequencies are crucial); label the MIDI notes on your diagram.
- Write an amplitude value in dB next to each pitch, possibly in a different colour
- Put 'tight' at the top left of your diagram and 'relaxed' at the bottom left, in a different colour: this is the *Q* scale
- Draw a line for changing *Q* in the *Q* colour, changing direction – or staying the same – at each time point.

Musical Applications

The facilities of FILTER VARIBANK are a composer's dream. Not only can the frequencies of the filterbank be specified in precise detail, but the frequencies may move in time. Not only can the tightness of the filters – the degree to which they focus on pitches – be specified, but this focus can also change over time. The number of harmonics and the amplitude 'rolloff' are also under the composer's control.

It is possible, therefore, to play with relationships between pitched tones and complex sounds with some degree of noise components. Working with sounds accepts all kinds of sonic material into the fabric of musical discourse – but pitch need not be excluded simply because a broader range of materials is being used. A tool like FILTER VARIBANK makes it possible gradually to focus complex sonic material into specific pitches, thus creating links with other pitched elements, such as a broader harmonic scheme and the use of acoustic musical instruments.

End of FILTER VARIBANK

FILTER VFILTERS – Make datafiles for fixed-pitch FILTER VARIBANK filters

Usage

filter vfilters *inpitchfile generic_outfilterbankfile*

Parameters

inpitchfile – contains a list of MIDI Pitch Values or frequency pitch values, with one or more values on each line
generic_outfilterbankfile – each line is converted into a data file for a pitched pitch(es) filter for **FILTER VARIBANK**. The outfile names are whatever you put for *generic_outfilterbankfile* plus a '0', '1', etc. appended, producing a different file for each line in *inmidifile*.

Understanding the FILTER VFILTERS Function

FILTER VFILTERS creates one or more data file(s) for the FILTER VARIBANK process from a list of MIDI Pitch Values (MPV's), one file for each line of input. The output basic filterbank file is a fixed (not time-varying) pitches filter. If the *inmidifile* is:

```
48 63.5 70 74 81
```

the *generic_outfilterbankfile* will be (edited here for conciseness):

```
[Time MPV Amp   MPV   Amp   MPV Amp   MPV Amp   MPV   Amp]
0     48  1     63.5  1     70  1     74  1     81     1
10000 48  1     63.5  1     70  1     74  1     81     1
```

The second line with *time* = 10000 means that the filter values given extend (without changing) from 0 seconds to 10000 seconds (i.e., the end of the file). The program adds it because it needs to have two time values in a varibank data file.

It actually doesn't matter whether the values in *inmidifile* are MIDI or frequency values. The input is just treated as numbers which go into the output. If you put in what you think is MIDI, the output will be in MIDI. If you put in what you think is frequency, the output will be frequency. Many musicians find it easier to think of harmonies in the MIDI Pitch Value system, so this is illustrated here. These MIDI Pitch Values, please note, may be fractional (microtonal).

Note that the data in VARIBANK filters can also be generated, and transposed from the *Sound Loom* Table Editor.

BATCHFILES can be expanded to use a sequence of different values on the same, or a series of different sources, using the facilities in the *Sound Loom* Table Editor.

Musical Applications

FILTER VFILTERS will help you to generate data files for the **FILTER VARIBANK** process more quickly.

End of FILTER VFILTERS

Last Updated 20 Mar 2021 -- HTML5 version
Documentation: Archer Endrich, revised Robert Fraser