pychoacoustics Documentation

Release ('0.2.52',)

Samuele Carcagno

July 04, 2013

CONTENTS

Contents:

CHAPTER

ONE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

CHAPTER

TWO

SNDLIB – SOUND SYNTHESIS LIBRARY

A module for generating sounds in python.

sndlib.**AMTone** (*frequency*, *AMFreq*, *AMDepth*, *phase*, *level*, *duration*, *ramp*, *channel*, *fs*, *maxLevel*)

Generate an amplitude modulated tone.

frequency [float] Carrier frequency in hertz.

AMFreq [float] Amplitude modulation frequency in Hz.

AMDepth [float] Amplitude modulation depth (a value of 1 corresponds to 100% modulation).

phase [float] Starting phase in radians.

level [float] Tone level in dB SPL.

duration [float] Tone duration (excluding ramps) in milliseconds.

ramp [float] Duration of the onset and offset ramps in milliseconds. The total duration of the sound will be duration+ramp*2.

channel [string ('Right', 'Left' or 'Both')] Channel in which the tone will be generated.

fs [int] Samplig frequency in Hz.

maxLevel [float] Level in dB SPL output by the soundcard for a sinusoid of amplitude 1.

snd : 2-dimensional array of floats

```
>>> snd = AMTone(frequency=1000, AMFreq=20, AMDepth=1, phase=0, level=65, duration=180,
...     ramp=10, channel='Both', fs=48000, maxLevel=100)
```

sndlib.**ERBDistance** (*f1*, *f2*)

sndlib.**FMTone** (*fc*, *fm*, *mi*, *phase*, *level*, *duration*, *ramp*, *channel*, *fs*, *maxLevel*)

Generate a frequency modulated tone.

fc [float] Carrier frequency in hertz. This is the frequency of the tone at fm zero crossing.

fm [float] Modulation frequency in Hz.

mi [float] Modulation index, also called beta and is equal to deltaF/fm, where deltaF is the maximum deviation of the instantaneous frequency from the carrier frequency.

phase [float] Starting phase in radians.

level [float] Tone level in dB SPL.

duration [float] Tone duration (excluding ramps) in milliseconds.

ramp [float] Duration of the onset and offset ramps in milliseconds. The total duration of the sound will be duration+ramp*2.

channel ['Right', 'Left' or 'Both'] Channel in which the tone will be generated.

fs [int] Samplig frequency in Hz.

maxLevel [float] Level in dB SPL output by the soundcard for a sinusoid of amplitude 1.

snd : 2-dimensional array of floats

```
>>> snd = FMTone(fc=1000, fm=40, mi=1, phase=0, level=55, duration=180,
...     ramp=10, channel='Both', fs=48000, maxLevel=100)
```

sndlib.**addSounds**(*snd1*, *snd2*, *delay*, *fs*)

Add or concatenate two sounds.

snd1 [array of floats] First sound.

snd2 [array of floats] Second sound.

delay [float] Delay in milliseconds between the onset of 'snd1' and the onset of 'snd2'

fs [float] Sampling frequency in hertz of the two sounds.

snd : 2-dimensional array of floats

```
>>> snd1 = pureTone(frequency=440, phase=0, level=65, duration=180,
...     ramp=10, channel='Right', fs=48000, maxLevel=100)
>>> snd2 = pureTone(frequency=880, phase=0, level=65, duration=180,
...     ramp=10, channel='Right', fs=48000, maxLevel=100)
>>> snd = addSounds(snd1=snd1, snd2=snd2, delay=100, fs=48000)
```

sndlib.**binauralPureTone**(*frequency*, *phase*, *level*, *duration*, *ramp*, *channel*, *itd*, *itdRef*, *ild*, *ildRef*, *fs*, *maxLevel*)

Generate a pure tone with an optional interaural time or level difference.

frequency [float] Tone frequency in hertz.

phase [float] Starting phase in radians.

level [float] Tone level in dB SPL. If 'ild' is different than zero, this will be the level of the tone in the reference channel.

duration [float] Tone duration (excluding ramps) in milliseconds.

ramp [float] Duration of the onset and offset ramps in milliseconds. The total duration of the sound will be duration+ramp*2.

channel [string ('Right', 'Left' or 'Both')] Channel in which the tone will be generated.

itd [float] Interaural time difference, in microseconds.

itdRef ['Right', 'Left' or None] The reference channel for the 'itd'. The interaural time difference will be applied to the other channel with respect to the reference channel.

ild [float] Interaural level difference in dB SPL.

ildRef ['Right', 'Left' or None] The reference channel for the 'ild'. The level of the other channel will be icreased of attenuated by 'ild' dB SPL with respect to the reference channel.

fs [int] Samplig frequency in Hz.

maxLevel [float] Level in dB SPL output by the soundcard for a sinusoid of amplitude 1.

snd [2-dimensional array of floats] The array has dimensions (nSamples, 2).

```
>>> itdTone = binauralPureTone(frequency=440, phase=0, level=65, duration=180,
...     ramp=10, channel='Both', itd=480, itdRef='Right', ild=0, ildRef=None,
...     fs=48000, maxLevel=100)
>>> ildTone = binauralPureTone(frequency=440, phase=0, level=65, duration=180,
...     ramp=10, channel='Both', itd=0, itdRef=None, ild=-20, ildRef='Right',
...     fs=48000, maxLevel=100)
```

sndlib.**broadbandNoise**(*spectrumLevel*, *duration*, *ramp*, *channel*, *fs*, *maxLevel*)

Synthetise a broadband noise.

spectrumLevel [float] Intensity spectrum level of the noise in dB SPL.

duration [float] Noise duration (excluding ramps) in milliseconds.

ramp [float] Duration of the onset and offset ramps in milliseconds. The total duration of the sound will be duration+ramp*2.

channel [string ('Right', 'Left' or 'Both')] Channel in which the noise will be generated.

fs [int] Samplig frequency in Hz.

maxLevel [float] Level in dB SPL output by the soundcard for a sinusoid of amplitude 1.

snd [2-dimensional array of floats] The array has dimensions (nSamples, 2).

```
>>> noise = broadbandNoise(spectrumLevel=40, duration=180, ramp=10,
...     channel='Both', fs=48000, maxLevel=100)
```

sndlib.**camSinFMComplex**(*F0*, *lowHarm*, *highHarm*, *harmPhase*, *fm*, *deltaCams*, *fmPhase*, *level*, *duration*, *ramp*, *channel*, *fs*, *maxLevel*)

Generate a tone frequency modulated with an exponential sinusoid.

fc [float] Carrier frequency in hertz.

fm [float] Modulation frequency in Hz.

deltaCams [float] Frequency excursion in cam units (ERBn number scale).

fmPhase [float] Starting fmPhase in radians.

level [float] Tone level in dB SPL.

duration [float] Tone duration (excluding ramps) in milliseconds.

ramp [float] Duration of the onset and offset ramps in milliseconds. The total duration of the sound will be duration+ramp*2.

channel ['Right', 'Left' or 'Both'] Channel in which the tone will be generated.

fs [int] Samplig frequency in Hz.

maxLevel [float] Level in dB SPL output by the soundcard for a sinusoid of amplitude 1.

snd : 2-dimensional array of floats

```
>>> snd = expSinFMTone(fc=1000, fm=40, deltaCents=1200, fmPhase=0, level=55,
...     duration=180, ramp=10, channel='Both', fs=48000, maxLevel=100)
```

sndlib.**camSinFMTone**(*fc*, *fm*, *deltaCams*, *fmPhase*, *startPhase*, *level*, *duration*, *ramp*, *channel*, *fs*, *maxLevel*)

Generate a tone frequency modulated with an exponential sinusoid.

fc [float] Carrier frequency in hertz.

fm [float] Modulation frequency in Hz.

deltaCams [float] Frequency excursion in cam units (ERBn number scale).

fmPhase [float] Starting fmPhase in radians.

level [float] Tone level in dB SPL.

duration [float] Tone duration (excluding ramps) in milliseconds.

ramp [float] Duration of the onset and offset ramps in milliseconds. The total duration of the sound will be duration+ramp*2.

channel ['Right', 'Left' or 'Both'] Channel in which the tone will be generated.

fs [int] Samplig frequency in Hz.

maxLevel [float] Level in dB SPL output by the soundcard for a sinusoid of amplitude 1.

snd : 2-dimensional array of floats

```
>>> snd = expSinFMTone(fc=1000, fm=40, deltaCents=1200, fmPhase=0, level=55,
...     duration=180, ramp=10, channel='Both', fs=48000, maxLevel=100)
```

sndlib.**chirp**(*freqStart*, *ftype*, *rate*, *level*, *duration*, *phase*, *ramp*, *channel*, *fs*, *maxLevel*)

Synthetize a chirp, that is a tone with frequency changing linearly or exponentially over time with a give rate.

freqStart [float] Starting frequency in hertz.

ftype [string] If 'linear', the frequency will change linearly on a Hz scale. If 'exponential', the frequency will change exponentially on a cents scale.

rate [float] Rate of frequency change, Hz/s if ftype is 'linear', and cents/s if ftype is 'exponential'.

level [float] Level of the tone in dB SPL.

duration [float] Tone duration (excluding ramps) in milliseconds.

ramp [float] Duration of the onset and offset ramps in milliseconds. The total duration of the sound will be duration+ramp*2.

channel [string ('Right', 'Left' or 'Both')] Channel in which the tone will be generated.

fs [int] Samplig frequency in Hz.

maxLevel [float] Level in dB SPL output by the soundcard for a sinusoid of amplitude 1.

snd [2-dimensional array of floats] The array has dimensions (nSamples, 2).

```
>>> gl = chirp(freqStart=440, ftype='linear', rate=500, level=55,
        duration=980, phase=0, ramp=10, channel='Both',
        fs=48000, maxLevel=100)
```

sndlib.**complexTone**(*F0*, *harmPhase*, *lowHarm*, *highHarm*, *stretch*, *level*, *duration*, *ramp*, *channel*, *fs*, *maxLevel*)

Synthetise a complex tone.

F0 [float] Tone fundamental frequency in hertz.

harmPhase [one of 'Sine', 'Cosine', 'Alternating', 'Random', 'Schroeder'] Phase relationship between the partials of the complex tone.

lowHarm [int] Lowest harmonic component number.

highHarm [int] Highest harmonic component number.

stretch [float] Harmonic stretch in %F0. Increase each harmonic frequency by a fixed value that is equal to (F0*stretch)/100. If 'stretch' is different than zero, an inhanmonic complex tone will be generated.

level [float] The level of each partial in dB SPL.

duration [float] Tone duration (excluding ramps) in milliseconds.

ramp [float] Duration of the onset and offset ramps in milliseconds. The total duration of the sound will be duration+ramp*2.

channel ['Right', 'Left', 'Both', 'Odd Right' or 'Odd Left'] Channel in which the tone will be generated. If 'channel' if 'Odd Right', odd numbered harmonics will be presented to the right channel and even number harmonics to the left channel. The opposite is true if 'channel' is 'Odd Left'.

fs [int] Samplig frequency in Hz.

maxLevel [float] Level in dB SPL output by the soundcard for a sinusoid of amplitude 1.

snd [2-dimensional array of floats] The array has dimensions (nSamples, 2).

```
>>> ct = complexTone(F0=440, harmPhase='Sine', lowHarm=3, highHarm=10,
...      stretch=0, level=55, duration=180, ramp=10, channel='Both',
...      fs=48000, maxLevel=100)
```

sndlib.**complexToneParallel**(*F0*, *harmPhase*, *lowHarm*, *highHarm*, *stretch*, *level*, *duration*, *ramp*, *channel*, *fs*, *maxLevel*)

Synthetise a complex tone.

This function produces the same results of complexTone. The only difference is that it uses the multiprocessing Python module to exploit multicore processors and compute the partials in a parallel fashion. Notice that there is a substantial overhead in setting up the parallel computations. This means that for relatively short sounds (in the order of seconds), this function will actually be *slower* than complexTone.

F0 [float] Tone fundamental frequency in hertz.

harmPhase [one of 'Sine', 'Cosine', 'Alternating', 'Random', 'Schroeder'] Phase relationship between the partials of the complex tone.

lowHarm [int] Lowest harmonic component number.

highHarm [int] Highest harmonic component number.

stretch [float] Harmonic stretch in %F0. Increase each harmonic frequency by a fixed value that is equal to (F0*stretch)/100. If 'stretch' is different than zero, an inhanmonic complex tone will be generated.

level [float] The level of each partial in dB SPL.

duration [float] Tone duration (excluding ramps) in milliseconds.

ramp [float] Duration of the onset and offset ramps in milliseconds. The total duration of the sound will be duration+ramp*2.

channel ['Right', 'Left', 'Both', 'Odd Right' or 'Odd Left'] Channel in which the tone will be generated. If 'channel' if 'Odd Right', odd numbered harmonics will be presented to the right channel and even number harmonics to the left channel. The opposite is true if 'channel' is 'Odd Left'.

fs [int] Samplig frequency in Hz.

maxLevel [float] Level in dB SPL output by the soundcard for a sinusoid of amplitude 1.

snd [2-dimensional array of floats] The array has dimensions (nSamples, 2).

```
>>> ct = complexTone(F0=440, harmPhase='Sine', lowHarm=3, highHarm=10,
...      stretch=0, level=55, duration=180, ramp=10, channel='Both',
...      fs=48000, maxLevel=100)
```

sndlib.**expAMNoise**(*fc*, *fm*, *deltaCents*, *fmPhase*, *AMDepth*, *spectrumLevel*, *duration*, *ramp*, *channel*, *fs*, *maxLevel*)

Generate a sinusoidally amplitude-modulated noise with an exponentially modulated AM frequency.

fc [float] Carrier AM frequency in hertz.

fm [float] Modulation of the AM frequency in Hz.

deltaCents [float]

AM frequency excursion in cents. The instataneous AM frequency of the noise will vary from fc**(-deltaCents/1200) to fc**(+deltaCents/1200).

fmPhase [float] Starting phase of the AM modulation in radians.

AMDepth [float] Amplitude modulation depth.

spectrumLevel [float] Noise spectrum level in dB SPL.

duration [float] Tone duration (excluding ramps) in milliseconds.

ramp [float] Duration of the onset and offset ramps in milliseconds. The total duration of the sound will be duration+ramp*2.

channel ['Right', 'Left' or 'Both'] Channel in which the tone will be generated.

fs [int] Samplig frequency in Hz.

maxLevel [float] Level in dB SPL output by the soundcard for a sinusoid of amplitude 1.

snd : 2-dimensional array of floats

```
>>> snd = expAMNoise(fc=150, fm=2.4, deltaCents=1200, fmPhase=3.14, AMDepth = 1,
...     spectrumLevel=24, duration=380, ramp=10, channel='Both', fs=48000, maxLevel=100)
```

sndlib.**expSinFMComplex**(*F0*, *lowHarm*, *highHarm*, *harmPhase*, *fm*, *deltaCents*, *fmPhase*, *level*, *duration*, *ramp*, *channel*, *fs*, *maxLevel*)

Generate a frequency-modulated complex tone with an exponential sinusoid.

fc [float] Carrier frequency in hertz.

fm [float] Modulation frequency in Hz.

deltaCents [float]

Frequency excursion in cents. The instataneous frequency of the tone will vary from fc**(-deltaCents/1200) to fc**(+deltaCents/1200).

fmPhase [float] Starting fmPhase in radians.

level [float] Tone level in dB SPL.

duration [float] Tone duration (excluding ramps) in milliseconds.

ramp [float] Duration of the onset and offset ramps in milliseconds. The total duration of the sound will be duration+ramp*2.

channel ['Right', 'Left' or 'Both'] Channel in which the tone will be generated.

fs [int] Samplig frequency in Hz.

maxLevel [float] Level in dB SPL output by the soundcard for a sinusoid of amplitude 1.

snd : 2-dimensional array of floats

```
>>> snd = expSinFMTone(fc=1000, fm=40, deltaCents=1200, fmPhase=0, level=55,
...     duration=180, ramp=10, channel='Both', fs=48000, maxLevel=100)
```

sndlib.**expSinFMTone**(*fc*, *fm*, *deltaCents*, *fmPhase*, *startPhase*, *level*, *duration*, *ramp*, *channel*, *fs*, *maxLevel*)

Generate a frequency-modulated tone with an exponential sinusoid.

fc [float] Carrier frequency in hertz.

fm [float] Modulation frequency in Hz.

deltaCents [float]

Frequency excursion in cents. The instataneous frequency of the tone will vary from fc**(-deltaCents/1200) to fc**(+deltaCents/1200).

fmPhase [float] Starting fmPhase in radians.

level [float] Tone level in dB SPL.

duration [float] Tone duration (excluding ramps) in milliseconds.

ramp [float] Duration of the onset and offset ramps in milliseconds. The total duration of the sound will be duration+ramp*2.

channel ['Right', 'Left' or 'Both'] Channel in which the tone will be generated.

fs [int] Samplig frequency in Hz.

maxLevel [float] Level in dB SPL output by the soundcard for a sinusoid of amplitude 1.

snd : 2-dimensional array of floats

```
>>> snd = expSinFMTone(fc=1000, fm=40, deltaCents=1200, fmPhase=0, level=55,
...     duration=180, ramp=10, channel='Both', fs=48000, maxLevel=100)
```

sndlib.**fir2Filt**(*f1*, *f2*, *f3*, *f4*, *snd*, *fs*)

Filter signal with a fir2 filter.

This function designs and applies a fir2 filter to a sound. The frequency response of the ideal filter will transition from 0 to 1 between 'f1' and 'f2', and from 1 to zero between 'f3' and 'f4'. The frequencies must be given in increasing order.

f1 [float] Frequency in hertz of the point at which the transition for the low-frequency cutoff ends.

f2 [float] Frequency in hertz of the point at which the transition for the low-frequency cutoff starts.

f3 [float] Frequency in hertz of the point at which the transition for the high-frequency cutoff starts.

f4 [float] Frequency in hertz of the point at which the transition for the high-frequency cutoff ends.

snd [array of floats] The sound to be filtered.

fs [int] Sampling frequency of 'snd'.

snd : 2-dimensional array of floats

If 'f1' and 'f2' are zero the filter will be lowpass. If 'f3' and 'f4' are equal to or greater than the nyquist frequency (fs/2) the filter will be highpass. In the other cases the filter will be bandpass.

The order of the filter (number of taps) is fixed at 256. This function uses internally 'scipy.signal.firwin2'.

```
>>> noise = broadbandNoise(spectrumLevel=40, duration=180, ramp=10,
...     channel='Both', fs=48000, maxLevel=100)
>>> lpNoise = fir2Filt(f1=0, f2=0, f3=1000, f4=1200,
...     snd=noise, fs=48000) #lowpass filter
>>> hpNoise = fir2Filt(f1=0, f2=0, f3=24000, f4=26000,
...     snd=noise, fs=48000) #highpass filter
>>> bpNoise = fir2Filt(f1=400, f2=600, f3=4000, f4=4400,
...     snd=noise, fs=48000) #bandpass filter
```

sndlib.**freqFromERBInterval**(*f1*, *deltaERB*)

sndlib.**gate**(*ramps*, *sig*, *fs*)

Impose onset and offset ramps to a sound.

ramps [float] The duration of the ramps.

sig [array of floats] The signal on which the ramps should be imposed.

fs [int] The sampling frequency os 'sig'

sig [array of floats] The ramped signal.

```
>>> noise = broadbandNoise(spectrumLevel=40, duration=200, ramp=0,
...     channel='Both', fs=48000, maxLevel=100)
>>> gate(ramps=10, sig=noise, fs=48000)
```

sndlib.**getRms**(*sig*)

Compute the root mean square (RMS) value of the signal.

sig [array of floats] The signal for which the RMS needs to be computed.

rms [float] The RMS of 'sig'.

```
>>> pt = pureTone(frequency=440, phase=0, level=65, duration=180,
...     ramp=10, channel='Right', fs=48000, maxLevel=100)
>>> getRms(pt)
```

sndlib.**glide**(*freqStart*, *ftype*, *excursion*, *level*, *duration*, *phase*, *ramp*, *channel*, *fs*, *maxLevel*)

Synthetize a rising or falling tone glide with frequency changing linearly or exponentially.

freqStart [float] Starting frequency in hertz.

ftype [string] If 'linear', the frequency will change linearly on a Hz scale. If 'exponential', the frequency will change exponentially on a cents scale.

excursion [float] If ftype is 'linear', excursion is the total frequency change in Hz. The final frequency will be freqStart + excursion. If ftype is 'exponential', excursion is the total frequency change in cents. The final frequency in Hz will be freqStart*2**(excusrion/1200).

level [float] Level of the tone in dB SPL.

duration [float] Tone duration (excluding ramps) in milliseconds.

ramp [float] Duration of the onset and offset ramps in milliseconds. The total duration of the sound will be duration+ramp*2.

channel [string ('Right', 'Left' or 'Both')] Channel in which the tone will be generated.

fs [int] Samplig frequency in Hz.

maxLevel [float] Level in dB SPL output by the soundcard for a sinusoid of amplitude 1.

snd [2-dimensional array of floats] The array has dimensions (nSamples, 2).

```
>>> gl = glide(freqStart=440, type='exponential', excursion=500,
        level=55, duration=180, phase=0, ramp=10, channel='Both',
        fs=48000, maxLevel=100)
```

sndlib.**harmComplFromNarrowbandNoise**(*F0*, *lowHarm*, *highHarm*, *level*, *bandwidth*, *duration*, *ramp*, *channel*, *fs*, *maxLevel*)

Generate an harmonic complex tone from narrow noise bands.

F0 [float] Fundamental frequency of the complex.

lowHarm [int] Lowest harmonic component number. The first component is #1.

highHarm [int] Highest harmonic component number.

level [float] The spectrum level of the noise bands in dB SPL.

bandwidth [float] The width of each noise band in hertz.

duration [float] Tone duration (excluding ramps) in milliseconds.

ramp [float] Duration of the onset and offset ramps in milliseconds. The total duration of the sound will be duration+ramp*2.

channel ['Right', 'Left', 'Both', 'Odd Right' or 'Odd Left'] Channel in which the tone will be generated. If 'channel' if 'Odd Right', odd numbered harmonics will be presented to the right channel and even number harmonics to the left channel. The opposite is true if 'channel' is 'Odd Left'.

fs [int] Samplig frequency in Hz.

maxLevel [float] Level in dB SPL output by the soundcard for a sinusoid of amplitude 1.

snd : array of floats

```
>>> c1 = harmComplFromNarrowbandNoise(F0=440, lowHarm=3, highHarm=8,
     level=40, bandwidth=80, duration=180, ramp=10, channel='Both',
     fs=48000, maxLevel=100)
```

sndlib.**imposeLevelGlide**(*sig*, *deltaL*, *startTime*, *endTime*, *channel*, *fs*)

Impose a glide in level to a sound.

This function changes the level of a sound with a smooth transition (an amplitude ramp) between 'startTime' and 'endTime'. If the signal input to the function has a level L, the signal output by the function will have a level L between time 0 and 'startTime', and a level L+deltaL between endTime and the end of the sound.

sig [float] Sound on which to impose the level change.

deltaL [float] Magnitude of the level change in dB SPL.

startTime [float] Start of the level transition in milliseconds.

endTime [float] End of the level transition in milliseconds.

channel [string ('Right', 'Left' or 'Both')] Channel to which apply the level transition.

fs [int] Samplig frequency of the sound in Hz.

snd : array of floats

```
>>> pt = pureTone(frequency=440, phase=0, level=65, duration=180,
...      ramp=10, channel='Right', fs=48000, maxLevel=100)
>>> pt2 = imposeLevelGlide(sig=pt, deltaL=10, startTime=100,
         endTime=120, channel='Both', fs=48000)
```

sndlib.**intNCyclesFreq**(*freq*, *duration*)

Compute the frequency closest to 'freq' that has an integer number of cycles for the given sound duration.

frequency [float] Frequency in hertz.

duration [float] Duration of the sound, in milliseconds.

adjFreq : float

```
>>> intNCyclesFreq(freq=2.1, duration=1000)
2.0
>>> intNCyclesFreq(freq=2, duration=1000)
2.0
```

sndlib.**itdtoipd**(*itd*, *freq*)

Convert an interaural time difference to an equivalent interaural phase difference for a given frequency.

itd [float] Interaural time difference in seconds.

freq [float] Frequency in hertz.

ipd : float

```
>>> itd = 300 #microseconds
>>> itd = 300/1000000 #convert to seconds
>>> itdtoipd(itd=itd, freq=1000)
```

sndlib.**joinSndISI**(*sndList*, *ISIList*, *fs*)

Join a list of sounds with given interstimulus intervals

sndList [list of arrays] The sounds to be joined.

ISIList [list of floats] The interstimulus intervals between the sounds in milliseconds. This list should have one element less than the sndList.

fs [int] Sampling frequency of the sounds in Hz.

snd : array of floats

```
>>> pt1 = pureTone(frequency=440, phase=0, level=65, duration=180,
...       ramp=10, channel='Right', fs=48000, maxLevel=100)
>>> pt2 = pureTone(frequency=440, phase=0, level=65, duration=180,
...       ramp=10, channel='Right', fs=48000, maxLevel=100)
>>> tone_seq = joinSndISI([pt1, pt2], [500], 48000)
```

sndlib.**makeAsynchChord**(*freqs*, *levels*, *phases*, *tonesDuration*, *tonesRamps*, *tonesChannel*, *SOA*, *fs*, *maxLevel*)

Generate an asynchronous chord.

This function will add a set of pure tones with a given stimulus onset asynchrony (SOA). The temporal order of the successive tones is random.

freqs [array or list of floats.] Frequencies of the chord components in hertz.

levels [array or list of floats.] Level of each chord component in dB SPL.

phases [array or list of floats.] Starting phase of each chord component.

tonesDuration [float] Duration of the tones composing the chord in milliseconds. All tones have the same duration.

tonesRamps [float] Duration of the onset and offset ramps in milliseconds. The total duration of the tones will be tonesDuration+ramp*2.

tonesChannel [string ('Right', 'Left' or 'Both')] Channel in which the tones will be generated.

SOA [float] Onset asynchrony between the chord components.

fs [int] Samplig frequency in Hz.

maxLevel [float] Level in dB SPL output by the soundcard for a sinusoid of amplitude 1.

snd : 2-dimensional array of floats

```
>>> freqs = [250, 500, 1000]
>>> levels = [50, 50, 50]
>>> phases = [0, 0, 0]
>>> c1 = makeAsynchChord(freqs=freqs, levels=levels, phases=phases,
        tonesDuration=180, tonesRamps=10, tonesChannel='Both',
        SOA=60, fs=48000, maxLevel=100)
```

sndlib.**makeHuggins**(*F0*, *lowHarm*, *highHarm*, *spectrumLevel*, *bandwidth*, *phaseRelationship*, *noiseType*, *duration*, *ramp*, *fs*, *maxLevel*)

Synthetise a complex Huggings Pitch.

F0 [float] The centre frequency of the F0 of the complex in hertz.

lowHarm [int] Lowest harmonic component number.

highHarm [int] Highest harmonic component number.

spectrumLevel [float] The spectrum level of the noise from which the complex is derived in dB SPL.

bandwidth [float] Bandwidth of the frequency regions in which the phase transitions occurr.

phaseRelationship [string ('NoSpi' or 'NpiSo')] If NoSpi, the phase of the regions within each frequency band will be shifted. If NpiSo, the phase of the regions between each frequency band will be shifted.

noiseType [string ('White' or 'Pink')] The type of noise used to derive the Huggins Pitch.

duration [float] Complex duration (excluding ramps) in milliseconds.

ramp [float] Duration of the onset and offset ramps in milliseconds. The total duration of the sound will be duration+ramp*2.

fs [int] Samplig frequency in Hz.

maxLevel [float] Level in dB SPL output by the soundcard for a sinusoid of amplitude 1.

snd [2-dimensional array of floats] The array has dimensions (nSamples, 2).

```
>>> hp = makeHuggins(F0=200, lowHarm=1, highHarm=5, spectrumLevel=40,
        bandwidth=65, phaseRelationship='NoSpi', noiseType='White',
        duration=280, ramp=10, fs=48000, maxLevel=100)
```

sndlib.**makePink**(*sig*, *fs*)

Convert a white noise into a pink noise.

The spectrum level of the pink noise at 1000 Hz will be equal to the spectrum level of the white noise input to the function.

sig [array of floats] The white noise to be turned into a pink noise.

fs [int] Sampling frequency of the sound.

snd [2-dimensional array of floats] The array has dimensions (nSamples, 2).

```
>>> noise = broadbandNoise(spectrumLevel=40, duration=180, ramp=10,
...     channel='Both', fs=48000, maxLevel=100)
>>> noise = makePink(sig=noise, fs=48000)
```

sndlib.**makePinkRef**(*sig*, *fs*, *refHz*)

Convert a white noise into a pink noise.

The spectrum level of the pink noise at the frequency 'refHz' will be equal to the spectrum level of the white noise input to the function.

sig [array of floats] The white noise to be turned into a pink noise.

fs [int] Sampling frequency of the sound.

refHz [int] Reference frequency in Hz. The amplitude of the other frequencies will be scaled with respect to the amplitude of this frequency.

snd [2-dimensional array of floats] The array has dimensions (nSamples, 2).

```
>>> noise = broadbandNoise(spectrumLevel=40, duration=180, ramp=10,
...     channel='Both', fs=48000, maxLevel=100)
>>> noise = makePink(sig=noise, fs=48000, refHz=1000)
```

sndlib.**makeSilence**(*duration*, *fs*)

Generate a silence.

This function just fills an array with zeros for the desired duration.

duration [float] Duration of the silence in milliseconds.

fs [int] Samplig frequency in Hz.

snd [2-dimensional array of floats] The array has dimensions (nSamples, 2).

```
>>> sil = makeSilence(duration=200, fs=48000)
```

sndlib.**makeSimpleDichotic**(*F0*, *lowHarm*, *highHarm*, *cmpLevel*, *lowFreq*, *highFreq*, *spacing*, *sigBandwidth*, *phaseRelationship*, *dichoticDifference*, *itd*, *ipd*, *narrowBandCmpLevel*, *duration*, *ramp*, *fs*, *maxLevel*)

Generate harmonically related dichotic pitches, or equivalent harmonically related narrowband tones in noise.

This function generates first a pink noise by adding closely spaced sinusoids in a wide frequency range. Then, it can apply an interaural time difference (ITD), an interaural phase difference (IPD) or a level increase to harmonically related narrow frequency bands within the noise. In the first two cases (ITD and IPD) the result is a dichotic pitch. In the last case the pitch can also be heard monaurally; adjusting the level increase its salience can be closely matched to that of a dichotic pitch.

F0 [float] Centre frequency of the fundamental in hertz.

lowHarm [int] Lowest harmonic component number.

highHarm [int] Highest harmonic component number.

cmpLevel [float] Level of each sinusoidal frequency component of the noise.

lowFreq [float] Lowest frequency in hertz of the noise.

highFreq [float] Highest frequency in hertz of the noise.

spacing [float] Spacing in cents between the sinusoidal components used to generate the noise.

sigBandwidth [float] Width in cents of each harmonically related frequency band.

phaseRelationship [string ('NoSpi' or 'NpiSo')] If NoSpi, the phase of the regions within each frequency band will be shifted. If NpiSo, the phase of the regions between each frequency band will be shifted.

dichoticDifference [string (one of 'IPD', 'ITD', 'Level')] The manipulation to apply to the heramonically related frequency bands.

itd [float] Interaural time difference in microseconds to apply to the harmonically related frequency bands. Applied only if 'dichoticDifference' is 'ITD'.

ipd [float] Interaural phase difference in radians to apply to the harmonically related frequency bands. Applied only if 'dichoticDifference' is 'IPD'.

narrowBandCmpLevel [float] Level of the sinusoidal components in the frequency bands. If the 'narrowBandCmpLevel' is greater than the level of the background noise ('cmpLevel'), a complex tone consisting of narrowband noises in noise will be generated.

duration [float] Sound duration (excluding ramps) in milliseconds.

ramp [float] Duration of the onset and offset ramps in milliseconds. The total duration of the sound will be duration+ramp*2.

fs [int] Samplig frequency in Hz.

maxLevel [float] Level in dB SPL output by the soundcard for a sinusoid of amplitude 1.

snd [2-dimensional array of floats] The array has dimensions (nSamples, 2).

```
>>> s1 = makeSimpleDichotic(F0=250, lowHarm=1, highHarm=3, cmpLevel=30,
    lowFreq=40, highFreq=1200, spacing=10, sigBandwidth=100,
    phaseRelationship='NoSpi', dichoticDifference='IPD', itd=0,
    ipd=3.14, narrowBandCmpLevel=0, duration=280, ramp=10,
    fs=48000, maxLevel=100)
```

Keyword arguments: F0 – Fundamental frequency (Hz) lowHarm – Number of the lowest harmonic highHarm – Number of the highest harmonic cmpLevel – level in dB SPL of each sinusoid that makes up the noise lowCmp – lowest frequency (Hz) highCmp – highest frequency (Hz) spacing – spacing between frequency components (Cents) sigBandwidth – bandwidth of each harmonic band (Cents) phaseRelationship – NoSpi or NpiSo dichotic difference – IPD, ITD or Level itd – interaural time difference microseconds ipd – interaural phase difference in radians narrowBandCmpLevel - level of frequency components in the harmonic bands (valid only if dichotic difference is Level) duration – duration (excluding ramps) in ms ramp – ramp duration in ms fs – sampling frequency maxLevel –

sndlib.**nextpow2**(*x*)

Next power of two.

x [int] Base number.

out [float] The power to which 2 should be raised.

```
>>> nextpow2(511)
9
>>> 2**9
512
```

sndlib.**phaseShift**(*sig*, *f1*, *f2*, *phase_shift*, *channel*, *fs*)

Shift the phases of a sound within a given frequency region.

sig [array of floats] Input signal.

f1 [float] The start point of the frequency region to be phase-shifted in hertz.

f2 [float] The end point of the frequency region to be phase-shifted in hertz.

phase_shift [float] The amount of phase shift in radians.

channel [string (one of 'Right', 'Left' or 'Both')] The channel in which to apply the phase shift.

fs [float] The sampling frequency of the sound.

out : 2-dimensional array of floats

```
>>> noise = broadbandNoise(spectrumLevel=40, duration=180, ramp=10,
...     channel='Both', fs=48000, maxLevel=100)
>>> hp = phaseShift(sig=noise, f1=500, f2=600, phase_shift=3.14,
        channel='Left', fs=48000) #this generates a Dichotic Pitch
```

sndlib.**pinkNoiseFromSin**(*compLevel*, *lowCmp*, *highCmp*, *spacing*, *duration*, *ramp*, *channel*, *fs*, *maxLevel*)

Generate a pink noise by adding sinusoids spaced by a fixed interval in cents.

compLevel [float] Level of each sinusoidal component in dB SPL.

lowCmp [float] Frequency of the lowest noise component in hertz.

highCmp [float] Frequency of the highest noise component in hertz.

spacing [float] Spacing between the frequencies of the sinusoidal components in hertz.

duration [float] Noise duration (excluding ramps) in milliseconds.

ramp [float] Duration of the onset and offset ramps in milliseconds. The total duration of the sound will be duration+ramp*2.

channel [string ('Right', 'Left' or 'Both')] Channel in which the noise will be generated.

fs [int] Samplig frequency in Hz.

maxLevel [float] Level in dB SPL output by the soundcard for a sinusoid of amplitude 1.

snd [2-dimensional array of floats] The array has dimensions (nSamples, 2).

```
>>> noise = pinkNoiseFromSin(compLevel=23, lowCmp=100, highCmp=1000,
    spacing=20, duration=180, ramp=10, channel='Both',
    fs=48000, maxLevel=100)
```

sndlib.**pinkNoiseFromSin2**(*compLevel*, *lowCmp*, *highCmp*, *spacing*, *duration*, *ramp*, *channel*, *fs*, *maxLevel*)

Generate a pink noise by adding sinusoids spaced by a fixed interval in cents.

This function should produce the same output of pinkNoiseFromSin, it simply uses a different algorithm that uses matrix operations instead of a for loop. It doesn't seem to be much faster though.

compLevel [float] Level of each sinusoidal component in dB SPL.

lowCmp [float] Frequency of the lowest noise component in hertz.

highCmp [float] Frequency of the highest noise component in hertz.

spacing [float] Spacing between the frequencies of the sinusoidal components in hertz.

duration [float] Noise duration (excluding ramps) in milliseconds.

ramp [float] Duration of the onset and offset ramps in milliseconds. The total duration of the sound will be duration+ramp*2.

channel [string ('Right', 'Left' or 'Both')] Channel in which the noise will be generated.

fs [int] Samplig frequency in Hz.

maxLevel [float] Level in dB SPL output by the soundcard for a sinusoid of amplitude 1.

snd [2-dimensional array of floats] The array has dimensions (nSamples, 2).

```
>>> noise = pinkNoiseFromSin2(compLevel=23, lowCmp=100, highCmp=1000,
    spacing=20, duration=180, ramp=10, channel='Both',
    fs=48000, maxLevel=100)
```

sndlib.**pureTone**(*frequency*, *phase*, *level*, *duration*, *ramp*, *channel*, *fs*, *maxLevel*)

Synthetise a pure tone.

frequency [float] Tone frequency in hertz.

phase [float] Starting phase in radians.

level [float] Tone level in dB SPL.

duration [float] Tone duration (excluding ramps) in milliseconds.

ramp [float] Duration of the onset and offset ramps in milliseconds. The total duration of the sound will be duration+ramp*2.

channel [string ('Right', 'Left' or 'Both')] Channel in which the tone will be generated.

fs [int] Samplig frequency in Hz.

maxLevel [float] Level in dB SPL output by the soundcard for a sinusoid of amplitude 1.

snd [2-dimensional array of floats] The array has dimensions (nSamples, 2).

```
>>> pt = pureTone(frequency=440, phase=0, level=65, duration=180,
...     ramp=10, channel='Right', fs=48000, maxLevel=100)
>>> pt.shape
(9600, 2)
```

sndlib.**scale**(*level*, *sig*)

Increase or decrease the amplitude of a sound signal.

level [float] Desired increment or decrement in dB SPL.

signal [array of floats] Signal to scale.

sig : 2-dimensional array of floats

```
>>> noise = broadbandNoise(spectrumLevel=40, duration=180, ramp=10,
...     channel='Both', fs=48000, maxLevel=100)
>>> noise = scale(level=-10, sig=noise) #reduce level by 10 dB
```

sndlib.**steepNoise**(*frequency1*, *frequency2*, *level*, *duration*, *ramp*, *channel*, *fs*, *maxLevel*)

Synthetise band-limited noise from the addition of random-phase sinusoids.

frequency1 [float] Start frequency of the noise.

frequency2 [float] End frequency of the noise.

level [float] Noise spectrum level.

duration [float] Tone duration (excluding ramps) in milliseconds.

ramp [float] Duration of the onset and offset ramps in milliseconds. The total duration of the sound will be duration+ramp*2.

channel [string ('Right', 'Left' or 'Both')] Channel in which the tone will be generated.

fs [int] Samplig frequency in Hz.

maxLevel [float] Level in dB SPL output by the soundcard for a sinusoid of amplitude 1.

snd [2-dimensional array of floats] The array has dimensions (nSamples, 2).

```
>>> nbNoise = steepNoise(frequency=440, frequency2=660, level=65,
...     duration=180, ramp=10, channel='Right', fs=48000, maxLevel=100)
```

CHAPTER

THREE

PYSDT – SIGNAL DETECTION THEORY MEASURES

A module for computing signal detection theory measures. Some of the functions in this module have been ported to python from the 'psyphy' R package of Kenneth Knoblauch http://cran.r-project.org/web/packages/psyphy/index.html

pysdt.**dprime_SD**(*H*, *FA*, *meth*)

Compute d' for one interval same/different task from 'hit' and 'false alarm' rates.

H [float] Hit rate.

FA [float] False alarms rate.

meth [string] 'diff' for differencing strategy or 'IO' for independent observations strategy.

dprime [float] d' value

```
>>> dp = dprime_SD(0.7, 0.2, 'IO')
```

pysdt.**dprime_SD_from_counts**(*nCA*, *nTA*, *nCB*, *nTB*, *meth*, *corr*)

Compute d' for one interval same/different task from counts of correct and total responses.

nCA [int] Number of correct responses in 'same' trials.

nTA [int] Total number of 'same' trials.

nCB [int] Number of correct responses in 'different' trials.

nTB [int] Total number of 'different' trials.

meth [string] 'diff' for differencing strategy or 'IO' for independent observations strategy.

corr [logical] if True, apply the correction to avoid hit and false alarm rates of 0 or one.

dprime [float] d' value

```
>>> dp = dprime_SD(0.7, 0.2, 'IO')
```

pysdt.**dprime_mAFC**(*Pc*, *m*)

Compute d' corresponding to a certain proportion of correct responses in m-AFC tasks.

Pc [float] Proportion of correct responses.

m [int] Number of alternatives.

dprime [float] d' value

```
>>> dp = dprime_mAFC(0.7, 3)
```

pysdt.**dprime_yes_no**(*H*, *FA*)

Compute d' for one interval 'yes/no' type tasks from hits and false alarm rates.

H [float] Hit rate.

FA [float] False alarms rate.

dprime [float] d' value

```
>>> dp = dprime_yes_no(0.7, 0.2)
```

pysdt.**dprime_yes_no_from_counts**(*nCA*, *nTA*, *nCB*, *nTB*, *corr*)

Compute d' for one interval 'yes/no' type tasks from counts of correct and total responses.

nCA [int] Number of correct responses in 'signal' trials.

nTA [int] Total number of 'signal' trials.

nCB [int] Number of correct responses in 'noise' trials.

nTB [int] Total number of 'noise' trials.

corr [logical] if True, apply the correction to avoid hit and false alarm rates of 0 or one.

dprime [float] d' value

```
>>> dp = dprime_yes_no_from_counts(nCA=70, nTA=100, nCB=80, nTB=100, corr=True)
```

CHAPTER

FOUR

DEFAULT EXPERIMENTS

audiogram

audiogram_mf

freq

PYTHON MODULE INDEX

INDEX