Waveform sonification descriptions, November 2, 2016, D. Parson

Basic waveform see PACISE 2016 paper at
http://faculty.kutztown.edu/parson/pubs/SonificationPacise2016Published.pdf

All of the other waveform approaches use the basic waveform algorithm as described in the PACISE 2016 paper. Here is how they differ.

First come sonifications from spring 2016 human sonic survey, ChucK program visarff2/data/sonify/genchuck_spring2016.py function genWaveform(…) generates these.

**waveformDouble** Add time domain signal for original waveform algorithm in the paper and an identical waveform with a frequency that is 2.0 X the original waveform, i.e., shifted up in pitch 1 octave. This second, superimposed sound waveform should be consonant (aurally "pleasing") with the original waveform.

Each of the two waveforms (original + 2X frequency) has its gain set to 0.5 in order not to increase the summed signal level. Both are mixed (summed) in a ChucK high pass filter (HPF) with a cutoff frequency of 4.5 the baseline frequency of 220Hz used by all of these waveforms (same as PACISE 2016 paper, see right side of Figure 5), and a Q of 10 (moderate frequency cutoff slope). Here is the generated ChucK code. If you are not a ChucK coder, just ignore it.

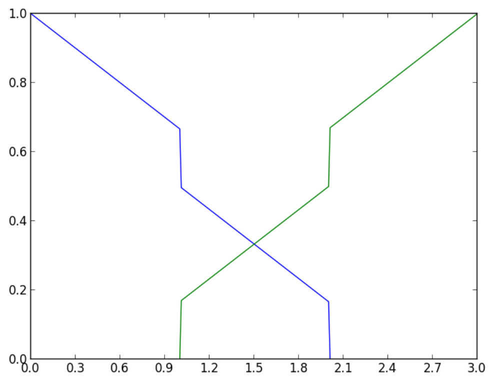
```
// file waveformDouble_0_-1.ck generated by genchuck.genWaveform from
../../SWedCgpaGprvGprjJfstNOQ.csv [0, -1, 0.21203703703703702,
0.6666666666666666, 0.9148148148148149, 0.7962962962962963,
0.49444444444444446]
[0.0,1.0,1.0,1.0,1.0] @=> float raw_waveform[];
5 => int raw_waveform_length;
220.0 => float tonicFreq;
440.0 => float otherFreq;
2.0::second => dur durationWAV;

dac => Gain __g__ => WvOut __w__ => blackhole;
"waveformDouble_0_-1.wav" => __w__.wavFilename ;
// LiSaTemplate2.ck from jan. 2016 adds a second LiSa.
// LiSaTemplate.ck from summer 2015
LiSa lisa => HPF hpf => dac ;
LiSa lisa2 => hpf ; // added Jan 2016
.5 => lisa.gain ; // added Jan 2016
.5 => lisa2.gain ; // added Jan 2016
tonicFreq * 4.5 => hpf.freq ;
10.0 => hpf.Q ;

// generate a waveform signal ranging -1.0, 1.0 and play
// it using LiSa, D. Parson, 7/2015
second/samp => float SRATE;
```

```
1.0 / tonicFreq => float tonicPeriod ;
raw_waveform_length * 2 => int sampleCount ; // half positive, half negative
sampleCount::samp => lisa.duration ;
sampleCount::samp => lisa2.duration ; // added Jan 2016
(tonicFreq * sampleCount) / SRATE => lisa.rate ;
(otherFreq * sampleCount) / SRATE => lisa2.rate ;
for (0 => int ix ; ix < raw_waveform_length ; ix+1 => ix) {
   lisa.valueAt(raw_waveform[ix], ix::samp);
   lisa2.valueAt(raw_waveform[ix], ix::samp);
}
for (raw_waveform_length => int ix ; ix < sampleCount ; ix+1 => ix) {
   lisa.valueAt(raw_waveform[ix%raw_waveform_length] * -1.0, ix::samp);
   lisa2.valueAt(raw_waveform[ix%raw_waveform_length] * -1.0, ix::samp);
}
<<< "Sampling rate is", SRATE, "sampleCount is ", sampleCount >>>;

1 => lisa.loop ;
1 => lisa2.loop ;
1 => lisa.play ;
1 => lisa2.play ;

durationWAV => now ;
null @=> __w__;
```

**waveformFourThirds** is identical to waveformDouble except that it uses a second frequency that is 4.0/3.0 X the original waveform instead of 2.0. Four thirds is a "just fourth" and should sound mostly consonant, but perhaps not as consonant as Double to most listeners. Double is an octave and is very consonant.
http://www.kylegann.com/tuning.html

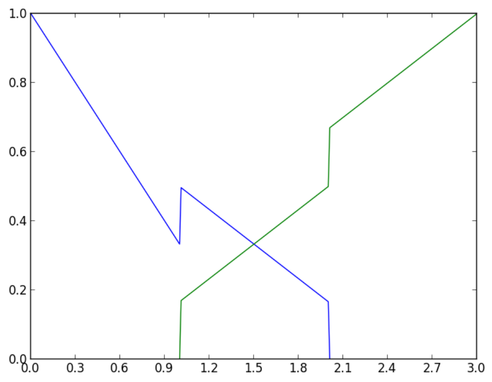
**waveformOnePt95** is identical to waveformDouble except that it uses a second frequency that is 1.95 X the original waveform instead of 2.0. This non-standard interval should sound very dissonant.
The above are the algorithms from the spring 2016 human sonic survey. Here are the subsequent sonifications subject to the machine listener ("virtual student" survey taker).

https://en.wikipedia.org/wiki/Sawtooth_wave :
The waveforms from genchuck_summer2016.py (August 28) **waveformdblsaw**, **waveform43rdssaw**, and **waveform195saw** use a sawtooth waveform that starts at the lowest level for an attribute and rises to the highest, then falling back to lowest, instead of a triangular waveform, as outlined in PACISE paper Figure 3. There are also **waveformdblrevs**, **waveform43rdsrevs**, and **waveform195revs** use that generate reverse-sawtooth (high-to-low ramps). There are also **waveformdblfixed**, **waveform43rdsfixed**, and **waveform195fixed** variants that use the following sweet&sour sonification curves (spring2016SweetSour.png – **ALL** of the algorithms in this paragraph use this "fixed" set of sweet&sour curves):



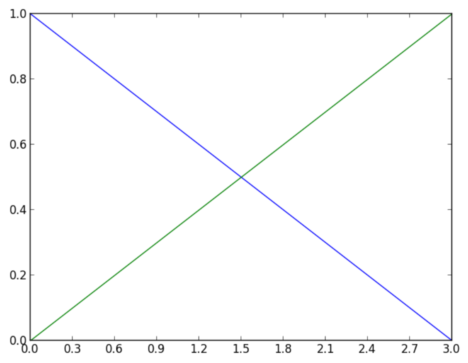
**fixed curves**

Instead of the curves in PACISE Figure 4, see the paper's Figure 4 for legends.



**original curves in PACISE paper Figure 4**

Finally, there are variants of the previous saw, revs, and fixed algorithms that include the string "lin" for linear in their names. They use this sweet&sour curve of September 3's genchuck_fall2016.py.



**sweet&sour for the "lin" linear sonifications**

https://sourceforge.net/projects/audacity/ Audacity is a waveform editor that you could use to extract time- and frequency-domain plots from wav files for the paper if you want.