Assessment 3: Bowed String Additive Synthesis

Due Date	February 20, 2023
Points	10

Points: 10

You must write a function that synthesizes one note at a time of a violin tone. You should submit your finished work to the Canvas assignment as a .scd file following the usual naming convention.

Required Classes

Here are the classes you are required to use to accomplish your task:

SinOsc // your sound source
Env // To control amplitudes over time
Xline // ???
Array // to store your frequency, amplitude, and other partial information
LFNoise1 // to add microvariation to frequencies and amplitudes
PinkNoise/WhiteNoise // to simulate the bow noise
Mix // to mix your sound down to a single channel
Pan2 // to pan your sound between two channels. You've got 'em, might as well use 'em
Out // so you can hear what you're doing

Conceptual Prompts: your sound accounting

You must account for:

- Transient (affects amplitude and frequency)
- Frequency and Amplitude variation for all partials
- Vibrato (hint: doesn't usually start right away)
- Timbrel change based on loudness. In general, the louder, the more high-freq energy*

- Bow noise
- Overall shape (loudness and duration) of the sound to be played

*(requires two spectra based on the same frequency for different dynamic levels. Requires you to either do analysis or “fake it till you make it” by using the provided example to create a new spectra that represents the “louder” or “softer” dynamic by changing the amplitudes relative to each other in a way that (non-linearly) adds more energy to higher partials or reduces them.)

You may need to consult the reading on sound and acoustics as well as the additive synthesis reading and the Perry Cook Chapters 7 and 8 to successfully understand all of the above (provided you don't remember from class lecture.)

Example Spectra

Here is a (cleaned-up) spectral analysis of five octaves of a violin G3 (lowest open string) showing both the frequency and amplitude of each partial. Please note that due to the FFT size, this spectra has the correct note names (G3, G4, etc) but the frequencies themselves are not integer multiples. This is a byproduct of the FFT and should be ignored, or rather, learned from. Practically, you should simply use integer multiples as described in class/handouts.

200 Hz (G3) = -32.4 dB
394 Hz (G4) = -9.5 dB
589 Hz (D5) = -20.8 dB
784 Hz (G5) = -20.3 dB
979 Hz (B5) = -19.2 dB
1174 Hz (D6) = -20.9 dB
1370 Hz (F6) = -26.1 dB
1565 Hz (G6) = -27.2 dB
1768 Hz (A6) = -23.9 dB
1961 Hz (B6) = -22.1 dB
2158 Hz (C#7) = -31.1 dB
2349 Hz (D7) = -34.1 dB
2547 Hz (D#7) = -28.2 dB
2742 Hz (F7) = -31.6 dB
2938 Hz (F#7) = -37.8 dB
3133 Hz (G7) = -35.9 dB
3328 Hz (G#7) = -30.5 dB
3524 Hz (A7) = -30.9 dB
3725 Hz (A#7) = -31.2 dB
3919 Hz (B7) = -53.6 dB
4110 Hz (C8) = -53.2 dB
4308 Hz (C8) = -50.2 dB
4505 Hz (C#8) = -46.8 dB
4700 Hz (D8) = -54.2 dB
4896 Hz (D#8) = -49.3 dB
5091 Hz (D#8) = -51.0 dB
5380 Hz (E8) = -72.1 dB
5677 Hz (F8) = -56.0 dB
5870 Hz (F#8) = -52.9 dB
6261 Hz (G8) = -56.9 dB

You may use this as a guide, or download your own string files and analyze them using any number of tools (Audacity, OcenAudio, Python, Matlab).

Here are string samples you can use: https://github.com/scottericpetersen/sflib/tree/master/string

If you download and analyze your own files, please 1) use at least 5 octaves worth of partials *or* until the partials are below somewhere around ~-65 dB and lower. Higher pitched samples will contain less overtones than lower ones. 2) Please indicate in a comment the source sound file so your grader knows what the source sound is and include that file along with any analysis files along with your .scd document when you submit your pset.

Requirements

You only need to synthesize one note at a time, and you can use the same set of spectral data (e.g., the above) regardless of fundamental frequency. To this end, your code outline should look like the blank code canvas below, where the `freq` argument supplies the fundamental frequency for your string-sound function.

When evaluating your code, the user should be able to easily:

- Change the fundamental frequency (via a float argument to the function).
- Change the length, shape, and overall loudness of your note (using Env) (via additional arguments, design left up to you).
- Adjust vibrato (via a float argument, from 0 or none up to 1 or "old soprano");
- Pan the sound left to right.

(Aside from `freq`, make sure to give all your arguments reasonable default values.)

When your note completes, the instrument (temporary synth created by play { }) should free itself so it doesn't sit around silently on the server (See Done).

Your static data (spectral analysis, Envs, other constant arrays) can go either inside or outside the `play { }` function. Only the sound-making code (.ar and .kr stuff) needs to be in the function itself. However, you *can* do the Array.fill lines in the function as well.

You must hard code your amplitudes.

You must *not* hard code the ~~partial~~ fundamental frequency (as this would not allow for frequency change).

****For graduate students and those enrolled in CPSC532. For others (MUSI 428 and CPSC 432) the following is a bonus feature and will earn you extra credit (2).**

- Choose from at least two (2) different spectra representing, for example, different strings or ranges of the stringed instrument, but that are only 3-4 semitones apart. If using the example spectra above, you should use the below spectra of a violin B3.******
  - https://github.com/scottericpetersen/sflib/blob/master/string/vln.b3.wav
  - Your program should take fundamental frequency input *only* between G3 and B3 (~196 - ~247 cps) and, as the input frequency moves between those pitches, linearly interpolate between the amplitudes from the example spectra (for G3) and those of the B3 (which you derive yourself using Audacity, Python, etc.)

Blank code canvas:

```
// this code block will play one sound of your designed when executed ( play {
|freq, ...| } ) // Alternately, this code block will allow the passing in of
arguments to a function to change it on the fly ( ~noteGen = {|args, ...| };
~noteGen.play(args: [\args]); )
```

Deliverables

- Your .scd code, submitted to the Canvas assignment
- Any code or supporting files (analysis output, etc) used in determining another (second) harmonic series (python code featuring scipy or numpy modules, MatLab code, etc)

If submitting multiple documents/files, please be sure to compress them as a .zip file and name according to convention.

- Completed blog post on the course Notion detailing your work, including discussion of analysis procedures, software, code examples, and *at least* one MP3 of your tone.

Grading Criteria

All of the above criteria will serve as the "minimum criteria" for the project. That means that any additional features (e.g., using multiple sources of spectral data) will serve to increase your feature-richness score. This is part of the following general criteria by which we will evaluate your work:

1. Completes all basic criteria: 85%
2. Feature Richness: 5% (Goes beyond minimum requirements)
3. Use of Language Features: 5% (Employs existing methods/classes rather than writing own)
4. Coding Style, Efficiency: 5% (Code is clean, minimal, well-commented, etc)
5. Artistic Merit (projects only) (2-5%, lowers basic criteria to 80-83%. Is artistically well-conceived and realized.)

Notes:

As you move the fundamental frequency away from the fundamental frequency at which your spectra was derived, the tone will become less and less convincing. This is to be expected!

CHECKLIST (in progress): My program...

- [ ] Takes fundamental frequency input and generates complex spectra using that frequency to generate integer multiple harmonic partials whose amplitudes are supplied by the example spectra or other spectra
- [ ] Varies the timbre based on the loudness specified by the “shape”
- [ ] Has vibrato effect that varies according to input from the user
- [ ] Uses one of the noise Ugens listed above to simulate bow noise.
- [ ] Allows the user to change the overall shape of the note (thus, also the transient) by inputting a specification for an Envelope

  (Note that you cannot default an argument to an Env object when using the `play { }` syntax—you could perhaps take other arguments instead which let you construct an Env in the function body yourself. How you do this is left up to your design.)
- [ ] Allows the user to specify panning (position between left and right channels) per note

Additional Criteria CHECKLIST for 532 Students: My program...

- [ ] Takes fundamental frequency between two spectra whose fundamentals are 3-4 semi-tones apart.
- [ ] Interpolates amplitude values based on position between the two amplitude arrays related to the two spectra.
- [ ] To-do