User Guideline

Zuheng Kang

INTRODUCTION

This project is a research project, however it is written in Mathematica ® at this time. The demo code for these functions are in `*/src/Piano Tuning.nb`.

USAGE

Download the file, and keep every folder in their path – do not change the path of files since the program is importing from other places.

To run the functions, the notebook should be created under `*/src/` folder. Where `*` represents the project folder. Otherwise, user need to assign variable `packageDirectory` to the source directory `*/src/`.

FUNCTIONS

pianoTuner[.]

The `pianoTuner` function is the traditional tuning method function.

Input

Audio samples folder path for piano keys. It support two naming systems:

- Key names, such as the standard frequency is named “A4”.
- Key numbers, the number start from lowest key as 1. If the lowest key is “A0”, then the number is the same as the number which is written inside the piano keys. If naming system is start from 0 as “A0”, you could configure the starting note is from “G#0”.

Output

- Tuning table (key + overtone ~ frequency).
- Tuning curve plot (key ~ cent).
- Pitch deviation plot (key ~ cent).

Options

- `noteRange` defines the lowest and highest note on piano, for 88 keys piano is “A0” and “C8”.
- `deleteNotes` will ignore the note for inharmonicity curve build.
- `noteStart` defines the starting note for naming systems. If “A0”, it said this key is 1 as its note number.
- `tuningSplit` defines the split point of two tuning method. If “C#4”, it said the split point is at “C#4/D4”.
- `tuningMethod` defines which frequency matches for the two sides of piano notes. If {“6:3”, “4:1”}, it said the lower part is one note’s 6^{th} overtone which will match its octave’s 3^{rd} overtone, since this frequency ideally should match. For the higher part, the note’s 4^{th} overtone (double octave) will match the note on its double octave.
- `polynomialOrder` defines the tuning curve fitting polynomial order.
- `temperament` defines the temperament of tuning. It could be the name of the temperament file in the `*/res/temperament/`, or the complete path of temperament file.
- `tempermentMajor` defines the major of temperament. It is “C”, “D”… “B”.

- `A4Frequency` defines the standard frequency, it is usually in 440 Hz.
- `saveTuningFile` defines the name of tuning file, the tuning file stores the inharmonicity information. If assigned, the file will be generated within the `*/src/` folder. Report and the frequencies information is also generated.
- `reportFormat` defines which format will be generated for report – tuning curves, tuning table.
- `exportTunedSamples` defines the path to export the tuned samples. The program will tune the sample.

entropyPianoTuner[.]

`entropyPianoTuner` defines the entropy piano tuning method function.

Intput

- *Same as* `pianoTuner`.

Output

- Tuning table (key + overtone ~ frequency).
- Tuning curve (key ~ cent).
- Frequency matching plot (key ~ volume).

Options

- `noteRange, noteStart, A4Frequency, exportTunedSamples, reportFormat` is the same as `pianoTuner`.
- `saveTuningFile` rather than generating inharmonicity information, the Tuning Shift will be generated, which defines the shift steps of frequency domain samples (which is proportional to the pitch that tuned, in our current program, the 10 shift is 1 cent).
- `peakSharpness` defines the power that been added to the Fourier transformation result.
- `loadTuneShift` defines the path of entropy tuning shift file. It will skip the entropy tuning process, and use this as the tuning strategy.

purePianoTuner[.]

Input

- Path that contains the audio samples for piano keys.
- Frequency information file that stores the real frequency information of audio samples.
- Inharmonicity file.
- A folder path for export samples.

Output

No

Files are in the export folder.

Options

- `noteRange, noteStart, A4Frequency` is the same.
- `loadEntropyShift` is similar to `loadTuningShift` in entropyPianoTuner.