TOT Tutorial

Torsten Anders

April 6, 2021

Contents

1 Introduction

This document presents in a tutorial fashion some functionality of the TOT library for Opusmodus. The TOT library is a loose collection of tools for

algorithmic composition. This library implements some features, where a number of definitions work together. This tutorial focusses on documenting such features. Other functions are self-contained, and their reference documentation is probably sufficient.

This document is unfinished...

2 Score processing

2.1 Score preview

2.2 Score transformations

3 Microtonal and xenharmonic music with regular temperaments

3.1 Introduction

The TOT library greatly expands Opusmodus' builtin support for microtonal music. Opusmodus' builtin support for microtonal music only allows for quarter tones (24 tone equal division of the octave, 24-EDO) and eighth tones (48 tone equal division of the octave, 48-EDO). The microtonal model of the TOT library, by contrast, allows users to define arbitrary equal temperaments (both equal divisions of the octave and other intervals), just intonation (JI) for arbitrary prime limits, and arbitrary regular temperaments (`https://en.xen.wiki/w/Tour_of_Regular_Temperaments`).

The library provides this tuning universe in a way that is controllable by a single uniform notation embedded in OMN. Still, the library tries to keep things relatively clear and simple by introducing mainly one actual new accidental symbol, and that symbol will then be combined with numbers (for prime limits) to express arbitrary JI pitches, which are then mapped to all the possible tunings. Technically, pitch deflections are expressed by OMN articulations, as a library cannot change the underlying OMN pitch format.

For microtonal playback, the library implements what could be called a subset of MIDI Polyphonic Expression (MPE), where chords are distributed automatically over multiple MIDI channels so that each tone is tuned independently by pitch bend messages. A considerable number of soft synth already support MPE directly (some incomplete list is shown here, scroll down and select Soft Synths), and every instrument plugin can be relatively easily made to support MPE by using multiple instances of that plugin in parallel (e.g., directly in a DAW or with a plugin host that itself is also a

plugin, like Plogue Bidule).

The core idea of this xemharmonic support is that JI, arbitrary equal temperaments and very many other tunings (`https://en.xen.wiki/w/Tour_of_Regular_Temperaments`) can all be expressed as regular temperaments. You can find an informal discussion of regular temperaments, its context and motivation – how it extends/generalises many other tone systems – at this link: `http://x31eq.com/paradigm.html`. Here is another introduction: `https://en.xen.wiki/w/Mike%27s_Lectures_On_Regular_Temperament_Theory`.

3.2 First steps

Importantly, regular temperaments can all be mapped to JI. Therefore, they can also all by notated by a pitch notation capable of notating JI for arbitrary prime limits. So, as a unifying pitch notation for all these temperaments I am using such a JI notation. Several recent JI staff notations [1] are based on the same fundamental idea: the traditional pitch nominals (A, B, C...) and the traditional accidentals (sharp, double-sharp, flat...) are denoting tones in a Pythagorean tuning (when the notation is read as a JI notation), i.e. the traditional nominals and accidentals notate all the pitches we can reach when stacking just fifths (plus their octaves). The core idea of these JI pitch notations is that they introduce a new accidental for the comma (`https://en.wikipedia.org/wiki/Comma_(music)`) of every prime limit (`https://en.wikipedia.org/wiki/Limit_(music)`). These notations mainly differ in what kind of symbols they propose for these comma-accidentals.

The JI notation of this library is based on the same principle idea. We notate Pythagorean nominals and accidentals using the standard OMN pitch notation. Here is a dominant seventh chord, which in a JI interpretation would be tuned in Pythagorean tuning.

```
(setf pythagorean-seventh '(h c4e4g4bb4))
```

We can play this chord in JI with the TOT macro `def-tempered-score`. This macro does almost exactly what the Opusmodus builtin `def-score` does, but it additionally receives a temperament as one of its arguments. Also, there are multiple MIDI channels specified per instrument, as simultaneous tones are played on different MIDI channels, so they can be tuned individually by pitch bend (the macro uses the `def-score` tuning argument

[1] I found so far 5 of such JI notations. Examples of highly developed and more widely used notations are Sagittal (`http://sagittal.org`) and the Helmholtz-Ellis JI Pitch Notation (`https://marsbat.space/pdfs/notation.pdf`).

in the background). Only chords with as many tones as we specify MIDI channels can be independently/correctly tuned.

```
(def-tempered-score my-score-name
    (:temperament '31-limit-JI
     :time-signature '(4 4))
  (instr1
   :omn pythagorean-seventh
   :channel '(1 2 3 4)
   :sound 'gm))
```

While `def-tempered-score` generates the score, it does not actually display it nor play it back. We can always play back and notate the last score explicitly with the next code snippet. The notation also reveals how `def-tempered-score` works in the background: it splits the given score instrument (`instr1`) with the chord into as many staves as there are MIDI channels, so that these resulting monophonic staves can be independently tuned via MIDI pitchbend.

```
(audition-last-score)
(display-musicxml *last-score* :display :window)
```

Instead of first defining a score with `def-tempered-score` and then explicitly triggering its notation or playback as shown, we can also use the TOT function `preview-score`, which then calls `def-tempered-score` in the background. We will commonly use a single temperament and a fixed set of instruments when working on a piece, so we might want to specify their settings only once and then use it for notating and playing back multiple snippets.

We can set the tuning separately using the variable `*current-temperament*`.

```
(setf *current-temperament* '31-limit-JI)
```

We can specify all other score settings via the two variables `*default-preview-score-header*` and `*default-preview-score-instruments*`, where `*default-preview-score-header*` is a keyword list (plist) supporting all keyword arguments of `def-score`.

```
(setf *default-preview-score-header*
      '(:title "Dummy title"
        :tempo 80))
```

Any instrument-specific settings can be specified with the variable `*default-preview-score-instru` for each instrument we want to use. The instrument labels we use here are the same that we will later use in the score short-hand format given to `preview-score`. Note that multiple MIDI channels are specified here again, and also note the nested quote (') signs.

```
(setf *default-preview-score-instruments*
      '(:instr1 (:sound 'gm
                 :channel '(1 2 3 4))))
```

We can now instead of `def-tempered-score` use the shorter call `preview-score` with these settings. The function `preview-score` is designed for playing back polyphonic scores, so we need to specify the instrument labels as well.

```
(preview-score (list :instr1 pythagorean-seventh))
```

OK, how about instead of the Pythagorean third we want to use a just major third – and also a harmonic seventh and the 11ths overtone? All pitches that go beyond Pythagorean tuning are notated using JI accidentals that express microtonal comma inflections.

I suggest we use the letter `K` for marking any OMN attribute that serves as a Komma accidental,[2] and then to simply complement that letter with the digits of the prime of the comma in question. So, the OMN accidental for the 5-limit comma (the syntonic comma) is notated `5K`, the 7-limit comma is notated `7K` and so forth. These accidentals raise the pitch by that comma. For a comma flat, put a minus in front of the accidental, e.g., `-5K`. So, here is how we can notate and play[3] the just harmonic seventh chord. `1K` is the natural sign, and multiple accidental attributes for a chord are assigned in ascending order of chord tones.

[2] Our letter `K` is similar to the Greek letter kappa, from where the word comma comes. (Besides, the letter `C` is already used in OMN for cent values). When extending OMN as users by microtonal accidentals, we are restricted to plain ASCII letters and numbers. Most of the ASCII special characters are already used for something else in OMN and are therefore unavailable for accidental symbols. Also, while the underlying CCL compiler does support unicode, unicode support in OMN attribute names is currently limited. I could use greek letters in the names of custon OMN attributes, but all unicode symbols I tried – e.g., any mathematical symbols or arrows – were not supported. Anyway, while such characters could currently not be used for the attribute name, they can be used for the string that is then shown for the attribute in the score.

[3] Unfortunately, we cannot yet define new shortcuts for the standard Opusmodus editor. I am instead using Opusmodus mainly with Emacs, and I defined a shortcut for me for such microtonal snippets that calls the TOT function `prevew-score`, which in turn calls `def-tempered-score` with the OMN expression before the cursor.

```
(preview-score (list :instr1 '(h c4e4g4bb4 1K+-5K+1K+-7K)))
```

JI leads to an infinite number of different pitches. Temperaments reduce that number. So, how about we want to play the above chord in, say, 22-tone equal temperament (`https://en.xen.wiki/w/22edo`). We only need to define that temperament. Each regular temperament (including all equal temperaments and also JI) is specified by only two settings: a small number of generator intervals, and a val for each generator. The vals together specify how each prime (up to the prime limit of the temperament) it is mapped to JI. These details are explained in the next section. For now, the following code simply shows the definition of 22-EDO, which is pretty brief.

```
(deftemperament 7-limit-22-EDO
    ;; List of vals
    (list (list 22
                (+ 13 22)
                (+ 7 (* 2 22))
                (+ 18 (* 2 22))))
  ;; List of generators
  (list (/ 1200.0 22)))

(setf *current-temperament* '7-limit-22-EDO)
```

After we defined 22-EDO this say, we can play the above chord (and any other 7-limit OMN intervals) in this temperament.

```
(preview-score (list :instr1 '(h c4e4g4bb4 1K+-5K+1K+-7K)))
```

3.3 Defining temperaments and scales

We can define arbitrary regular temperaments simply by specifying their vals and generators. This section briefly introduces these concept, so that you can define your own temperaments. We will first define the temperament for 12-tone equal temperament (12-EDO), because that is most widely used and best understood by most Western musicians.

All pitches of a regular temperament are specified by a small number of *generators*. For an equal temperament like 12-EDO, we only need a single generator. We can use the semitone of this temperament as its generator. All tones of the temperament 12-EDO can then be derived by stacking that interval multiple times. The semitone of 12-EDO and thus our generator is tuned to 100 cents.

The *vals* of a temperament specify how the intervals of the prime frequency ratios 2, 3, 5... up to the prime limit of our temperament are mapped to intervals in this temperament.

TODO

```
(deftemperament 11-limit-12-EDO
    (list (edo-val 12 '(0 7 4 10 6)))
  '(100.0)
  "12-EDO temperament with an 11-limit mapping.")
```

3.4 Obtaining information

3.4.1 Information on intervals

3.4.2 Information on scales

3.5 Defining harmony

3.6 Constraining music to an underlying harmony

3.7 Dynamic temperaments

At a later stage, the library is also meant to support dynamic temperaments, so that the tuning can change during the course of a piece, but the implementation of dynamic temperaments is still unfinished.

4 Form

`fn-unfold`

5 Karnatic rhythms

5.1 Creating a higher-level plan

5.2 Filling in details