musictheory

Release 0.0.1

Fabian C. Moss

Apr 22, 2020

CONTENTS

Welcome!

This is not a pedagogical resource for basic music theory concepts but an in-depth introduction into the structures of Western music, built axiomatically from tones and their relations. The logo, a hemidemisemiquaver (or a sixty-fourth note), symbolically reflects this level of difficulty. A PDF version of the contents of this page can be found here.

This project has been inspired by this great book:

- Lewin, D. (2007). *Generalized Intervals and Transformations*. Oxford: Oxford University Press.

I recently also discovered Music for Geeks and Nerds by Pedro Kroger which looks very interesting.

Since this is ongoing work, I can give no guarantee for completeness or accuracy. Feel free to contact me with your questions and suggestions!

CHAPTER

ONE

INTRODUCTION

1.1 What is CM?

- Description and relation to related fields – musicology – music theory – music information retrieval – music cognition – music psychology
- modeling (!!!) very important chapter, maybe deserves its own paragraph. Modeling as a form of question/hypothesis guided research as opposed to wild tool application (''We did machine learning!")
- How to do CM? conferences, journals, twitter, GitHub repositories, libraries
- which language to use? Matlab, R, Python, Julia…

1.2 General notes

- Overarching goal is to have an ACCESSIBLE introduction for musicologists with elementary understanding of a programming language such as Python or R. Requirement should be a sound understanding of how functions, loops, and conditionals work.
- Every chapter must have: – a very clear focus on one musicological question – one particular method to answer this question – a range of exercises (not always involving programming, also listening and composing) – and a list of relevant references
- the algorithms/methods used in each chapter should be one of the most basic instances of a class of methods. The point is not to have the best classifier, the best dimensionality reduction, the best regression model etc. but rather to understand the class of problems that we are dealing with. Thinking in these abstract problem classes helps to recognize and understand the nature of other problems more easily.

1.3 Notes (old)

General remark: Create excercises with listening, composing and analyzing tasks.

- Sounds in the external world
- Perception, constraints (e.g. audible range)
- discretization - Musical Universals (3-7 note scales) - allows symbolic representation
- scales (independent from tuning/temperament): collections of pitches
- members of scale: notes - neighborness - Schenkerian terms: neighbor notes - pitch classes - pitch class sets
- intervals - counterpoint - consonance / dissonance - interval classes - interval class vectors

- special pitch class sets: chords - Triads - Euler space - tonnetz - seventh chords
- notes in time: durations, rhythm - Schenkerian terms: passing notes - cognitive framework: meter - metrical hierarchies
- visualisations (pitch-time plots) - pianoroll - MIDI - modern Western notation - different keys (not only treble and bass)

CHAPTER

TWO

FUNDAMENTALS

2.1 Representations and Formats

(This part could e.g. feature (algorithmic) composition exercises)

2.1.1 Scores

- MEI
- music21
- MusicXML
- **kern
- MIDI

2.1.2 Chords

(symbolic)

2.2 Tones, pitches, pitch classes

2.2.1 Tones

2.2.2 pitches / pitch classes / pitch class sets

2.2.3 GISs

2.2.4 Euler Space / Tonnetz / Tonal Space

2.2.5 Frequency / Temperament

2.3 Time

2.3.1 Notes

(Tones + Duration)

2.3.2 Rhythm

(Duration patterns)

2.3.3 Meter

(Hierarchy)

2.3.4 Musical time vs. performance time

CHAPTER

THREE

MODELING MUSICAL SEQUENCES

3.1 Regular Expressions

(chord symbols, rhythms)

3.2 n-gram models

(melody, rhythms)

3.3 Hidden Markov models

(harmony)

3.4 Probabilistic Context-Free Grammars

(form; choro)

3.5 More advanced models

Neural nets

CHAPTER

FOUR

ADVANCED APPLICATIONS

4.1 Style (classification)

4.1.1 Feature clustering

(k-means, PCA, . . .)

4.1.2 Hierarchical clustering

4.2 History

(regression, GPs)

- trends (maybe with a non note-based dataset e.g. metadata)

4.3 Performance

- Spotify API to compare different recordings

CHAPTER

FIVE

DEVELOPERS

5.1 Installation

Warning: These instructions do not work yet.

To install `musictheory` type the following in your terminal:

```
pip install musictheory
```

5.2 API

The entire `musictheory` API.

5.2.1 Tone

class `main.`**`Tone`**(*octave=None*, *fifth=None*, *third=None*, *name=None*)

Class for tones.

`get_accidentals`()

Gets the accidentals of the tone (flats (*b*) or sharps (*#*))..

Parameters **`None`** –

Returns The accidentals of the tone.

Return type str

Example

```
>>> t = Tone(0,7,0) # C sharp
>>> t.get_accidentals()
`#`
```

`get_frequency`(*chamber_tone=440.0*, *precision=2*)

Get the frequency of the tone.

Parameters

- **chamber_tone** (*float*) – The frequency in Hz of the chamber tone. Default: 440.0 (A)
- **precision** (*int*) – Rounding precision.

Returns The frequency of the tone in Hertz (Hz).

Return type float

Example

```
>>> t = Tone(0,0,0)
>>> t.get_frequency(precision=3)
261.626
```

get_label()

Gets the complete label of the tone, consisting of its note name, syntonic position, and octave.

Parameters **None** –

Returns The accidentals of the tone.

Return type str

Example

```
>>> c = Tone(0,0,0)
>>> ab = Tone(0,1,-1)
>>> c.get_label(), ab.get_label()
`C_0` `Ab,1`
```

get_midi_pitch()

Get the MIDI pitch of the tone.

Parameters **None** –

Returns The MIDI pitch of the tone if it is in MIDI pitch range (0–128)

Return type int

Example

```
>>> t = Tone(0,0,0)
>>> t.get_midi_pitch()
60
```

get_pitch_class(*start=0*, *order='chromatic'*)

Get the pitch-class number on the circle of fifths or the chromatic circle.

Parameters

- **start** (*int*) – Pitch-class number that gets mapped to C (default: 0).
- **order** (*str*) – Return pitch-class number on the chromatic circle (default) or the circle of fifths.

Returns The pitch class of the tone on the circle of fifths or the chromatic circle.

Return type int

Example

```
>>> t = Tone(0,7,0) # C sharp
>>> t.get_pitch_class(order="chromatic")
1
```

```
>>> t = Tone(0,7,0) # C sharp
>>> t.get_pitch_class(order="fifths")
7
```

get_step()

Gets the diatonic letter name (C, D, E, F, G, A, or B) of the tone *without* accidentals.

Parameters **None** –

Returns The diatonic step of the tone.

Return type str

Example

```
>>> t = Tone(0,7,0) # C sharp
>>> t.get_step()
`C`
```

get_syntonic()

Gets the value of the syntonic level in Euler space. Tones on the same syntonic line as central C are marked with _, and those above or below this line with ‘ or ,, respectively.

Parameters **None** –

Returns The number of thirds above or below the central C.

Return type int

Example

```
>>> e1 = Tone(0,4,0) # Pythagorean major third above C
>>> e2 = Tone(0,0,1) # Just major third above C
>>> e3 = Tone(0,8,-1) # Just major third below G sharp
>>> e1.get_syntonic(), e2.get_syntonic(), e3.get_syntonic()
`'` `_` `,`
```

5.2.2 Interval

class main.**Interval**(*source*, *target*)

Class for an interval between two tones *s* (source) and *t* (target).

get_euclidean_distance(*precision=2*)

Calculates the Euclidean distance between two tones with coordinates in Euler space.

Parameters **precision** (*int*) – Rounding precision.

Returns The Euclidean distance between two tones *s* (source) and *t* (target).

Return type float

Example

```
>>> s = Tone(0,0,0) # C_0
>>> t = Tone(1,2,1) # D'1
>>> i = Interval(s,t)
>>> i.get_euclidean_distance()
2.45
```

get_generic_interval(*directed=True*)

Generic interval (directed) between two tones.

Parameters **directed** (*bool*) – Affects whether the returned interval is directed or not.

Returns (Directed) generic interval from *s* to *t*.

Return type int

Example

```
>>> db = Tone(0,-1,-1) # Db,0
>>> b = Tone(0,1,1) # B'0
>>> i1 = Interval(db, b) # the interval between Db0 and B1 is an ascending␣
↪thirteenth
>>> i1.generic_interval()
13
```

```
>>> i2 = Interval(b, db) # the interval between B1 and Db0 is a descending␣
↪thirteenth
>>> i2.generic_interval()
-13
```

```
>>> i3 = Interval(b, db) # the interval between B1 and Db0 is a descending␣
↪thirteenth
>>> i3.generic_interval(directed=False)
13
```

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

INDEX