musictheory

Release 0.0.1

Fabian C. Moss

May 25, 2020

CONTENTS

Warning: The content on these pages is very much under construction!

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 longa (or a quadruple whole note), symbolically reflects this level of difficulty. since it is the longest note value in Western music notation.

This project is inspired by a number of great books, e.g.

- Aldwell & Schachter (2010). *Harmony and Voice Leading*.
- Cadwallader & Gagné (1998). *Analysis of Tonal Music. A Schenkerian Approach*.
- Forte (1977). *The Structure of Atonal Music*.
- Lewin (1987). *Generalized Intervals and Transformations*.
- Müller (2015). *Fundamentals of Music Processing*.
- Straus (2005). *Introduction to Post-Tonal Theory*.

What is new and unique about the approach taken here is that we take a computational perspective and implement all introduced concepts. This does not only provide us with sharp and unequivocal definitions, but also allows us to scale music theory up from the analysis of individual bars, sections, or pieces to that of entire repertoires and corpora!

I recently also discovered Music for Geeks and Nerds by Pedro Kroger which looks very interesting. The Python project musthe also seems to pursue a similar goal.

Note: 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...
- Music as the "missing Humboldt system" (Merker, 2002) with its "orthogonal discretization of spectro-temporal space" (Merker, 2002:4)

1.2 Quickstart

> **Warning:** These instructions do not work yet.

To install the Python library `musictheory`, type the following in your terminal:

```
pip install musictheory
```

1.3 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.

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)

1.4 Acknowledgements

- DCML

CHAPTER

TWO

FUNDAMENTALS

The theory presented in here can be described as a *tonal theory* in the sense that its most fundamental objects are *tones*, discrete musical entities that have a certain location in tonal space. A tonal space is then a metrical space describing all possible tone locations, and the metric is given by an *interval function* between the tones. Note that by this definition, there are as many different tonal spaces as there are interval functions.

While many aspects and examples will be taken from Western (classical) music, the theory is in principle not restricted to this tradition but extends well to virtually all musical cultures where a tone is a meaningful concept.

2.1 Tones

Let's start with a mental exercise: imagine a tone. Contemplate for a while what this means. Does this tone have a pitch? A duration? A velocity (volume)?

- Riemann (1916). *Ideen zu einer Lehre von den Tonvorstellungen*:

"The ultimate elements of the tonal imagination are single tones." (Wason & Martin, 1992, 92)

Bearing that in mind, let's create (or *instantiate*) a tone. To do so, we need to conceptualize ("vorstellen" in Riemann's terminology) a *tone location* ("Tonort", Mazzola 1985, 241). There are many different ways to do this. In fact, the way we specify the location of a tone defines the tonal space in which it is situated.

2.1.1 Euler Space

One option is to locate a tone t as a point $p = (o, q, t)$ in Euler Space, defined by a number of octaves o, fifths q, and thirds t. We will use the `musictheory.Tone` class for this

```
from musictheory import Tone

t = Tone(octave=0, fifth=0, third=0)
```

Printing this tone will give us the common name.

```
t
>>> Tone(C)
```

A shorter way to initialize a tone is by just passing a triple to the class:

```
g = Tone(0,1,0)
>>> Tone(G)
```

From this representation we can derive a variety of others, corrsponding to transformations of tonal space.

Tone triples can be assigned a more-readable label, e.g. `C4',Dbb2,,`.

Regular expression: `[A-G](#|b)*d(,|')+`.

2.1.2 Frequencies

Each tone corresponds to some *fundamental frequency* f in Hertz (Hz), oscillations per second.

- Overtone series
- frequency ratios
- logarithm: multiplication => addition

In order to determine the *frequency* of a tone, it is necessary to define a reference point. This reference point, also called the *Kammerton* is nowadays set to $\mathrm{A}4 = 440$Hz, i.e. the frequency of the tone `A4_` is 440 oscillations per seconds. If m is the MIDI number of the tone in questions, then

$$f = K \cdot 2^{(m-69)/12}$$

The frequency of a tone can be accessed via the *`Tone.get_frequency()`* method. attribute:

```
t.get_frequency()
>>> 392.0
```

Changing the chamber tone will, of course, change the frequency for each tone:

```
t.get_frequency(chamber_tone=442.0)
>>> 393.78
```

2.1.3 Octave equivalence

Octave equivalance considers all tones to be equivalent that are separated by one or multiple octaves, e.g C1, C2, C4, C10 etc. More precisely, all tones whose fundamental frequencies are related by multiples of 2 are octave equivalent.

2.1.4 Tonnetz

The *Tonnetz* does not contain octaves and thus corresponds to a projection

$$\pi : (o, q, t) \mapsto (q, t).$$

2.2 Pitch classes

A very common object in music theory is that of a *pitch class*. Pitch classes are equivalence classes of tones that incorporate some kind of invariance. The two most common equivalences are *octave equivalence* and *enharmonic equivalence*.

2.2.1 Enharmonic equivalence

If, in addition to octave equivalence, one further assumes enharmonic equivalence, all tones separated by 12 fifths on the line of fifths are considered to be equivalent, e.g. A♯ and B♭, F♯ and G♭, G♯, and A♭ etc.

The notion of a pitch class usually entails both octave and enharmonic equivalence. Consequently, there are twelve pitch classes. If not mentioned otherwise, we adopt this convention here. The twelve pitch classes are usually referred to by their most simple representatives, i.e.

$$\text{C, C♯, D, E♭, F, F♯, G, A♭, A, B♭, B,}$$

but it is more appropriate to use *integer notation* in which each pitch class is represented by an integer $k \in \mathbb{Z}_{12}$.

$$\mathbb{Z}_{12} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\},$$

and usually one sets $0 \equiv$ C. This allows to use *modular arithmetic* do calculations with pitch classes.

2.2.2 MIDI

The Musical Instrument Digital Interface format (MIDI) was developed to communicate between electronic musical instruments. Each pitch class (assuming enharmonic but not octave equivalence) is represented by an integer between 0 and 127, and the MIDI number for pitch C4 is set to 60. Increasing a MIDI number corresponds to the number of semitones. Since an octave contains twelve semitones, a fifth contains seven semitones, and a major third contains four semitones, we can determine the MIDI pitch number for any tone *t* by multiplying its Euler coordinates with the respective number of semitones and add it to the MIDI pitch for the tonal center (60).

$$m = 60 + 12 \cdot o + 7 \cdot f + 4 \cdot t$$

2.2.3 Other invariances

OPTIC

2.2.4 Tuning / Temperament

2.3 Intervals

We can add an interval to a tone:

```
>>> t = Tone(0,1,0) # G

>>> f = Tone(0,-1,0) # F
>>> a = Tone(0,-1,1) # A
>>> i = Interval(f,a) # +M3

>>> t + i
B
```

Analogously, we can also substract an interval from a tone:

```
>>> t - i
Eb
```

Moreover, we can add or substract intervals from each other:

```
>>> j = Interval(a, f) # +m6
>>> i + j
P8
```

```
>>> i - j
-A4
```

- Pitch intervals
- Ordered pitch-class intervals (-> rather directed)
- Unordered pitch-class intervals
- Interval classes
- Interval-class content
- Interval-class vector

2.3.1 GISs

2.4 Pitch-Class Sets

Let $y = \{y_1, \ldots, y_m\}$ be a pitch-class set.

- Sets that contain pitch classes
- ordered: {0,4,7}
- unordered: {7,0,4}

2.4.1 Normal Form

- smallest difference between last and first element
- (see algorithm in Straus,2005)

2.4.2 Transposition

transposition: adding n to each pc (mod 12)

- {0,4,7} + 7 = {7,11,14} = {7,11,2}

The *transposition* of a pitch-class set *y* by *n* semitones is given by

$$\begin{aligned} T_n(y) &= y + n \mod 12 \\ &= \{y_1 + n \mod 12, \ldots, y_m + n \mod 12\} \end{aligned}$$

2.4.3 Inversion

inversion: reversing the sign of each pc (mod 12)

- [0,4,7] => [0,-4,-7] = [0,8,5]

The *inversion* of a pitch-class set y is given by

$$\begin{aligned} I(y) &= -y \mod 12 \\ &= \{-y_1 \mod 12, \ldots, -y_m \mod 12\} \end{aligned}$$

- Inversion In, Ixy

Note: Note that this definition is an entirely different concept than *chord inversion* with which we will deal in later chapters.

2.4.4 Index number

- Forte numbers: <cardinal number>-<ordinal number>
- ordinal number is position on the list
- [0,1,3,6,9] => 5-31

2.4.5 Set Class

2.4.6 Prime Form

- 0 is first entry
- 220 different pc sets in prime form (equivalence by transposition or inversion)

Transformations between representations of tones are actually *transformations of tonal space*.

[Diagram of relations between different representations.]

2.5 The diatonic scale

Music in the Western tradition fundamentally builds on so-called *diatonic* scales, an arrangement of seven tones that are named with latin letters from A to G. “Diatonic” can be roughly translated into “through all tones”. Within this scale, no tone is privileged, so the diatonic scale can be appropriately represented by a circle with seven points on it. Mathemacally, this structure is equivalent to $\mathbb{Z}_7$.

[tikz figure here]

Now, if we want to determine the relative relations between the tones, it is necessary to assign a reference tone that is commonly called the *tonic*, or *finalis* in older music.

For example, if the tone D is the tonic, we can determine all other scale degrees as distance to this tone. Scale degrees are commonly notated with arabic numbers with a caret:

$$\text{D} : \hat{1}$$
$$\text{E} : \hat{2}$$
$$\text{F} : \hat{3}$$
$$\text{G} : \hat{4}$$
$$\text{A} : \hat{5}$$
$$\text{B} : \hat{6}$$
$$\text{C} : \hat{7}$$

2.6 Other scales

2.6.1 Pentatonic

2.6.2 Scales based on chromaticism

- chromatic
- hexatonic
- octatonic
- whole tone

2.7 Modes

scale plus order plus hierarchy (but order already defined above?)

Different terminologies:

- Messiaen's Modes
- Church modes
- Indian modes (ragas)
- other modes?

2.8 Keys

2.9 Time

2.9.1 Notes

(Tones + Duration) blablabla. . .

2.9.2 Rhythm

(Duration patterns)

2.9.3 Meter

(Hierarchy)

2.9.4 Musical time vs. performance time

2.10 Notes on Segmentation

- Straus 2005
- Hanninen 2012

CHAPTER

THREE

INDICES AND TABLES

- genindex
- search

CHAPTER

FOUR

DEVELOPERS

4.1 API Reference - musictheory

4.1.1 Tone

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

Class for tones.

get_accidentals()

Gets the accidentals of the tone (flats (♭) 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_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_name()
Gets the complete name of the tone, consisting of its step, 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_name(), ab.get_name()
`C_0` `Ab,1`
```

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()
`'` `_` `,`
```

4.1.2 Interval

class musictheory.**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_frequency_ratio()
Frequency ratio of the interval. Needs `fractions.Fraction` class.

Returns Frac – Integer ratio of the interval.

get_generic_interval(*directed=True*, *octaves=True*)
Generic interval (directed) between two tones.

Parameters

- **directed** (*bool*) – Affects whether the returned interval is directed or not.
- **octaves** (*bool*) – returns generic interval class if *False*.

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
```

get_specific_interval(*directed=True*, *octaves=True*)
Specific interval (directed) between two tones.

Parameters

- **directed** (*bool*) – Affects whether the returned interval is directed or not.
- **octaves** (*bool*) – returns specific interval class if *False*.

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

Return type int

Example

```
>>> fs = Tone(0,2,1) # F#'0
>>> db = Tone(0,-1,-1) # Db,0
>>> i1 = Interval(fs, db)
>>> i1.specific_interval()
17
```

```
>>> i1.specific_interval(octaves=False)
5
```

PYTHON MODULE INDEX

m

INDEX

G

I

M

T