musictheory

Release 0.0.1

Fabian C. Moss

May 09, 2021

CONTENTS

Warning: The content on these pages is very much under construction! Since this is ongoing work, I can give no guarantee for completeness or accuracy. Feel free to contact me with your questions and suggestions!

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 maxima, the longest note value in medieval music, symbolically reflects this level of difficulty.

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

- Aldwell *et al.* [[1]]
- Lewin [[8]]
- Straus [[18]]
- Cadwallader and Gagné [[2]]
- Müller [[13]]

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.

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 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 I

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 Frequencies

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

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

2.1.2 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(o=0, q=0, t=0)
```

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

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.

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 Other invariances

OPTIC

2.2.3 Tuning / Temperament

2.3 Intervals

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

2.3.1 GISs

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

[Diagram of relations between different representations.]

CHAPTER

THREE

FUNDAMENTALS II

3.1 Scales and Modes

3.1.1 Indian classical music

3.1.2 Western classical music

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:

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

Modes

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

Keys

3.1.3 Jazz

3.1.4 Other scales

- chromatic
- hexatonic
- octatonic
- whole tone
- Messiaen

3.2 Time

- beats
- seconds
- onsets

3.2.1 Notes

(Tones + Duration) blablabla. . .

3.2.2 Rhythm

(Duration patterns)

3.2.3 Meter

(Hierarchy)

3.2.4 Musical time vs. performance time

CHAPTER

FOUR

PITCH-CLASS SET THEORY

Content adapted from Forte [[5]], Straus [[18]]. In this chapter, "pitch class" always entails octave *and* enharmonic equicalenve.

4.1 Pitch classes

4.1.1 Octave equivalence

Octave equivalence maps all pitch classes with the same label/name to one class, e.g. all C's on the piano get mapped to pitch class C, all Ab's get mapped to pitch class Ab, and all G#'s get mapped to G#.

Octave equivalence has perceptual correlates (cite paper) and is frequently employed in composition. For instance, the double bass sounds an octave lower as a cello but they are notated identically, or a fugue subject enters in different registers. The octave is the most fundamental interval with a frequency ratio of 2:1. Octave equivalence transforms the Euler space to the Tonnetz $\mathbb{Z} \times \mathbb{Z}$.

[diagram]

4.1.2 Enharmonic equivalence

Enharmonic equivalence describes the identification of pitches like F#4 and G#4. Enharmonic equivalence transforms the Euler Space to a tube, namely $\mathbb{Z}_{12} \times \mathbb{Z}$ (the chromatic circle and the octave line).

[diagram]

Taking octave equivalence and enharmonic equivalence together, we arrive at the torus that contains all twelve enharmonic (and octave-related) pitch classes, and which can be represented by numbers $k \in \mathbb{Z}_{12}$

[diagram]

Both equivalences are indepentend of one another, i.e. we have the following commuting diagram between tonal spaces:

[diagram]

Assuming enharmonic equivalence has a number of implications:

- we cannot distinguish between diatonic and chromatic semitones
- we cannot distinguish between augmented fourths and diminished fifths
- we cannot distinguish between dominant seventh chords and augmented sixth chords
- and more

Note: Describe relation to equal temperament.

For the remainder of this chapter, pitch classes designate entities for which both equivalences are assumed. It is thus appropriate to represent them on a circle. Since each pitch class can have infinitely many spellings, going the reverse direction is a difficult inference problem, for which a number of algorithms have been proposed (see advanced chapter on *Pitch Spelling*).

4.2 Intervals

Generally, intervals describe the distance between two points (see Chapter *Intervals*). But depending on which representation for tones we chose (or, equivalently, which equivalences we assume) the types of the intervals changes as well.

Note: Add paragraph about what types are.

4.2.1 Pitch Intervals

Pitch intervals describe the distance between two pitches in semitones. That is, we do assume enharmonic equivalence but not octave equivalence. Consequently, we can visualize pitch intervals in

Ordered pitch-class intervals

Unordered pitch-class intervals

Interval class

Interval class content

Interval class vector

4.3 Pitch-class sets

Normal form

Transposition

Inversion I

Index number

Inversion II

Set class

Prime form

Segmentation and analysis

4.4 Relationships

Common tones under transposition

Transpositional symmetry

Common tones under inversion

Inversional symmetry

Z-relation

Complement relation

Subset and superset relations

Transpositional combination

Contour relations

Composing out

Voice-leading

Atonal pitch space

4.5 Advanced concepts

Tonality

Centricity

Inversional axis

The diatonic collection

The octatonic collection

The whole-tone collection

The hexatonic collection

Collectional interaction

Interval cycles

Triadic post-tonality

4.6 Twelve-tone theory

Twelve-tone series

Basic operations

Subset structure

Invariants

CHAPTER

FIVE

HARMONY

- major-minor tonality
- chords: - root on (altered) diatonic scale degree - stack of thirds - chromatic alterations

First note based (Tone-based)

Then arrive at annotation standard (simple version) and do simple regex filtering (find all Vs. . .)

Moss *et al.* [[12]], Neuwirth *et al.* [[14]]

CHAPTER

SIX

SEGMENTATION

6.1 Free segmentation

(pc set analysis)

- Straus [[18]]
- Hanninen [[6]], Hanninen [[7]]

relating segments creates specific graph structure

6.2 Contiguous segmentation

chord labels, Roman numerals, etc.

graph: chain

6.3 Hierarchical segmentation

CFGs, Schenker

graph: tree

6.4 Exhaustive segmentation

Keyscapes Sapp [[15]], Sapp [[16]], Pitchscapes Lieck and Rohrmeier [[9]], Wavescapes Viaccoz *et al.* [[20]]

See also Müller (form)

CHAPTER

SEVEN

ADVANCED APPLICATIONS

7.1 Pitch Spelling

Meredith [[10]], Meredith and Wiggins [[11]] Cambouropoulos [[3]], Chew and Chen [[4]] Stoddard *et al.* [[17]] Temperley [[19]]

7.2 Style (classification)

7.2.1 Feature clustering

(k-means, PCA, . . .)

7.2.2 Hierarchical clustering

7.3 History

(regression, GPs)

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

7.4 Performance

- Spotify API to compare different recordings

CHAPTER

EIGHT

DEVELOPERS

8.1 Installation

Warning: These instructions do not work yet.

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

```
pip install musictheory
```

8.2 API - musictheory

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_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
```

class musictheory.**PitchClass**

Pitch class instance in $\mathbb{Z}_{12}$.

class musictheory.**PitchClassSet**(*st*)

Pitch class sets

interval_class_vector()

Interval-class vector for given pitch-class set

Returns interval-class vector

Return type list

invert(*t*)
Invert pitch-class set

Parameters **t** (*int*) – inversion pitch class

Returns inverted pitch-class set

Return type set

normal_form()
Normal form of pitch-class set

Returns Normal form

Return type set

transpose(*t*)
Transposition by *t* semitones.

Parameters **t** (*int*) – number of semitones to transpose up

Returns transposed pitch-class set

Return type set

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

BIBLIOGRAPHY

[1] Edward Aldwell, Carl Schachter, and Allen Cadwallader. *Harmony and Voice Leading*. Cengage Learning, 4th edition, 2010.

[2] Allen Cadwallader and David Gagné. *Analysis of Tonal Music. A Schenkerian Approach*. Oxford University Press, 1998.

[3] Emilios Cambouropoulos. Pitch Spelling: A Computational Model. *Music Perception: An Interdisciplinary Journal*, 20(4):411–429, June 2003. doi:10.1525/mp.2003.20.4.411.

[4] Elaine Chew and Yun-Ching Chen. Real-Time Pitch Spelling Using the Spiral Array. *Computer Music Journal*, 29(2):61–76, June 2005. doi:10.1162/0148926054094378.

[5] Allen Forte. *The Structure of Atonal Music*. Yale University Press, New Haven and London, 1977.

[6] Dora A Hanninen. Orientations, Criteria, Segments: A General Theory of Segmentation for Music Analysis. *Journal of Music Theory*, 45(2):345–433, 2001.

[7] Dora A. Hanninen. *A Theory of Music Analysis: On Segmentation and Associative Organization*. University of Rochester Press, Rochester, NY, first edition, December 2012.

[8] David Lewin. *Generalized Musical Intervals and Transformations*. Oxford University Press, Oxford, 1987.

[9] Robert Lieck and Martin Rohrmeier. Modelling Hierarchical Key Structure with Pitch Scapes. In *Proceedings of the 21st International Society for Music Information Retrieval Conference (ISMIR 2020)*, 811–818. Montreal, Canada, 2020. doi:10.5281/zenodo.4245558.

[10] David Meredith. The ps13 pitch spelling algorithm. *Journal of New Music Research*, 35(2):121–159, June 2006. doi:10.1080/09298210600834961.

[11] David Meredith and Geraint A Wiggins. Comparing Pitch Spelling Algorithms. *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 280–287, 2005. doi:10.1007/978-3-540-31807-1_14.

[12] Fabian C. Moss, Markus Neuwirth, Daniel Harasim, and Martin Rohrmeier. Statistical characteristics of tonal harmony: a corpus study of Beethoven's string quartets. *PLoS ONE*, 14(6):e0217242, 2019. doi:10.1371/journal.pone.0217242.

[13] Meinhard Müller. *Fundamentals of Music Processing*. Springer, 2015.

[14] Markus Neuwirth, Daniel Harasim, Fabian C. Moss, and Martin Rohrmeier. The Annotated Beethoven Corpus (ABC): A Dataset of Harmonic Analyses of All Beethoven String Quartets. *Frontiers in Digital Humanities*, 5(July):1–5, 2018. doi:10.3389/fdigh.2018.00016.

[15] Craig Stuart Sapp. Harmonic Visualizations of Tonal Music. In *The International Computer Music Conference (ICMC)*, 423–430. 2001.

[16] Craig Stuart Sapp. Visual hierarchical key analysis. *Computers in Entertainment*, 3(4):3, 2005. doi:10.1145/1095534.1095544.

[17] Joshua Stoddard, Christopher Raphael, and Paul E Utgoff. Well-tempered spelling: A key-invariant pitch spelling algorithm. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR 2004)*. Barcelona, Spain, 2004.

[18] Joseph Nathan Straus. *Introduction to post-tonal theory*. Pearson Prentice Hall, New York, 3rd edition, 2005.

[19] David Temperley. *The Cognition of Basic Musical Structures*. MIT Press, 2001.

[20] Cédric Viaccoz, Daniel Harasim, Fabian C. Moss, and Martin Rohrmeier. Wavescapes: A Visual Hierarchical Analysis of Tonality using the Discrete Fourier Transformation. *Musicae Scientiae*, 2021.

PYTHON MODULE INDEX

m

INDEX

G

I

M

N

P

T