musictheory

Release 0.0.1

Fabian C. Moss

Mar 29, 2021

CONTENTS

1 Introduction **3**
- 1.1 What is CM? . . . 3
- 1.2 General notes . . . 3
- 1.3 Notes (old) . . . 3

2 Fundamentals **5**
- 2.1 Tones . . . 5
- 2.2 Pitch classes . . . 6
- 2.3 Intervals . . . 6
- 2.4 Pitch-Class Sets . . . 7
- 2.5 Western tonal music . . . 7
- 2.6 Time . . . 8
- 2.7 Notes on Segmentation . . . 8

3 Pitch-Class Set Theory **9**
- 3.1 Pitch classes . . . 9
- 3.2 Intervals . . . 9
- 3.3 Pitch-class sets . . . 9
- 3.4 Relationships . . . 10
- 3.5 Advanced concepts . . . 10
- 3.6 Twelve-tone theory . . . 10

4 Fourier Pitch Space **11**
- 4.1 A hierarchy of pitch-class vectors . . . 11
- 4.2 Discrete Fourier Transform . . . 11
- 4.3 Color mapping of Fourier coefficients . . . 12
- 4.4 Wavescapes . . . 13

5 Indices and tables **15**

6 Developers **17**
- 6.1 Installation . . . 17
- 6.2 API - musictheory . . . 17

Python Module Index **23**

Index **25**

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 hemidemisemiquaver (or a sixty-fourth note), symbolically reflects this level of difficulty.

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 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

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

2.4 Pitch-Class Sets

- Sets that contain pitch classes

2.4.1 Normal Form

2.4.2 Transposition

2.4.3 Inversion

- Inversion In, Ixy

2.4.4 Index number

2.4.5 Set Class

2.4.6 Prime Form

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

[Diagram of relations between different representations.]

2.5 Western tonal music

2.5.1 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:

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

2.5.2 Modes

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

2.5.3 Keys

2.5.4 Other scales

- chromatic
- hexatonic
- octatonic
- whole tone

2.6 Time

2.6.1 Notes

(Tones + Duration) blablabla. . .

2.6.2 Rhythm

(Duration patterns)

2.6.3 Meter

(Hierarchy)

2.6.4 Musical time vs. performance time

2.7 Notes on Segmentation

- Straus 2005
- Hanninen 2012

CHAPTER

THREE

PITCH-CLASS SET THEORY

Content adapted from Straus (2005). In this chapter, "pitch class" always entails octave *and* enharmonic equicalenve.

3.1 Pitch classes

Octave equivalence

Enharmonic equivalence

3.2 Intervals

Pitch Intervals

Ordered pitch-class intervals

Unordered pitch-class intervals

Interval class

Interval class content

Interval class vector

3.3 Pitch-class sets

Normal form

Transposition

Inversion I

Index number

Inversion II

Set class

Prime form

Segmentation and analysis

3.4 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

3.5 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

3.6 Twelve-tone theory

Twelve-tone series

Basic operations

Subset structure

Invariants

CHAPTER

FOUR

FOURIER PITCH SPACE

Note: The following is taken from the Wavescapes paper!

We first introduce the segmentation of a musical piece into a hierarchy of pitch-class vectors. Then, we describe the Discrete Fourier Transform (DFT) and its application to this hierarchy, and a color mapping of the obtained Fourier coefficients. This procedure results in a visual representation that we call *wavescapes*.

4.1 A hierarchy of pitch-class vectors

A *segmentation* of a musical piece is a partition into N non-overlapping segments of equal length r. In this general sense, segments can be defined by musical units in symbolic scores (e.g., measures, note durations) as well as by continuous durations in audio recordings (e.g., seconds, onsets). We represent the q-th segment by a *pitch-class vector* (PCV) $x_q \in \mathbb{R}^{12}_{\geq 0}, 1 \leq q \leq N$, whose entries contain the total durations (also called weights) of the twelve pitch classes in this segment. A pitch class is the equivalence class of all octave-related pitches in twelve-tone equal temperament, assuming enharmonic equivalence. The value $x_q[0]$ is the weight of pitch class C, $x_q[1]$ is the weight of C♯, and so forth. For example, the PCV of the first four measures of J.~S. Bach's Prelude in C~major is $x = (14, 0, 9, 0, 9, 2, 0, 3, 0, 1, 0, 4)$, where the duration of each pitch class is given in quarter notes.

A complete piece is modeled as a *hierarchy of segments* given by a function P that inclusively returns the pitch-class content from the n-th to the m-th segment,

$$\begin{aligned} P\colon \mathbb{N}^2 &\longrightarrow \mathbb{R}^{12}_{\geq 0} \\ (n, m) &\longmapsto x = \sum_{q=n}^{m} x_q, \text{for } 1 \leq n \leq m \leq N. \end{aligned}$$

The size as well as the hierarchical level of any segment with pitch-class content $P[n, m]$ is then $m - n + 1$, and there are $\sum_{n=1}^{N} n = \binom{n+1}{2}$ segments in total.

4.2 Discrete Fourier Transform

The *Discrete Fourier Transform* (DFT) decomposes a vector into a sum of sinusoidal waves of unique frequencies with varying amplitudes and phases. Mathematically, the DFT of any PCV x is given by

$$\begin{aligned} F\colon \mathbb{R}^{12}_{\geq 0} &\longrightarrow \mathbb{C}^{12} \\ x &\longmapsto X, \end{aligned}$$

where the k-th component of the complex-valued vector X is defined as

$$X[k] = \sum_{n=0}^{11} x[n] e^{-i2\pi n \frac{k}{12}}, \ \forall k \in [0, \dots, 11].$$

The values of $X[k]$ are referred to as *Fourier coefficients*, or simply *coefficients*. The zeroeth coefficient $X[0]$ is always equal to the sum of x. By symmetry, the coefficients for $k \in [1, \dots, 5]$ are conjugate to the ones for $k \in [11, \dots, 7]$ while the sixth coefficient is its own conjugate,

$$X[k] = \bar{X}[12 - k] \text{for } k \in [1, \dots, 11].$$

For this reason, we consider only coefficients 1 through 6 in accordance with previous research citep{amiot_david_2007,yust_generalized_2019}.

Since the Fourier coefficients are complex numbers, they have a representation in Cartesian coordinates through their real and imaginary parts,

$$X[k] = \mathrm{Re}(X[k]) + i \cdot \mathrm{Im}(X[k]),$$

and a representation in polar coordinates in terms of their *magnitude* μ_k and *phase* ϕ_k,

$$\begin{aligned} X[k] &= \mu_k \cdot e^{i\phi_k} \\ &= \mu_k \cdot (\cos(\phi_k) + i \cdot \sin(\phi_k)). \end{aligned}$$

The phase is defined as

$$\begin{aligned} \phi_k &= \measuredangle(X[k]) \\ &= \tan^{-1}\left(\frac{\mathrm{Im}(X[k])}{\mathrm{Re}(X[k])}\right) \end{aligned}$$

and the magnitude is defined as

$$\mu_k = |X[k]| = \sqrt{\mathrm{Re}(X[k])^2 + \mathrm{Im}(X[k])^2},$$

where $\measuredangle$ is the angle function $\measuredangle : \mathbb{C} \to [0, 2\pi[$.

Consider again the PCV $x = (14, 0, 9, 0, 9, 2, 0, 3, 0, 1, 0, 4)$. The fifth Fourier coefficient of x has the Cartesian representation $X[5] = (14.87 + i \cdot 19.09)$, phase $\phi_5 = 0.91$, and magnitude $\mu_5 = 24.19$.

4.3 Color mapping of Fourier coefficients

To visualize the Fourier coefficients of PCVs, we represent them in polar coordinates and map their phases and magnitudes to a color. Given the periodic nature of the phase, it can be associated to a color through a *circular hue*. Here, we choose the hue function $h : [0, 2\pi[\longrightarrow [0, 1]^3$ that maps ϕ_k to a triple (r, g, b), representing the strengths of the red, green, and blue components of the color citep{ong_generalization_2014}:

$$h(\phi_k) = \begin{cases} (1, \frac{3\phi_k}{\pi}, 0) & \text{if } 0 \leq \phi_k < \frac{\pi}{3} \\ (2 - \frac{3\phi_k}{\pi}, 1, 0) & \text{if } \frac{\pi}{3} \leq \phi_k < \frac{2\pi}{3} \\ (0, 1, \frac{3\theta}{\pi} - 2) & \text{if } \frac{2\pi}{3} \leq \phi_k < \pi \\ (0, 4 - \frac{3\phi_k}{\pi}, 1) & \text{if } \pi \leq \phi_k < \frac{4\pi}{3} \\ (\frac{3\phi_k}{\pi} - 4, 0, 1) & \text{if } \frac{4\pi}{3} \leq \phi_k < \frac{5\pi}{3} \\ (1, 0, 6 - \frac{3\phi_k}{\pi}) & \text{if } \frac{5\pi}{3} \leq \phi_k < 2\pi \end{cases}$$

The magnitude μ_k of a Fourier coefficient can be mapped to an opacity value $\alpha = \mu_k / X[0]$ by normalizing it with the sum of PCV x, given by its zeroeth coefficient $X[0]$.

citet{amiot_entropy_2020} uses the normalization

$$\alpha_k = \frac{|X[k]|^2}{\sum_{j=1}^{11} |X[j]|^2}, \text{ for } k = 1, \ldots, 11.$$

(A saturation mapping is also possible, but is overall inferior in visual quality compared to the opacity mapping.) The normalization of the magnitude also facilitates the comparison of different PCVs with one another.

We represent the phase and magnitude mappings by a coloring function C_k,

$$\begin{aligned} C_k : \mathbb{C}^{12} &\longrightarrow [0,1]^3 \times [0,1] \\ X &\longmapsto ((r,g,b),\alpha), \end{aligned}$$

which selects the k-th coefficient of X and uses the previous mappings on its phase and magnitude to return a color,

$$\begin{aligned} C_k(X) &= \left(h(\measuredangle(X[k])), \frac{|X[k]|}{X[0]} \right) \\ &= \left(h(\phi_k), \frac{\mu_k}{X[0]} \right). \end{aligned}$$

4.4 Wavescapes

To summarize, we showed how to generate a color given a pitch-class vector by successively applying the mappings F and C_k. Together with P, the mappings define an arrangement of colors for a given piece of music that we call a *wavescape*. More precisely, the wavescape for the k-th Fourier coefficient is given by

$$\begin{aligned} W_k : \mathbb{N}^2 &\longrightarrow [0,1]^3 \times [0,1] \\ W_k[n,m] &= (C_k \circ F \circ P)[n,m], \end{aligned}$$

for segment indices $n, m \in \mathbb{N}$ with $0 \leq n \leq m < N$. Following the hierarchical visual structure in Figure~ref{tab:visuhierarchy}, wavescapes are displayed as triangles of colors, similarly to keyscapes (Sapp).

Applying the procedure described above for a given piece renders a wavescape for each of the six coefficients. Each of them may show interesting properties for music analysis. In order to determine on which wavescape to concentrate our analyses, we focus on those with the largest average normalized magnitude to which we refer by $\bar{\alpha}_k$ for a given piece and coefficient k.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- search

CHAPTER

SIX

DEVELOPERS

6.1 Installation

Warning: These instructions do not work yet.

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

```
pip install musictheory
```

6.2 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**(*octave=None*, *fifth=None*, *third=None*, *name=None*)
Pitch class instance in $\mathbb{Z}_{12}$.

class musictheory.**PitchClassSet**(*octave=None*, *fifth=None*, *third=None*, *name=None*)
Pitch class sets

interval_vector()
Interval vector given a pitch-class 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()
```

PYTHON MODULE INDEX

m

musictheory, 17

INDEX

G

get_accidentals() (*musictheory.Tone method*), 18
get_euclidean_distance() (*musictheory.Interval method*), 17
get_frequency() (*musictheory.Tone method*), 19
get_generic_interval() (*musictheory.Interval method*), 17
get_label() (*musictheory.Tone method*), 19
get_midi_pitch() (*musictheory.Tone method*), 19
get_pitch_class() (*musictheory.Tone method*), 20
get_specific_interval() (*musictheory.Interval method*), 18
get_step() (*musictheory.Tone method*), 20
get_syntonic() (*musictheory.Tone method*), 20

I

Interval (*class in musictheory*), 17
interval_vector() (*musictheory.PitchClassSet method*), 18

M

module
    musictheory, 17
musictheory
    module, 17

P

PitchClass (*class in musictheory*), 18
PitchClassSet (*class in musictheory*), 18

T

Tone (*class in musictheory*), 18