Computational melody analysis

Klaus Frieler

Introduction

This chapter provides an introduction into both the basic principles of computational melody analysis and their application and implementation in MeloSpySuite/GUI as developed and applied within the Jazzomat Research Project, especially within the case studies in the second part of this volume. The chapter addresses at a wide and diverse audience ranging from jazz researchers and musicologists seeking for an understanding of MeloSpyGUI, its possibilities and functionalities, to computer scientists who wishes to comprehend or build on the concepts used within the software. Of course, both groups have different needs and different sets of previous knowledge. We tried to accommodate as many of these needs as possible and decided on a dual strategy of using exact mathematical notations on the one hand and verbal descriptions and concrete examples on the other. Mathematical notation might be hard to read or even off-putting for non-experts, but in the spirit of reproducibility and transparency it is inevitable if one wishes to be as precise as possible for those who want to know or to recreate the details of an algorithm. Some mathematical basics as well as details regarding the implementation of features and categories are included in 'info boxes' along the main text. Additionally, there are short introductions to the FlexQ quantization algorithm and into circular mathematics in the appendix. We think that those readers who are more interested in the applications and the functionality of the software for jazz research as well as music research in general can safely skip the formulas and stick to the verbal contexts and particularly the examples. Nevertheless, a minimum openness to the math might be necessary for a more comprehensive picture.

This chapter focuses on three main topics: Firstly, the basic mathematical concepts for the computational representation of music used within the Wei-

mar Jazz Database as well as the main annotation categories (segmentation, chords, meter and midlevel units) are explained. Secondly, both the general concepts of feature extraction and the most important musical features that can be extracted with our software are discussed. The features are grouped into the main types in alphabetical order: accents, auxiliary, melodic contour, intervals, metadata, meter, pitch, rhythm, sequences of intervals pitch or rhythm, structure, and tone formation (including dynamics, articulation, and intonation). The last section is dedicated to the concepts of pattern search and pattern mining which is a powerful tool within the MeloSpy software.

Music representation and annotation

Computational musicology aims to extract useful information from music by using suitable (digital) representations of music. This is a very general task with many different approaches, depending on the concrete goals at hand. It is very unlikely that any representational system will fulfill all possible needs. In fact, quite a few representational systems have already been proposed, implemented, and used. The best-known approach is common Western musical notation, which was originally developed for practical purposes. But since it features rather good mapping between scores and musical objects, musicologists and music theoreticians have used common Western musical notation for many centuries and are still using it today. In fact, it is still the most commonly used analytical representation. Many musical representation systems aim towards a digital version of common Western musical notation, e. g., MusicXML, MEI, MuseScore and Lilypond. But since common Western musical notation is basically a prescriptive system fraught with a lot of historic conventions, idiosyncrasies, and implicit assumptions, it is not always well-suited for analytical tasks, e. g., for performance research where information on micro-timing, loudness, and timbre is needed.

Moreover, since common Western musical notation is basically a graphical (two-dimensional) representation, encoding it to a digital format is not always straight-forward and the results might not be well suited for analytical purposes due to of many purely graphical elements. Hence, computational analysis very often relies on more task-compatible encoding formats, e. g., `**kern`, ABC, EsAC or MIDI, all of which have all their own advantages and disadvantages. They are all based on certain (simplifying) assumptions, which reduce the expressiveness of the encoding, i. e., the amount of available information.

In regard to expressiveness, audio files contain maximal information in the sense that they facilitate a re-creation of the sounding music with the aid

of a suitable audio player. However, this maximal expressiveness comes at a high analytical cost, since a large amount of effort is necessary to extract the analytically relevant information. This poses technological problems that can not be deemed fully solved today, though impressive progress has been made in this regard in the field of music information retrieval (MIR) in the last two decades. Music is ultimately a psychological phenomenon and the human cognitive system acts as a very powerful and specific filter system which, for example, can easily extract the main melody from a complex polyphonic track, a task that modern algorithms are still struggling with. Also, musically very important phenomena such as beat induction and meter or tonality perception are not easily defined and extracted from complex audio files, although these are often the focus of analytical endeavors.

In a more formal sense, a musical performance[1] can be viewed (physically) as a sound-wave $p(t)$ over a bounded time interval T. This sound wave is then transformed into a psychological representation $p'(t)$ during listening. We further assume that music differentiates into streams of (sonic, musical) event series. This assumption is strongly supported by research on auditory scene analysis (Bregman, 1990). Events in this context have a defined beginning (onset) and end (offset). The content of a musical event, however, cannot be described easily in full generality, particularly not in psychological terms. But very often it can be agreed upon that musical events have certain qualities (*qualias*) such as pitch, loudness, and timbre as well as modulations of these qualities.

Infobox 1: Mathematical basics: sets, maps, and sequences

The math used here deals mostly with 'sets', 'maps', and 'sequences'. A set is basically a collection of items, each counted only once and in no particular order. Sets are notated using curly brackets which embrace either a list of elements or some sort of condition that defines the elements in the set. For example, $A = \{1, \ldots, N\}$ is the set of the first N natural numbers. Standard sets often used in the context of computational musicology are the natural numbers (symbol $\mathbb{N}$), the integer numbers (symbol $\mathbb{Z}$), and the real numbers (symbol $\mathbb{R}$). To indicate that x is contained in a set A, one writes $x \in A$. For example, $17 \in \mathbb{N}$. The empty set with no elements is notated $\emptyset$. The numbers of elements in a set A is written as $|A|$. Sets can have infinitely or finitely many elements. Sets can also have subsets, notated $A \subset B$. Sets of consecutive numbers are

[1] We do not wish to go into the complicated discussion of what music actually *is* here, but simply take this as an externally given label. However, in certain audio engineering applications, it is an important to classify chunks of audio files into music or non-music, e. g., speech, environmental noise, or bird song.

very important in the context of computational musicology and are written as $[N:M] = \{N, N+1, \ldots, M\}$, which means all integers between N and M including M. It holds $[N:M] \subset \mathbb{N}$. Intervals of real numbers are written with a comma instead of a colon, e. g., $[x, y]$ is the interval of all real numbers between and including x and y. Sets have intersections $A \cap B$, i. e., the set of all elements both in A and B, and unions $A \cup B$, i. e., the set of elements either in A or B. Large symbols are used to indicate intersections and unions of more than one set, e. g., $\bigcup_{1 \leq i \leq N} A_i$ is a shorthand notation for $A_1 \cup A_2 \cup \cdots \cup A_N$. Another important construction is the so-called Cartesian product. For two sets A and B the Cartesian product $A \times B$ is the set of pairs (x, y) where $x \in A$ and $y \in B$. This also works for more than two sets, and the elements are then called N-tuples for a Cartesian product of N sets. The order is important, i. e., $A \times B \neq B \times A$. Maps are collections of arrows between two sets, starting in a domain set, say A, and ending in a image set (or co-domain), say B. A map from A to B is written as $M : A \to B$. If one wants to express the mapping of a concrete element in the domain to an element in the image of M, one writes $x \mapsto y = M(x)$. Not all elements in B have to be the endpoint of an arrow. Likewise, not all elements in A need to be a starting point, but in most cases we will assume this to be the case. We also assume that at most one arrow points from each element in the domain to an element in the image set. Maps can also be defined between maps, which will be encountered quite often in computational musicology, e. g., in the case of transformations or vector features. This comes from the fact that sequences play an important role.
A sequence is simply a collection of elements with an order; thus, one can say such things as "element x appears earlier than element y in the sequence". Since the integer numbers have a natural order of this kind, they can be used to define sequences as maps from integer intervals onto a certain image set X. For example, $x : [1:N] \to X$, with $i \mapsto x(i) \in X$. Very often, index notation is used for this, i. e., $i \mapsto x_i$. Another compact notation for this is with simple parentheses $(x_i)_{1 \leq i \leq N}$, or, even lazier, just x_i. An explicit listing of elements in parentheses means that it is a sequence or a tuple and not just a set.

In some cases, e. g., for percussive events, a simple (task-relative) categorization might be sufficient, e. g., naming the percussive instrument that has produced the event. In other cases, a justifiable simplification is to use a constant single pitch as a sole descriptor, whereby pitch is represented with a symbol (e. g., a note name) or a number as a proxy derived from the fundamental frequency (f_0) of the event. These events are called tone events. Chords can then be represented as collections of pitches (and further abstracted into chord symbols).

In the context of the Jazzomat Research Project, we are dealing exclusively with monophonic solos and therefore use exactly this simplified represen-

tation as a core, complemented by various annotations. Formally, thus, a melody is a discrete times series

$$t_i \mapsto e_i = (t_i, d_i, p_i),$$

where an event e_i is described by a triple (t_i, d_i, p_i) of numbers which represent

- onset t_i (seconds),
- duration d_i (seconds),
- and a (constant) pitch value p_i (MIDI number),

of a tone event.

Pitch values are represented by indices in the 12-tone equal tempered system, using the well-known and common MIDI indexing scheme (Selfridge-Field, 1997), which represents an associated f_0 value via the formula[2]

$$p = 12 \left\lfloor \log_2 \frac{f_0}{440\,\mathrm{Hz}} \right\rfloor + 69,$$

where the square brackets mean taking only the integer part. The note $A4 = 440\,\mathrm{Hz}$ is mapped to MIDI index 69 and every octave equals 12 semi-tone steps, e. g.,

$$\mathrm{A3}\,(220\,\mathrm{Hz}) \mapsto 12 \log_2 \frac{1}{2} + 69 = -12 + 69 = 57,$$

or

$$\mathrm{A5}\,(880\,\mathrm{Hz}) \mapsto 12 \log_2 2 + 69 = 12 + 69 = 81.$$

The number of events in a melody is called its length, whereas the duration of a melody is defined as the difference between the onset of its first element and the offset of its last event. For the following, we will use the convention that the first element in an event series is indexed by 0 (zero-indexing). Hence, a melody of length N is indexed with $0 \ldots N-1$. For intervals of integer numbers we will adopt the notation $[N : M]$ for the set of integers between and including N and M (with $M \geq N$). Formally, a melody of length N is then a map $e : [0 : N-1] \to \mathbb{R} \times \mathbb{R}^+ \times [0 : 127]$.

[2]The actual transcription process for the Weimar Jazz Database was performed by human transcribers who assigned the most fitting pitch to a tone event. See Chapter **??** for measuring the exact intonation of the played tones.

A melody is defined as monophonic, which means that its onsets should be strictly ordered ($t_i < t_{i+1}$, for all i). In a strictly monophonic melody, events do not overlap, i. e., the offset $t_i + d_i$ of an event should be smaller than or at most equal to the onset of the following event, $t_i + d_i \leq t_{i+1}$ for all i. The inter-onset intervals (IOI) are then defined as the differences of onsets between consecutive events, $\mathrm{IOI}_i = t_{i+1} - t_i$. Note that there are one fewer IOI than there are events. Semitone intervals are defined as differences of pitches between consecutive events, $\Delta p_i = p_{i+1} - p_i$. An elementary set of operations that can be applied to melodies or series is a projection, which is a formal term for 'forgetting information'. A projection means to only use a subset of the information, e. g., onsets, durations, or pitches or combinations thereof. For example, the sequence (t_i) is the list of onsets, the sequence (p_i) is the pitch sequence derived from a melody, and so on. Using only part of the information in a melody (or annotated melody, see below) is a very common technique for constructing features and defining patterns.

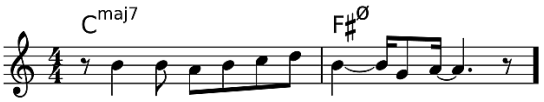

Figure 1: Beginning of Chet Baker's solo on "Let's Get Lost", also known as the "The Lick".

Example. The core representation and some derived values of the melody depicted in Figure 1 (in common Western musical notation) can be found in Table 1. In output files of the MeloSpyGUI the value `NA` (not available) is mostly used for this. Note that the difference values IOI and interval are attached to the first interval here, which is the convention used in the Jazzomat Research Project; other authors prefer to attach it to the second value. However, attaching difference values to the first element often simplifies subsequent computations.

Annotations

Obviously, this core representation of a melody is rather bare-bone and of only limited analytical value (although it already allows for the derivation of pitch and interval features as well as Markov models and very basic rhythm analysis). Thus, it is desirable to 'thicken' this description of tone events with a flexible annotation system, which allows the 'tagging' of additional information to an event or a series of events. Most annotations are generated externally, mostly by humans, and are not directly derivable from the basic

Table 1: Core representation and some derived values of the beginning of Chet Baker's solo on "Let's Get Lost".

	Core representation			Derived values		
Note	Onset (s)	Duration (s)	Pitch	Offset (s)	IOI (s)	Interval
1	2.667	0.195	71	2.862	0.395	0
2	3.062	0.087	71	3.150	0.166	-2
3	3.228	0.194	69	3.422	0.221	2
4	3.448	0.190	71	3.638	0.192	1
5	3.640	0.190	72	3.830	0.200	2
6	3.840	0.161	74	4.001	0.200	-3
7	4.040	0.280	71	4.310	0.360	-4
8	4.400	0.155	67	4.555	0.244	2
9	4.644	0.673	69	5.317	n. d.	n. d.

Note. n. d. = not defined.

representation. Most annotations are generated only once and then stored along with the basic presentations. Hence, the term annotation expresses a more practical distinction to transformations and features, as described below in p. 20.

One has to differentiate two types of annotations: local and contextual. Local annotations pertain to the event itself and are directly derived from it. Important examples are acoustical features such as f_0 modulations (vibrato, slides etc.), intensities/loudness or timbre (cf. Chapter **??**). Contextual annotations are only derivable with respect to a certain context of an event, e. g., segmentations (cf. Infobox 2). However, contextual annotation can always be represented (and stored) as local annotations to certain events, e. g., a phrase segmentation can be expressed by annotating phrase IDs. The contextuality must then be reflected by certain consistency conditions, e. g., the phrase IDs of two adjacent events must be equal or the second must be greater than that of the first.

Formally, we define an annotated melody as the map

$$t_i \mapsto (e_i, a_i),$$

where e_i is the core representation (t_i, d_i, p_i) and the annotations

$$a_i = (a_{i1}, a_{i2}, \ldots, a_{iM})$$

are M-dimensional tuples of values (i. e., vectors of M values for each event).

The only condition imposed on an annotation is applicability to each event in a series (which can always be fulfilled trivially by allowing the empty set or a similar special symbol as a valid annotation value). We will encounter several important annotations (e. g., metrical and chord annotations) in the following sections.

Segmentations

Segmentations play an important role in computational musicology. They occur naturally, e. g., in the form of melodic phrases, during melody perception (Lerdahl & Jackendoff, 1983). Segmentation simply refers to sets of subsequences or segments. A subsequence is any consecutive patch of a series, i. e., without 'holes'. Formally, a subsequence of a melody $e : [0 : N-1] \to e_i$ is a restriction on $[k : l]$ with $0 \leq k \leq l \leq N-1$. We will notate such a subsequence as $e_{k:l}$.

Two main kinds of segmentations can be differentiated: overlapping and non-overlapping. Overlapping segmentations contain segments that can share common events, whereas in non-overlapping segmentations segments are disjunctive. Exhaustive segmentations are those for which every event is contained in at least one segment. Pattern partitions (s. p. 42) are an important example of non-exhaustive overlapping segmentations, whereas phrase and chord annotations are an example for exhaustive non-overlapping segmentations. Very often, segmentations are themselves equipped with some kind of annotation, e. g., an ID or a value of arbitrary type, cf. Infobox 2 for examples. In the following, we will briefly discuss three important segmentation types—form parts, chord and midlevel annotation—which are specific to the jazz solos in the Weimar Jazz Database.

Infobox 2: Segmentations in the Weimar Jazz Database

Several important segmentations are included in the Weimar Jazz Database and accessible via different mechanisms in the MeloSpyGUI. On the database level, the following (exhaustive, non-overlapping) segmentations are available.

- *Phrase IDs* These are integer IDs, starting with phrase 1, for each phrase annotated by the transcriber.
- *Chord annotations*. These contain chord symbols for each event under the scope of this chord, which were manually annotated to the beat track. See Infobox 3 and Chapter **??**.
- *Form parts*. Annotated according to the lead sheet. Each event is also

part of a form part which is represented by specific form part symbols (e. g., A1, B2).

- *Chorus IDs*. Each solo consists of one or more choruses, i. e., cycles of the underlying form as embodied in the chord sequences. Chorus IDs are integers, starting from chorus 1. However, sometimes there are pick-up bars, i. e., a solo starting slightly before the beginning of a new form cycle. These pick-ups are notated as chorus ID -1.
- Midlevel Units (MLU) → p. 14.

Segmentations are stored in the Weimar Jazz Database in the `SECTIONS` table using the IDs of the first and last element index of the segmentation in the annotated melody plus the annotated value/ID. All these segmentations can be used in the MeloSpyGUI during feature extraction to chop a melody into segments for which features are calculated. There are also auxiliary features which add the values/IDs for direct event-wise annotations. Additionally, segmentations with bar chunks can be chosen, which will segment the melodies/solos into consecutive patches of bars of arbitrary length with user-definable overlap (i. e., these are overlapping, exhaustive segmentations if the overlap is not empty).

Chord symbols

Chords play a major role in jazz improvisation, thus a comprehensive representation of chords is vital for the computational analysis of jazz solos. Consequently, the beat tracks in the Sonic Visualiser project files of the solo transcriptions have been annotated with chord symbols taken from lead sheets. Chord annotations are included in the Weimar Jazz Database and are, for instance, used for chord-related pitch representations (s. p. 23).

A chord is basically a pitch class set with a certain structure, in which the root is the most important tone. Chords are represented by chord labels, which hold all relevant information and follow a certain syntax. The MeloSpyGUI chord syntax is designed to match the chord symbols that are in practical (jazz) use as closely as possible. Internally, a `CHORD` object consists of a root note, a triad type (major, minor, diminished, augmented, half-diminished/7♭5) and an optional bass note (for slash chords and inversions). Sevenths (major, minor, or diminished), ninths (natural, flat, or sharp), elevenths (natural or sharp), and thirteenths (natural or flat) are optional.

Instead of using existing chord syntax (e. g., Harte, Christopher, Sandler, Samer, & Gómez, 2005), we decided to devise our own, closer to traditional chord symbols, in order to facilitate annotation and readability. However,

very often users will not have to deal with the low-level chord annotations, since during exporting, internal chord symbols are translated back to standard notation. For sake of completeness, and for those users who would like to produce their own Sonic Visualiser project files in our format, the syntax can be found in Infobox 3.

Infobox 3: Chord syntax

Chords are expressed using the following grammar. The grammar should be read as follows: Each line contains a definition of an expression, indicated by the ::= sign, where small letter names represent other expressions and quoted strings are the terminal symbols. For example, a chord can either be the 'no chord' expression, which is the string `NC`, or a normal chord, defined on the third line, or a special chord, defined on the fourth line. Vertical bars denote alternatives and square brackets denote optional elements. For example, an accidental can be either a sharp (♯) or a flat (♭) and a 'generic note name' is note letter ('A' to 'G') optionally followed by an accidental.
Note that some conveniences and optional expressions are included to facilitate annotation. The main difference to standard notation (e. g., in the Realbook) is that accidentals of chord tensions are placed after the number. For instance, a $C^{7\sharp9\sharp11\flat13}$ chord is translated to `C79♯11♯13b`. The usual implications of lower tension in the presence of a higher tension also apply. Hence, a Cm^{11} notated as `C-11` or `Cm11` automatically includes the sevenths and the ninth as well and is therefore equivalent to `C-7/9/11`. Currently, no mechanism is included for 'adding' single tension without implications. For example, a C^{add9} with added ninth but without the seventh is not expressible in our chord syntax.

```
chord ::= no_chord | normal_chord | special_chord
no_chord ::= "NC"
normal_chord ::= generic_note_name triad_type
                 [seventh] [ninth] [eleventh] [thirteenth]
                 [slash_expr]
special_chord ::= generic_note_name exception [slash_expr]
generic_note_name ::= note_letter [accidental]
note_letter ::= "A"-"G"
accidental ::= sharp | flat
sharp ::= "#"
flat  ::= "b"
triad_type ::= major | minor | augmented |
               diminished    | suspended
major ::= "maj" | ""
minor ::= "min" | "m" | "-"
augmented ::= "aug" | "+"
diminished ::= "dim" | "o"
```

```
suspended ::= "sus"
seventh ::= ["j"]"7"
ninth ::= "9" accidental
eleventh ::= "11" [sharp]
thirteenth ::= "13" [flat]
slash_expr ::= "/" generic_note_name
exception ::= "7b5"  | "alt"   | "7#9" |
              "m7b5" | "7sus4" | "maj7"
```

Metrical annotation

Metrical frameworks are very important for the perception and production of music around the world. The perception and induction of a beat is a precondition for the occurrence of the phenomenon of meter. For our representation of the musical surface, we adopted a very general framework which works with minimal assumptions, e. g., without presupposing a deeply nested metrical hierarchy.

The basic conception puts the beat level in the center, with beats not being assumed to be isochronous, or even regular, but are essentially just a series of time points. Internally, a beat is stored as an onset with a duration, which is the interval between consecutive beats, i. e., the inter-beat interval (IBI).

In our metrical framework, there are only three hierarchical levels: Beats are grouped into units ('measures') of certain length (called 'period') on one hand, and divided into sub-beat levels ('tatums') on the other; no regularity assumptions are made for either. A sequence of measures with constantly varying periods can be represented as well as beat-wise changing subdivisions, which are not required to be of equal length although this is common. A metrical framework has to be differentiated from a concrete realization of this framework. For transcriptions, the task is often to infer an underlying metrical framework to a certain realization of a musical rhythm, which might not be unequivocal. For the transcriptions in the Weimar Jazz Database, the core part is implemented via human annotations, i. e., based on manual beat tracks with annotated time signatures. To accelerate and facilitate the transcription process, the annotation of musical tones to metrical position is done algorithmically based on the beat track and the tone onsets. This algorithm, called 'FlexQ', was specifically devised in the context of the Jazzomat Research Project and is described in more detail in Appendix **??** (see also

Frieler and Pfleiderer (2017) for a discussion of general issues with respect to metrical annotations).

First, we will introduce some notations and concepts. A beat track is a sequence of time points t_i, with inter-beat intervals $\Delta t_i := t_{i+1} - t_i$. The inverse $\frac{1}{\Delta t_i}$ of the inter-beat interval is the (instantaneous) tempo of the beat track.

A metrical annotation of the beat track is a mapping of beat times to pairs of integer values $t_i \to (p_j(i), b_{k_j}(i))$ subject to the following conditions: $p_j \geq 1$ and $1 \leq k_j \leq p_j$ for all j. The numbers j are the bar numbers, the p_j are the periods (number of beats in a bar j) and the b_{k_j} are the beat indices in the j-th bar, running from 1 to p_j. A shorthand notation for the metrical annotation is $m_i = (j(i), p_j(i), b_{k_j}(i))$.

Example. Assume an (isochronous) series of six beats (cf. Figure 2). A 2-period metrical annotation, using the shorthand notation, would then be given by

$$(1,2,1),(1,2,2),(2,2,1),(2,2,2),(3,2,1),(3,2,2),$$

and a 3-period one by

$$(1,3,1),(1,3,2),(1,3,3),(2,3,1),(2,3,2),(2,3,3).$$

A mixed 4-period/2-period annotation is also feasible:

$$(1,4,1),(1,4,2),(1,4,3),(1,4,4),(2,2,1),(2,2,2).$$

Since the numbers are only a function of the beat indices i, but not of the actual beat times t_i, this representation is purely abstract. Consequently, given such a metric representation, any series of beat onsets can be viewed as a realization of such a metric framework.

To establish a connection to an actual perceived meter, the beat times should fulfill certain criteria, e. g., approximate isochrony. Note that, in contrast to common Western musical notation's time signatures, only the number of beats per group (periods) are provided. No distinction is made between $\frac{2}{4}$ or $\frac{2}{2}$, or $\frac{6}{8}$, all of which have two beats grouped into a bar. The difference between $\frac{2}{4}$ and $\frac{6}{8}$ is that in the former one beats are more likely to be subdivided binarily, whereas in the latter the prevalent subdivision is ternary. Note, that binary or ternary subdivisions are not a fixed rule, but only prevalences, e. g., triplets occur less often in $\frac{2}{4}$ and duplets (dotted eighths notes) in $\frac{6}{8}$. The system proposed here leaves this mostly to the user and simply provides a system for annotating arbitrary subdivisions for each beat. This brings us to the general definition of a metrical annotation.

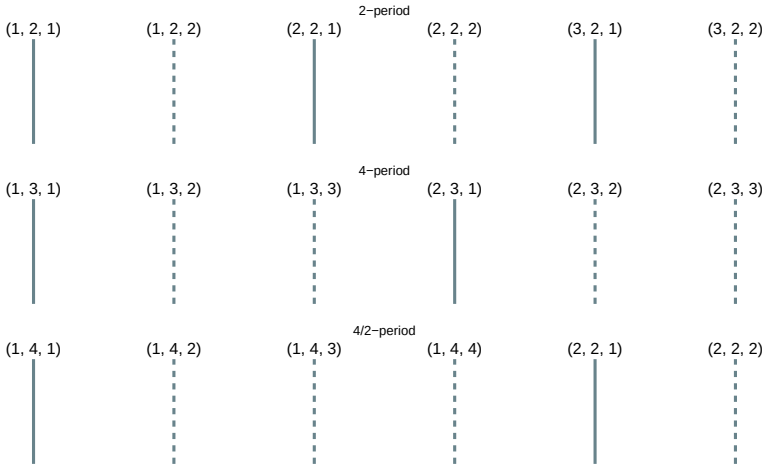

Figure 2: Three different sample metrical annotations of a six beat sequence. Solid lines: first beat of a bar; dashed lines: all other beats. Annotations are given in the format (bar number, period, beat index).

Consider an arbitrary rhythm, i. e., a series of time-points t_i. A metrical annotation for this rhythm is then a map onto tuples of five integers

$$(j, p_j, b_{k_j}, d_{k_j}, s_m),$$

where (j, p_j, b_{k_j}) is a beat annotation as defined above, $d_{k_j} \in \mathbb{N}$ is the division of the beat b_{k_j}, and $1 \leq s_m \leq d_{k_j}$ is the tatum position below the beat b_{k_j}, which runs between 1 and the division of the beat.

Example. Consider the rhythm of the 'The Lick' in Figure 1. The metrical annotation consists of two bars (mm. 1 and 2), both of period 4. The first two notes are each placed on the second tatums of beat 1 and 2. The following four eighth notes are on tatum 1 or 2 of the binary subdivided beats 3 and 4. The next note is on tatum 1 of the undivided first beat of the second bar, while the last two notes are on tatum positions 2 and 4 of beat 2 with a 4-subdivision (sixteenth notes). Rests are not considered here. The resulting metrical annotations can be found in Table 2 and Figure 3.

Infobox 4: Metrical position notation

For output purposes, the MeloSpyGUI uses a special dot-notation for metrical positions. The format is

```
period.division.bar_number.beat_number.tatum_position
```

period is the number of beats in a bar. If beat_proportions are set (for non-isochronous beats), beat_proportions will be used instead of period in an 'additive' notation, e. g., $(3+2+2)$. division is the number of tatums in the current beat. The other fields are bar number, beat number, and tatum position.

Midlevel annotation

Midlevel analysis (MLA) can be regarded as a short and compact system to segment the musical surfaces into sections of sufficiently distinct character. These units are called midlevel units (MLU). It was developed in an attempt to identify underlying playing ideas of jazz improvisations. The approach was first developed for piano solos (Frieler & Lothwesen, 2012; Schütz, 2015) and then adapted for monophonic solos (Frieler, Pfleiderer, Abeßer, & Zaddach, 2016).

MLA is based on the hypothesis that jazz players are more likely to make decisions on a middle level (with regard to playing details), comprising times spans of a few seconds, which are then actualized using preconceived or spontaneous material from motor memory. This guiding principle led to the identification of several distinct types of ideas by analyzing a large sample of solos. First, solos were segmented into parts of discernibly different character, which were tentatively named. In a second step, these types were condensed and re-ordered by simultaneously establishing a comprehensive system of sub- and sub-subtypes. In principle, this typology of playing ideas is not closed, but can be considered saturated for the Weimar Jazz Database, in the sense that it is possible to typify all playing ideas into one of the devised types. However, with new data or data of a different kind, it might be desirable to add new types if enough ideas are encountered that cannot be reasonably well fitted into one of the existing categories. For example, due to instrument specifics, playing ideas for piano solos and monophonic solos are not identical, although a large overlap exists. There is evidence that piano players, when

Table 2: Metrical annotation of the rhythm of 'The Lick'.

Note	TAS	Metrical annotation $(j, p_j, b_{k_j}, d_{k_j}, s_m)$	Dot notation $(p_j, d_{k_j}, j, b_{k_j}, s_m)$
1	1+	(1, 4, 1, 2, 2)	`4.2.1.1.2`
2	2+	(1, 4, 2, 2, 2)	`4.2.1.2.2`
3	3	(1, 4, 3, 2, 1)	`4.2.1.3.1`
4	3+	(1, 4, 3, 2, 2)	`4.2.1.3.2`
5	4	(1, 4, 4, 2, 1)	`4.2.1.4.1`
6	4+	(1, 4, 4, 2, 2)	`4.2.1.4.2`
7	1	(2, 4, 1, 1, 1)	`4.1.2.1.1`
8	2e	(2, 4, 2, 4, 2)	`4.4.2.2.2`
9	2a	(2, 4, 2, 4, 4)	`4.4.2.2.4`

Note. Note = note number in the example from Figure 1 and Table 1; TAS = traditional American system for counting metrical positions (second sixteenth = e, eighth = +, fourth sixteenth = a); metrical annotation as defined in the text; dot notation = metrical dot notation used in the MeloSpyGUI, see Infobox 4.

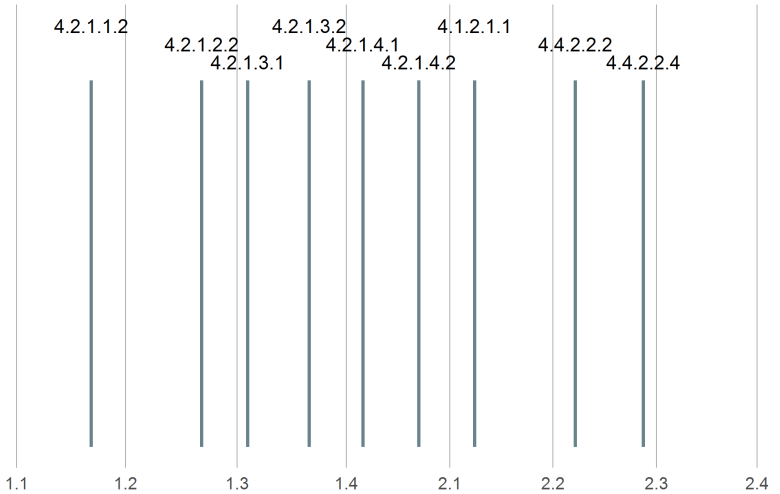

Figure 3: Metrical annotation for 'The Lick'. Thick vertical lines correspond to the onsets, thin lines to annotated beats. Labels are given in metrical dot notation for the onsets and in the format (bar number, beat index) for the beats.

confronted with the system, find the concept at least partly adequate for internal improvisation processes, though possibly not with regard to all details of the typology (Schütz, 2015). However, the exact connection to actual playing ideas is still an open research question. Besides identifying distinct MLUs, the system also incorporates an easy way to encode musical relationships and similarities between MLUs, e. g., variations, which permit interesting insights into the process of improvisation and an examination of dramaturgy and form.

1. *line*:[3] A line is a series of tones for the most part proceeding in small, step-sized intervals with high rhythmical uniformity and a salient trajectory in pitch space. Depending on the trajectory, there are several sub- and sub-subcategories. The main subcategories are simple (default), tick, interwoven, and wavy lines. The main shapes are 'horizontal', 'ascending', 'descending', 'concave', and 'convex'. Tick lines are lines of exclusively convex and concave shapes but with asymmetric arms, i. e., a longer or shorter descent or ascent combined with an accordingly longer or shorter ascent or descent. Simple lines show a straight direction, i. e., without too many twist and turns, which are characteristic of wavy lines. Wavy lines tend to be rather long and may have an overall direction besides 'wiggling around'. Interwoven lines consist of two independent horizontal ascending or descending lines that are played in tone-wise alternation.

2. *licks*: In the context of MLA, a lick is a rather short and concise melodic figure that often includes rhythmical and intervallic salience. Licks have a clear gestalt-like quality, which distinguishes them from fragments. They comprise mostly short tones and sometimes large intervals and chromaticism, which distinguishes them from melodies. Shortness, rhythmic diversity, or both qualities together separate licks from lines. We have two proper subtypes in this category: *lick_bebop* and *lick_blues*. All other licks are grouped into a residual subclass *lick*. Blues licks are defined by tonal features such as blue notes as well as typical constructions. Historically, the blues played (and still plays) a special role in jazz improvisation (Schuller, 1958), so it seemed worth defining a special subcategory. Bebop licks on the other hand use certain techniques which are typical for bebop lines, such as approach notes and chromatic passing tones.

[3]The following description of the nine main MLU types is a slightly edited version taken from Frieler et al. (2016).

3. *melody*: A melodic figure that is not derived from the theme of the song and embodies some kind of song-like, lyrical, cantabile character. A rule of thumb may be: If an MLU sounds more like scatting (if sung), it should be termed a lick or a line; if it sounds more like a Broadway tune, a pop song or a folk tune, it should be labeled as *melody*.

4. *rhythm*: This category describes units in which the rhythmical expression is the single most prominent feature. There are four subtypes that differ according to the number of pitches (single or multiple) and basic rhythm quality (regular or irregular). The most important subtypes are single tone repetitions, predominantly regular and isochronous, and oscillations with multiple tone repetitions, predominantly regular.

5. *theme*: Denotes material taken directly from the theme of the tune, possibly with variations. The characteristics of a *theme* MLU are often similar to those of melody, but because of its relationship to the tune, it is a distinct playing idea.

6. *quote*: These are direct quotes from another piece of music (jazz tune, classical tune etc.), which might resemble a melody or a theme. Playing a pattern taken from another jazz musician as part of a longer line or a lick does not count as a quote if it is not clearly recognizable as such.

7. *fragment*: A small set of tones which neither form a clear contour-based succession or motivic/thematic figure nor are very expressive. Fragments are most often single tones or very short segments which can even sound like 'mistakes'.

8. *expressive*: These are figures or single tones with a sound- or gesture-like character in which aspects of expressivity are clearly focused, e. g., scream-like sounds.

9. *void*: This category refers to moments of 'actively playing nothing'. Generally, jazz soloists add short breaks between phrases, e. g., just for breathing, which do not belong to this category. The length of the break in the flow of a solo should clearly exceed these usual gaps between phrases.

All solos in the Weimar Jazz Database were manually annotated with MLUs by three annotators. Special care has been taken to ensure that the MLU types are consistently annotated. One annotator double-checked all other annotations. As this is basically a qualitative system with categories that cannot be defined unequivocally and which have continuous and overlapping

boundaries, there is some leeway for the annotators. However, when comparing annotations of three to four different coders using a small subset of solos, high agreement (Fleiss's $\varkappa = .81$) for assignment of MLU boundaries was found, whereas the agreement for MLU types was lower (Fleiss's $\varkappa = .60$), but still sufficiently good. The main disagreement was between *line* and *lick* as well as between *melody* and *lick*. For specific MLUs, disagreements between different coders might come about due to the fact that several equally adequate solutions exist, but each MLU has to be assigned to exactly one type. Even when annotations might disagree, they can often still be considered as 'correct' (or 'not wrong'). Hence it can be safely assumed that for statistical analyses the disagreements level each other out and that the resulting statistics are sufficiently valid. The most common MLU type is, not surprisingly, *lick* with about 45 % followed by *lines* 30 %. However, since *lick*s are much shorter than *line*s, they only account for about 37 % of the duration of all solos, whereas *line*s account for about 40 %. For more details about MLU statistics and distributions in the Weimar Jazz Database, we refer to Frieler et al. (2016).

Infobox 5: MLU syntax

For the MLA annotation in Sonic Visualiser, a specific short mnemonic code was devised (see Online Supplementary Material S2 of Frieler et al., 2016). The basic rule is that a musical phrase starts an MLU, but that more than one MLU can be contained in one musical phrase, which is called 'glueing'. Each MLU contains information about its type (full type with all possible subsubtypes), whether the MLU is glued on, possible reference or relationship to a foregoing MLU (called 'back reference'), some additional information about this relationship, and finally comments of further specification.
The basic syntax is:

```
[~][#[N][*+-=]]cat-label[:specifier]
```

where square brackets denote optional elements. The first element, the tilde `~`, if present, indicates a glued MLU. The hash `#`, followed by an optional integer number `N` indicates a back reference to the MLU `N` MLUs before. If `N` is missing, a value of one is assumed, i. e., a relationship to the immediately preceding MLU. Equivalently, this can also be expressed by sequences of `N` hashes. The following symbols `*+-=` indicate the type of relationship, if clearly identifiable. The asterisk `*` indicates an unspecified variant and can be omitted. The plus and minus signs denote transpositions up `+` or down `-` in pitch space, whereby the transposition does not need to be exact (hence an asterisk is implied). For exact (possibly transposed) repetitions, the equal sign `=` can be used.
The `cat-label` has to be one of the following:

- Licks: *lick* (unspecified), *lick_bebop* (bebop lick), *lick_blues* (blues lick).
- Simple lines: *line_a* (ascending), *line_d* (descending), *line_cv* (concave), *line_cx* (convex).
- Tick lines (a = ascending, d = descending, s = short, l = long): *line_t_asdl*, *line_t_alds*, *line_t_dsal*, *line_t_dlas*.
- Interwoven lines (a = ascending, h = horizontal, d = descending, top line first) : *line_ah*, *line_ha*, *line_ad*, *line_hd*, *line_aa*, *line_ad*, *line_da*, *line_dd*.
- Wavy lines *line_w[_x]*, where x can be any specifier from simple and tick lines or missing, which is short for *line_w_h*, a wavy line that starts and ends roughly at the same pitch.
- Rhythms (s = single pitch, m = multiple pitches, r = regular rhythm, i = irregular rhythm): *rhythm_sr*, *rhythm_si*, *rhythm_mr*, *rhythm_mi*.
- Others: *melody*, *void*, *expressive*, *quote*, *theme*.

The specifier part is optional and can be any string, e. g., the name of the musical work a *quote* was taken from or the location in the theme of a *theme* reference. *Examples.* *~ #2+melody* is a glued melody that is an upwardly transposed variation of the MLU (not necessarily also a *melody*) two units before; *theme:t1* is a reference to measure 1 of the theme of the tune; *lick*, *#lick*, *#lick* is a chain *aa'a"* of successively varied licks. If more than one back reference is possible, as probably here for the last one, the shortest back reference should be taken.

Feature extraction

Feature extraction is one of the central tasks in computational musicology and a prerequisite to many analytical investigations. In a general sense, features can be conceived as characteristics, traits, or properties that describe specific entities, e. g., musical objects. They range from very simple features, which require only a trivial measurement or are part of the definition of an object, to very complex ones ('deep stucture'), which require elaborate measurement procedures or cognitive models. Moreover, features can be ascribed to the whole entity of interest or only to parts of it. They can be single values (e. g., the number of notes in a melody) or vectors of symbols (e. g., the title of a piece.)

Features often serve as dependent variables in models to predict target variables, e. g., music perceptions or style classification. For example, certain musical properties, such as tempo, articulation, or tonality, are known to con-

tribute to the emotional expression or impact a piece of music can have. For classification, objects are grouped into classes, categories, or types according to certain properties, i. e., feature values, using suitable similarity measures. Features are essentially human constructions and seldom merit ontological status. They coordinate the relationship between the objects and operational models for the prediction and description of these objects, but are principally arbitrary and contingent on the purpose of the scientific model. In analogy to coordinate systems that describe physical objects in space, one can also speak of 'feature spaces' for the description of an object. Like coordinate systems in the physical world, certain features are more apt or easier to handle than others, depending on the tasks and objects at hand. Finding the most suitable features for a problem is often a central task, called 'feature selection'.

There are always possibilities for constructing new features from old ones using algorithmic or mathematical operations. This is basically the approach we followed in designing our feature extraction software (`melfeatures`). It has a flexible module ('feature machine') at its core that allows the definition of a large class of features based on certain predefined properties of the musical object (i. e., its basic representation) and a set of mathematical and statistical operations.

General concepts

It is useful to first define some general properties of features. Let $\mathcal{O} = \{o_i\}$ be a set of objects and $\mathcal{F} = F_1 \times F_2 \times \cdots \times F_N$ an arbitrary feature space. The number N is called the dimension of the feature space, where we assume that the components F_i are mostly elementary (one-dimensional). In most cases, the components are either real or integer numbers ($\mathcal{F}_i = \mathbb{R}, \mathbb{N}$) or a set of labels, i. e., an arbitrary set with no further internal structure.

A feature is then a map from the objects into the feature space. The primary or defining features are those which are used to define the objects. Please note that the abstract representation of objects here is not identical to the objects themselves, i. e., the objects of interest here are the digital representations of musical objects as defined in section p. 1. Hence, in our representation, the primary feature of a melody is the representation itself, i. e., tuples of onsets, durations, and pitches etc. In this approach, each object has to have its own feature space, since the dimension of the basic representation is dependent on the length of the melody. The representation space for a melody of length N is then effectively $3N$ dimensional with components $F_i = \mathbb{R} \times \mathbb{R}^+ \times [0 : 127]$. Feature maps where the feature spaces vary with the object are called sequential features or, alternatively and probably less confusingly,

transformations. Such transformations can themselves, of course, become the starting point of another feature map and in this way transformation chains can be built. This is also the basic logic behind the 'feature machine' of the `melfeature` module of MeloSpyGUI software, where transformation chains are defined with the aid of a simple syntax.

An important special case of feature maps are global features, which map an object to one and the same feature space. If the feature space is one-dimensional, these are called scalar features, even if the feature space is not numerical. Metadata are an important case of global and scalar feature. For example, the solos in the Weimar Jazz Database have an annotated rhythm feel which is represented by the set of labels

$$\mathscr{R} = \{\texttt{SWING}, \texttt{LATIN}, \texttt{FUNK}, \texttt{TWOBEAT}, \texttt{MIX}\}.$$

Then, the 'rhythm feel feature' is a map from the solos $\mathscr{S} \to \mathscr{R}$ onto the set of rhythm feel labels. Similar constructions hold for all metadata, even though the feature space is not always a fixed set of labels, but often strings (character sequences), e. g., the title of a piece.

Phrase IDs are an example of a non-primary sequential feature. Since each solo has a different number of phrases, a global finite-dimensional feature space is not sufficient to represent all possibilities. We can formally solve this with the aid of sequence spaces, e. g., spaces of sequences of arbitrary length into a fixed target space. Let T be such a target space (in the case of phrase IDs, these are the integers, $T = \mathbb{N}$). A sequence of length N over T, denoted $\mathscr{S}_N(T)$, is a map $[0 : N-1] \to T$. The space of all sequences over T is defined as the union of all sequences of positive length:

$$\mathscr{S}(T) = \bigcup_{N=1}^{\infty} \mathscr{S}_N(T) = \{\mathscr{S}_1(T), \mathscr{S}_2(T), \dots\}.$$

Note that sequences of length 1 are identical to the target space itself, $\mathscr{S}_1(T) = T$. A sequential feature (also sometimes, a bit misleadingly, called vector feature) is a feature map with feature space $\mathscr{S}(T)$. The list of phrase IDs of a solo is then a sequence of integers from 1 to M, where M is the number of phrases in a solo. Of course, such a list of integers is not very informative, so it can be combined, for example, with the indices of start and end tones to unequivocally identify phrases within a certain solo.

Our basic representation of melodies can then be viewed as a sequential feature with feature space $\mathscr{S}(\mathbb{R} \times \mathbb{R}^+ \times [0 : 127])$. In the case of sequential musical objects, a sequential feature is called point-wise or local if the length of the sequence space is (a simple function of) the length of the melody. In

other words, a transformation that results in a sequence of the same length as the original melody is 'local' in this sense.

Finally, a matrix feature is a feature map into the space of matrices (mostly over $\mathbb{R}$), where the dimension of the matrices can be dependent on the object itself. A prominent example, available in the MeloSpyGUI, is the self-similarity matrix of phrases (see p. 23).

An important class of—mostly global—features are statistical features. These are constructed by using descriptive statistics (e. g., mean, median, standard deviation etc.) of a sample distribution which in this case are the elements of a transformation chain. For example, the mean pitch of a melody is constructed by the projection on the pitch component and subsequent averaging all (MIDI) pitches p_i of a melody of length N, i. e.,

$$\overline{p} = \frac{1}{N}\sum_{i=0}^{N-1} p_i.$$

A more complicated example is the relative frequencies of a chordal pitch class. Chordal pitch classes are defined for a pitch p and an annotated chord C_p with root pitch class $r(C_p) \in [0:11]$. The chordal pitch class of p is then defined as the interval of p to the root modulo 12. The modulo 12 operation returns the remainder after an integer division by 12, e. g., $23 \mod 12 = 11$. This operation effectively disregards the octave. The full formula is then:

$$\gamma(p, C_p) = (p - r(C_p)) \mod 12.$$

Applying this transformation to a melody of length N results in a sequence of chord pitch classes $\gamma_i = \gamma(p_i, C_{p_i})$, also of length N, which can then be treated as a random sample over the numbers from 0 to 11. The frequency of chordal pitch class $\Gamma_j, j \in [0:11]$, $f(\Gamma_j)$ is the number of times it occurs in the sample $f(\Gamma_j) = \left|\{\gamma_i = \Gamma_j, i \in [0:N-1]\}\right|$. Clearly, the sum of all frequencies over all possible pitch classes equals the size of the sample, i. e., $\sum_{j=0}^{11} f(\Gamma_j) = N$. The relative frequency of a chordal pitch class $d(\Gamma_j)$ is then the absolute frequency divided by the size of the sample, which is a number between 0 and 1. This relative frequency can be regarded as an estimation of the probability that a certain chordal pitch class will occur. The twelve relative frequencies of single chordal pitch classes can also serve as global, scalar features for a solo.

Selected features

A classification of melodic features can be performed according to many different criteria. A first classification is with respect to the musical dimension they pertain to, e. g., pitch/intervals or rhythm/meter. But since there are also features that relate to several different musical dimensions at once, this schema is not without problems. Another approach is more technical, like the one presented in the preceding section, e. g., global, sequential, scalar, vector, matrix features etc., which gives some information about the structure but not about the meaning. In the feature definition files of the MeloSpySuite/-GUI, a rough classification, mainly based on musical dimension, is already in use, so it seems natural to follow this approach. The reader can thus easily make a connection to the software. We will discuss global characteristics of each feature class and the most important features, but we have to leave many details to our online documentation[4] due to limitations on space.

Accents

Accent features (or better, accent transformations, also called 'structural markers') are exclusively sequential, local transformations. They result in a sequence of numerical values of the same length as the melody. Nearly all accent features provide boolean values (binary markers), i. e., `TRUE` (1) or `FALSE` (0), indicating, whether or not a certain condition is fulfilled for an event in a melody. They originated in a paper (Müllensiefen, Frieler, & Pfleiderer, 2009), in which perception of perceived accents (in pop music) was modeled with the help of a long list of so-called 'accent rules' taken from the literature. The MeloSpySuite/GUI currently includes all accent rules used in the paper and several others. A slight generalization leads to the concept of structural markers, which provide point-wise boolean values with respect to some structural conditions, e. g., phrase and bar beginnings and endings. These are particular useful for more complex analyses in conjunction with other features. For example, if one wants to find out the distribution of metrical positions of phrase endings in a solo, one could extract metrical positions and phrase ending markers, and examine the distribution of metrical position with respect to phrase endings.

Auxiliary

Auxiliary features (transformations) either export the basic representation of the melody (which also permits the export of raw melodic data, though we recommend using the `melconv` module of the MeloSpySuite/GUI for

[4] http://jazzomat.hfm-weimar.de/documentation.html

this purpose), or annotated values, such as phrase IDs, form labels, metrical position, and chords.

Contour

Pitch contour is an important concept in music psychology since melody perception and melodic memory seem to work mostly with the rough contour of pitch sequences (Dowling & Fujitani, 1971), i. e., the ups and downs, and its global shape.

Huron and Abeßer contour map a pitch sequence to a fixed contour shape class. The Huron contour was proposed by Huron (1996) in a study about the melodic arch in Western folk song. For a sequence of N pitch values (p_i) the algorithm proceeds as follows: Take the first and last pitch p_0 and p_{N-1} and the mean value of all pitches in between $\overline{p} = \frac{1}{N-2}\sum_{i=1}^{N-2} p_i$. Between the first and the mean pitch and the mean and the last pitch, three size relationships are possible: equal (=), less (<), or greater (>). This yields nine possible combinations, which represent certain global contour shapes, e. g., == is a horizontal, <> a convex, and < < an ascending contour. A reduced version with only five classes is obtained by mapping mixed horizontal-ascending/descending contours to the ascending/descending parts. This procedure is more suited for shorter melodies, e. g., folk song phrases of about 7–10 pitches, and not meaningful for very long melodies such as entire jazz solos or even a long bebop line. For this, an extension of Huron's method proposed by Abeßer et al. (2015) is better suited. It was developed for frequency contours (cf. p. ??), but carries over to pitch sequences without modification. Generally, contour shapes are best calculated for phrases or other shorter melody sections in order to obtain meaningful results.

Interval

Intervals are another very important musical dimension. They are derived from the pitch dimension, but are generally invariant under pitch transposition which reflects the psychological phenomenon that most people have only relative and not absolute pitch. All interval features provided in the MeloSpySuite/GUI use the semitone interval transformation as a starting point, i. e.,

$$\Delta p_i = p_{i+1} - p_i.$$

This transformation has one fewer element than the original melody. It can be made a point-wise transformation by adding an empty element at the beginning or the end. Intervals have a sign indicating the direction of the pitch change, but sometimes only the magnitude is of interest, which can be

retrieved by taking the absolute value, $|\Delta p_i|$. The interval features provided are mostly statistical descriptors (maximum/minimum, range, mode, mean, median, variance, standard deviation, entropy, and Zipf's coefficient).

Infobox 6: Information entropy and Zipf's coefficient

Information entropy is a useful concept in computational musicology. It was developed by Claude Shannon (Shannon, 1948) to measure the information content of messages. The definition is generally applicable to the distribution of random events. The information of an event is defined as being inversely proportional to its probability, since very common events are expectable and thus less informative, whereas rare events provide a lot of information. If we have a set of possible events e_i with probabilities p_i, the information content of e_i is defined as the logarithm of the inverse probability:

$$h_i = \log \frac{1}{p_i} = -\log_2 p_i.$$

The information content of an event can be interpreted as the number of average Yes/No-question (called *bits*) one has to ask to guess the outcome of an experiment. That is also why the logarithm appears in the above, because with every Yes/No-question one halves the space of possibilities. For example, to obtain the result of a fair coin toss, one has to ask one question on average. (For an unfair coin, which always lands heads up, no question has to be asked, because the result is always the same).
The information entropy is then the expected information gain over all possible events:

$$H = E[h] = \sum_i p_i h_i = -\sum_i p_i \log_2 p_i.$$

Entropy will have low values if few very likely events dominate the process and will have very high values if all outcomes are roughly equally likely. Indeed, for uniform distributions, i. e., $p_i = \frac{1}{N}$ if there are N possible events, the entropy is maximal:

$$H_{\max} = -\sum_{i=1}^{N} \frac{1}{N} \log_2 \frac{1}{N} = -\log_2 \frac{1}{N} = \log_2 N.$$

This can be used to define normalized entropy for a random process with N outcomes:

$$H_0 = \frac{H}{H_{\max}} = \frac{H}{\log_2 N}.$$

The Zipf coefficient can be regarded as a measure for the non-uniformity of a probability distribution: the higher the value, the more the distribution is dominated by the most frequent elements. Power laws are a very common

phenomenon in a wide area of applications from income distribution to city sizes. Zipf observed his law for word frequencies in English texts, where he found that the frequency of words fell as $\propto 1/r$ with the frequency rank ($\alpha = 1$). Zipf's coefficient is generally defined as the slope of the logarithm of class frequencies vs. the frequency ranks, i. e., the coefficient α of a (supposed) power law $f(r) \propto r^{-\alpha}$, where r is the frequency rank (see Zanette, 2006, for discussion of Zipf's law in the context of music). Very often, however, the Zipf coefficient is strongly correlated with the information entropy of a distribution. In practice, information entropy is mostly preferred as a measure for non-uniformity of a sample distribution. since it is more well-known and more easily and robustly calculated.

For semitone intervals, two classifications are available: fuzzy intervals and Parsons's code. The latter is sometimes called 'contour', which is not to be confused with the contour measures in the sense defined above. Parsons (1975) devised his code in the context of cataloging musical themes, where he considered only the basic interval directions, i. e., `U`(p) (+1), `D`(own) (-1), or `S`(ame)/`R`(epeat) (`0`), to achieve a very rough but compact code for melodies. Formally, this is just the mapping of semitone intervals to their sign, i. e., $\Delta p_i \mapsto \mathrm{sgn}(\Delta p_i)$, where positive numbers are mapped to +1, negative numbers to -1 and zero onto itself.

A more detailed classification is given by the fuzzy interval transformation, which is sometimes called 'refined contour' in the literature. There are, of course, many different possible classifications of intervals. We used one with nine symmetric classes which seemed to be a good compromise between compactness and specificity (cf. Table 3).

Metadata

This category consists of all available metadata (in the Weimar Jazz Database, or other corpora such as the EsAC folk song database). These are all global, scalar features. For a list of the most important metadata in the Weimar Jazz Database, see **??**.

Meter

This category collects global and local features and transformations connected to the metrical annotation. Besides raw exports of metrical annotations such as bar, beat, and tatum positions, the Metrical Circle Map transformation (`MCM`) and derived statistical features are also provided. The Metrical Circle

Table 3: Fuzzy interval classes.

Name	Interval range	Numerical value
large jump down	< -7	-4
jump down	$[-7 : -5]$	-3
leap down	$[-4 : -3]$	-2
step down	$[-2 : -1]$	-1
repetition	0	0
step up	$[1 : 2]$	1
leap up	$[3 : 4]$	2
jump up	$[5 : 7]$	3
large jump up	> 7	4

Map (Frieler, 2007, 2008) was introduced to enable a comparison of melodies with all types of meters and highly diverse rhythms. Metrical measures are binned into N equally long segments and metrical positions are mapped to the closest bin. This ensures comparability, admittedly rather brute-force, of metrical positions in any metrical frame, as well as for music with changing meters. However, for actual comparisons using the MCM, care should be taken in regard to the different metrical frames in a set of melodies in order to achieve interpretable results. In the MeloSpySuite/GUI a `MCM` with $N = 48$ is implemented, i. e., each tone event is mapped to a number $m \in [0 : 47]$. This is a local transformation (provided a metrical annotation is available). For this mapping, several circular statistical descriptors are available (see Appendix **??** for a short introduction to circular statistics).

There are two metrical complexity measures available, *compression complexity* and *division complexity* as well as the arithmetic mean of both. The first is based on the idea that metrical local sub-beat grids are more complex if fewer positions are occupied. The beat-local grid is defined by the local division of the beat. For example, a group of four sixteenths is 'simpler' than a group of one eighth and two sixteenths, since the third sixteenth position in the sixteenth sub-beat grid is unoccupied in the latter. The second complexity measure is inspired by the idea that frequently changing beat divisions are more complex, particularly if these are not related by doubling or quadrupling. Let us take again 'The Lick' as an example (Figure 1, p. 6), see Table 4. Compression complexity values are calculated as follows. For each beat i in a set of N beats with local divisions d_i, the number $1 \leq n_i \leq d_i$ of occupied tatum positions is counted. Then, the total compression complexity

is given by the mean over all beats:

$$\frac{1}{N}\sum_{i=1}^{N}\frac{d_i - n_i}{d_i - 1}.$$

A rhythm where all tatum positions are occupied will receive the minimal value of zero and a rhythm with only syncopated events the maximal value of one. (Note that the division is ensured to be optimal due to the FlexQ algorithm in our metrical representation).

The division complexity is similarly calculated for all beats with at least one event, where a change in division will be penalized by 1 if the ratio of divisions is not a power of two, in which case the penalty is set to 1/2.

$$\frac{1}{N-1}\sum_{i=1}^{N-1}\theta(d_i, d_{i+1}),$$

where

$$\theta(x,y) = \begin{cases} 0 & x = y, \\ 0.5 & |\log_2(x/y)| \in \mathbb{N}, \\ 1 & \text{else.} \end{cases}$$

A rhythm with a constant beat division will be awarded the minimal division complexity of 0; a rhythm with constantly changing divisions, not related by a power of 2, will receive the maximal value. (Note that this construction also relies on the fact that the divisions are guaranteed to be local and optimal as provided by the FlexQ algorithm.)

MLA

From midlevel annotations as described above (p. 14), a set of sequential features can be derived, e. g., lists of raw MLUs, main types, duration, and back references.

Pitch

The base for all pitch features is the pitch dimension of the basic representation from which several representations are derived. The most important transformation of pitch is pitch classes (pc), which reflect the octave equivalence. In the common 12-tone equal tempered tone system, this is easily done by taking the MIDI pitch values modulo 12. Since, by convention, $C4 = 60$, all Cs are mapped to $\hat{0}$, where the hat over the zero conventionally indicates pitch classes (Forte, 1973). The other tones are enumerated accordingly, i. e.,

Table 4: Compression and division complexity for 'The Lick'.

Measure	Beat	Division	Occupation	CC	DC
1	1	2	.X	0.5	0.0
1	2	2	.X	0.5	0.0
1	3	2	XX	0.0	0.0
1	4	2	XX	0.0	0.0
2	1	1	X	0.0	0.5
2	2	4	.X.X	0.5	0.5
			Sum	1.5	1.0

Note: ./X = unoccupied/occupied tatum position, CC = compression complexity value, DC = division complexity value.

C♯/D♭→$\hat{1}$, D →$\hat{2}$, D♯/E♭→$\hat{3}$ etc. Formally, given a fixed anchor pitch P, the pitch class transformation (PC) for pitches p_i with respect to P is given by:

$$p_i \mapsto (p_i - P) \mod 12.$$

It is an established tradition to use $P = 0$ for (absolute) pitch classes (Forte, 1973), but the anchor is basically arbitrary.

The idea is to use different anchors with respect to context. One option for this is the tonic of the key of the melody (if available). For example, for a melody in A♭ major, one sets the anchor to $P = 8$, and all A♭s are then mapped to $\hat{0}$, with the other pitches mapped accordingly. This transformation is called tonal pitch class (TPC) and is available in the MeloSpySuite/GUI.

Another option, which is especially important in chord-based jazz improvisation, is to use the root of the underlying chord as the anchor. For example, a C will be mapped to $\hat{0}$ if played in a Cmaj^7 context, but to $\hat{1}$ in a D♭$^{\text{min7}}$ context, and to $\hat{4}$ in a A♭$^{\circ 7}$ context. This is called chordal pitch class (CPC).

Another modification of the approach is to reflect the diatonic major/minor system, since under TPC and CPC minor thirds are mapped to $\hat{3}$ whereas major thirds are mapped to $\hat{4}$, even though both serve the function of a diatonic third scale degree in minor or major, resp. Diatonic scale degrees are mapped to the values 1 to 7 and special symbols are used for the remaining 'chromatic' tones. This leads to tonal and chordal diatonic pitch classes (TDCP, CDPC).

An extension of CDPC, called extended chordal pitch class (CDPCX), proved to be more important in practice and is often used in this book. For chords with a major third, the major scale is used as the underlying diatonic scale; for chords with a minor third (including diminished and half-diminished chords), the dorian scale is used[5] as the underlying diatonic scale. Exceptions are dominant seventh chords, where the minor seventh is mapped to the seventh scale degree (mixolydian). See Table 5 for a mapping of chord types to base scales. The remaining non-diatonic pitches are thus also dependent on the chord, or—to be more precise—from the diatonic context derived from the chord and are thus mapped to different symbols (Table 6). The resulting alphabet as implemented in the MeloSpySuite/GUI contains 13 symbols plus the symbol X for pitches without chord context (NC). Note that flat ninth, sharp fourth/flat fifth, and flat sixth are always considered to be non-diatonic, even when the chord prescribes these as a tension. The major/minor third is considered non-diatonic over chords with a minor/major third. The same rule also holds for major and minor sevenths.

Table 5: Mapping of chord types to diatonic scales for CDPCX.

Chord type	Diatonic scale
$\text{maj}^{(6,7)}$, aug	ionian
$\text{dom}^{7,\,\text{alt}}$, aug^{7}	mixolydian
$\text{min}^{(7)}$, $\text{dim}^{(7)}$, $\text{min}^{7\flat5}$, $\text{sus}^{(7)}$	dorian

Generally, all higher chord tensions are ignored for reasons of simplicity in conjunction with the fact that the chord annotations in the Weimar Jazz Database are taken from lead sheets. One should bear this in mind when interpreting CDPCX values. Some chords, such as the diminished seventh chords, do not fit to a classical diatonic scale over any of its chord tones or are, at least, ambiguous. These are in someway intrinsically non-diatonic chords (even though, e. g., a fully diminished seventh chord can be interpreted as a dominant seventh chord with an added minor ninth and without a root in a minor scale.) Moreover, linear movements should be considered too, since many of the non-diatonic pitch classes are produced not as 'free chromatics' but using passing and neighboring tones. Features derived from these pitch

[5]The choice to use the dorian scale and not the minor (aeolian) scale was based on the rationale that (1) a minor sixth (or ♭13) is generally considered a dissonant extension over a minor chord, (2) dorian minor chords are somehow more common in jazz due to the frequent use of ii7 chords, and (3) the transformation becomes simpler this way.

Table 6: Mapping of CPC to CDPCX.

CPC	Diatonic context			Name
	Ionian	Dorian	Mixolydian	
$\hat{0}$	1	1	1	
$\hat{1}$	-	-	-	♭9
$\hat{2}$	2	2	2	
$\hat{3}$	B	3	B	♯9
$\hat{4}$	3	>	3	♯10
$\hat{5}$	4	4	4	
$\hat{6}$	T	T	T	♯11
$\hat{7}$	5	5	5	
$\hat{8}$	%	%	%	♭13
$\hat{9}$	6	6	6	
$\hat{10}$	<	7	7	♭7
$\hat{11}$	7	L	L	♯7
NC	X	X	X	

transformations are mostly statistical descriptors, either continuous (PITCH), circular (PC and CPC), or nominal (CDPC, CDPCX), TDCPC).

Example. In Table 7 the most important pitch transformations are listed for 'The Lick' (cf. Figure 1, p. 6.)

Rhythm

For rhythms, the most important distinction is the one between durations and inter-onset intervals (IOI). Generally, IOIs are more relevant for rhythm perception whereas durations are mainly a matter of articulation and tone formation. Durations are already part of the basic representation of a tone event and IOIs can be easily calculated given the list of onsets. However, due to micro-timings and transcription imprecisions, the bare numerical values of either durations and IOIs are of little analytical help and require some binning processes. In the MeloSpySuite/GUI a classifications for time spans (durations and IOIs) are predefined in two different variants, absolute and relative. This classification has five classes, VERY SHORT, SHORT, MEDIUM, LONG, and VERY LONG. To calculate the classes, one first has to relate the time spans to a reference value. Absolute time span classes are constructed using a fixed reference value of 500 ms corresponding to the beat duration in

Table 7: Pitch representations of 'The Lick'.

Note	Chord	Name	PITCH	PC	TPC	TDPC	CPC	CDPCX
1	C^{maj7}	B4	71	11	11	7	11	7
2	C^{maj7}	B4	71	11	11	7	11	7
3	C^{maj7}	A4	69	9	9	6	9	6
4	C^{maj7}	B4	71	11	11	7	11	7
5	C^{maj7}	C5	72	0	0	1	0	1
6	C^{maj7}	D5	74	2	2	2	2	2
7	$F\sharp^{7\flat5}$	B4	71	11	11	7	5	4
8	$F\sharp^{7\flat5}$	G4	67	7	7	5	1	-
9	$F\sharp^{7\flat5}$	A4	69	9	9	6	3	3

Note. PITCH = MIDI pitch, PC = Pitch class, TPC/TDPC = Tonal (diatonic) pitch class (based on C major), CPC/CDPCX Chordal (diatonic, extended) pitch class.

120 bpm. This value is a good approximation to preferred and spontaneous tempos reported in the rhythm research literature. In a way, it represents a 'natural' time unit for humans (e. g., Fraisse, 1982). Relative time span classes are constructed using the duration of the momentary beat as the reference value.

Formally, let T_i denote time-spans, i. e., durations d_i or IOIs Δt_i, and fix a reference duration $T_{0,i}$, possibly a function of the time-span itself, e. g., the local beat duration for relative time-span classes, or a fixed value (e. g., $T_0 = 0.5$ for absolute time-span classes). Furthermore, let $\epsilon < 1$ be a small numerical offset. Then the time-span classification $\hat{\Theta}$ is defined for normalized time-spans

$$\hat{\Theta}(T_i) = K\left(\frac{T_i}{T_{0,i}}\right)$$

with the time-span classification function $K(t)$:

$$K_\epsilon(t) = \begin{cases} -2 & t < 2^{-2+\epsilon} \\ -1 & 2^{-2+\epsilon} < t \leq 2^{-\frac{1}{2}-\epsilon} \\ +0 & 2^{-\frac{1}{2}-\epsilon} < t \leq 2^{\frac{1}{2}+\epsilon} \\ +1 & 2^{\frac{1}{2}+\epsilon} < t \leq 2^{2-\epsilon} \\ +2 & > 2^{2-\epsilon}. \end{cases}$$

In the time-span classifications in the MeloSpySuite/GUI $\epsilon = 0.1$ was used.

This gives a classification function, using verbal class labels, of

$$K(t) = \begin{cases} \texttt{VERY SHORT} & t < 0.268, \\ \texttt{SHORT} & 0.268 < t \leq 0.660, \\ \texttt{MEDIUM} & 0.660 < t \leq 1.560, \\ \texttt{LONG} & 1.560 < t \leq 3.732, \\ \texttt{VERY LONG} & t > 3.732. \end{cases}$$

The rationale behind these numerical class boundaries is based on a few ideas. First, a five-fold classification seems a good choice between specificity and compactness. Secondly, sixteenth notes should be mapped to `VERY SHORT`, eighth notes to `SHORT`, quarter notes to `MEDIUM`, half notes to `LONG`, and whole notes to `VERY LONG`. These correspond to normalized time-spans of $\frac{1}{4}, \frac{1}{2}, 1, 2, 4$, which are mapped to the desired classes. But besides these binary metrical time-spans, there are also triplets, quintuplets etc., as well as dotted and double dotted durations which have to be accommodated. Furthermore, in performance-based data such as Weimar Jazz Database, the time-spans are actually real-valued numbers. The choice of class boundaries was further motivated by the conditions that triplets should be mapped to the class `short` and dotted quarters to `medium`. Finally, the boundaries were determined by programming parsimony by tweaking powers of two, since—accidentally—$2^{\frac{1}{2}} = \sqrt{2} = 1.4142\ldots$ is less then 1.5, which is the normalized length of a dotted quarter note.

In the MeloSpySuite/GUI four different time-span classifications are available: relative and absolute duration and IOI classes. For these, standard statistical descriptors such as class frequencies and entropies are implemented as scalar global features.

Example. Let's calculate the relative IOI classes of 'The Lick' (Figure 1, p. 6). The first quarter note is of medium class (numerical 0), the eighth notes are of class short (-1), the prolonged quarter note in measure 2 still falls in the medium class, and for the last tone, no IOI class can be given, but let us assume that it ends a phrase, so it might be assigned to the class very long (2). As a result, we have a sequence of relative IOI classes

$$(0, -1, -1, -1, -1, -1, 0, -1, 2).$$

Another important set of features comes from the field of micro-timing (micro-rhythm). Micro-timing can be generally defined as the deviations of performed timing from a nominal timing as for instance prescribed in a score. However, in the case of jazz improvisation, where there is no score

which could define nominal values, the definition of micro-timing is not as straightforward and is intrinsically intertwined with the metrical annotation process (see Frieler & Pfleiderer, 2017 for a more in-depth discussion of this topic). Nevertheless, if a metrical framework is given, the deviation from nominal onset can be measured. The MeloSpySuite/GUI contains some local features in this respect. For jazz solos, the swing ratio is a very important special case of systematic micro-timing. It is defined as the ratio of the first to the second eighth note in a binarily divided beat. The MeloSpySuite/GUI contains some local transformations and global scalar values for measuring swing ratios.

Sequence/Interval, Sequence/Pitch, Sequence/Rhythm

These classes of features deal all with direct sequential aspects of melodies, based on different abstractions (transformations) of the three main types: interval, pitch, and rhythm. Moreover, there are two main types of construction: N-gram based and run-length based sequential features.

N-gram-based features are closely related to the concept of patterns (see p. 38). N-grams are defined as subsequences $e_{j:k}$ of length $N = k - j + 1$ of a sequence $\{e_i\}_{1 \leq i \leq L}$. The sequences of interest mostly originate from simple point-wise one-dimensional transformations such as semitone intervals, pitch classes, or duration classes. A sequence of length L has $L - N + 1$ subsequences of length N, since at each position of a sequence except the last $N - 1$ a subsequence can be found. Of interest then are distributions of subsequences, e. g., frequencies of N-grams and other statistical properties, particularly information entropies. This is also connected to Markov chains, since the frequencies of bigrams are an estimator for the transition probabilities between elements. Generally, N-grams are connected to Markov chains of order $N - 1$.

Example. Consider all unigrams (1-grams), bigrams (2-grams), and trigrams (3-grams) for the relative IOI classes of 'The Lick' (numerical representation) listed above. For this sequence of nine elements, nine unigrams, eight bigrams, and seven trigrams can be extracted; see Table 8, where the N-grams are shown along with their frequencies. In the predefined features of the MeloSpySuite/-GUI only bi- and trigrams are available. This has mostly practical reasons, but is also motivated by the fact that N-gram spaces for finite sequences quickly become rather sparse with increasing N, e. g., the set of observed N-grams is much smaller than the space of all possible N-grams, which has size M^N, exponential in the size M of the alphabet (i. e., the set of all possible outcomes). This means that each N-gram is mostly observed only once, and certain features, such as the entropy, cannot be meaningfully estimated.

Table 8: Uni-, bi-, trigrams and runs of relative IOI classes for 'The Lick'.

Position	Unigram			Bigram		Trigram	
	Value	Count	Run length	Value	Count	Value	Count
0	M	2	1	MS	2	MSS	1
1	S	6	5	SS	4	SSS	3
2	S	6		SS	4	SSS	3
3	S	6		SS	4	SSS	3
4	S	6		SS	4	SSM	1
5	S	6		SM	1	SMS	1
6	M	2	1	MS	2	MSL	1
7	S	6	1	SL	1	—	—
8	L	1	1	—	—	—	—

Note. M = MEDIUM (0), S = SHORT (-1), L = VERY LONG (2).

If the alphabet is finite, the normalized (sample) entropy of an N-gram distribution of a sequence can be estimated to (cf. Infobox 6)

$$H_N^0 = -\frac{\sum_{n \in \text{all N-grams}} f_n \log f_n}{N \log M},$$

where the relative frequencies $f_n = F_n/(L-N+1)$ are based on the count F_n of N-grams n and the number $(L-N+1)$ of all subsequences of length N in a sequence of length L. The denominator is the maximal N-gram entropy for N-grams over an alphabet of M elements, i. e.,

$$H_N^{\max} = \log \frac{1}{M^N} = -N \log M.$$

.

Example. To calculate the normalized relative IOI class bigram entropy (ioiclass_bigram_entropy_norm) of 'The Lick' we need the relative frequencies $f_{\texttt{MS}} = \frac{2}{8} = \frac{1}{4}$, $f_{\texttt{SS}} = \frac{4}{8} = \frac{1}{2}$, and $f_{\texttt{SM}} = f_{\texttt{SL}} = \frac{1}{8}$, which can easily be read off from Table 8. The denominator is $2\log(5) = -3.22$, since the time-span classes have five possible outcomes. We have

$$\begin{aligned} H_2^0(\text{The Lick}) &= \frac{\frac{1}{2}\log\frac{1}{2} + \frac{1}{4}\log\frac{1}{4} + 2\frac{1}{8}\log\frac{1}{8}}{2\log(5)} \\ &= \frac{-1.23}{-3.22} \\ &= 0.377. \end{aligned}$$

Another important type of sequence features is based on run lengths. A 'run' in any sequence is a subsequence with only identical elements. Run lengths are a natural way of looking at sequences and have several applications, e. g., for statistical tests or for data compression. For example, in the relative IOI class abstraction of 'The Lick' there are four runs of length 1 and one run of length 5 (a sequence of eighth notes); see Table 8, Columns 2 and 4. Run lengths can be used to construct features either as mean lengths of specific elements, e. g., descending or ascending semitone intervals, or—more generally—as mean run lengths of all runs of all elements of a certain alphabet as well as derived constructions, e. g., ratio_chromatic_sequences, which is the percentage of chromatic (semitone) passages with respect to all passages, where a passage is defined as a run of at least three elements.

Structure

Structural features relate to musical form, which is viewed here as the self-similarity of a melody with respect to a given segmentation, e. g., phrases. To quantize self-similarity for melodies, one needs a melodic similarity measure (Müllensiefen & Frieler, 2004; Frieler, 2009), which is normally conceptualized as a (symmetric) map

$$\sigma(m_1, m_2) \to [0, 1],$$

for two melodies m_1 and m_2 with values in the unit interval, where the value 1 is interpreted as 'identity' and 0 as maximal dissimilarity. If a segmentation of a melody m into M segments $(m_i)_{1 \leq i \leq M}$ is given, then the self-similarity matrix is given by the values

$$\sigma_{ij} = \sigma(m_i, m_j).$$

Currently, in the MeloSpySuite/GUI only interval-based self-similarity matrices for bar and phrase segmentations are available and the similarity measure is defined via the edit distance (Müllensiefen & Frieler, 2004). From a self-similarity matrix, scalar features can be derived via statistical descriptors of distributions of similarity values, e. g., for adjacent segments or non-adjacent segments. Using a clustering algorithm to find clusters of segments with high inter-group similarity, the `SSM` can also be expressed as a form string, e. g., where similar segments are labeled with letters and primes, e. g., `aaba'`. (This is also available for duration classes.)

Tone formation

Tone formation means the actual process of performing a note (the map of abstract notes to performed tones). Currently, this falls into three main

categories: articulation, f_0-modulation, and loudness. Micro-timing as an important fourth dimension is sorted under rhythm features.

Articulation is traditionally addressed with musical labels such as staccato, legato, portamento etc., which often also have special symbols in common Western musical notation, e. g., dots for staccato and horizontal lines for legato. There are also instrument specific articulations, such as spizzicato for bowed instruments. For our purposes, we approximately define the articulation of a tone event simply as the ratio of duration to inter-onset interval. Higher ratios indicate more legato and smaller ones more staccato articulation. This is only a first approximation, e. g., there is an absolute, tempo-dependent aspect to articulation which is ignored when using time-spans ratios—which are tempo invariant by definition. With this definition, the articulation values with a range of `0` to 1 for monophonic melodies can be easily calculated from the basic representation and scalar global features can be derived using statistical descriptors such as mean, median, and standard deviation.

f_0-modulation means the changing of the fundamental frequency (f_0) of a tone. The abstraction that a tone event has one single pitch is very often not justified in practice, particularly not for wind instruments and singing. Frequently, bends, slides, and vibratos can be observed. Measuring f_0-modulations is not an easy task. The techniques used for the solos in the Weimar Jazz Database are explained in detail in Chapter **??**. In the MeloSpySuite/GUI only a basic set of features is available, which includes the deviation of the f_0 from the nominal 12-tone equal tempered pitch, the vibrato (modulation) frequency and the modulation range (in cents). Moreover, for the solos in the Weimar Jazz Database manual tone-wise annotations of f_0-modulation types are available, which can take on the values `BEND`, `SLIDE`, `FALL-OFF`, `STRAIGHT` (no modulation), and the empty value (no annotation).

Loudness and intensity are the third aspect of tone formation. Intensity values are not easily measured and the corresponding loudness (a psycho-physical concept) even less so. Nevertheless, for each tone event in the Weimar Jazz Database, intensity estimations were extracted using sophisticated audio algorithms, as described in detail in Chapter **??**. This results in a frame-wise estimation of intensity values per tone, for which the maximum, median, and standard deviations are available. Furthermore, the relative position of the loudness maximum with respect to the duration of the tone and the temporal centroid of the frame-wise loudness distribution are provided. Finally, the ratio of the median loudness to the median value of the background track is included (signal-to-background ratio).

Patterns

In the context of jazz research, we are interested in the use of patterns during improvisation (e. g., Owens, 1974; Norgaard, 2014). To measure the pattern content in improvisations, we devised the melpat module of the MeloSpy-Suite/GUI, which helps with two basic tasks in this context: pattern search and pattern mining/extraction. In the first task, a certain pattern is given and instances of this pattern are to be found in a corpus of melodies. In the second task, all patterns, possibly subject to certain conditions, are to be extracted automatically from a corpus of melodies.

More specifically, patterns in the context of melodies are defined as subsequences, or N-grams, of a certain representation of the melody, e. g., interval or pitch sequences. Therefore, we recall and extend first the notations for sequences and N-grams as introduced earlier (p. 23).

Basic definitions

Our starting point is a corpus $\mathscr{C} = \{m_i\}$ of melodies and a local (point-wise) transformation F with target space T_F, so that the object of interest is the set of transformed sequences in the corpus $\mathscr{F} = \{s_i \equiv F(m_i), m_i \in \mathscr{C}\}$. For the sake of simplicity, we will write $\mathscr{C}$ for a transformed corpus as well.

The maximal length of the corpus is the length of the longest sequence in $\mathscr{C}$, notated as $l_{\max}(\mathscr{C}) = \max_{s\in\mathscr{C}} l(s)$, where $l(s)$ is the length of the sequence s. The size of the corpus is the number of sequences in $\mathscr{C}$, notated as $\sigma(\mathscr{C})$. The total length of the corpus is the sum of all lengths of all sequences

$$l_\infty(\mathscr{C}) = \sum_{s\in\mathscr{C}} l(s).$$

We denote subsequences of a sequence s with

$$s_{i:j} = (s_i, s_{i+1}, \ldots, s_j)$$

for $0 \leq i \leq j < l(s)$. The length of the subsequence is $l(s_{i:j}) = j - i + 1$. We denote the set of all subsequences of length n of s as $\mathbf{n}(s)$, e. g., $\mathbf{2}(s)$ is the set of all bigrams, $\mathbf{3}(s)$ the set of all trigrams of s.

Two sequences are said to be identical if they have the same length and the same sequence of elements. An N-gram is then the equivalence class of all identical sequences of length N (over the target space T_F). The standard representative of an N-gram is then the sequence $[0 : N-1] \to T_F$. The

frequency of an N-gram n in a sequence s is the size of the set of occurrences of n in s:

$$\chi_s(n) = \{i \in [0 : l(s)-1], \text{where } s_{i:i+N-1} = n\},$$

hence, $F_s(n) = |\chi_s(x)|$. $\chi_s(n)$ is the set of all indices for s where the N-gram n can be found. We say n is contained in s, $n \in s$, if the set of occurrences is not empty, i. e., $\chi_s(n) \neq \emptyset$.

The relative frequency of n in s is defined as the ratio of frequency to the maximal number of subsequences of length N in s, i. e.,

$$f_s(n) = \frac{F_s(n)}{l(s)-N+1},$$

see Table 8 for an example.

The frequency of an N-gram in a corpus $\mathscr{C}$ is the sum of all frequencies of the N-gram over all sequences in the corpus, i. e., the size of the set of occurrences of n in $\mathscr{C}$:

$$\chi_{\mathscr{C}}(n) = \bigcup_{s \in \mathscr{C}} \chi_s(n).$$

Hence,

$$F_{\mathscr{C}}(n) = |\chi_{\mathscr{C}}(n)| = \sum_{s \in \mathscr{C}} F_s(n).$$

The relative frequency of n in the corpus is then defined as

$$f_{\mathscr{C}}(n) = \frac{F_{\mathscr{C}}(n)}{\sum_{s \in \mathscr{C}} l(s)-N+1}.$$

The number of occurrences of an N-gram n in the corpus is the number of sequences where n appears at least once, i. e., the size of the set of embedding sequences:

$$\omega_{\mathscr{C}}(n) = \{s \in \mathscr{C} \text{ where } n \in s\}.$$

Pattern search

Pattern search can be defined as searching for a given sequence in a corpus (after suitable transformation). For a given search pattern (N-gram) n in a corpus $\mathscr{C}$, the search result is just the set of subsequences and indices where the pattern can be found:

$$\rho_{\mathscr{C}}(n) = \{(j, i, l(n)) \mid i \in \chi_{s_j}(n) \neq \emptyset\}.$$

Example. Let us say we want to find all instances of 'The Lick' in the Weimar Jazz Database. To find a (possibly too large) set of candidates, it is recommended that we start with a plain interval search. The semitone interval sequence of the lick is $(0, -2, 2, 1, 2, -3, -4, 2)$ (cf. Table 1). Hence, we can first select the transformation intervals in the pattern search tab of the MeloSpyGUI and enter the string

```
0 -2 2 1 2 -3 -4 2
```

and start processing. This will result in a list of instances of this interval pattern (as a CSV file). Indeed, this search finds only one instance, in the Chet Baker solo.

This simple search process can be amended in two ways. Firstly, by secondary search, i. e., searching within the result set using secondary criteria, and secondly, by using regular expressions. Of course, both extensions can also be combined.

To define secondary search, we fix two transformations F_1 and F_2 over a given corpus and two search patterns with respect to the two transformations, say n_1 and n_2. The result of the secondary search is then simply the intersection of the single searches:

$$\rho_{\mathscr{C}_1 \times \mathscr{C}_2}(n_1, n_2) = \rho_{\mathscr{C}_1}(n_1) \cap \rho_{\mathscr{C}_2}(n_2).$$

It is desirable to have compatible results from both searches, in the sense that they should refer to exactly the same subsequences in the original untransformed melodies. This might be a problem where a local transformation is actually defined over subsequences, i. e., semitone or inter-onset intervals which need two consecutive elements for calculation. Retrieving the original subsequence in the melody is called back-lifting. We refrain from formalizing the process of back-lifting, but give an example for semitone interval transformations. An interval at position i in an interval sequence is back-lifted to the subsequence $m_{i:i+1}$ in the original sequence, if the indexing of the interval transformation was done in forward fashion, i. e., via $\Delta p_i = p_{i+1} - p_i$, or to the subsequence $m_{i-1:i}$ if the indexing is done in backward fashion $\Delta p_i = p_{i-1} - p_i$.

Regular expressions (regex for short) are a very powerful method for defining flexible search patterns for strings, i. e., sequences of characters. They have a longstanding tradition in computer science. They are defined using a rather cryptic syntax, which can be viewed as a very specialized computer language. Nearly all modern computer languages provide some native support for regular expressions and there are different flavors or variants of regular

Table 9: List of special characters for regular expression search.

Special character	Meaning
`.`	matches any element
`^`	matches beginning of sequence
`$`	matches end of sequence
`*`	zero or more repetitions
`+`	one or more repetitions
`?`	zero or one repetition
`{m}`	exactly *m* repetitions
`{m,n}`	between *m* and *n* repetitions
`|`	matches either regex before or after the symbol

expressions. The basic elements for regular expressions are character sets, quantifiers, and position indicators. For instance, the string `[a-z]` indicates the set of lower-case letters in most regex implementations. Wildcards, e. g., a single dot, which stand in for an arbitrary character are also very important. Examples for quantifiers are `*`, meaning 'zero or more instances', and `+` for 'one or more instances'. Examples for position parameters are `^` ('caret') for 'beginning of a string' and `$` for 'end of a string' (cf. Table 9).

With only these few elements, rather complex search patterns can already be defined. For example, the regex `^A[a-z]+C` will find all strings that start with an upper-case `A`, followed by at least one lower-case letter, and an upper-case letter `C`. For example. the strings `ABC`, `AAbC`, and `AC` would not match this pattern, but `AbC` and `AabcC` would.

Regular expressions are defined for finite alphabets, i. e., ASCII or unicode. To use the powerful regex machinery for melodic sequences, only transformations which have a finite alphabet can be used. For continuous transformations, i. e., inter-onset intervals, they are not applicable. On using regular expressions with the MeloSpyGUI, see Infobox 7.

Infobox 7: Using regular expressions for pattern search

The use of regular expressions (regex) for finite-sized transformations is done in the MeloSpySuite/GUI by mapping the target domain of the transformation to a consecutive segment in the unicode alphabet and using the regular expression library for Python. The basic syntax of regular expressions thus

follows the Python rules (https://docs.python.org/2/library/re.html). However, this mapping to unicode characters cannot be used for actually inputting regular expressions, which makes a hybrid syntax necessary. Moreover, using fixed character classes such as `\w`, which means all alphanumeric characters (i. e., `[a-zA-Z0-9_]`) in standard Python regex syntax is of no use. Instead, the standard representations have to be used to define 'character sets' manually. For example, for semitone intervals, integer numbers are used, e. g., the sequence `-1 0 1 -2` (without commas) means a sequence of a semitone down, a tone repetition, a semitone up, and a whole-tone down. All quantifiers and position parameters have to be (singly or doubly) quoted, e. g., `'[' -1 0 ']+'` means one or more occurrences of the elements `-1 0` (semitone down and repetition). The most important special characters for the use of regexes in pattern search can be found in Table 9.
In case of the above example, the interval search for 'The Lick', one might think that the first tone repetition could be absent and the last tone could be varied. This can be expressed with the regex

```
0 "?" -2 2 1 2 -3 -4 "."
```

This will find variations of 'The Lick' with and without the first tone repetition, as expressed by the quantifier `?` translating to 'zero or one tone repetition', and with all possible end tones, expressed by the dot `.` as a stand-in for all possible intervals. The actual search in the Weimar Jazz Database brings up six instances of four different patterns: the original instance with the tone repetition at the beginning, and five beginning with a descending whole tone (-2), two of which end in a descending fourth (-5) and two in a descending minor third (-3).

Pattern mining

The task of pattern mining or pattern extraction is to extract all patterns from a given corpus $\mathscr{C}$, which is subject to certain conditions for the patterns. A pattern database with minimal length K, maximal length M is the set of all N-grams with $k \leq N \leq M$. Often, we like to subject the pattern database to further conditions, e. g., minimal frequencies or occurrences. In fact, (sequential) patterns are often defined as N-grams that occur at least k times in at least l different entities in the corpus, with the minimal definition of $K = 2$ and $L = 1$. Obviously, $L \geq 1$ implies also $K > 1$.

Another useful restriction, available as a submodule to the melpat module in the MeloSpySuite/GUI, is called pattern partition. A pattern partition is defined for a single sequence s in the corpus with respect to the pattern database of the entire corpus (including the borderline case of a single-sequence corpus).

It is constructed by filtering sub-N-grams from a pattern database as defined above (possibly subject to length, frequency, and occurrence conditions). A sub-N-gram n is proper if there exists a super-K-gram k with $K > N$ and $n \in k$ such that all occurrences of n also imply the occurrence of k. If an N-gram n only occurs as a certain sub-N-gram, then it is not considered to be an independent entity but as completely derivative and hence as not interesting in itself. A pattern database can be substantially pruned by filtering out all derivative N-grams. The pattern partition of a sequence s is then the subset of N-grams of s in the so pruned pattern database.

Example. Consider the mini-corpus $\mathscr{C} = \{s_1, s_2\}$ of two sequences given in Parsons's code:

$$\begin{aligned} s_1 &= \texttt{UUDUUD}, \\ s_2 &= \texttt{UUDUDD}. \end{aligned}$$

Here, the pattern database of bi- and trigrams occurring at least twice is composed of the trigrams `UUD` (3) and `UDU` (2), and the bigrams `UU` (3), `UD` (4), and `DU` (2) (counts in parentheses). As one can readily see, the bigram `UU` only occurs as part of the trigram `UUD`. Hence, it would be filtered out for a pattern partition. On the other hand, the bigram `UD`, which is also a sub-bigram of `UDD`, would be kept, since it also occurs as a sub-bigram of `DUD` and `UDD`. (See also **??** in Chapter **??** for a graphical representation of an interval pattern partition of at least length $N = 7$ in a solo by Woody Shaw.)

For pattern partitions and databases, certain global scalar features can be defined that try to capture how much the sequences in a corpus are determined by patterns, which are defined here as N-grams with the minimal condition of occurring at least twice in at least one sequence. For the following, we fix a corpus and drop the index $\mathscr{C}$ for simplicity. Likewise, we fix a pattern database

$$\mathscr{P} = \bigcup_{N \le i \le M} \mathscr{P}_i$$

as the union of (pruned) N- to M-grams. The pattern partition of a sequence s is then the intersection of $\mathscr{P}$ with all N-grams of s (of the same length range):

$$\mathscr{P}(s) = \mathscr{P} \cap \bigcup_{N \le n \le M} \mathbf{n}(s).$$

The coverage of a sequence s by patterns from $\mathscr{P}$ is the percentage of elements in s contained in at least one N-gram. If we define the set of index intervals in s where any n covered by any N-gram from $\mathscr{P}$ as

$$\Gamma(s) = \{\{i, \ldots, i+N-1\} \mid \exists_{n \in \mathscr{P}} : i \in \chi_s(n)\},$$

then the coverage is simply

$$\gamma(s) = \frac{|\Gamma(s)|}{l(s)}.$$

The over coverage is the average number of N-grams to which an element in s belongs. The average overlap is defined as the average number of elements two adjacent patterns $\mathscr{P}$ share, where the patterns are sorted with respect to start positions in s. The average N-gram length is defined as the mean length of N-grams contained in the pattern partition of s. Finally, the mean logarithm excess probability is defined using the ratio of the observed frequency of a pattern to the expected frequency derived from a Markov process of zeroth order. The latter is just a complicated term for the product of the probabilities of the elements that constitute an N-gram. For an illustration, consider the mini-corpus of Parsons's code sequences from above. The relative frequencies of the unigrams are $f_{\mathtt{U}} = \frac{7}{12}$ and $F_{\mathtt{D}} = \frac{5}{12}$. Using elementary probability theory, the *a priori* frequency is the probability under the assumption that all elements in the sequence are independently produced, i. e., the product of the single probabilities. The trigram `UUD` thus has an expected probability of

$$p_e(\mathtt{UUD}) = p(\mathtt{U})p(\mathtt{U})p(\mathtt{D}) = \frac{7}{12} \cdot \frac{7}{12} \cdot \frac{5}{12} = 0.142.$$

On the other hand, the observed frequency is three times out of a possible eight times, or $p_o = \frac{3}{8} = 0.375$. The ratio of observed to expected probabilities, the excess probability, is then

$$\frac{p_o}{p_e} = \frac{0.375}{0.142} = 2.64.$$

Hence the trigram `UUD` appears about 2.6 times more often than an independent (Markov) process would suggest. The logarithm of excess probability is then used to make the process of averaging excess probabilities over all patterns more meaningful.

References

Abeßer, J., Cano, E., Frieler, K., Pfleiderer, M., & Zaddach, W.-G. (2015). Score-informed analysis of intonation and pitch modulation in jazz solos. In *Proceedings of the 16th International Society for Music Information Retrieval Conference (ISMIR).* Málaga, Spain.

Bregman, A. S. (1990). *Auditory scene analysis. The perceptual organization of sound* (2nd ed.). Cambridge: MIT-Press.

Dowling, W. J., & Fujitani, D. (1971). Contour, interval, and pitch recognition in memory for melodies. *The Journal of the Acoustical Society of America*, *49*(2, Suppl. 2), 524–531.

Forte, A. (1973). *The structure of atonal music*. New Haven: Yale University Press.

Fraisse, P. (1982). Rhythm and tempo. In D. Deutsch (Ed.), *The psychology of music* (pp. 149–180). New York: Academic Press.

Frieler, K. (2007). Visualizing music on the metrical circle. In *Proceedings of the 8th International Symposium on Music Information Retrieval, ISMIR 2007.* Wien: OCG.

Frieler, K. (2008). Metrical circle map and metrical Markov Chains. In A. Schneider (Ed.), *Systematic and comparative musicology.* Frankfurt/M., Bern: P. Lang.

Frieler, K. (2009). *Mathematik und kognitive Melodieforschung. Grundlagen für quantitative Modelle.* Hamburg: Dr. Kovač.

Frieler, K., & Lothwesen, K. (2012). Gestaltungsmuster und Ideenfluss in Jazzpiano-Improvisationen. Eine Pilotstudie zum Einfluss von Tempo, Tonalität und Expertise. In A. C. Lehmann, A. Jeßulat, & C. Wünsch (Eds.), *Kreativität – Struktur und Emotion* (pp. 256–265). Würzburg: Königshausen & Neumann.

Frieler, K., & Pfleiderer, M. (2017). Onbeat oder offbeat? Überlegungen zur symbolischen Darstellung von Musik am Beispiel der metrischen Quantisierung. In M. Eibl & M. Gaedke (Eds.), *INFORMATIK 2017* (pp. 111–125). Bonn: Gesellschaft für Informatik.

Frieler, K., Pfleiderer, M., Abeßer, J., & Zaddach, W.-G. (2016). Midlevel analysis of monophonic jazz solos. A new approach to the study of improvisation. *Musicae Scientiae*, *20*(2), 143–162.

Harte, Christopher, Sandler, M., Samer, A., & Gómez, E. (2005). Symbolic representation of musical chords. A proposed syntax for text annotations. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)* (pp. 66–71). Queen Mary, London. Retrieved from http://ismir2005.ismir.net/proceedings/1080.pdf

Huron, D. (1996). The melodic arch in western folksongs. *Computing in Musicology*, *10*, 3–23.

Lerdahl, F., & Jackendoff, R. (1983). *A generative theory of tonal music*. Cambridge, MA: The MIT press.

Müllensiefen, D., & Frieler, K. (2004). Cognitive adequacy in the measurement of melodic similarity. Algorithmic vs. human judgments. *Computing in Musicology*, *13*, 147–176.

Müllensiefen, D., Frieler, K., & Pfleiderer, M. (2009). The perception of accents in pop music melodies. *Journal of New Music Research*, *38*(1), 19–44.

Norgaard, M. (2014). How jazz musicians improvise. The central role of auditory and motor patterns. *Music Perception: An Interdisciplinary Journal*, *31*(3), 271–287.

Owens, T. (1974). *Charlie Parker. Techniques of improvisation* (Unpublished doctoral dissertation). University of California, Los Angeles.

Parsons, D. (1975). *The directory of tunes and musical themes*. London: British Library.

Schuller, G. (1958). Sonny Rollins and the challenge of thematic improvisation. *Jazz Review*, *November 58*, 6–11.

Schütz, M. (2015). *Improvisation im Jazz. Eine empirische Untersuchung bei Jazzpianisten auf der Basis der Ideenflussanalyse*. Hamburg: Dr. Kovač.

Selfridge-Field, E. (Ed.). (1997). *Beyond MIDI. The handbook of musical codes*. Cambridge: The MIT Press.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, *27*(3), 379–423 and 623–656.

Zanette, D. H. (2006). Zipf's law and the creation of musical context. *Musicae Scientiae*, *10*, 3–18.