# FANTASTIC: Feature ANalysis Technology Accessing STatistics (In a Corpus): Technical Report v1.5

Daniel Müllensiefen

June 19, 2009

## Contents

**Version History**

| | | |
|---|---|---|
| May 2009 | Version 1.0 | Describes three main functions: `compute.features()`, `compute.corpus.based.feature.frequencies()`, `compute.m.type.corpus.based.features()`. The latter is still very slow. |
| June 2009 | Version 1.5 | Describes new main function `feature.similarity()`. |

# 1 Introduction

FANTASTIC is a program, written in *R* [1], that analyses melodies by computing features. The aim is to characterise a melody or a melodic phrase by a set of numerical or categorical values reflecting different aspects of musical structure. This feature representation of melodies can then be applied in Music Information Retrieval algorithms or computational models of melody cognition. Apart from characterising melodies individually by feature values, FANTASTIC also allows for similarity comparisons between melodies that are based on feature computation.

Similarly to existing melody analysis tools (e.g. the MIDI tool box, Eerola and Toiviainen (2004)), the feature computation algorithms in FANTASTIC make use of ideas and concepts from *descriptive statistics*, *music theory*, and *music cognition*. But in contrast to most existing software, FANTASTIC also incorporates approaches derived from computational linguistics and provides the option to characterise a melody by a set of features with respect to a particular corpus of melodies.

The idea of characterising and analysing melodies in terms of features is not new and owes much to great predecessors in the work of, for example, Lomax (1977), Steinbeck (1982), Jesser (1990), Sagrillo (1999), Eerola and Toiviainen (2004). Introductions to the concept of feature-based melody analysis and examples of the application of this concept in Ethnomusicology, Musicology, Music Psychology, and Music Information Retrieval can be found in those publications.

This technical report documents the internal processing structure of FANTASTIC as well as the features currently implemented.

# 2 Input format

FANTASTIC uses symbolic representations of monophonic melodies as melodic input data. It accepts MCSV files (Frieler, 2005) which also are the input format to the melodic similarity computation program SIMILE by Klaus Frieler (Müllensiefen and Frieler, 2006). MCSV files are similar to kern files in that they represent musical events in a tabular way but are conceptually simpler and more limited (e.g. cannot handle polyphony yet). Subsequent lines in the MCSV file represent note events in the order appear in the melody and the different columns contain different information about the note events. The most important types of information are pitch (represented as MIDI number), note onset in seconds, note onset in metrical time, note duration (in metrical time and in seconds), and phrase boundary (binary information for every note event: 1 indicated that note is a phrase boundary, 0 means note is not a boundary). Currently Temperley's Grouper (Temperley, 2001) is used as melody segmentation algorithm, column `Tempe` in MCSV file).

MCSV files can be created by using Klaus Frieler's melody conversion program MELCONV. Currently MELCONV enables conversion between a number of symbolic music formats such as *MIDI*, the *Essener Assoziativ Code (EsAC)*, *notes* (Temperley, 2001), and

[1] see http://www.r-project.org

MCSV. But since they are ordinary text files with a tabular format it is easy to create MCSV files from most melody processing software.

# 3 Running the program

The source code of FANTASTIC comprises five .R files[2] which are packaged together and available online[3]. They need to be unpacked and placed into the same directory, preferably the working directory of the R-analysis. `Fantastic` makes use of the following, non-standard R-packages which have to be installed before running the program: `MASS` and `zipfR`.

Currently, there are three top-level functions[4] for computing features and feature frequencies from melodies encoded in MCSV files:

1. `compute.features`: Computes feature summary statistics and m-type summary statistics for a melodies given as a list of MCSV files. This includes features on the pitches, intervals, and rhythmic durations of a melody as well as features summarising the contour and the harmonic content. In addition, features characterising the repetitiveness of short melodic-rhythmic elements (which we call *m-types* see section 6.2) in a melody are also computed.

2. `compute.corpus.based.feature.frequencies`: For each melody, this function computes the density or relative frequency of the summary features with respect to the distribution of summary features in a specified corpus. That is, for each melody each feature value is replaced by its corresponding frequency value from the frequency distribution of a corpus. Thus, this computation makes a step from the feature scale to a *commonness vs. rarity* scale. The input to this function has therefore two main components: A list of analysis melodies for each of which frequency values are computed and a corpus of melodies from which the frequency distribution are derived.

3. `compute.m.type.corpus.based.features`: For each melody, this function computes features on the basis of how common or frequent the short melodic-rhythmic elements (*m-types*) are of which a melody is composed. This function has two main components as input: A list of analysis melodies for each of which m-types are extracted and a corpus of melodies from which the distribution of m-types is derived.

The three main functions are executed following three steps:

[2] The five source files are names `Fantastic.R`, `Feature_Value_Summary_Statistics.R`, `M-Type_Summary_Statistics.R`, `Frequencies_Summary_Statistics.R`, and `M-Type_Corpus_Features.R`.

[3] http://www.doc.gold.ac.uk/isms/m4s/FANTASTIC.zip

[4] We are planning to integrate `compute.corpus.based.feature.frequencies` and `compute.m.type.corpus.based.features` into a single function that computes information about melodies in the context of a corpus.

1. Start the *R* installation on your computer and load the file *Fantastic.R*.

2. Preferably, make the directory that contains the .R source files and the MCSV files for analysis your working directory (`setwd("<path to directory>")`).

3. Call the either of the three functions with the appropriate argument values (s. below) and assign an output object to it: e.g. `output <- compute.features(c("file1.csv", "file2.csv"))`.

## 3.1 Computing features of melodies

**Function:** `compute.features(melody.filenames = list.files(path = dir, pattern = ".csv"), dir = ".", output = "melody.wise", use.segmentation = TRUE, write.out = FALSE)`

**Description:** This is the main function for computing features of melodies encoded in MCSV files. This means that it computes features just for the melodies given, without any reference to a corpus of music, and returns a table containing feature values for each melody (or melodic phrase).

**Arguments:** The function `compute.features` takes the following arguments that control the analysis procedure and have been set to default values in the current implementation:

- `melody.filenames`: Takes the names of the files to be analysed. These can be either concatenated with the function `c()`, (e.g. `c("file1.csv", "file2.csv")`) or you can use the the R-function `list.files()` to list all the .csv files in the working directory (`list.files(pattern=".csv")`. By default, all the .csv files found in the directory specified by `dir` are used as `melody.filenames`.

- `dir`: Takes the absolute path (starting with "/" or "\") or relative path (starting with any other symbol) to and name of the directory that contains the .csv files for analysis (e.g. "`../analysis_directory`"). The default is the present working directory. If the value(s) given to `melody.filenames` contain the directory path to the files then the argument to `dir` is ignored.

- `output` takes the argument values `"melody.wise"` (default) and `"phrase.wise"`. This arguments determines whether the analysis information in the output object is given for the melody as a whole or on the basis of the individual melodic phrases as indicated in the MCSV file (see above).

- `use.segmentation` takes argument values `TRUE` (default) and `FALSE` and determines whether the feature computation is done on the melody as a whole or phrase by phrase.

- `write.out` takes argument values `TRUE` and `FALSE` (default) and determines whether a file with the analysis results should be written out.

The interactions between the arguments `output` and `use.segmentation` is specifically defined for the following two cases:

- If `use.segmentation` is set to `TRUE` and `output` is set to `"melody.wise"` then numerical features are averaged over all phrases in the melody and the most frequent value of categorical features is computed as output.
- If `use.segmentation` is set to `FALSE` and `output` is set to `"phrase.wise"` then the program emits an error message and terminates because in order to output features on a phrase-level the program has to make use of segmentation information.

**Output:** The output is a table (i.e. an *R*-data frame) that has as lines the data for each melody analysed (or the data for each phrase of each melody analysed if `output="phrase.wise"` is requested). The columns of the output table comprise file name (and phrase number) as well as analytic features that FANTASTIC computes. These can be numeric feature values or feature labels as character strings. For all melodies and melody phrase for which feature values cannot be computed (e.g. because the length of a phrase is outside the phrase length limits as given as a global parameter to the program see 4.2), then `NAs` are written out as features values for that phrase.[5]

## 3.2 Computing frequencies of melody features in the context of a melody corpus

**Function:** `compute.corpus.based.feature.frequencies(analysis.melodies = "analysis_dir", ana.dir = "analysis_melodies", corpus = "corpus_dir", write.out.corp.freq = TRUE, comp.feat.use.seg = TRUE, comp.feat.output = "phrase.wise")`

**Description:** This is the main function for computing the frequencies of features of melodies (the *analysis melodies*) in the context of a corpus of melodies. It returns a table containing frequency values for each analysis melody (or each melodic phrase).

**Arguments:** `compute.corpus.based.feature.frequencies` is run with the following arguments and default settings:

- `analysis.melodies`: Takes either a feature file as produced by `compute.features` and ending in ".txt" or a list of MCSV files or a directory name (including path) in which the melodies for analysis can be found. The default is a directory with name "analysis_dir" that is below the present working directory.

[5] For technical reasons and as an exception, if feature values cannot be computed for the first phrase of an analysis melody, then this phrase is skipped and no `NAs` are written out.

- `ana.dir`: Takes the path to the directory in which the analysis melodies as listed by `analysis.melodies` can be found. It is only necessary to specify a value for `ana.dir` if the value to `analysis.melodies` is a list of melodies. if the argument value to `analysis.melodies` is a feature file or a directory `ana.dir` is ignored.

- `corpus`: Takes either a feature file produced by `compute.features` or a file containing corpus frequencies (`feature_densities_list.txt`)[6] that was produced by a previous run of `compute.corpus.based.feature.frequencies` on the corpus or the name of the directory containing the MCSV files of the corpus or the same directory name or files list as the argument `analysis.melodies`. A corpus frequencies files is a binary file produced by saving a list of frequency tables as an R-object. Each frequency table contains the binned distribution of a feature in a corpus. Reading in the frequency distribution from such a corpus frequencies file or using feature file from frequencies can be derived directly saves time over computing feature values for the melodies of a corpus and then deriving frequencies from them. The default of the `corpus` argument is a directory with name "corpus_dir" that is below the present working directory.

- `write.out.corp.freq`: Takes `TRUE` (default) or `FALSE` and determines whether a corpus frequencies file should be written to the present working directory. This file contains a list of the binned distributions of all summary features and is in the binary format that R uses to write save R-objects. Its default name is `feature_densities_list.txt`. The name of this file can then be used as a value for the argument `corpus` when `compute.corpus.based.feature.frequencies` is run again with the same melody corpus. This will save a considerable amount of computing time.

- `comp.feat.use.seg`: Takes `TRUE` (default) or `FALSE` and determines whether phrase segmentation information should be used when melody features are computed. This argument applies to the computation of features from the analysis melodies and from the corpus melodies (if argument `corpus` is a list of files or a directory containing MCSV files).

- `comp.feat.output`: Takes "`phrase.wise`" (default) or "`melody.wise`" and determines whether features should be computed on phrase level or on melody level when feature computation from MCSV is necessary. This argument applies to the computation of features from the analysis melodies and from the corpus melodies (if argument `corpus` is a list of files or a directory containing MCSV files).

**Output:** In the output table, rows stand for analysis melodies and columns represent feature frequencies. The column names have the prefix `dens.` (e.g. `dens.d.entropy,`

[6] For technical reasons, the function gives an error message to the standard output when the argument to `corpus` is a file with corpus frequencies `feature_densities_list.txt`. However, it continues its computation and delivers correct results as the final output. The error message should thus be ignored.

dens.h.contour) in order to distinguish them from similarly named columns containing the values of the corresponding features. This table is automatically written to a file named densities_of_feature_values.txt.

## 3.3 Computing features on the occurrence of m-types in the context of a melody corpus

**Function:** compute.m.type.corpus.based.features(analysis.melodies, ana.dir =".", corpus, corpus.dir = ".")

**Description:** This is the main function for computing features that informs about the m-types occurring in a melody in the context of a corpus. Its usage follows the basically the same logic as compute.corpus.based.feature.frequencies and its main input components are a set of *analysis melodies* and a corpus of melodies. It returns a table containing m-type corpus features for each analysis melody (or melodic phrase).

**Arguments:** compute.m.type.corpus.based.features is run with the following arguments and default settings:

- analysis.melodies: Takes a list of files for which the m-type occurrence feature are to be computed.
- ana.dir: Takes the directory name (path) in which the analysis melodies are to be found. The default is the present working directory (".").
- corpus: Takes either the name of a *m-type frequency* file ending in ".txt" or a a list of files or the same list of files as the argument analysis.melodies.
- corpus.dir: Takes either the name of the directory in which the melodies of the corpus are to be found or the same value as the argument ana.dir. The default is the present working directory (".").

**Output:** The output of this function is a table where each row represents one analysis melody and columns stand for m-type occurrence features. Column names are prefixed with "mtcf." (= *m-type corpus feature*) to distinguish them form the m-type features that are computed from compute.features. The results are also written to a file named mtype_corpus_based_feat.txt. With every run compute.m.type.corpus.based.features writes out an *m-type frequency* file containing the m-types and their frequencies for every melody in the corpus is written to the present working directory. This *m-type frequency* file has the name m-type_counts_several_melodies.txt. Using this files as an argument to corpus in subsequent runs of compute.m.type.corpus.based.features with the same corpus information will save a considerable amount of computing time. A warning message occurs

on standard output during the computation of some rank correlations if all m-types occur with the same frequency and have thus the same rank.The full output is still computed and the warning message can be ignored.

## 3.4 Computing similarities between melodies based on features

**Function:** `feature.similarity(mel.fns=list.files(path=dir,pattern=".csv"), dir=".", features=c("p.range","step.cont.glob.var","tonalness","d.eq.trans"), use.segmentation=FALSE, method="euclidean", eucl.stand=TRUE, corpus.dens.list.fn=NULL, average=TRUE)`

**Description:** This is the main function for computing similarity values between melodies. It computes features using the function `compute.features` underneath and offers three different methods how similarity values can be derived from the feature values of two melodies. It takes a list of melodies as input and its output is a similarity matrix containing similarity values for all (symmetric) pairwise comparisons possible for the melodies given as input.

**Arguments:** `compute.features` is run with the following arguments:

- `mel.fns`: Takes a list of MCSV files for which pairwise similarities should be computed. By default, all the .csv files found in the directory specified by argument `dir` are used as `mel.fns`. The files must contain segmentation information even if it not used in the computation.[7]
- `dir`: Takes the directory names (path) in which the input files are to be found. The default is the present working directory (".").
- `features`: Takes a character vector of feature names on the basis of which similarities should be computed. Currently, only features described in 6.1 are allowed. The default is rather arbitrarily set to `c("p.range","step.cont.glob.var","tonalness","d.eq.trans")`. If the value to argument `method` is ``euclidean" only numerical features can be used, i.e.the categorical features `h.contour`, `mode`, and `int.contour.class` are not allowed.
- `use.segmentation`: Takes `TRUE` or `FALSE` (the default) as input and determines whether phrase segmentation information should be used by `compute.features` to compute the feature values for each melody.
- `method`: Takes a character string to determine the method which should be used to derive similarity values from from the feature values of each melody. The method must be one of ``euclidean" (the default), ``gower" or ``corpus".

[7] Unfortunately, the function doesn't currently accept a feature data.frame as input, such as the output of `compute.features()`. Therefore, the feature computation has to be run again with every call to `feature.similarity()`, even if it is on a set of melodies that has been used as input previously.

- `eucl.stand`: Takes either `TRUE` or `FALSE` (the default) and determines whether features values should be standardised over all input melodies when `euclidean` is the value of argument ``` ``method" ```. The standardisation used is the so-called *z-standardisation* (subtraction of feature mean and division by feature variance).

- `corpus.dens.list.fn`: Takes filename and path of a file containing information about the frequency distributions of features in a corpus (default file name: `feature_densities_list.txt`) as produced by the function `compute.corpus.based.feature.frequencies`. This corpus information is only used when the value to argument ``` ``method" ``` is ``` ``corpus" ```.

- `average`: Takes the values `TRUE` (the default) or `FALSE` and determines whether the mean of the similarity values based on different features should be taken. If `FALSE` one similarity matrix for each feature is outputted.

**Output:** The output of this function is a list of similarity matrices. Each matrix is an R-object of class `dist` and is of size $(n-1)^2$ where $n$ is the number of melodies given as input. Only the lower triangle of each matrix is fully (no diagonal). Note that while the matrix is of class *dist*, the values represent similarities on a scale from 0 (minimum similarity) to 1 (maximal similarity / identity). In order to obtain a distance matrix that could be used as input to existing R-functions, such as `hclust()`, the matrix has to be transformed first by subtracting it from 1, e.g. `dist.matrix <- 1 - sim.matrix`. When the input to argument `feature` is more than 1 feature and the value of argument `average` is `FALSE`, then the output is a list of similarity matrices, one for each feature. A conversion from this matrix of class `dist` to a normal $n \times n$ matrix can be achieved using the R-function `as.matrix()`.

# 4 Overview

## 4.1 Global architecture

The global architecture is displayed graphically in the processing flow chart **??**.

## 4.2 Global parameters

FANTASTIC operates on the basis of a few global parameters.

1. `phr.length.limits <- c(2, 24)`: Is a vector of length 2 that holds the lower and upper limit of the length of a phrase. Defaults are `2` and `24`.

2. `int.class.scheme`: Is a data frame that holds pitch intervals (in semitones) and corresponding interval classes (represented as 2-digit sequences of letters and numbers). This pitch interval classification scheme is used for constructing the so-called

*m-types* (see section 6.2). The default classification scheme is summarised in table 1:

| Interval in semitones | Interval Class Value |
|---|---|
| -12 | d8 |
| -11 | d7 |
| -10 | d7 |
| -9 | d6 |
| -8 | d6 |
| -7 | d5 |
| -6 | dt |
| - 5 | d4 |
| -4 | d3 |
| -3 | d3 |
| -2 | d2 |
| -1 | d2 |
| 0 | s1 |
| 1 | u2 |
| 2 | u2 |
| 3 | u3 |
| 4 | u3 |
| 5 | u4 |
| 6 | ut |
| 7 | u5 |
| 8 | u6 |
| 9 | u6 |
| 10 | u7 |
| 11 | u7 |
| 12 | u8 |

Table 1: Interval classification scheme for construction of m-types

3. `tr.class.scheme`: Is a list of two vectors. The first vector holds the labels for three relative rhythm classes (represented as 1-letter strings). Defaults are `class.symbols = c("q", "e", "t")`. The second vector has the two upper limits of rhythm class 1 and 2 (represented as numeric time ratios). Defaults are `upper.limits = c(0.8118987, 1.4945858)`. This duration ratio classification scheme is used for constructing the so-called *m-types* (see section 6.2).

4. `n.limits <- c(1, 5)`: Is a vector of length 2 that holds the lower and the upper limit of the length of the *m-types* to be used for analysis (see section 6.2). Defaults are `1` and `5`.

# 5 Basic representation of melodic data

In a similar way to the MCSV file format (see section 2), FANTASTIC represents a melody internally as a sequence of notes which are represented as tuples of time and pitch information:

$$n_i = (t_i, p_i)$$

The basic unit of pitch information is always MIDI pitch while the basic unit of time information is expressed in milliseconds as a unit of absolute time and as the smallest metrical unit occurring in a given melody which we will call *tatum* in the remainder of this document.[8] Whether timing information expressed in milliseconds or tatums depends on the purpose and technical construction of the feature it is used in. The MCSV file provides both types of information.

# 6 Features based on the content of a single melody

## 6.1 Feature Value Summary Statistics

The functions for computing Feature Value Summary Statistics can be found in the file *Feature_Value_Summary_Statistics.R*. The main function for computing these features is `summary.phr.features`. As its first argument (`phr.data`) this function takes an R-dataframe having the same tabular structure as an MCSV file. For using the content of any MCSV file with this function the user first has to load the content of the file into an R-object by: `example.melody <- read.table(filename,sep=";",dec=",",skip=1,header=TRUE)`. Then `example.melody` can be the first argument to `summary.phr.features`. The second argument (`poly.contour`) specifies whether features from the polynomial contour representation ( 6.1.5) should be computed. Its default value is `TRUE`.

The features in this section use simple descriptive statistics on pitch, interval, and duration information of the melodies as well as some global features regarding melody extension in time, melodic contour, and tonality.

### 6.1.1 Descriptive statistics on pitch

**Feature 1 (Pitch Range: `p.range`)**

$$\text{p.range} = \max(p) - \min(p) \tag{1}$$

[8] The term *tatum* was invented in computational music analysis by (Bilmes, 1993, footnote p. 21) to denote the high frequency pulse or smallest metrical unit that can be perceived in a piece. Before this concept was appropriated by computational musicology, it was used to describe the smallest division of the beat and used in the description of, for example, west-african music. The term for this concept varied between authors and was called *density referent* Hood (1971), *elementary pulse* Kubik (1998), or *minimal operational value* Arom (1991). For a comparative discussion see Pfleiderer (2006).

**Feature 2 (Pitch Standard Deviation: `p.std`)**

$$\text{p.std} = \sqrt{\frac{\sum_{i=1}^{N}(p_i - \bar{p})^2}{N-1}} \tag{2}$$

**Feature 3 (Pitch Entropy: `p.entropy`)** *This is a variant of Shannon entropy (Shannon, 1948). It is computed on the basis of the relative frequencies $f_i$ of the pitch classes $p_i$ of a melody and normalised by maximum entropy given the upper phrase length limit assumed above (24).*[9] *We denote the absolute frequency of pitch class $i$ by $F_i$ and the relative frequency by $f_i$.*

$$f_i = \frac{F(p_i)}{\sum_i F(p_i)}$$

$$\text{p.entropy} = -\frac{\sum_i f_i \cdot \log_2 f_i}{\log_2 24} \tag{3}$$

### 6.1.2 Descriptive statistics on pitch intervals

Pitch intervals are derived from pitches by calculating the difference between two consecutive pitches:

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

In addition to raw intervals that have magnitude and direction information, features are also computed on the absolute intervals of a melody which are only characterised by their magnitude:

$$|\Delta p_i|$$

**Feature 4 (Absolute Interval Range: `i.abs.range`)**

$$\text{i.abs.range} = \max(|\Delta p|) - \min(|\Delta p|) \tag{4}$$

**Feature 5 (Mean Absolute Interval: `i.abs.mean`)**

$$\text{i.abs.mean} = \frac{\sum_i |\Delta p_i|}{N} \tag{5}$$

*where $N$ is the length of the interval vector $\Delta p$.*

[9] Note that the standard way of normalising entropy is by diving by the maximum entropy as given by the log of the size of the symbol alphabet - in this case the pitch alphabet. Unfortunately, it can not be known in advance what size of the pitch alphabet is for any given set of melodies. However, empirically, the length of a melody correlates quite strongly with its entropy and therefore it seemed reasonable to standardise entropy by the maximum phrase length that is given to the program as a global parameter. Thus, the maximum entropy is here assumed to be the entropy of a melody with maximum phrase length that has a different pitch class value for each note.

**Feature 6 (Standard Deviation Absolute Interval: `i.abs.std`)**

$$\text{i.abs.std} = \sqrt{\frac{\sum_i (|\Delta p_i| - \overline{|\Delta p|})^2}{N-1}} \tag{6}$$

**Feature 7 (Modal Interval: `i.mode`)** *The modal interval is the most frequent interval in a melody. In case that there is no single most frequent interval, the interval with the highest (positive) number of semitones is chosen.*[10]

**Feature 8 (Interval Entropy: `i.entropy`)** *Inerval entropy is computed analogous to pitch entropy but using* $\log_2 23$ *since the maximum number of different intervals given the phrase lengths limits is 23.*

$$f_i = \frac{F(\Delta p_i)}{\sum_i F(\Delta p_i)}$$

$$\text{i.entropy} = -\frac{\sum_i f_i \cdot \log_2 f_i}{\log_2 23} \tag{7}$$

### 6.1.3 Descriptive statistics on note durations

Since the MCSV format does not represent rests adequately FANTASTIC represents note durations as inter-onset intervals (IOIs). For durations in milliseconds we write $\Delta t$ and for durations measured in metrical tatums we write $\Delta T$. $\Delta t$ is used as a quasi-continuous representation for durations and $\Delta T$ serves as a discrete numerical representation.

**Feature 9 (Duration Range: `d.range`)**

$$\text{d.range} = \max(|\Delta t|) - \min(|\Delta t|) \tag{8}$$

**Feature 10 (Median of Durations: `d.median`)** *The median of the distributions of a melody is the value of* $\Delta T$ *that divides the frequency distribution of discrete duration values into two half with the same number of duration values (50%).*

**Feature 11 (Modal Duration: `d.mode`)** *The modal duration is the most frequent value of* $\Delta T$*. In case that there is no single most frequent value, the highest value of* $\Delta T$ *among the most frequent ones is chosen.*

**Feature 12 (Duration Entropy: `d.entropy`)** *Duration entropy is computed analogous to pitch and interval entropy, using* $\log_2 24$ *as the maximum entropy for normalisation given the upper phrase lengths limit of 24.*

$$f_i = \frac{F(\Delta T_i)}{\sum_n F(\Delta T_i)}$$

[10] Since typical interval distributions for tonal music show that larger intervals are much rarer than smaller intervals, picking the largest most frequent pitch interval is a reasonable strategy to arrive a discriminative and characteristic feature value.

$$\text{d.entropy} = -\frac{\sum_i f_i \cdot \log_2 f_i}{\log_2 24} \tag{9}$$

**Feature 13 (Equal Duration Transitions: `d.eq.trans`)** *This feature as well the the two subsequent features are derived from features proposed by Steinbeck (1982, p. 152f) and measure the relative frequency of note duration transitions. First, duration ratios between subsequent note durations as measured in tatums are computed and stored in a vector $R$:*

$$r_i = \frac{\Delta T_i}{\Delta T_{i+1}}$$

*Then, the relative number of subsequent duration ratio with equal value is counted while appropriate rounding is applied to the duration ratio values. The sum is the divided by the total number of duration ratios of subsequent note, i.e. the length of the vector $R$.*

$$\text{d.eq.trans} = \frac{\sum_{r_i=1} 1}{|R|} \tag{10}$$

**Feature 14 (Half Duration Transitions: `d.half.trans`)** *This features counts the number of note transitions where the first note is either twice as long or half as long as the second note, i.e. their duration ratio is either approximately 2 or 0.5.*

$$\text{d.half.trans} = \frac{\sum_{r_i=0.5 \vee r_i=2} 1}{|R|} \tag{11}$$

**Feature 15 (Dotted Duration Transitions: `d.dotted.trans`)** *This feature counts the number of dotted note transitions, i.e. the first note has either a duration that is three times as long as the second one or vice versa.*

$$\text{d.dotted.trans} = \frac{\sum_{r_i=\frac{1}{3} \vee r_i=3} 1}{|R|} \tag{12}$$

### 6.1.4 Global extension

**Feature 16 (Length: `len`)** *The length of a melody is the number of the notes it encompasses.*

**Feature 17 (Global Duration: `glob.duration`)** *The global duration of a melody is defined as the difference between the onset of the last note and the onset of the first note measured in milliseconds.*

$$\text{glob.duration} = t_n - t_1 \tag{13}$$

**Feature 18 (Note Density: `note.dens`)** *Note density is the number of notes per (milli)second.*

$$\text{note.dens} = \frac{\text{len}}{\text{glob.duration}} \tag{14}$$

### 6.1.5 Melodic Contour

**Feature 19 (Huron Contour: `h.contour`)** *Huron Contour is an implementation of the contour classification scheme proposed by Huron (1996). It is defined by the greater-than, equal, and less-than relations between the first pitch of a melody $p_1$, the mean of pitches $p_2, \ldots, p_{n-1}$, $\bar{p}$, and its last pitch $p_n$. The mean pitch is rounded to the nearest integer. The nine possible shapes are defined in table 2. The categorical values of this features are the verbal labels denoting the contour classes.*

| Pitch Relations | Contour Class |
|---|---|
| $p_1 < \bar{p} > p_n$ | Convex |
| $p_1 < \bar{p} = p_n$ | Ascending-Horizontal |
| $p_1 < \bar{p} < p_n$ | Ascending |
| $p_1 = \bar{p} = p_n$ | Horizontal |
| $p_1 = \bar{p} > p_n$ | Horizontal-Descending |
| $p_1 = \bar{p} < p_n$ | Horizontal-Ascending |
| $p_1 > \bar{p} = p_n$ | Descending-Horizontal |
| $p_1 > \bar{p} > p_n$ | Descending |
| $p_1 > \bar{p} < p_n$ | Concave |

Table 2: Relations between first pitch, mean pitch and last pitch of a melody and contour classes assigned according to Huron (1996)

**Step Contour** The next three features are derived from a representation of melodic contour that is conceived as a step curve drawn from the duration values ( sections on the x-axis) and pitch values (points on the y-axis). This representation is not reductive in as much it is possible to reconstruct the original melody from the contour step curve[11]. Step contour is widely used as a representation in many automatic melody analysis application, examples are Steinbeck (1982), Juhász (2000), Eerola and Toiviainen (2004). The step contour is computed in the following steps:

1. Normalise the duration values (measured in tatums) of all notes (to a norm of 4 bars of 4/4, assuming semi-quavers as tatum):

$$\widehat{\Delta T_i} = 64 \frac{\Delta T_i}{\sum_i \Delta T_i}$$

[11] With the exception of rests which are treated as extension of the duration of the previous note

2. Create a vector of length 64 by repeating each value of $p_i$ proportionally to its normalised duration $\widehat{\Delta T_i}$.

Melodic step contour is then represented as a vector of length 64 with its elements being samples at equally-spaced positions of the raw pitch values of the melody.

**Feature 20 (Step Contour Global Variation: `step.cont.glob.var`)** *This is defined as the standard deviation of the step contour vector* $\boldsymbol{x}$*:*

$$\text{step.cont.glob.var} = \sqrt{\frac{\sum_i (x_i - \bar{x})^2}{N-1}} \tag{15}$$

**Feature 21 (Step Contour Global Direction: `step.cont.glob.dir`)** *The step contour vector is correlated with the vector of bin numbers* ($n = 1, \ldots, 64$) *and we define the value of the Pearson-Bravais correlation coefficient to be the global direction of a step contour representation. The feature value has thus a value range from* $-1$ *to* $1$ *which can be interpreted as falling and rising contour shapes.*

**Feature 22 (Step Contour Local Variation: `step.cont.loc.var`)** *The local variation in a step contour representation is captured as mean absolute difference between adjacent values in the step contour vector* $\boldsymbol{x}$*:*

$$\text{step.cont.loc.var} = \frac{\sum_{i=1}^{N-1} |x_{i+1} - x_i|}{N-1} \tag{16}$$

**Interpolation Contour** The three following features are derived from a representation of melodic contour that is based on the idea of interpolating between the high and low points (i.e. contour turning points or contour extremum notes) of a melody using straight lines. This contour representation was formalised by Steinbeck (1982) and termed *Polygonzug* (= frequency polygon). Müllensiefen and Frieler (2004) discuss this representation under the term *Contourization*. The idea of the definition of interpolation contour given here is to substitute the pitch values of a melody with the sequence gradients that represent the direction and steepness of the melodic motion at evenly spaced points in time. An *interpolation contour* representation is obtained from the raw onset in milliseconds and pitch vaules by the following steps:

1. Determine all contour extremum notes. The contour extremum notes are the first note $n_1$, the last note $n_N$, and of every note $n_i$ inbetween, where $n_{i-1}$ and $n_{i+1}$ are either both greater or both lower than $n_i$, or in the cases where $n_{i-1}$ or $n_{i+1}$ are equal to $n_i$ but $n_{i-2}$ and $n_{i+1}$ or $n_{i-1}$ or $n_{i+2}$ are either both greater or both lower than $n_i$.

2. As pure changing notes (notae cambiatae) generally do not make perceptually contour extrema, the changing notes are excluded from the set of potential contour extrema. A changing note $n_i$ is a note where the pitches of $n_{i-1}$ and $n_{i+1}$ or are equal. The changing notes are deleted from the set of contour extrema. This is a variant from interpolation contour definition given by Müllensiefen and Frieler (2004).

3. Calculate the gradients of the lines between two subsequent contour extremum notes $n_i = (t_i, p_i)$ and $n_j = (t_j, p_j)$ $(j > i)$ by $m = \frac{p_j - p_i}{t_j - t_i}$

4. Calculate the duration for each line between subsequent contour extremum points by $\Delta t_i = t_j - t_i$.

5. Obtain an integer value representing each duration by integer.duration = round$(10\Delta t_i)$. Thus, any duration below 50 milliseconds is not any longer represented after this rounding step.

6. Create a vector of the gradients where each gradient value is repeated corresponding to its `integer.duration`. The length of this `weighted.gradients` vector is the sum of the `integer.duration` vector.

The interpolation contour representation is a vector of varying length containing that the gradient values of the interpolation lines. The relative length of each interpolation line is represented by the number of times its gradient value is repeated.

**Feature 23 (Interpolation Contour Global Direction: `int.cont.glob.dir`)** *This is the overall direction of the interpolation contour and takes the values* 1 *(up),* 0 *(flat), or* $-1$ *(down).*

$$\text{int.cont.glob.dir} = \text{sgn}(\sum_i x_i) \tag{17}$$

**Feature 24 (Interpolation Contour Mean Gradient: `int.cont.grad.mean`)** *The mean of the absolute gradient values informs about the degree of inclination at which the interpolation contour is rising or falling on average.*

$$\text{int.cont.grad.mean} = \frac{\sum_i^N |x_i|}{N} \tag{18}$$

**Feature 25 (Interpolation Contour Gradients Std. Dev.: `int.cont.grad.std`)** *This is defined as the standard deviation of the interpolation contour vector* $\boldsymbol{x}$*:*

$$\text{int.cont.grad.std} = \sqrt{\frac{\sum_i (x_i - \bar{x})^2}{N - 1}} \tag{19}$$

**Feature 26 (Interpolation Contour Direction Changes: `int.cont.dir.changes`)** *This feature measures the number of changes in contour direction relative to the number of interpolation lines (i.e. number of different gradient values).*

$$\text{int.cont.dir.changes} = \frac{\sum\limits_{\text{sgn}(x_i) \neq \text{sgn}(x_{i+1})} 1}{\sum\limits_{x_i \neq x_{i+1}} 1} \tag{20}$$

**Feature 27 (Interpolation Contour Class: `int.contour.class`)** *For this feature the gradients of the interpolation contour vector are transformed into four symbols and the resulting letter string is interpreted as the Interpolation Contour Class. The feature is computed in the following steps:*

1. *The interpolation contour vector (containing the gradient values) is sampled at four equally spaced points. The resulting vector with length 4 is a very compact representation of the contour of a melody. It represents only the major up- and downward movements while all minor contour movements are filtered out by this downsampling.*

2. *Normalise the value range of the the interpolation gradients to a norm where the value of* 1 *corresponds to pitch change of a semitone over the time interval of a quaver at 120bpm (i.e. 250ms). Since the basic units of pitch and time representation are 1 second and 1 semitone, the normalisation is achieved simply by dividing the vector of gradients by four:* $\text{norm.gradients} = (\frac{1}{4} \cdot \mathbf{x})$

3. *Classify the normalised gradient values into five different classes:*

$$\texttt{num.gradient.class} = \begin{cases} -2 \text{ (strong down)} & : \text{ norm.grad} \leq -1.45 \\ -1 \text{ (down)} & : -1.45 < \text{norm.grad} \leq -0.45 \\ 0 \text{ (flat)} & : -0.45 < \text{norm.grad} < 0.45 \\ 1 \text{ (up)} & : 0.45 \leq \text{norm.grad} < 1.45 \\ 2 \text{ (strong up)} & : 1.45 \leq \text{norm.grad} \end{cases}$$

4. *For better readability, convert numerical gradient symbols to letters with letter a being assigned to gradient value* $-2$ *(and letter c being gradient value* $0$*).*

*Thus, the value of the interpolation contour class is a string of four letters.*[12]

**Polynomial Contour** The next feature is derived from the representation of melodic contour as a polynomial curve. The concept, motivations, technical details, and potential usages of the polynomial contour representation is discussed in detail by Müllensiefen and Wiggins (2009). A polynomial contour representation is computed from onset $t_i$ and pitch $p_i$ values of the notes of a melody in three steps:

[12] The resolution of this feature is determined by two main factors: The length of the vector of interpolation gradients (currently 4) and the classification of the gradient values into different classes (currently 5). These values are inspired by Huron's contour idea but aim at a resolution that is about twice as fine: Instead of two contour lines (possible "V"-shapes etc.) interpolation contour has four (possible "W"-shapes etc.) and instead of one class for positive and negative gradients (up and down movements) it has two. On the contrary, the Interpolation Contour Global Direction feature can be seen analogous to Huron Contour but having a *lower* resolution. While Huron Contour has nine classes that are theoretically possible, interpolation contour can assume $5^4 = 625$ different class values (and Interpolation Contour Global Direction only 3). The two resolution parameters haven been chosen on theoretic grounds, but the resolution of interpolation contour could be determined from a large corpus of contour data in a future step, e.g. using entropy or minimum description length discretization.

1. **Centre around origin on time axis**: First all onset values are shifted on the time axis such that the onset of the first and the last note are symmetrical with respect to the origin. The motivation for centering is the assumption that melodic phrases often exhibit a certain symmetry over time (e.g. rise and fall or fall and rise). The centering is done according to:

$$t'_i = t_i - (t_1 + \frac{t_n - t_1}{2})$$

2. **Fit a full polynomial model**: Define all full polynomial model by:

$$p = c_0 + c_1 t + c_2 t^2 + \ldots + c_m t^m$$

   where $m = \lfloor \frac{n}{2} \rfloor$. To obtain the parameters $c_i$ use least squares regression and treat the exponential transformations $t, t^2, \ldots, t^m$ of the vector of onsets $t$ as predictor variables and the vector of pitch vaules $p$ as the response variable.

3. **Model selection**: Least squares regression generally overfits the data of the response variable and therefore use the *Bayes' Information Criterion* (BIC) as a model selection procedure that balances the fit to the response variable against the complexity of the model. The procedure is applied in a step-wise backwards fashion and returns a model containing only those time components that make a significant contribution in the prediction of the pitch data. The coefficients of the selected time components represent the full polynomial contour curve, the coefficients of non-selected components are set to 0.

**Feature 28 (Polyn. Cont. Coefficients: `poly.coeff1, poly.coeff2, poly.coeff3`)** *The number of non-zero coefficients can vary considerably for different contour curves, but at the same time it is necessary for subsequent usage to define a features with a set number of preferably only few dimensions. Therefore, only coefficients $c_1, c_2, c_3$ are retained as the numerical values of this 3-dimensional feature from the polynomial model of a melodic contour curve. These coefficients are believed to capture the major variations in polynomial contour shape.*

### 6.1.6 Implicit Tonality

The features in this section are derived from a representation of the tonalities that are implied by a melody. The Krumhansl-Schmuckler algorithm (Krumhansl, 1990) is used to compute a `tonality.vector` of length 24 where each vector element is the Pearson-Bravais correlation between one of the 24 major and minor keys and the analysed melody. The Krumhansl-Kessler profiles (Krumhansl and Kessler, 1982) for major an minor keys (`maj.vector` and `min.vector`) are given as global parameters to the function `compute.tonality.vector` and could be easily swapped e.g. for Temperley's binary vectors (Temperley, 2001).

**Feature 29 (Tonalness: `tonalness`)** *Tonalness is defined as the magnitude of the highest correlation value in the* `tonality.vector`*. It expresses how strongly a melody correlates to a single key.*

**Feature 30 (Tonal Clarity: `tonal.clarity`)** *This features is inspired by Temperley's notion of tonal clarity (Temperley, 2007) and is defined as the ratio between the magnitude of the highest correlation in the* `tonality.vector` $A_0$ *and the second highest correlation* $A_1$*:*

$$\text{tonal.clarity} = \frac{A_0}{A_1} \tag{21}$$

**Feature 31 (Tonal Spike: `tonal.spike`)** *Similar to* `tonal.clarity`*,* `tonal.spike` *depends on the magnitude of the highest correlation but in contrast to the previous feature is divided by the sum of all correlation values* $> 0$*:*

$$\text{tonal.spike} = \frac{A_0}{\sum_{A_i > 0} A_i} \tag{22}$$

**Feature 32 (Mode: `mode`)** `Mode` *is defined as the mode of the tonality with the highest correlation in the* `tonality.vector`*. It can assume the values* `major` *and* `minor`*.*

## 6.2 m-type Summary Statistics

The features in 6.1 summarise the melodic content of a phrase (or a whole melody), with many of them not paying attention to the order in which the notes of the phrase appear (the melodic contour features are an exception of course). This means that a phrase and its retrograde would receive the same feature value. But it has been shown on several occasions (e.g. Dowling, 1972) that note order is a very decisive factor for perceived melodic content. The basis for the construction of the features described in this section is that they pay attention to note order. These features have two conceptual roots:

**Creating *m-types*** A moving window is slid over the notes of a melody and the content of each window is recorded. This idea has been developed by Downie (2003), Uitdenbogerd (2002), Müllensiefen and Frieler (2004), where these authors have called the short melodic substrings *n-grams*. We shall call these short melodic substrings *m-tokens* and the set of different m-tokens in a melody is called the *m-types* of a melody. These terms make reference to the technical terms *token* and *type* from linguistics denoting the set of all words in a text and the set of all distinct verbal terms in a text. Types can be conceptually compared to entries in a dictionary.

**Computing m-type summary statistics** In computational linguistics several features have been proposed to describe the usage of types within a text based on their frequency distribution. We compute these features to summarise the distribution of m-types within a melody.

### 6.2.1 Creating m-types

The function `n.grams.from.melody.main` creates m-tokens and counts the frequency of m-types. It returns a frequency table of all the m-types in a melody. This is achieved in a number of stages. The important arguments to this function are the lower and upper limit of the size of the moving window in numbers of note adjacent note pairs. In other words, m-tokens and m-types of varying length $n$ can be requested. M-types of length $n = 1$ represent the pitch interval and duration ratio only between two adjacent notes. M-types of length $n = 3$ represent the intervals and duration ratios between in a substring of 4 notes. The current default limits are $n \in \{1, \ldots, 5\}$ and are given as a global variable. There are $N-n+1$ m-tokens of length $n$ in a melody with $N$ notes. The number of m-types in a melody depends on its repetitiveness. Maximally it can be equal to the number of m-tokens and minimally it is 1.

1. The melody is segmented into phrases using the phrase information provided in the MCSV file.

2. For each phrase that is within the phrase length limits, pitch intervals are computed from adjacent raw pitch values and duration ratios are computed from adjacent raw inter-onset intervals in milliseconds.

    - Pitch intervals are classified into 19 interval classes according to the classification scheme in the appendix. The scheme classifies intervals that could be diatonically altered (e.g. ascending major and minor seconds or descending major and minor thirds) into the same interval class and has collective classes for upwards and downwards intervals larger than an octave. Pitch interval classes are denoted by a two-digit string.
    - Duration ratios are classified into 3 different classes, *shorter*, *equal*, and *longer*, according to the classification scheme in the appendix. The duration ratio classification scheme is based on empirical perceptual limits describing from experiments on the similarity of adjacent tone durations (Sadakata et al., 2006). Duration ratio classes are denoted by a single letter string.

3. For each pair of adjacent notes in a phrase, the interval class and duration ratio class strings are hashed into one string of 3 digits.

4. For each length $n$, a window of length $n$ is slid over the string of 3-digit hash symbols and the frequency of each substring of hash symbols, the m-type, is counted.

5. The frequency counts for each m-type are summed up over all phrases of the melody.

The result of this procedure is a table with three columns that contains the m-type as as a letter string with "_" as separator between subsequent hash symbols, the frequency count of the m-type in the melody and $n$, i.e. the length of the m-type.

### 6.2.2 Computing m-type summary statistics

The distribution of the m-type frequency counts is generally very different from the frequency distribution of other categorical features, such as Huron Contour Class or Interpolation Contour Class, explained above. As with the frequency distributions of words in written text, there are usually very many m-types in a melody that occur only once and very few that occur very often. The following features take the special frequency distribution of the m-types into account and summarise it in different ways. Thus, they measure the repetitiveness of m-types. Most of the following features are taken from publications by Harald Baayen (Baayen, 1992; 2001). To facilitate the understanding of the following features we introduce the following concepts and notation (Baayen, 2001, p. 2-12):

$n$ : The length of m-tokens and m-types. The current defaults are $n \in \{1, \ldots, 5\}$.

$|n|$ : The number of different lengths values used where $|n| = 5$ by default.

$\tau_i$ : The $i$-th m-type of a set of m-types in a melody.

$N$ : The number of m-tokens in a melody.

$f(i, N)$ : The frequency of m-type $\tau_i$ in a melody with $N$ m-tokens.

$V(N)$ : The number of m-types in a melody with $N$ m-tokens. This can be interpreted as the size of the m-type vocabulary of the melody.

$m$ : The index for the frequency class in the frequency distribution of the m-types.

$V(m, N)$ : The number of m-types with frequency $m$ in a melody with $N$ m-tokens: $V(m, N) = \sum_{i=1}^{V(N)} I_{f(i,N)=m}$ where the identity operator takes the value of 1 if the index condition is satisfied and a value of 0 otherwise.

**Feature 33 (Mean m-type entropy: `mean.entropy`)** *For each m-type length n the entropy of the m-type distribution is calculated analogous to equation 3 and then divided by the maximal entropy for this length, i.e.* $\log_2 N$*, where* $N$ *is the number of m-tokens in the melody. Then, the mean is taken over these relative entropy values of all lengths:*

$$\text{mean.entropy} = \frac{\sum_n H_r(n)}{|n|} \tag{23}$$

**Feature 34 (Mean Productivity: `mean.productivity`)** *This is the mean over all lengths of the number of m-types only occurring once divided by the number of m-tokens. In linguistics the words only occurring once in a text is know as hapax legommena.*

$$\text{mean.productivity} = \frac{\sum_n \frac{V(1,N)}{N}}{|n|} \tag{24}$$

**Feature 35 (Yules's k: `mean.Yules.K`)** *Yule proposed this feature in 1944 to measure the rate which words are repeated in a text, see (Baayen, 2001, p. 25) for a discussion.*

$$\text{mean.Yules.K} = \frac{1}{|n|} \cdot 1000 \cdot \frac{(\sum_m m^2 V(m,N)) - N}{N^2} \tag{25}$$

**Feature 36 (Simpsons's D: `mean.Simpsons.D`)** *Simpson's D is was conceived to measure word repetition rate as well and is mathematically closely related to Yule's k.*

$$\text{mean.Simpsons.D} = \frac{1}{|n|} \sum_m V(m,N) \cdot \frac{m}{N} \cdot \frac{m-1}{N-1} \tag{26}$$

**Feature 37 (Sichel's S: `mean.Sichels.S`)** *Sichel's S is based on the empirical observation that the proportion of dis legommena (the words that occur twice) with respect to the vocabulary size is often constant throughout a text. It is defined by:*

$$\text{mean.Sichels.S} = \frac{1}{|n|} \cdot \frac{V(2,N)}{V(N)} \tag{27}$$

**Feature 38 (Honoré's H: `mean.Honores.H`)** *This feature is based on the assumption that the proportion of m-types only occurring once (hapax legommena) is a logarithmic function of the number of m-tokens.*

$$\text{mean.Honores.H} = \frac{1}{|n|} \cdot 100 \cdot \frac{\log N}{1.01 - \frac{V(1,N)}{V(N)}} \tag{28}$$

# 7 Corpus-based Feature Statistics

There is substantial support - both theoretical and empirical - in the literature for the assertion that the frequencies with which feature values occur in a domain might be an important determinant for how these features are cognitively processed.[13] This section describes how the two main functions derive frequency values from melody features with respect to corpus of melodic phrases.

## 7.1 Frequencies of Summary Features

For the features summarising the content of a melody or a melodic phrase 6.1 obtaining frequency-related information for a particular melody and from a corpus is straightforward. We replace the feature values with the relative frequency or frequency density with which they occur in the corpus. We, thus, eliminate the specific information about how large the

[13] Essentially, David Huron's latest book (2006) is all about how statistical properties of music influence music cognition. Looking outside music research and, for example, turning to psychological memory research, the importance of word frequency for verbal memory and and cognitive text processing has been found in a large number of experiments.

pitch range of a melody is or how varied the distribution of rhythmic duration classes is, for example. But we replace with information about how common or unusual the values of these features are with respect to a melody corpus. From a data-analytic point of view the feature value summary statistics described above 6.1 fall principally into three different classes: Numerical features that measure a melodic feature on a continuous scale, numerical features that measure a melodic feature on a discrete scale with not too many different values, and categorical features which assign category labels as feature values to a melody or melodic phrase. The estimation of frequency information is different for these three cases. In the former case frequency densities are estimated and in the two latter cases relative frequencies are computed. There is no clear-cut criterion for distinguishing between continuous and discrete features and the categorisation is mainly based on how the feature is constructed. If the number of different possible feature values is a reasonably direct function of the number of notes of the melody, then the feature is classified as discrete (e.g. `p.range`, `len`, `d.mode`). However, if the possible feature values can exceed the number of notes of a melody by orders of magnitude then the feature is classified as continuous.

### 7.1.1 Frequency Densities for numerical continuous features

The estimation of frequency densities for numerical features[14] is done in two steps:

1. z-standardisation: The values of each feature $x$ of the melodies of the corpus are standardised by subtracting each value from the mean of the feature values and dividing it by the standard deviation of the feature values:

$$z(x) = \frac{x - \bar{x}}{\sqrt{\text{var}(x)}}$$

2. Density estimation: R-function `density()` from the package `stats` is applied to the standardised distribution of the values of a feature in the corpus. The function applies a gaussian kernel to the frequency counts for all values of $x$. The bandwidth of the kernel is chosen according to Silverman's 'rule of thumb' (Silverman, 1986). Then, the density of $x$ is estimated at 512 (default value) equally spaced points over the range of $x$. As a result a table is returned that contains a vector $x'$ the 512 values over the range of $x$ where densities are estimated and the corresponding density estimates $y$. If the argument `write.out.corp.freq` is `TRUE` this table of feature value densities is saved as a file.

3. Replacing feature values with densities: For each original value $x_i$ the corresponding value $x'_i$ that has minimal distance to $x_i$ is determined and its corresponding $y_i$ value is looked up. The value of $y_i$ is then used to replace the original value of $x_i$.

[14] Actually, densities are only computed for one-dimensional features because of the additional conceptual complexity and the high computational resources needed to estimate densities for features with two or more dimensions. The only multi-dimensional feature that is currently implemented, `poly.coeff1, poly.coeff2, poly,coeff3`, is therefore excluded from density estimation at this point.

As a result a table is returned where all values $x_i$ are replaced by values $y_i$. The set of continuous numerical features is `p.entropy, p.std, i.abs.mean, i.abs.std, i.entropy, d.range, d.median, d.entropy, d.eq.trans, d.half.trans, d.dotted.trans, glob.duration, note.dens, tonalness, tonal.clarity, tonal.spike, int.cont.grad.mean, int.cont.grad.std, step.cont.glob.var, step.cont.glob.dir, step.cont.loc.var`.

### 7.1.2 Relative frequencies for categorical features

The computation of relative frequencies for categorical features is done simply by dividing the number of counts in each category by the sum of counts of all categories (i.e. the number of melodies or melodic phrases).

1. Relative frequency computation:

$$f(x_i) = \frac{\text{count}(x_i)}{\sum_i \text{count}(x_i)}$$

   As a result a table is returned that contains the categorical feature values and their relative frequencies. If the argument `write.out.corp.freq` is `TRUE` this table of relative feature value frequencies value is saved as a file together with the table of densities of numerical features.

2. Replacing feature values with relative frequencies: Each categorical value of $x$ is replaced by its relative frequency.

As a result a table is returned where all category labels $x_i$ are replaced by their relative frequencies $y_i$. The set of categorical features is `mode, h.contour, int.contour.class`.

### 7.1.3 Relative frequencies for numerical discrete features

The computation of relative frequencies for discrete features is based a mixture between the procedure for categorical and continuous features.

1. Relative frequency computation:

$$f(x_i) = \frac{\text{count}(x_i)}{\sum_i \text{count}(x_i)}$$

   As a result a table is returned that contains the numerical feature values and their relative frequencies.

2. Replacing feature values with densities: For each original value $x_i$ the corresponding value $x_i'$ that has minimal distance to $x_i$. The value of $y_i$ is then used to replaced $x_i$.

As a result a table is returned where all numerical values $x_i$ are replaced by their relative frequencies $y_i$. The set of discrete numerical features is `p.range, i.abs.range, i.mode, d.mode, len, int.cont.glob.dir, int.cont.dir.change`.

## 7.2 Features derived from m-type distributions in melody and corpus

The distribution of the summary features and the m-types are fundamentally different. While summary features are generally assumed to be distributed according to a Gaussian distribution (or a mixture of several Gaussians), m-types are assumed to be distributed according to a powerlaw, very similar to word frequency distributions in written texts. Deriving information from these two kinds of distribution is therefore an inherently different procedure. There are several ways in which the distributions of m-types in a melody with respect to a corpus can be condensed into single-value features.

### 7.2.1 Comparisons between m-type distributions from melody and corpus

A very straightforward way to determine whether m-types are used in an *unusual* way in a melody is to compare their frequency of occurrence in the melody with their frequency of occurrence in the corpus. Similarity measures such as correlations, differencing, and vector multiplication are used for comparison. We denote the frequency count of m-type $\tau$ in a melody $m$ by $\mathrm{TF}_\tau$ (derived from *term frequency* in information retrieval), and we use $\mathrm{DF}_\tau$ (from *Document Frequency*) for the number of melodies in the corpus containing $\tau$.

**Feature 39 (Spearman corr. of term and doc. frequencies: `mtcf.TFDF.spearman`)** *For this feature the ranks of* $\mathrm{TF}_{\tau \in m}$ *and* $\mathrm{DF}_{\tau \in m}$ *are correlated using Spearman's rank correlation. For ties in the values (frequency counts) in TF and DF the minimum rank is assigned to all ranks in the same set of ties. The value is range from* $-1$ *to* $1$*.*

**Feature 40 (Kendall's $\tau$ of term and document frequencies: `mtcf.TFDF.kendall`)** *This feature is similar to* `mtcf.TFDF.spearman` *but uses the well-known correlation method known as Kendall's* $\tau$*. For ties in the values (frequency counts) in TF and DF the minimum rank is assigned to all ranks in the same set of ties. The value is range from* $-1$ *to* $1$*.*

**Feature 41 (Mean dot product of term and doc. frequencies: `mean.log.TFDF`)** *The rationale behind this feature is to use the dot product to measure the similarity between the vector of the term frequencies and the vector of the document frequencies for all m-types* $\tau$ *contained in a melody. Assuming that m-type frequencies, like word frequencies in texts, are approximately distributed according to an exponential distribution, both vectors are first transformed logarithmically and then standardised:*

$$\mathrm{TF}'_{\tau_i \in m} = \frac{\log_2(\mathrm{TF}_{\tau_i \in m})}{\sum_i \log_2(\mathrm{TF}_{\tau_i \in m})}$$

*where the same transformations are applied to* $\mathrm{DF}_{\tau \in m}$*. We then take the mean of the vector resulting from vector multiplication:*

$$\text{mtcf.mean.log.TFDF} = \frac{\mathrm{TF}'_{\tau \in m} \cdot \mathrm{DF}'_{\tau \in m}}{|\tau \in m|} \tag{29}$$

**Feature 42 (Normalised distance of term and doc. frequencies: `norm.log.dist`)** *The same logarithmic and standardisation transformations used for* ***`mean.log.TFDF`*** *are applied as a first stage for this feature. Then, instead of multiplying the elements of both vectors their difference is taken and averaged:*

$$\text{mtcf.norm.log.dist} = \frac{\sum_{\tau_i \in m} |TF'_{\tau_i} - DF'_{\tau_i}|}{|TF_{\tau \in m}|} \tag{30}$$

### 7.2.2 Features derived from m-type corpus frequencies

These features are the closest equivalent to word frequencies derived from a corpus in linguistic or psycho-linguistic applications and are intended to reflect how common or rare the m-types are that are found in an analysis melody.

**Feature 43 (Maximum document frequency: `mtcf.max.log.DF`)** *This is simply the logarithm of the m-type contained in the analysis that also occurs in a maximum number of melodies in the corpus.*

$$\text{max.log.DF} = \log_2(\max(DF_{\tau \in m})) \tag{31}$$

**Feature 44 (Minimum document frequency: `mtcf.min.log.DF`)** *Conversely to* `mtcf.max.log.DF` *this feature measures how rare m-type with the least occurrences in the corpus is.*

$$\text{min.log.DF} = \log_2(\min(DF_{\tau \in m})) \tag{32}$$

**Feature 45 (Mean document frequency: `mtcf.mean.log.DF`)** *This feature reflects how common or rare the m-types contained in a melody are on average:*

$$\text{mean.log.DF} = \frac{\sum DF_{\tau \in m}}{|DF_{\tau \in m}|} \tag{33}$$

**Feature 46 (Mean document frequency entropy: `mtcf.mean.entropy`)** *This feature is very similar to* `mean.entropy`*, but instead of the frequencies of the m-types in the analysis melody $f_c(\tau_i)$ it uses the frequencies of the in the corpus, $f_C(\tau_i)$. Just like for* `mean.entropy`*, the entropy values are averaged over the various m-types lengths.*

**Feature 47 (Mean document frequency productivity: `mtcf.mean.productivity`)** *This feature is very similar to* `mean.productivity`*, but instead of the frequencies of the m-types in the analysis melody $f_c(\tau_i)$ it uses the frequencies of the in the corpus, $f_C(\tau_i)$. Just like for* `mean.productivity`*, the productivity values are averaged over the various m-types lengths.*

**Feature 48 (Mean document frequency Yule's K: `mtcf.mean.Yules.K`)** *This feature is very similar to* `mean.Yules.K`*, but instead of the frequencies of the m-types in the analysis melody $f_c(\tau_i)$ it uses the frequencies of the in the corpus, $f_C(\tau_i)$. Just like for* `mean.Yules.K`*, the K values are averaged over the various m-types lengths.*

**Feature 49 (Mean document frequency Simpson's D: `mtcf.mean.Simpsons.D`)** *This feature is very similar to* `mean.Simpsons.D`*, but instead of the frequencies of the m-types in the analysis melody $f_c(\tau_i)$ it uses the frequencies of the in the corpus, $f_C(\tau_i)$. Just like for* `mean.Simpsons.D`*, the D values are averaged over the various m-types lengths.*

**Feature 50 (Mean document frequency Sichel's S: `mtcf.mean.Sichels.S`)** *This feature is very similar to* `mean.Sichels.S`*, but instead of the frequencies of the m-types in the analysis melody $f_c(\tau_i)$ it uses the frequencies of the in the corpus, $f_C(\tau_i)$. Just like for* `mean.Sichels.S`*, the S values are averaged over the various m-types lengths.*

**Feature 51 (Mean document frequency Honoré's H: `mtcf.mean.Honores.H`)** *This feature is very similar to* `mean.Honores.H`*, but instead of the frequencies of the m-types in the analysis melody $f_c(\tau_i)$ it uses the frequencies of the in the corpus, $f_C(\tau_i)$. Just like for* `mean.Honores.H`*, the H values are averaged over the various m-types lengths.*

### 7.2.3 Features derived from m-type melody frequencies and inverted m-type corpus frequencies

A simple and very wide-spread scheme for weighting words in text retrieval is to multiply the frequency with which a type or term occurs in a text with the inverse of the frequency with which it occurs in a text corpus. Types that are rare in general but frequent in a given document thus get a high weight. The analogue can be constructed for m-types in melodies. With reference to text retrieval we shall use the here$\mathrm{IDF}_C(\tau)$ to denote the inverse document frequency of the m-type $\tau$ in melody corpus $C$:

$$\mathrm{IDF}_C(\tau) = \frac{|C|}{|c \in C : \tau \in c|}$$

With $\mathrm{TF}_c(\tau)$ we denote here what we called above $f(\tau, N)$, i.e. the frequency of m-type $\tau$ in melody $c$(with $N$ tokens which is irrelevant here). The final TF-IDF weight for $\tau$ ist derived by multiplication of term and inverted document frequency, $TF_c(\tau) \times IDF_C(\tau)$.

**Feature 52 (Mean TF-IDF entropy: `mtcf.TFIDF.m.entropy`)** *This feature is very similar to* ***`mean.entropy`****, but instead of the frequencies of the m-types in the analysis melody $f_c(\tau_i)$ it uses the TF-IDF weights. Just like for* `mean.entropy`*, the entropy values are averaged over the various m-types lengths.*

**Feature 53 (Mean TF-IDF Yule's K: `mtcf.TFIDF.m.K`)** *This feature is very similar to* `mean.Yules.K`*, but instead of the frequencies of the m-types in the analysis melody $f_c(\tau_i)$ it uses the TF-IDF weights. Just like for* `mean.Yules.K`*, the K values are averaged over the various m-types lengths.*

**Feature 54 (Mean TF-IDF Sichel's S: `mtcf.TFIDF.m.S`)** *This feature is very similar to* `mean.Sichels.S`*, but instead of the frequencies of the m-types in the analysis melody $f_c(\tau_i)$ it uses the TF-IDF weights. Just like for* `mean.Sichels.S`*, the S values are averaged over the various m-types lengths.*

### 7.2.4 Features derived from entropy-based weightings

In modern text retrieval and psycholinguistics a considerable amount of attention has been given to weighting schemes that are based on word frequencies and should reflect the importance, weight, or specificity of a word to a particular context (section or paragraph of a written text). We use the weighting scheme suggested for use with the *Latent Semantic Analysis* algorithm in Quesada (2007, p. 80-81). Similar to the TFDF-based features above the entropy-based weighting consists of local weight and a global weight. The local weight of an m-type $\tau$ is the logarithm of its frequency in the analysis melody +1:

$$\text{loc.w}(\tau) = \log_2(f(\tau) + 1)$$

The global weight of an m-type is based on the ratio $P_\tau$, i.e. the ratio of its its local frequency in each melody $c$ of the corpus to the overall frequency in the corpus $C$:

$$P_c(\tau) = \frac{f_c(\tau)}{f_C(\tau)}$$

The global weight is then calculated as follows:

$$\text{glob.w} = 1 + \frac{\sum_{c \in C} P_c(\tau) \cdot \log_2(P_c(\tau))}{\log_2(|C|)}$$

The final weight for m-type $\tau$ is then obtained by

$$\text{glob.loc.w}(\tau) = \text{loc.w}(\tau) \cdot \text{glob.w}(\tau)$$

**Feature 55 (Mean global weight: `mtcf.mean.g.weight`)** *This is simply the mean of the global weights (`glob.w`) of all m-types $\tau$ in the analysis melody.*

**Feature 56 (Standard deviation of global weights: `mtcf.std.g.weight`)** *Accordingly, `mtcf.std.g.weight` is an indicator of the variability of the global weights the m-types $\tau$ in the analysis melody.*

**Feature 57 (Mean global-local weight: `mtcf.mean.gl.weight`)** *This is simply the mean of the combined global-local weights (`glob.loc.w`) of all m-types $\tau$ in the analysis melody.*

**Feature 58 (Standard deviation of global-local weights: `mtcf.std.gl.weight`)** *Accordingly, `mtcf.std.gl.weight` is an indicator of the variability of the combined global-local weights (`glob.loc.w`) of all m-types $\tau$ in the analysis melody.*

# 8 Similarity computation based on features and feature distributions

The function `feature.similarity()` offers several ways to compute the similarity between melodies based on summary features as computed by `compute.features()` and their frequency distributions as computed by `corpus.based.feature.frequencies()`.[15] The computation of similarities is carried out in two steps:

1. Compute features for all input melodies by calling the function `compute.features()`. The value of the argument `use.segmentation` is passed on to `compute.features()` and the function is always called with `output="melody.wise"`.

2. Similarities between melodies are computed based on their values for each feature and according to one of three different methods (see below). If argument `average=FALSE`, a list of separate matrices, one for each feature, is outputted, otherwise the output list contains only 1 matrix where similarity values are averaged over the features specified.

## 8.1 Methods for computing similarities from features

Three different methods can be used to compute similarities from features. They all output similarity values between 0 (minimal similarity) and 1 (maximal similarity / identity) for every pair of melodies. The similarities computed here are currently always symmetric, meaning that $\sigma(m_i, m_j) = \sigma(m_j, m_i)$, i.e. the order in which two melodies $m_i$ and $m_j$ are compared does not matter.[16]

### 8.1.1 Euclidean Distance / Similarity

Euclidean distance is a well-known distance measure and is only suitable for determining the distance between melodies $m_i$ and $m_j$ on the basis of numerical features $k_i$. It is defined as

$$d_{eucl}(m_i, m_j) = \sqrt{\sum_K (x_{ik} - x_{jk})^2}$$

where $x_{ik}$ is the value of melody $m_i$ on feature $k$ and $x_{jk}$ is the value of melody $m_j$ on feature $k$. This distance value is then divided by the number of features $K$ and in order to arrive at a similarity value from the interval $[0, 1]$ an exponential transform is used:

$$\sigma_{eucl}(m_i, m_j) = e^{-\frac{d_{eucl}(m_i, m_j)}{K}} \quad \in [0, 1]$$

[15] While similarity computations based on m-types have already been applied quite successfully to real-world problems (see Müllensiefen and Pendzich, 2009), `feature.similarity()` doesn't currently carry out similarity computations based on m-types or m-type features.

[16] Asymmetric measures that performed well in a previous applications (Müllensiefen and Pendzich, 2009) are to be integrated at some later stage.

If the argument `eucl.stand` is `TRUE`, then all feature values $x_{i,k}$ are z-standardised by

$$z(x_{i,k}) = \frac{x_{i,k} - \bar{x}_k}{\text{var}(x_k)}$$

before computing the euclidean distance. The standardisation is recommended when features with different value ranges are combined into one similarity measure.

### 8.1.2 Similarity based on Gower's coefficient

Gower's *general coefficient of similarity* (Gower, 1971) has become a standard way of combining similarity measurements from variables of different measurement scales (binary (dichotomous), categorical (qualitative), and numerical (quantitative) variables). For categorical variables Gower's coefficient assigns a value of 1 when both objects have are from the same class and 0 if they are from different classes. For a numerical variable k Gower defines the similarity between objects (melodies) $m_i$ and $m_j$ by

$$\sigma_{Gower,k}(m_i, m_j) = 1 - \frac{|x_i - x_j|}{R_k}$$

In contrast to the euclidean distance, the standardisation of the distance is done here by dividing by the range $R_k$ of feature $k$ (and not by its variance). Similarity values are then averaged over the $K$ features specified. Details of this similarity coefficient can be found in Gower (1971). Since Gower's coefficient always standardises distance/similarity values by the range of the feature values, it gives misleading results when only few melodies are to be compared. In the case of just two input melodies their similarity is necessarily 0 because the distance of their values on any (numerical) feature equals the range of that feature variable. It is therefore recommended to use Gower similarity only with larger sets of melodies as input.

### 8.1.3 Corpus-based similarity

This is a novel approach to similarity computation that makes use of the musical background information as is manifest in the frequency distribution of numerical features in a corpus. The core idea is to replace the actual difference between the feature values for two melodies by the difference in the cumulative sums of melodies from a corpus at these feature values. Thus, a difference of 0.2 arising from feature values $x_i = 0.6$ and $x_j = 0.4$ can mean something else than a difference of 0.2 obtained from $x_i = 0.9$ and $x_j = 0.7$ if the bulk of melodies from a corpus scores values between, say 0.3 and 0.7 on that feature. In that case, the difference between 0.9 and 0.7 seems to indicate a higher similarity because high feature values happen less frequently than feature values broadly around 0.5. The similarity between two melodies $m_i$ and $m_j$ on a numerical feature $k$ in the context of a corpus $C$ is thus defined as

$$\sigma_{corpus,k,C}(m_i, m_j) = |\sum_{l=1}^{l=x_i} f_C(x_l) - \sum_{l=1}^{l=x_j} f_C(x_l)|$$

where $l$ is the index of the discrete frequency distribution from corpus $C$ and $\sum_{l=1}^{l=x_i} f_C(x_l)$ is the cumulative frequency count of the distribution between the first bin $l = 1$ and the bin at feature value $x_i$.

For categorical features corpus-based similarity equals Gower similarity, giving 1 to melodies having the same class and 0 having different classes. If `average=TRUE` similarity values from different features are combined by a weighted average. The weighting is done according to the entropy of the frequency distribution of each feature. Features with higher entropy are given a higher weight.

# Acknowledgements

This research was funded by EPSRC grant EP/D0388551. The author likes to thank Klaus Frieler, Andrea Halpern, Ruben Hillewaere, and David Lewis for very valuable comments on earlier drafts of this report and on the FANTASTIC code itself.

# References

Arom, S. (1991). *African Polyphony and Polyrhythm. Musical Structure and Methodology.* Cambridge University Press, Cambridge.

Baayen, H. (1992). Quantitative aspects of morphological productivity. In Booij, G. and van Marle, J., editors, *Yearbook of Morphology 1991. Theme: Morphological Classes*, pages 109–150. Springer.

Baayen, H. (2001). *Word Frequency Distributions.* Kluwer Academic Publishers.

Bilmes, J. A. (1993). Timing is of the essence: Perceptual and computational techniques for representing, learning, and reproducing expressive timing in percussive rhythm. Master's thesis, School of Architecture and Planning, MIT, Boston [MA].

Dowling, W. J. (1972). Recognition of melodic transformations: Inversion, retrograde, and retrograde inversion. *Perception & Psychophysics*, 12(5):417–421.

Downie, J. S. (2003). *Evaluating a simple approach to music information retrieval. Evaluating a simple approach to music information retrieval. Conceiving melodic n-grams as text.* PhD thesis, Faculty of Information and Media Studies, University of Western Ontario, London (Ontario), Canada.

Eerola, T. and Toiviainen, P. (2004). Mir in matlab: The midi toolbox. In *Proceedings of the 5th International Conference on Music Information Retrieval.*

Frieler, K. (2005). Melody - csv file format (mcsv), technical documentation. unpublished.

Gower, J. C. (1971). A general coefficient of similarity and some of its properties. *Biometrics*, 27(4):857–871.

Hood, M. (1971). *The Ethnomusicologist*. McGraw-Hill, New York.

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

Jesser, B. (1990). *Interaktive Melodieanalyse: Methodik und Anwendung computergestützter Analyseverfahren in Musikethnologie und Volksliedforschung. Typologische Untersuchung der Balladensammlung des DVA*. Peter Lang, Bern.

Juhász, Z. (2000). A model of variation in the music of a hungarian ethnic group. *Journal of New Music Research*, 29(2):159–172.

Krumhansl, C. L. (1990). *Cognitive Foundations of Musical Pitch*. Oxford Psychology Series 17. Oxford University Press, Oxford.

Krumhansl, C. L. and Kessler, E. (1982). Tracing the dynamic changes in perceived tonal organization in a spatial representation of musical keys. *Psychological Review*, 89:334–368.

Kubik, G. (1998). *Zum Verstehen afrikanischer Musik*. Philipp Reclam junior, Leipzig.

Lomax, A. (1977). *Cantometrics: An Approach to the Anthropology of Music*. University of California, Berkley.

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

Müllensiefen, D. and Frieler, K. (2006). *The SIMILE algorithms documentation 0.3*.

Müllensiefen, D. and Wiggins, G. A. (2009). Polynomial functions as a representation of melodic phrase contour. *under review*.

Pfleiderer, M. (2006). *Rhythmus: Psychologische, theoretische und stilanalytische Aspekte populärer Musik*. transcript, Bielefeld.

Quesada, J. (2007). Creating your own LSA spaces. In Landauer, T. K., McNamara, D. S., Dennis, S., and Kintch, W., editors, *Handbook of Latent Semantic Analysis*, chapter 4, pages 71–85. Lawrence Erlbaum Associates.

Sadakata, M., Desain, P., and Honing, H. (2006). The bayesian way to relate rhythm perception and production. *Music Perception*, 23(3):269–288.

Sagrillo, D. (1999). *Melodiegestalten im luxemburgischen Volkslied: Zur Anwendung computergestützter Verfahren bei der Klassifikation von Volksliedabschnitten*. Holos, Bonn.

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

Silverman, B. W. (1986). *Density Estimation*. Chapman and Hall, London.

Steinbeck, W. (1982). *Struktur und Ähnlichkeit: Methoden automatisierter Melodieanalyse*. Bärenreiter, Kassel.

Temperley, D. (2001). *The Cognition of Basic Musical Structures*. MIT Press, Cambridge, MA.

Temperley, D. (2007). *Music and Probability*. MIT Press, Cambridge, MA.

Uitdenbogerd, A. L. (2002). *Music Information Retrieval Rechnology*. PhD thesis, RMIT University of Melbourne, Australia.

# Index