Package 'FlexibleChoraleHarmonicAnalysis'

December 22, 2020

Type Package

Title Flexible Harmonic Analyses of Chorales by Bach and Praetorius

Version 0.8.0

Description This package is a companion to the paper presented at the International Society of Music Information Retrieval (ISMIR) 2018 conference entitled ``A Flexible Approach to Automated Harmonic Analysis: Multiple Annotations of Chorales by Back and Praetorius" (Condit-Schultz, Ju, and Fujinaga). The package includes several collections of data and tools for generating harmonic analysis data, as described in that paper. The data includes ``raw" humdrum/kern files of 571 chorales by J.S. Bach and M. Praetorius, as well as a wide variety of useful features parsed from this data, saved as RData objects. Most importantly, the package includes another RData object which encodes a large number of alternative harmonic analyses of these chorales, a set of functions which can be used to ``filter" these possibilities to get specific analyses, and functions to output these analyses as humdrum files, aligned with the original kern data.

License GPL (>= 2)

URL https://github.com/natsguitar/FlexibleChoraleHarmonicAnnotations

Encoding UTF-8

LazyData true

Depends R (>= 3.2.0), data.table (>= 1.13.2), stringr, rlang

Roxygen list(markdown = TRUE)

RoxygenNote 7.11

StagedInstall no

R topics documented:

analyzeChorales 2
ChoraleTable 7
humdrumDirectory 11
humdrumPaths 11
viewHumdrum 12
writeHumdrum 13
%ranks% 13

Index **15**

`analyzeChorales` *Produce harmonic analyses of chorales*

Description

This is the master API for the package, which generates specific harmonic analyses of the 571 chorales. You may specify one or more criteria for either *filtering* or *ranking* analyses. These criteria will be used to produced specific harmonic analyses of the 571 chorales, which you may inspect within an R session (`viewHumdrum`) or export to new humdrum data files (`writeHumdrum`).

Usage

```
analyzeChorales(rank = list(), filter = list(), ...)

ChoraleAnalyses
```

Arguments

`rank`	A single formula or a list of formulae used to rank the harmonic analyses of each contextual window.
`filter`	A single formula or a list of formulae used to filter out undesired harmonic interpretations.
`...`	If an optional `path` argument is included, the analysis is sent directly to writeHumdrum, using the `path` (and `label` argument if included) from `....`

Format

An object of class `list` of length 4.

Filtering and Ranking

This package contains (internally) precomputed "permutational" analyses of the 571 chorales. These "permutations" are *every* possible legal harmonic analysis of each of the 24,852 "contextual windows" in the chorale dataset. Most windows are a single chord or slice and have only one possible analysis (13,466 cases). However, other windows are more complicated and can have hundreds of possible analyses (1,546 is the maximum!). Most windows (21,241 out of 24,852) have five or fewer possible analyses. We can extract our "flexible" output analyses by specifying criteria for filtering and/or ranking the various possible analyses.

Filtering and ranking criteria are specified as R formulae, using the ~ (tilde) operator. Each formula must be expression that evaluates using the fields of the (hidden) ChoraleAnalyses data object. ChoraleAnalyses`` contains parsed features about each of the individual harmonic *analysis* of each contextual window in a (shallow) tree format. You will specify criteria for harmonic analyses by referring to the fields in theChoraleAnalysestree; thus, getting to know the fields/features of theChoraleAnalyses‘ tree is essential to making use of the package.

ChoraleAnalyses is a nested list, so R's $ operator can be used to step through levels in the tree: Note that some of the field names in the table below end with $—these are non-terminal branches of the tree. Terminal branches (actual features you can use for filtering/ranking) are followed by a parenthetical indicating their data type. The top level of the tree divides analysis features into four categories: windows, chords, chord-tones, non-chord tones. The full tree's fields (and their respective data types) are:

- Window$: Information about the window itself.
  - Duration (numeric): duration of window in tactus units.
  - Number (integer): enumeration of window.
  - Slices (integer): number of slices in the window.
- Chord$: Information about the chords in the analysis.
  - CompletionDelay$ (list of two):
    - In some cases all the notes of a harmony don't appear at the same time, yet we recognize the chord to have "arrived." The completion delay indicates how long it takes for the "complete" chord to be sounded after the chord arrives.
    - Durations (list of numeric):
      - Completion delay in tactus units.
    - Slices (list of integer):
      - Completion delay in number of slices.
  - Count (integer): How many chords in the window?
  - Durations (list of numeric): The duration of the chord(s) in tactus units.
  - Inversions (list of integer): The inversion of the chord:
    - 0: root position
    - 1: first inversion
    - 2: second inversion
    - 3: third inversion.
  - Qualities (list of character): The quality of the chord(s). Complete triads and 7th chords are indicated:
    - "M" (major)
    - "m" (minor)
    - "d" (diminished)
    - "MM" (major 7th)
    - "Mm" (dominant 7th)
    - "mm" (minor 7th)
    - "dm" (half-diminished 7th)
    - "dd" (fully-diminished 7th)

    Incomplete triads are indicated:
    - "m(?5)": a minor third with missing 5th.
    - "M(?5)": a major third with missing 5th.
    - "P5": perfect 5th with no 3rd.

    Incomplete seventh chords are indicated:
    - "(MM)": Apparent "MM" but missing 5th; i.e., major 3rd and major 7th are present.
    - "(mm)": minor 3rd and minor 7th but missing 5th. Could be "mm" or "dm".

    - "(Mm)": major 3rd and minor 7th but missing 5th. Only makes sense as incomplete "Mm".
    - "(dm)": diminished 5th and minor 7th but missing 3rd. Only makes sense as incomplete "dm".
    - "(dd)": diminished 5th and diminished 7th but missing 3rd. Only makes sense as incomplete "dd".
    - "(?m)": perfect 5th and minor 7th but missing 3rd. Could be "Mm" or "mm".
    - "(??)": minor 7th but no fifth *or* third.
  - Roots (list of character): The root of the chord(s): combinations of the letters A-G with zero or more sharps (#) or flats (-). For example, "G", "A-", or "F##".
  - SeventhResolves (list of logical): A logical (TRUE/FALSE) indicating if 7ths in the chord resolve downward by step (eventually). If there is no 7th in the chord, value is NA.
  - Slices (list of integer): How many slices per chord?
- CTs$ (chord tones)
  - Contour (data.table; 11 columns of integers): How many of each contour type are counted as chord tones, each column being one type. See the chorale table documentation for a list of the types.
  - Count (integer):
    - How many chord tones are there in the window?
  - Durations (list):
    - Duration (in tactus units) of the chord-tones in the analysis. Ordered by voice (soprano to bass), then by slice.
- NCTs$ (non-chord tones)
  - Contour (data.table; 11 columns of integers): How many of each contour type are counted as non-chord tones, each column being one type. See the chorale table documentation for a list of the types.
  - Count (integer):
    - How many non-chord tones are there in the window?
  - Durations (list):
    - Duration (in tactus units) of the chord-tones in the analysis. Ordered by voice (soprano to bass), then by slice.

(Read the choraleTable documentation for additional information about the features.)

To illustrate how this works, we can refer to ~NCTs$Contour$PT to get the field containing the count of non-chord-passing-tones in each analysis. Similarly, we can access the field indicating the inversion of chords using ~Chord$Inversion. As a shorthand, you can place some of the tree nodes (including $s) on the left-hand side of a formula's ~: Variables on the right-hand side are then evaluated within the left-hand node. For example, we write NCTs$Contour ~ PT instead of ~ NCTs$Contour$PT. This is most useful when referring to multiple fields within one branch of the tree: for example, NCTs$Contour ~ PT > NT is a legal way of using *both* the NCTs$Contour$PT and NCTs$Contour$NT fields.

Creating Filters/Rankings:

You can compose any R expression you want using fields of the ChoraleAnalyses tree! This can include arithmetic, comparison operators (>, <, etc.), or even your own arbitrary functions. Since you have the entire ChoraleAnalyses feature tree (above) at your finger tips, you can make a

huge variety of possible criteria for selecting the harmonic analyses you prefer. For example, `NCTs$Contour ~ PT > NT` accesses the `NCTs$Contour$PT` and `NCTs$Contour$NT` and asks which analyses have more non-chord-passing tones than non-chord-neighboring tones (using >). Used as a rank or filter, this formula would favor analyses where there are more passing tone NCTs than neighboring tone NCTs.

Filtering formulae must return logical (`TRUE` or `FALSE`) values: only harmonic analyses for which the filter(s) return `TRUE` are used. Ranking formulae, on the other hand, can return any numeric values: harmonic analysis are ranked from best to worst (high to low rank) according the results of your ranking formulae and the highest rank analysis for each window is used in the output analyses. When analyses are tied in rank you'll just get a more or less random pick from the ties—the solution is to be more specific in your ranking/filtering! Logical values in a ranking formulae are treated as numeric 0 (FALSE) and 1 (TRUE).

Watch out! Overly zealous filters can easily lead to some harmonic slices with no legal analysis, which are then marked "X" in the output. Prefer using ranking formulae over filters to avoid this problem—ranking formulae will always return something.

Having ranking/filtering criteria expressed as R formulae makes it possible to store, combine, and compose criteria, as R formulae can be held in lists. You can concatenate formulae using `c(...)`. For example, if you have ranking formula that you use a lot you could assign it to a variable, like `ccount = Count ~ Chord`. You can then mix `ccount` with other formula by concatenating them: c(ccount, ~ another criteria, ~ a third criteria, etc). You may want to try out various combinations of 4–6 different ranking criteria.

If you find this all confusing, that is understandable! Some examples should help:

More Chords:
Lets say you prefer analyses that use as many chords as possible (generally minimizing non-chord tones). You could just use a ranking formulae of `rank = Chord ~ Count`. This formula will return the value of `ChoraleAnalyses$Chord$Count`, which are integers indicating how many chords there are in the window. These numbers will be used to rank the analysis, with higher numbers being better—the result will be analyses with as many chords as possible. If you want to be more specific you could add another ranking criteria, minimizing the number of non-chord tones: `rank = c(Chord ~ Count,NCTs ~ Count * -1)`. This will return the number of chords *and* the negative number of non-chord tones—the maximum rank will be the analysis with the most chords *and* the least non-chord tones.

Fewer chords:
Lets say you instead want to *minimize* the number of chords, getting a more abstract, "high-level" analysis. You can just reverse the formulae in the last example: `rank = Chord ~ Count * -1`, or `rank = c(Chord ~ Count * -1,NCTs ~ Count)`.

Lists of X:

You'll note that many fields of the `ChoraleAnalyses` tree are "lists of" vectors. These are vectors of varying lengths corresponding to the number of chords in each analysis. If, for a particular analysis, `Chord$Count` has a value if `3` that means the window is being analyzed with three chords. Thus, `Chord$Roots` will be a vector of length three, representing the roots of the three chords in order. Similarly, `Chord$SeventhResolves` will be a vector of length three indicating if 7ths in each of the three chords resolve or not (`NA` if there is no 7th). All of these fields have plural names, so be careful: Its `Chord$Durations` not `Chord$Duration`.

The package performs some extra magic to help us work with these lists of vectors. If you use calls to `any`, `all`, `max`, `min`, `mean`, or `sum` in your filter/rank formulae they will automatically

apply to the nested values. For example, if you say `rank = Chord ~ any(SeventhResolves)`, the analyses will be ranked to prefer analyses where are at least one (i.e., any) of the 7ths in the chords (if any) resolve. Similarly, `rank = Chord ~ all(SeventhResolves)` will prefer analyses where *all* 7ths resolve. For a finer-grain ranking: `rank = Chord ~ mean(SeventhResolves)` will return the average of the logicals treated as `0`s and `1`s, so you get the proportion of chords sevenths which resolve—more resolving sevenths the better.

Another example would be to use `rank = ~ Chord ~ min(Durations) >= 1`, which will prefer analyses where the minimum chord duration is a tactus or greater.

If more than one list fields are included in the nested call (to `any`, `all`, etc.), they will all be mapped across in parallel. Non-nested fields will also be mapped across as well. (Note that all these special nested calls are called with the `na.rm = TRUE`.)

Ranking:
Another useful function is the %ranks% function. The %ranks% function is most useful for ranking `Chord$Qualities`. You can also use the standard R base function to cut numeric values into ranked categories (set `ordered_result = TRUE`).

Standard Analyses

You will find that there are a few ranking criteria that you almost always want to use to get reasonable analyses. These include various combinations of:

- `Chord ~ mean(SeventhResolves)`: more sevenths resolving is good!
- `Chord ~ min(Qualities %ranks% c('M|m|d|Mm'=4,'MM|mm|dm|dd' = 2,0))`: prefer analyses without any rarer chords.
- `NCTs ~ -1 * max(Durations)`: having long-duration non-chord tones is a little weird, so prefer shorter ones. You might alternatively use `mean(Duration)`.
- `Chord ~ abs(mean(Durations) -1)`: favor analyses where chord durations average close to 1 (the tactus).
- `Chord$CompletionDelay ~ max(Durations) * -1`: prefer analyses with short completion delay...i.e., transition directly to complete chords.

Write Output

If you want to directly output your analyses to humdrum files, add a `path` (and optional `label`) argument to your call. This will pass the analysis and the `path`/`label` arguments directly to writeHumdrum.

Examples

```
analysis1 <- analyzeChorales(Chord ~ Count)


viewHumdrum('bach', 10, analysis1)
writeHumdrum(analysis1, label = 'analysis1')

###

shortNCTs    <- NCTs ~ -max(Durations)
triads       <- Chord ~ min(Qualities %ranks% c('M|m|d|Mm'=4, 'MM|mm|dm|dd' = 2, 0))
```

```
resolved7ths <- Chord ~ mean(SeventhResolves)

analysis2 <- analyzeChorales(c(shortNCTs, triads, resolved7ths, Chord$CompletionDelay ~ max(Durations) * -1 ))

###

analyses <- c(analysis1, analysis2)
viewHumdrum('bach', 10, analyses)

writeHumdrum(analyses, label = 'twoAnalyses')
```

`ChoraleTable` *Features parsed from 572 chorale **kern files*

Description

The FlexibleChoraleHarmonicAnalysis package's "ChoraleTables" contain musical features parsed from the 572 "raw" chorale (**kern) files. These features are used as the basis for the harmonic analyses. The package actually defines two "ChoraleTable" objects: `ChoraleTable_Slices` and `ChoraleTable_Notes`. These tables contain the same information, parsed "by slice" and "by note" respectively.

Usage

```
ChoraleTable_Slices

ChoraleTable_Notes
```

Format

An object of class `data.table` (inherits from `data.frame`) with 42928 rows and 40 columns.

An object of class `data.table` (inherits from `data.frame`) with 171915 rows and 40 columns.

Slice vs Note

A "slice" refers to the set of notes (or rests) occuring at the same time in a chorale. A new slice is formed whenever any voice in the chorale articulates a new onset (or rests). (Each data record in the original humdrum data is one slice). There are 42,928 slices in the 571 chorales, corresponding to the 42,928 rows of the `ChoraleTable_Slices` table. These slices do not all contain/represent the same number of notes: there are 567 four-voice chorales (370 by Bach) and 4 five-voice chorales (1 by Bach). In addition, Praetorius' 130th chorale is divided between a three-voice part and a four-voice response. In total, the 42,928 slices in `ChoraleTable_Slices` include: 99 three-note slices, 42,527 four-note slices, and 302 five-note slices. As a result, many columns in the `ChoraleTable_Slices` table are actually lists of vectors varying in length from three to five (details below).

The ChoraleTable_Notes table contains the same information as the the ChoraleTable_Slices table, except each row represents a single note or rest. There are 171,915 notes/rests in the data, corresponding to the 171,915 rows of the ChoraleTable_Notes table.

Each of the 42,928 slices and 171,915 notes in the dataset are enumerate in the Slice_Number and Note_Number fields. This enumeration begins with the first slice/note of the Bach Chorales, proceeds through each Bach chorale in turn (following their numbering), and then proceeds with the first slice/note of the first Praetorius Chorale, etc. Alignment between the ChoraleTable_Notes and ChoraleTable_Slices tables can be determined using the Slice_Number and Note_Number fields.

Windows and Phrases

The Bach chorales indicate phrase boundaries using fermata marks. The Praetorius chorales indicate phrases using barlines. In total, the dataset consists of 3,225 phrases: 2,277 phrases in the Bach; 948 phrases in the Praetorius.

The 42,928 slices in the chorales are also parsed into 24,852 "contextual windows." These windows were identified through a combination of programmatic rules and expert hard-coding of some edge cases. These windows vary in "size" from 1 to 10 slices—or in more musical terms, from half a tactus (usually an 8th note) to 32 tactus beats (four measures). The vast majority (17,200) of windows are one tactus in length. The vast majority of windows are either one slice (13,276), two slices (8,154), three slices (1,544), or four slices (1,142).

All the package's "flexible" harmonic analysis take place within these windows. See the paper http://ismir2018.ircam.fr/doc/pdfs/283_Paper.pdf for more details. The window enumeration is indicated in the Slice_WindowNumber field of the chorale tables.

Fields

The chorale tables each have 40 columns of information, parsed from the raw chorale (**kern) scores.

Twenty-three columns (prefixed "Slice_") contain features that pertain to an entire slice—for instance, the metric position. (These data points are duplicated for each note in each slice in the ChoraleTable_Notes table.) The remaining seventeen columns (prefixed "Note_") contain features specific to individual notes. In the ChoraleTable_Slices table, these 17 columns actually contain lists of nested vectors varying in length from three to five (depending on the number of voices/notes in the slice): The first element in each vector represents the highest voice (e.g., the soprano) in the slice, with the last element representing the lowest voice (e.g., the bass). All 40 columns of the ChoraleTable_Notes table are vectors not lists.

The twenty-three columns of slice-specific features are:

Slice_Composer Character: Either "Bach" or "Praetorius."

Slice_Duration Numeric: Duration of slice in tactus-beat units.

Slice_Duration Numeric: Duration, or "offset," of slice from the first downbeat of the chorale, in tactus units. Pickups at the beginning of a piece receive negative values.

Slice_FileName Character: Name of chorale humdrum file.

Slice_FileNumber Integer: The chorale-file "number," as enumerated from 1 to 572. 1–371 are Bach. 372–572 are Praetorius.

Slice_isNewWindow Logical: Is this slice the beginning of one of the 24,852 "contextual windows" used in the analysis process?

Slice_Measure Integer: The measure number within the chorale. Pickup notes are in measure 0.

Slice_Measure Character: The mensuration sign, if any, included in the humdrum score. Values are "2", "3", "3/2", "c", "3", "c", "C", "c3", "c|", "c|2", "c|3", "c2", or "c3." This field is NA for all slices in scores that have no mensuration sign.

Slice_Meter Character: The time signature indicated in the humdrum score. Values are "2/1", "2/2", "3/1", "3/2", "3/4", and "4/4."

Slice_MetricLevel Numeric: The "metric level" of each slice indicates the duration of the highest metric beat that the slice lands on, in tactus-note units. For instance, the downbeat of a 4/4 measure corresponds to a whole-note beat, and thus has a metric level of 4. Eighth-note offbeats have a metric level of 0.5. The third beat in a triple meter is accorded a metric level 1.1.

Slice_Metric_Position Numeric: The duration from the beginning of the measure, in tactus units. The first slice in each measure has a Metric_Position of 0.

Slice_Number Integer: The slice's "number," enumerated from the beginning of the first Bach chorale through the rest of the Bach chorales, then proceeding through the Praetorius chorales. The maximum value is 42,928.

Slice_NumberOf_NewPitchClasses Integer: How many pitch classes (i.e., pitches regardless of octave) didn't appear in the previous slice of the chorale? If a pitch class that wasn't present before appears more than once, it is counted as many times as it appears. The first slice of each chorale always counts all the notes present.

Slice_NumberOfNewPitchClasses_Unique Integer: How many unique pitch classes (i.e., pitches regardless of octave) didn't appear in the previous slice of the chorale? If a pitch class that wasn't present before appears more than once, it is counted only once. The first slice of each chorale always counts all the unique notes present.

Slice_NumberOf_Onsets Integer: How many note onsets occur in the slice? This excludes rests and notes that are tied from the previous slice.

Slice_NumberOfSoundingNotes Integer: How many notes are being sounded? This only excludes rests.

Slice_NumberOfVoices Integer: How many voices are present in this chorale, whether or not they are singing at this moment?

Slice_NumberOfVoices_Active Integer: How many voices are "active" in the chorale during this passage, even if they might be resting during this slice? This feature is useful for a view chorales (mostly by Praetorius) which engage in antiphony (e.g., call and response) between voices. We may want to register that some of the voices are still "active" when they rest briefly in between phrases.

Slice_PhraseBoundary Logical: Is this slice the boundary at the end of a phrase? Phrases are indicated by fermatas (Bach) and barlines (Praetorius).

Slice_Phrase_Number Integer: The number of the current phrase, enumerated from the beginning of the Bach chorales and continuing through the Praetorius. There are a total of 3,225 phrases in the dataset.

Slice_Record Integer: The record (line) number of the slice in it's original humdrum file.

The seventeen columns of note-specific features are:

Note_Accented_Relative Logical: Is this note rhythmically accented relative to the previous note? Tied notes inherit their status from their onset. Rests have a value of <NA>.

Note_CircleOfFifths Integer: The pitch-classes position in the line of fifths; i.e., C = 0, G = 1, D = 2, F# = 6, Eb = -3.

Note_Duration Numeric: Note's duration in tactus units.

Note_Duration_Remaining Numeric: Note's duration in tactus units (redundant).

Note_ExtendsPastNextBeat Logical: Does this note sustain past the next beat stronger than the beat it attacked on?

Note_isLongerThanSlice Logical: Does this note sustain longer than the duration of the current slice?

Note_isNewPitchClass Logical: Is this note a pitch class (i.e., pitch regardless of octave) a pitch class which wasn't sounding in the previous slice?

Note_isOnset Logical: Is this note an onset? Excludes rests and ties.

Note_isRest Logical: Is this note a rest? There are 904 rests in total in the dataset.

Note_Metric_Level_Relative Integer: What is the metric level of this note (see the `Slice_MetricLevel` definition above) compared to the metric level of the previous note? Values are -1 (note is lower than previous note), 0 (note is same level as previous note), 1 (note is higher than previous note). Rests and notes are phrase boundaries are <NA>.

Note_Number Integer: Notes are enumerated from highest voice to lowest voice within each slice, from the beginning of the first Bach chorale through the end of the last Praetorius Chorale. The maximum value is 171,915. Rests are numbered too!

Note_PotentialNonChordToneType Character: A string indicating what type of non-chord-tone this note COULD be. Notes which do not match ANY possible non-chord pattern are <NA>. Options are: `"Ant"` (anticipation), `"App"` (appogiatura), `"Camb1"` (first note of cambiatta), `"Camb2"` (second note of cambiatta), `"CT"` (chord tone; this note cannot legally be a non-chord tone), `"DPT1"` (dirst note of double passing tone), `"DPT2"` (second note of double passing tone), `"Esc"` (escape tone), `"Inc"` (incomplete neighbor), `"NT"` (neighbor tone), `"NTchrom"` (chromatic neighbor tone), `"Ped"` (pedal tone), `"PT"` (passing tone), `"Ret"` (retardation), `"Sus"` (suspension), `"Sus+"` followed by `"SusInc"` (a suspension decorated by a lower incomplete neighbor to the resolution), `"syncNT"` (syncopated neighbor), and `"syncPT"` (syncopated passing tone).

Note_Semits Integer: The note's pitch in semitones from middle-C; i.e., middle-C = 0, and the G at the top of the treble clef = 19. Rests are <NA>.

Note_TonalName Character: The "tonal name" of the pitch regardless of octave. Often referred to as the "pitch class" in this documentation. Tonal names are combinations of one of seven letter names (A, B, C, D, E ,F, G) and zero or more accidentals (# = sharp, - = flat). Rests are <NA>.

Note_VoiceLeading_Approach Integer: What is the interval (in semitones) between the previous note and the current note? If the previous note is G5 and the current note is C5, than the VoiceLeading_Approach is `-7`. Rests are <NA>.

Note_VoiceLeading_Departure Integer: What is the interval (in semitones) between the current note and the next? If the current note is G5 and the next note is C5, than the VoiceLeading_Approach is `-7`. Rests are <NA>.

Note_VoiceNumber Integer: Which voice, numbered from high to low, is this note sung by. For instance, the Tenor voice in a four voice chorale would be VoiceNumber = 3.

Note that the "tactus" for each chorale was expertly labeled, and all durations are encoded relative to the tactus. In the vast majority of Bach cases, the tactus is the quarter-note, so the duration units are quarter notes. However, some Bach chorales and many Praetorius chorales have a half-note tactus, so 1 = half note.

`humdrumDirectory` *Raw humdrum data directory*

Description

This character string stores the directory where the package's raw humdrum data is stored on your machine.

Usage

```
humdrumDirectory
```

Format

An object of class `character` of length 1.

`humdrumPaths` *Raw humdrum data paths.*

Description

This character vector stores the path to all 572 of the package's raw humdrum data files.

Usage

```
humdrumPaths
```

Format

An object of class `character` of length 572.

viewHumdrum *Inspect a humdrum file from the dataset.*

Description

This function prints humdrum data in the R console. It can be used to inspect the "raw" humdrum files from the dataset, or to view the annotated humdrum generated by the package. When this function is called, the humdrum file(s) are printed on the screen.

Usage

```
viewHumdrum(composer = "Bach", chorale = 1, analyses = NULL)
```

Arguments

composer	Character string: The name of the desired composer (Bach or Praetorius). Partial matches are allowed; Case insensitive.
chorale	Numeric (integer): The desired chorale number(s).
analyses	An analyses object, created by analyzeChorales. If NULL the original humdrum files are printed. analyzeChorales outputs a single analysis, but multiple analyses can be concatenated and fed to viewHumdrum: i.e., viewHumdrum('bach',10,c(analysis1,ana

Value

A list of character vector: one for each viewed chorale. Each element is a vector of humdrum records/slices.

Examples

```
viewHumdrum('Bach', 11)
# Prints the 11th Bach chorale, unanalyzed.

viewHumdrum('Praetorius', 1:10)
# Prints the first 10 Praetorius chorales, unanalyzed.

viewHumdrum('Pr', 1:10, analyses = analyzeChorales(rank = list(Chord ~ Count)))
# Prints the first 10 Praetorius chorales, with a particular harmonic analysis.
```

writeHumdrum *Write annotated humdrum data to files*

Description

This function writes new humdrum data (original **kern with added harmonic analyses spines) to text files. The original krn file names are used for the output files with the addition an optional label specified by the user and a different extension (.hum).

Usage

```
writeHumdrum(analyses, path = NULL, label = NULL)
```

Arguments

analyses	An analyses object, created by analyzeChorales. analyzeChorales outputs a single analysis, but multiple analyses can be concatenated and fed to writeHumdrum: i.e., writeHumdrum(c(analysis1,analysis2))
path	A single character string indicating the file path to write the files into. Defaults to NULL, in which case files are written to the current working directory.
label	A single character string to append to each filename. Defaults to NULL, which causes no label to be appended. Use this to differentiate different batches of analyses you generate.

Value

A data.table with two columns (Records and FileName) and nrow(ChoraleTables_Slices) rows. Records contains a character vector of new humdrum records. FileName a character vector indicating the new humdrum filenames, with the added path and label, for each record/slice.

%ranks% *Apply arbitrary ranks to values*

Description

The %ranks% infix function allows you to apply arbitrary ranks to an input vector. This is mainly used with the Chord$Qualities field.

Usage

```
x %ranks% patterns
```

Arguments

x	A vector. Usually the Chord$Qualities field. This field will always be coerced to a character vector.
patterns	A named vector of numbers.

Details

Each of the names of the right-side patterns argument is treated as one or more character strings to match exactly the left-side input vector x. The output is assigned the value of the last pattern that matches. Multiple patterns can be given the same weight by putting them in the same name, but separated by "|".

If one, unnamed, pattern value is included on the right-side, it is used as the default value for anything that matches no patterns.

Examples

```
test <- c('M', 'm', 'd', 'Mm', 'mm', 'MM', 'P5')

test %ranks% c('M' = 5, 'm|d' = 4, 'Mm' = 3, 'mm|MM' = 2, 0)

# returns: c(5, 4, 4, 3, 2, 2, 0)


analyzeChorales(Chord ~ min(Qualities %ranks% c('M' = 5, 'm|d' = 4, 'Mm' = 3, 'mm|MM' = 2, 0)))
```

Index

∗ **datasets**
    analyzeChorales, 2
    ChoraleTable, 7
    humdrumDirectory, 11
    humdrumPaths, 11
%ranks%, *6*, 13
3/2, *9*

2, *9*
3, *9*

analyzeChorales, 2, *12, 13*

base, *6*

c, *9*
chorale dataset, *2*
chorale table, *4*
ChoraleAnalyses *(*analyzeChorales*)*, 2
ChoraleTable, 7
choraleTable, *4*
ChoraleTable_Notes *(*ChoraleTable*)*, 7
ChoraleTable_Slices *(*ChoraleTable*)*, 7

humdrumDirectory, 11
humdrumPaths, 11

viewHumdrum, *2*, 12

writeHumdrum, *2*, *6*, 13