Package ‘musicMCT’

November 3, 2025

Title Analyze the Structure of Musical Scales

Version 0.3.0

Description Analysis of musical scales (& modes, grooves, etc.) in the vein of Sherrill 2025 <doi:10.1215/00222909-11595194>.
The initials MCT in the package title refer to the article's title: ``Modal Color Theory.'' Offers support for conventional musical pitch class set theory as developed by Forte (1973, ISBN: 9780300016109) and David Lewin (1987, ISBN: 9780300034936), as well as for the continuous geometries of Callender, Quinn, & Tymoczko (2008) <doi:10.1126/science.1153021>.
Identifies structural properties of scales and calculates derived values (sign vector, color number, brightness ratio, etc.). Creates plots such as ``brightness graphs'' which visualize these properties.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 3.5)

LazyData true

Imports igraph, utils, stats, graphics

Suggests grDevices, knitr, rmarkdown, testthat (>= 3.0.0), vdiffr, withr,

VignetteBuilder knitr

URL `https://satbq.github.io/musicMCT/`, `https://github.com/satbq/musicMCT`

BugReports `https://github.com/satbq/musicMCT/issues`

Config/testthat/edition 3

NeedsCompilation no

Author Paul Sherrill [aut, cre, cph] (ORCID: <`https://orcid.org/0009-0002-3617-016X`>)

Maintainer Paul Sherrill <`paul.sherrill@utah.edu`>

Repository CRAN

Date/Publication 2025-11-03 03:20:08 UTC

Contents

anazero_fingerprint *Are regularities within or between sets in a pair?*

Description

As for other hyperplane arrangements, it is useful to consider the number of entries which equal 0 in an anaglyph signvector. However, such entries can represent three different types of regularity: regularity within the first set, regularity within the second set, or regularity in the comparison between them. This function distinguishes between those three types of hyperplanes.

Usage

```
anazero_fingerprint(set, edo = 12, rounder = 10)
```

Arguments

`set`	A vector of even length representing a pair of sets
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

A vector with three entries, representing regularities in the first set, regularities in the second set, and regularities between them.

See Also

`make_anaglyph_ineqmat()`, `svzero_fingerprint()`

Examples

```
maj <- c(0, 4, 7)
sus2 <- c(0, 2, 7)
anazero_fingerprint(c(maj, sus2))

# The first zero shows that the major triad has no regularities.
# This is equivalent to:
countsvzeroes(maj)

# The second zero shows that the sus2 trichord has 1 regularity.
# This is equivalent to:
countsvzeroes(sus2)

# The final zero shows that the major triad's perfect fifth
# equals the size of the *two* perfect fifths in the sus2 trichord.
# We can visualize the whole set of relationships using a brightness
# graph:
```

```
brightnessgraph(maj, sus2)
```

asword *Algebraic word of a set's step sizes*

Description

Among others, Carey & Clampitt (1989) and Clampitt (1997) have shown that much can be learned about a set by representing it as a word on m "letters" which represent the m distinct steps between adjacent members of the set. This is more or less what is done in theory fundamentals classes when a major scale is represented as TTSTTTS (if we temporarily forget that T and S represent specific interval sizes). In scholarship the algebraic letters are usually represented as letters of the Latin alphabet, but for some computational purposes it is useful for these to be explicitly ordered. That is, the major scale should be represented as integers 2212221, which is distinct from 1121112. (Thus `asword` makes finer distinctions than you might expect coming from a word-theoretic context.)

Usage

```
asword(set, edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

Vector of integers of the same length as `set`. `1` should always be the lowest value, representing the smallest step size in the set.

Examples

```
dia_12edo <- c(0, 2, 4, 5, 7, 9, 11)
qcm_fifth <- meantone_fifth()
qcm_dia <- sort(((0:6)*qcm_fifth)%%12)
just_dia <- j(dia)
asword(dia_12edo)
asword(qcm_dia)
asword(just_dia)

#### asword() is less discriminating than colornum().
#### See "Modal Color Theory," 16
set1 <- c(0, 1, 4, 7, 8)
set2 <- c(0, 1, 3, 5, 6)
set1_word <- asword(set1)
```

```
set2_word <- asword(set2)
isTRUE(all.equal(set1_word, set2_word))
colornum(set1) == colornum(set2)
# (Last line only works with representative_signvectors loaded.)
```

brightnessgraph *Visualize brightness relationships among modes of a scale*

Description

Discussed in "Modal Color Theory" (pp. 7-11), the brightness graph of a scale is a Hasse diagram that represents the sum- and voice-leading brightness relationships between the modes of a scale. Each node of the graph represents a mode. With default options, the large Roman numeral of each node indicates which mode of the input scale it represents. (The input scale is roman numeral I.) Small Arabic numerals beneath the Roman numeral indicate the pitch-classes of the mode (relative to scale degree 1 as 0). In parentheses, the sum brightness of the mode is shown. Modes with higher sum brightness are farther up on the graph. Arrows connect modes that can be compared by voice-leading brightness. The arrows only show a transitive reduction of all VL-brightness comparisons, so that if you can travel between two sets by only going "up" or "down" the arrows, the source and destination are indeed related by voice-leading brightness.

If `goal=NULL` (as it is by default), the brightness graph includes simply the modes of `set`. However, `goal` can be any other scale of the same length as `set`, in which case the brightness graph includes modes of both sets and their interconnections. The modes of `goal` are represented by lower-case roman numerals, while upper-case numerals represent the modes of `set`.

Various visual parameters can be configured: `numdigits` determines how many digits of each pitch-class to display; `show_sums` toggles on or off the sum brightness values; `show_pitches` toggles on or off the individual pitch classes of each mode; `fixed_do`, if set to `TRUE` switches the graph from showing "parallel" modes (e.g. C ionian vs C aeolian) to showing "relative" modes (e.g. C ionian to A aeolian).

For now, the function doesn't have a smart way to determine the horizontal positioning of modes in the graph. It uses a heuristic that works well for many sets, but sometimes it will create too much visual overlap or won't clarify underlying structure particularly well. Think of these automatically generated graphs as the starting point for manual fine tuning.

Usage

```
brightnessgraph(
  set,
  goal = NULL,
  numdigits = 2,
  show_sums = TRUE,
  show_pitches = TRUE,
  fixed_do = FALSE,
  edo = 12,
  rounder = 10
)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`goal`	Numeric vector of same length as set. Defaults to `NULL`.
`numdigits`	Integer: how many digits of each pitch-class to show? Defaults to `2`.
`show_sums`	Boolean: should the graph show sum brightness values? Defaults to `TRUE`.
`show_pitches`	Boolean: should the graph show values for each note of the scale? Defaults to `TRUE`.
`fixed_do`	Boolean: should the graph use only the fixed pitches of the input set? Defaults to `FALSE`.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

Invisibly, an igraph graph object (the structure of the plotted brightness graph)

Examples

```
brightnessgraph(c(0,2,4,5,7,9,11))
brightnessgraph(c(0,2,4,5,7,9,11), fixed_do=TRUE)
brightnessgraph(c(0,1,4,9,11),edo=15)

#### A more complicated graph
werck_ratios <- c(1, 256/243, 64*sqrt(2)/81, 32/27, (256/243)*2^(1/4), 4/3,
  1024/729, (8/9)*2^(3/4), 128/81, (1024/729)*2^(1/4), 16/9, (128/81)*2^(1/4))
werckmeister_3 <- z(werck_ratios)
brightnessgraph(werckmeister_3, show_sums=FALSE, show_pitches=FALSE)


#### Graph for both inversions of the Tristan genus:
dom7 <- c(0, 4, 7, 10)
halfdim <- c(0, 3, 6, 10)
brightnessgraph(dom7, halfdim)
```

`brightness_comparisons` *Voice-leading brightness relationships for a scale's modes*

Description

The essential step in creating the brightness graph of a scale's modes is to compute the pairwise comparisons between all the modes. Which ones are strictly brighter than others according to "voice-leading brightness" (see "Modal Color Theory," 6-7)? This function makes those pairwise comparisons in a manner that's useful for more computation.

Usage

```
brightness_comparisons(set, goal = NULL, edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`goal`	Numeric vector of same length as set. Defaults to `NULL`.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Details

Note that the returned value shows all voice-leading brightness comparisons, not just the transitive reduction of those comparisons. (That is, dorian is shown as darker than ionian even though mixolydian intervenes in the brightness graph.)

Value

If `goal=NULL`, an n-by-n matrix where n is the size of the scale. Row i represents mode i of the scale in comparison to all n modes. If the entry in row i, column j is `-1`, then mode i is "voice-leading darker" than mode j. If `1`, mode i is "voice-leading brighter". If 0, mode i is neither brighter nor darker, either because contrary motion is involved or because mode i is identical to mode j. (Entries on the principal diagonal are always 0.)

If `goal` is a set, the result is a 2n-by-2n matrix whose first n rows and columns represent the modes of `set` and whose last n rows and columns represent the modes of `goal`. (Thus the upper left n-by-n square is the same as if `goal` were `NULL` and the lower right n-by-n square is the result of entering `goal` as `set` with an empty goal parameter. The upper-right and lower-left quadrants of the matrix make comparisons between the modes of `set` and `goal`.) The meaning of entries `-1`, `0`, and `1` are as above.

See Also

`brightnessgraph()` for a human-readable presentation of the same information.

Examples

```
# Because the diatonic scale, sc7-35, is non-degenerate well-formed, the only
# 0 entries should be on its diagonal.
brightness_comparisons(sc(7, 35))

mystic_chord <- sc(6,34)
colSums(sim(mystic_chord)) # The sum brightnesses of the mystic chord's 6 modes
brightness_comparisons(mystic_chord)
# Almost all 0s because very few mode pairs are comparable.
# That's because nearly all modes have the same sum, which means they have sum-brightness
# ties, and voice-leading brightness can't break a sum-brightness tie.
# (See "Modal Color Theory," 7.)
```

```
major <- c(0, 4, 7)
minor <- c(0, 3, 7)
brightness_comparisons(major, minor)
```

carlos_step *Define a step size for one of Wendy Carlos's scales*

Description

For her album *Beauty in the Beast*, Wendy Carlos developed several non-octave scales whose step sizes are calculated to optimize approximations of three intervals: the 3:2 fifth, the 5:4 major third, and the 6:5 minor third. The alpha, beta, gamma, and delta scales differ in terms of how strongly they privilege each of those just intervals. The basic step size for each scale is created by calling this function with the appropriate `name` argument (e.g. "alpha"). You can also choose your own weights for the three approximated just intervals, in which case the `name` argument is overridden.

Usage

```
carlos_step(name = "alpha", weights = NULL, edo = 12)
```

Arguments

`name`	Which of Carlos's four scales to create: `"alpha"`, `"beta"`, `"gamma"`, or `"delta"`. Defaults to `"alpha"`
`weights`	Numeric vector of length 3 assigning the number of steps that correspond to 3:2, 5:4, and 6:5, respectively. Overrides `name` if specified.
`edo`	Number of unit steps in an octave. Defaults to `12`.

Value

Single numeric value containing the step size for the desired scale

Examples

```
alpha_scale <- (0:31) * carlos_step()
practically_12tet <- (0:24) * carlos_step(weights=c(7, 4, 3))
```

clampitt_q *Voice leadings between inversions with maximal common tones*

Description

Clampitt (2007, 467; doi:10.1007/9783642045790_46) defines two n-note sets to be Q-related if they:

- Have all but one tone in common
- Are related by `tni()`
- Have a strictly crossing-free voice leading which preserves all $n - 1$ common tones This function finds all sets which are Q-related to an input `set` in this sense. The relation is defined to generalize the smooth voice leadings between consonant triads and diatonic scales to other sets, in particular demonstrating that non-singular pairwise well-formed scales (see `isgwf()`) demonstrate similarly nice voice leading properties.

(Strictly speaking, Clampitt includes `tn()` in the second part of the definition. However, the first criterion is only possible under `tn()` if the set is generated and therefore inversionally symmetrical. Therefore if a set satisfies Clampitt's definition by `tn()`, it also satisfies the `tni()` requirement.)

If the third part of the definition is relaxed, allowing the voice leading to involve voice crossing, Clampitt (1997, 121) identifies this as the Q*-relation. The Q*-relation can be computed with this function by setting `method="hamming"`. (All other methods provided by `vl_dist()` give equivalent results in this context.)

Usage

```
clampitt_q(
  set,
  index = NULL,
  method = c("taxicab", "euclidean", "chebyshev", "hamming"),
  edo = 12,
  rounder = 10
)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`index`	Integer: which Q-related set and voice leading should be returned? Defaults to `NULL`, in which case all options are returned.
`method`	What distance metric should be used? Defaults to `"taxicab"` but can be `"euclidean"`, `"chebyshev"`, or `"hamming"`.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

A list with two entries, "sets" and "vls". The former is a matrix whose columns are the sets which are Q-related to the input set, in OPC-normal form. The latter is a matrix whose rows represent the voice-leading motions which transform set into its goals. (This follows the general practice of musicMCT of representing scales as columns and voice leadings as rows.) The rows of "vls" correspond to the columns of "sets", but the columns of "vls" correspond to the order of the input set, which may not match the normal form of the output sets. (See the last example.)

See Also

isgwf(), minimize_vl(), normal_form()

Examples

```
# The Neo-Riemannian P, L, and R transformations on triads are all Q-relations:
major_triad <- c(0, 4, 7)
clampitt_q(major_triad)

# A well-formed scale like the diatonic has two Q-relations given by its signature transformations:
major_scale <- c(0, 2, 4, 5, 7, 9, 11)
clampitt_q(major_scale)

# A non-singular pairwise well-formed scale also has Q-relations:
clampitt_q(j(dia))

# Set-class 7-31 is pairwise well-formed:
clampitt_q(sc(7, 31))
# It also has two additional Q*-related sets:
clampitt_q(sc(7, 31), method="hamming")

# Most other types of scales have at most one Q-relation:
dominant_seventh <- c(0, 4, 7, 10)
clampitt_q(dominant_seventh)

# The order of "sets" may not match the order of "vls":
clampitt_q(c(0, 1, 4, 7))
```

---

clockface *Visualize a set in pitch-class space*

Description

No-frills way to plot the elements of a set on the circular "clockface" of pc-set theory pedagogy. (See e.g. Straus 2016, ISBN: 9781324045076.)

Usage

```
clockface(set, edo = 12)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`edo`	Number of unit steps in an octave. Defaults to `12`.

Value

Invisible copy of the input `set`

Examples

```
just_diatonic <- j(dia)
clockface(just_diatonic)

double_tresillo <- c(0, 3, 6, 9, 12, 14)
clockface(double_tresillo, edo=16)
```

`colornum` *Reference numbers for scale structures*

Description

As described on p. 28 of "Modal Color Theory," it's convenient to have a systematic labeling system ("color numbers") to refer to the distinct colors in the hyperplane arrangements. This serves a similar function as Forte's set class numbers do in traditional pitch-class set theory. Color numbers are defined with reference to a complete list of the possible sign vectors for each cardinality, so they depend on the extensive prior computation that is stored in the object `representative_signvectors`. (This is a large file that cannot be included in the package musicMCT itself. It needs to be downloaded separately, saved in your working directory, and loaded by entering `representative_signvectors <- readRDS("representative_signvectors.rds")`. Color numbers are currently only defined for scales with 7 or fewer notes.

Usage

```
colornum(set, ineqmat = NULL, signvector_list = NULL, edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`ineqmat`	Specifies which hyperplane arrangement to consider. By default (or by explicitly entering "mct") it supplies the standard "Modal Color Theory" arrangements of `getineqmat()`, but can be set to strings "white," "black", "gray", "roth", "infrared", "pastel", "rosy", "infrared", or "anaglyph", giving the `ineqmat`s of `make_white_ineqmat()`, `make_black_ineqmat()`, `make_gray_ineqmat()`, `make_roth_ineqmat()`, `make_infrared_ineqmat()`, `make_pastel_ineqmat()`, `make_rosy_ineqmat()`, `make_infrared_ineqmat()`, or `make_anaglyph_ineqmat()`. For other arrangements, this parameter accepts explicit matrices.

`signvector_list`
: A list of signvectors to use as the reference by which `colornum` assigns a value. Defaults to `NULL` and will attempt to use `representative_signvectors`, which needs to be downloaded and assigned separately from the package musicMCT. (If a named `ineqmat` other than "mct" is chosen, the function attempts to replace a `NULL` signvector list with a corresponding object in the global environment. For instance, if `ineqmat="pastel"` then the function tries to use `pastel_signvectors` for `signvector_list`.)

`edo`
: Number of unit steps in an octave. Defaults to `12`.

`rounder`
: Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Details

Note that the perfectly even "white" scale is number `0` for every cardinality by definition.

The function assumes that you don't need to be reminded of the cardinality of the set you've entered. That is, there's a color number 2 for every cardinality, so you can get that value returned by entering a trichord, tetrachord, etc.

Value

Single non-negative integer (the color number) if a `signvector_list` is specified or `representative_signvectors` is loaded; otherwise `NULL`

Examples

```
colornum(edoo(5))
colornum(c(0, 3, 7))
colornum(c(0, 2, 7))
colornum(c(0, 1, 3, 7))
colornum(c(0, 1, 3, 6, 10, 15, 21), edo=33)
colornum(c(0, 2, 4, 5, 7, 9, 11))
```

`comparesignvecs` *Do two sign vectors represent adjacent colors?*

Description

As "Modal Color Theory" (pp. 31ff.) describes, it can be useful to know whether two colors are adjacent to each other in the MCT space. That is, can one scalar color be continuously modified until it becomes the other, without crossing through any third color? For instance, the 5-limit just diatonic scale is a two-dimensional color that is adjacent to the 1-d line of meantone diatonic scales. This means, in some sense, that the meantone structure is a good approximation of the 5-limit just structure.

Usage

```
comparesignvecs(signvecX, signvecY)
```

Arguments

signvecX, signvecY	A pair of sign vectors to be compared. Note that these must be sign vectors, not scales themselves.

Value

Integer: `0` if the sign vectors represent the same color, `1` if they are adjacent, and `-1` if they are neither adjacent nor identical.

Examples

```
meantone_major_sv <- signvector(c(0, 2, 4, 5, 7, 9, 11))
meantone_dorian_sv <- signvector(c(0, 2, 3, 5, 7, 9, 10))
just_major <- j(dia)
just_dorian <- sim(just_major)[,2]
just_major_sv <- signvector(just_major)
just_dorian_sv <- signvector(just_dorian)

comparesignvecs(meantone_major_sv, just_major_sv)
comparesignvecs(meantone_dorian_sv, just_major_sv)
comparesignvecs(meantone_dorian_sv, just_dorian_sv)
```

`convert` *Convert between octave measurements*

Description

By default the period of a scale (normally the octave) has a size of 12 units (semitones). But it can be useful to convert to a different measurement unit, e.g. to compare a scale defined in 19-tone equal temperament (19edo) to the size of its intervals when measured in normal 12edo semitones, or vice versa.

Usage

```
convert(x, edo1, edo2)
```

Arguments

x	The set to convert as a numeric vector.
edo1	The size of the period measured in the same units as the input `x`. Numeric.
edo2	The period size to convert to. Numeric.

Value

A numeric vector the same length as `x` representing the input set converted to the desired cardinality (`edo2`).

Examples

```
maqam_rast <- c(0, 2, 3.5, 5, 7, 9, 10.5)
convert(maqam_rast, 12, 24)

perfect_fifth <- z(3/2)
lydian_scale <- sort((perfect_fifth * (0:6)) %% 12)
convert(lydian_scale, 12, 53)
```

`coord_to_edo` *Coordinate systems for scale representation*

Description

Usually, it is most intuitive to music theorists to represent a scale as a vector of the pitch-classes it contains. However, for certain computations in the setting of Modal Color Theory, it is more convenient to use a coordinate system with the "white" perfectly even scale as the origin (because this is the point where all of the hyperplanes in the arrangement defining scalar "colors" intersect). Therefore, these two functions convert between the two coordinate systems: `coord_to_edo` takes in a scale represented by its pitch classes and returns its displacement vector from "white" and `coord_from_edo` does the reverse.

Usage

```
coord_to_edo(set, edo = 12)

coord_from_edo(set, edo = 12)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`edo`	Number of unit steps in an octave. Defaults to `12`.

Details

It should be noted that the representative "white" scale used is not necessarily the *closest* one to the scale in question. Instead, it is the unique transposition of white that has 0 as its first coordinate. This is natural in the context of Modal Color Theory, which essentially always assumes transpositional equivalence with $x_0 = 0$. The closest transposition of "white" to `set` will be the one that has the same sum class as `set`, guaranteeing that the voice leading between them is "pure contrary" (Tymoczko 2011, 81ff; explored further in Straus 2018 doi:10.1215/002229097127694).

Value

Numeric vector of same length as `set`. Same scale, new coordinate system.

Examples

```
dominant_seventh_chord <- c(0, 2, 6, 9)
coord_to_edo(dominant_seventh_chord)

ait1 <- c(0, 1, 4, 6)
ait2 <- c(0, 1, 3, 7)
coord_to_edo(ait1)
coord_to_edo(ait2) # !

weitzmann_pentachord <- coord_from_edo(c(0, -1, 0, 0, 0)) # See note 53 of "Modal Color Theory"
convert(weitzmann_pentachord, 12, 60)
coord_to_edo(weitzmann_pentachord)
```

---

dft *The musical Discrete Fourier Transform of a pitch-class set*

Description

Computes the magnitudes and phases of the DFT components for a given (multi)set which can be input as either a vector of elements or as a distribution. (See Amiot (2016, doi:10.1007/9783319-455815) for an overview of applications of the DFT in this vein.) Entering a distribution takes priority over an entered `set`.

Usage

```
dft(set, distro = NULL, edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`distro`	Numeric vector representing a pitch-class distribution. Defaults to `NULL` and overrides `set` and `edo` if entered.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Details

The scaling and orientation of phases corresponds to that used in Yust (2021) `doi:10.1093/mts/mtaa0017`: phases are reported as multiples of one kth of an octave (where the set is entered in k-edo), and oriented so that the $\hat{f}_1$ component of a singleton points in the direction of the singleton (i.e. the phase of $\hat{f}_1$ for pitch class 4 is 4). This differs from the phase values use in other publications, such as Yust (2015, doi:10.1215/002229092863409). Magnitudes are not squared, following Amiot (2016) rather than Yust (2021).

Value

A 2-by-k real matrix, where k is the number of independent components. The ith column corresponds to the (i-1)th component (so that the first column gives the zeroth component). The first row gives the magnitudes of the components and the second row gives the phases. (See details regarding interpretation of the values: they are scaled by edo/(2*pi) from radians.)

Examples

```
# Compare to Yust (2021), Example 10
reich_signature <- c(0, 1, 2, 4, 5, 7, 9, 10)
dft(reich_signature)
# Magnitudes differ from Yust by squaring:
round(dft(reich_signature)[1, ]^2, digits=3)

reich_sig_distribution <- c(1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0)
dft(distro=reich_sig_distribution)

# Z-related AITs differ in phase but not magnitude:
ait1 <- c(0, 1, 4, 6)
ait2 <- c(0, 1, 3, 7)
dft(ait1)
dft(ait2)
```

`edoo` *Perfectly even scales (the color white)*

Description

Creates a perfectly even scale that divides the octave into n equal steps. Such scales serve as the origin for the hyperplane arrangements of Modal Color Theory, whence the name `edoo` for "**e**qual **d**ivision of the **o**ctave **o**rigin."

Usage

```
edoo(card, edo = 12)
```

Arguments

`card`	Number of notes in the scale. Numeric.
`edo`	Number of unit steps in an octave. Defaults to `12`.

Value

Numeric vector of length `card` representing a scale of `card` notes.

Examples

```
edoo(5)
edoo(5, edo=15)
octatonic_scale <- tc(edoo(4), c(0, 1))
print(octatonic_scale)
```

---

emb *How many instances of a subset-type exist within a scale? How many scales embed a subset?*

Description

David Lewin's EMB and COV functions: see Lewin, *Generalized Musical Intervals and Transformations* (New Haven, CT: Yale University Press, 1987), 105-120. For EMB, given a group ("CANON") of transformations which are considered to preserve a set's type, find the number of instances of that type in a larger set (`scale`). Lewin characterizes this generally, but `emb()` only offers T_n and T_n/T_nI transformation groups as available canonical groups. Conversely, Lewin's COV function asks how many instances of a `scale` type include `subset`: this is implemented as `cover()` (not `cov()`!).

Usage

```
emb(subset, scale, canon = c("tni", "tn"), edo = 12, rounder = 10)

cover(subset, scale, canon = c("tni", "tn"), edo = 12, rounder = 10)
```

Arguments

`subset`	Numeric vector of pitch-classes in any representative of the subset type (Lewin's X)
`scale`	Numeric vector of pitch-classes in the larger set to embed into (Lewin's Y)
`canon`	What transformations should be considered equivalent? Defaults to "tni" (using standard set classes) but can be "tn" (using transposition classes)
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

Integer: count of `subset` or `scale` types satisfying the desired relation.

Examples

```
emb(c(0, 4, 7), sc(7, 35))
emb(c(0, 4, 7), sc(7, 35), canon="tn")

# Works for continuous pc-space too:
emb(j(1, 3, 5), j(dia))
emb(j(1, 2, 3, 5, 6), j(dia))
emb(j(1, 2, 4, 5, 6), j(dia), canon="tn")

emb(c(0, 4, 7), c(0, 1, 3, 7))
emb(c(0, 4, 7), c(0, 1, 3, 7), canon="tn")

emb(c(0, 4), c(0, 4, 8))
cover(c(0, 4), c(0, 4, 8))

harmonic_minor <- c(0, 2, 3, 5, 7, 8, 11)
cover(c(0, 4, 8), harmonic_minor)
cover(c(0, 4, 8), harmonic_minor, canon="tn")
```

`eps` *The brightness ratio*

Description

Section 3.3 of "Modal Color Theory" describes a "brightness ratio" which characterizes the modes of a scale in terms of how well "sum brightness" acts as a proxy for "voice-leading brightness." Scales with a brightness ratio less than 1 are pretty well behaved from this perspective, while ones with a brightness ratio greater than 1 are poorly behaved. When the brightness ratio is 0, sum brightness and voice-leading brightness give exactly the same results. (This can happen for sets on two extremes: those like the diatonic scale which are well formed and those like the Weitzmann scales, which differ from "white" in only one scale degree.)

I wish I had come up with a more descriptive name than "brightness ratio" for this property, because it's not really a ratio of brightness in the sense you might expect (i.e. "this scale is 20% bright"). Rather, it's a ratio of two brightness-related properties, `delta` and `eps`. "Modal Color Theory" (p. 20) offers definitions of these. Delta is "the largest sum difference between (voice-leading) incomparable modes," with value 0 by definition if all of the modes are comparable. ("This, in a sense, is a measure of how badly voice-leading brightness breaks down from the perspective of sum brightness.") **Eps**ilon "represents the smallest sum difference between non-identical but comparable modes." This is harder to give an intuitive gloss on, but my attempt in "MCT" was "Essentially, epsilon measures the finest distinction that voice-leading brightness is capable of parsing."

The brightness ratio (`ratio`) itself is simply delta divided by epsilon.

Usage

```
eps(set, edo = 12, rounder = 10)
```

```
delta(set, edo = 12, rounder = 10)

ratio(set, edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

Single non-negative numeric value

Examples

```
harmonic_minor <- c(0, 2, 3, 5, 7, 8, 11)
hypersaturated_harmonic_minor <- saturate(2, harmonic_minor)
c(delta(harmonic_minor), eps(harmonic_minor))
c(delta(hypersaturated_harmonic_minor), eps(hypersaturated_harmonic_minor))

# Delta and epsilon depend on the precise scale, but ratio() is constant on a hue
ratio(harmonic_minor)
ratio(hypersaturated_harmonic_minor)

#### Sort all 12tet heptachords by brightness ratio
heptas12 <- unique(apply(combn(12, 7), 2, primeform),MARGIN=2)
hepta_ratios <- apply(heptas12, 2, ratio)
sorted_heptas <- heptas12[, order(hepta_ratios)]
colnames(sorted_heptas) <- apply(sorted_heptas, 2, fortenum)
sorted_heptas

#### Compare evenness to ratio for 12tet hetpachords
plot(apply(heptas12, 2, evenness), hepta_ratios, xlab="Evenness", ylab="Brightness Ratio")
```

`evenness` *How even is a scale?*

Description

Calculates the distance from a set to the nearest perfectly even division of the octave, which will *not* be the one with a first entry of 0, unlike almost every other usage in this package. That's because, for most purposes, we do want to distinguish between different modes of a set, but it seems counterintuitive to me to say that one mode of a scale is less even than another. Since this value is a distance from the perfectly even ("white") scale, lower values indicate more evenness.

Usage

```
evenness(
  set,
  method = c("euclidean", "taxicab", "chebyshev", "hamming"),
  edo = 12,
  rounder = 10
)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`method`	What distance metric should be used? Defaults to `"euclidean"` (unlike most functions with a method parameter in musicMCT) but can be `"taxicab"`, `"chebyshev"`, or `"hamming"`.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Details

Note that the values this function returns depend on what measurement unit you're using (i.e. are you in 12edo or 16edo?). Their absolute value isn't terribly significant: you should only make relative comparisons between calculations done with the same value for `edo`.

Currently, `methods` other than "Euclidean" are somewhat experimental.

Value

Single non-negative numeric value

Examples

```
evenness(c(0, 4, 8))
evenness(c(0, 4, 7)) < evenness(c(0, 1, 2))

dim_triad <- c(0, 3, 6)
sus_2 <- c(0, 2, 7)
coord_to_edo(dim_triad)
coord_to_edo(sus_2)
evenness(dim_triad) == evenness(sus_2)
```

flex_points *Voice-leading inflection points*

Description

When considering an n-note set's potential voice leadings to transpositions of a goal (along the lines of `vl_rolodex()` and `tndists()`), there will always be some transposition in continuous pc-space for which a given modal rotation is the best potential target for voice leading. (That is, there is always some `x` such that `whichmodebest(set, tn(set, x)) == k` for any `k` between `1` and `n`.) Moreover, there will always be a transposition level at the boundary between two different ideal modes, where both modes require the same amount of voice leading work. `flex_points()` identifies those inflection points where one mode gives way to another. (Note: `flex_points()` identifies these points by numerical approximation, so it may not give exact values. For more precision, increase the value of `subdivide`.)

Usage

```
flex_points(
  set,
  goal = NULL,
  method = c("taxicab", "euclidean", "chebyshev", "hamming"),
  subdivide = 100,
  edo = 12,
  rounder = 10
)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`goal`	Numeric vector like set: what is the tn-type of the voice leading's destination? Defaults to `NULL`, in which case the function uses `set` as the tn-type.
`method`	What distance metric should be used? Defaults to `"taxicab"` but can be `"euclidean"`, `"chebyshev"`, or `"hamming"`.
`subdivide`	Numeric: how many small amounts should each edo step be divided into? Defaults to `100`.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

Numeric vector of the transposition indices that are inflection points. Length of result matches size of `set`, except in the case of some multisets, which can have fewer inflection points.

Examples

```
major_triad_12tet <- c(0, 4, 7)
major_triad_just <- z(1, 5/4, 3/2)
major_triad_19tet <- c(0, 6, 11)

flex_points(major_triad_12tet, method="euclidean", subdivide=1000)
flex_points(major_triad_just, method="euclidean", subdivide=1000)

# Note that the units of measurement correspond to edo.
# The value 3.16 here corresponds to exactly 1/6 of an octave.
flex_points(major_triad_19tet, edo=19)
```

`fortenum` *Forte number from set class*

Description

Given a pitch-class set (in 12edo only), look up Forte 1973's catalog number for the corresponding set class.

Usage

```
fortenum(set)
```

Arguments

`set`	Numeric vector of pitch-classes in the set

Value

Character string in the form "n-x" where n is the number of notes in the set and x is the ordinal position in Forte's list.

Examples

```
fortenum(c(0, 4, 7))
fortenum(c(0, 3, 7))
fortenum(c(4, 8, 11))
```

`fortenums` *Allen Forte's list of set classes*

Description

For compatibility with music theory's traditional pitch-class set theory, whose landmark text is Allen Forte's 1973 *The Structure of Atonal Music*, the data set `fortenums` hard-codes the ordinal positions of 12-equal pitch-class set classes from Allen Forte's list. This allows us to look up specific set classes from Forte numbers or vice versa. `sc()` does the former and `fortenum()` does the latter. There's very little need to ever interact with the file `fortenums` itself: you should be able to get anything you need from this data through either `sc()` or `fortenum()`.

Note that `primeform()` in `musicMCT` uses Rahn's algorithm rather than Forte's for finding a canonical representative of each set class. Consequently, the entries of `fortenums` also use Rahn's prime forms rather than Forte's.

Usage

```
fortenums
```

Format

A list of length 12. The nth entry of the list corresponds to set classes of cardinality n. Each list entry is a vector of character strings; every element of the vector contains a Rahn prime form as a comma-delimited string. These prime forms are ordered in the same sequence as Forte's list. Thus, for instance, the set class of the minor triad is represented by the string `"0, 3, 7"`, which is the 11th element in `fortenums[[3]]`.

Source

Forte, Allen. 1973. *The Structure of Atonal Music*. New Haven, CT: Yale University Press. Appendix 1, pp. 179-181.

`fpmod` *Modulo division with rounding*

Description

When working with sets in continuous pitch-class spaces (i.e., where octave equivalence is needed), R's normal operator for modulo division `%%` does not always give ideal results. Values that are very close to (but below) the octave appear to be far from 0. This function uses rounding to give octave-equivalent results that music theorists expect.

Usage

```
fpmod(set, edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

Numeric vector the same length as `set`

Examples

```
really_small <- 1e-13
c_major <- c(0, 4, 7, 12-really_small)
c_major %% 12
fpmod(c_major, 12)
```

`fpunique` *Unique real values up to some tolerance*

Description

Working with scales in continuous pitch space, many pitches and intervals are irrationals represented as **f**loating **p**oint numbers. This can cause arithmetic and rounding errors, leading to it looking like there are more distinct pitches/intervals in the set than there really are. Use `fpunique` rather than `base::unique()` whenever you handle scales in continuous pitch space.

Usage

```
fpunique(x, MARGIN = 0, rounder = 10)
```

Arguments

`x`	Numeric array whose unique elements are to be determined
`MARGIN`	Numeric `0`, `1`, or `2` depending on whether you want unique individual numbers, unique rows, or unique columns, respectively. Defaults to `0`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Details

Sometimes you may need to adjust the tolerance (`rounder`) to get correct results, especially if you have done several operations in a row which may have introduced rounding errors.

Value

Numeric array of unique elements (vector if `MARGIN` is 0; matrix otherwise)

Examples

```
just_dia <- j(dia)
intervals_in_just_dia <- sort(as.vector(sim(just_dia)))
failed_unique_intervals <- unique(intervals_in_just_dia)
fpunique_intervals <- fpunique(intervals_in_just_dia)
length(failed_unique_intervals)
length(fpunique_intervals)
```

`get_relevant_rows` *Which hyperplanes affect a given generic interval?*

Description

Given an `ineqmat` (i.e. a matrix representing a hyperplane arrangement), this function tells us which of those hyperplanes affect a specific generic interval size. (One specific application of this is is `step_signvector()`, which pays attention only to the comparisons between step sizes in a scale.)

Usage

```
get_relevant_rows(generic_intervals, ineqmat)
```

Arguments

`generic_intervals`	A vector of one or more integers representing generic intervals that can be found within the scale. Unisons are `0`, generic steps are `1`, etc.
`ineqmat`	The matrix of hyperplane normal vectors that you want to search.

Value

Vector of integers indicating the relevant hyperplanes from `ineqmat`

Examples

```
heptachord_ineqmat <- getineqmat(7)
heptachord_step_comparisons <- get_relevant_rows(1, heptachord_ineqmat)

# Create an ineqmat that attends only to the quality of (024) trichordal
# subsets in a heptachord.
heptachord_triads <- get_relevant_rows(c(0, 2, 4), heptachord_ineqmat)
triads_in_7_ineqmat <- heptachord_ineqmat[heptachord_triads,]

# Now, the following two heptachords have different colors
# but the same pattern of (024) trichordal subsets, so their signvector
# using triads_in_7_ineqmat is identical:
heptachord_1 <- convert(c(0, 1, 3, 6, 8, 12, 13), 17, 12)
heptachord_2 <- convert(c(0, 1, 3, 5, 7, 10, 11), 14, 12)
colornum(heptachord_1) == colornum(heptachord_2)
sv_1 <- signvector(heptachord_1, ineqmat=triads_in_7_ineqmat)
sv_2 <- signvector(heptachord_2, ineqmat=triads_in_7_ineqmat)
isTRUE(all.equal(sv_1, sv_2))
subset_varieties(c(0, 2, 4), heptachord_1, unique=FALSE)
subset_varieties(c(0, 2, 4), heptachord_2, unique=FALSE)
# Both have identical qualities for triads on scale degree 3, 5, and 7,
# which you can see by comparing columns 3, 5, and 7 in the two
# matrices above.
```

howfree — *Count a scale's degrees of freedom*

Description

Some scalar structures can vary their specific pitches much more flexibly than others while retaining the same overall "color." For instance, the meantone family of diatonic scales is generated by a line of fifths and can only vary along one dimension: the size of the generating fifth. This literally defines a line in the MCT geometry, and if the scale moves off that line it ceases to have the same structure as the diatonic scale. (Notably, it stops being non-degenerate well-formed.) By contrast, the 5-limit just diatonic scale is defined by two distinct parameters: the size of its major third and the size of its perfect fifth. See "Modal Color Theory," pp. 26-27, for more discussion.

Usage

```
howfree(set, ineqmat = NULL, edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`ineqmat`	Specifies which hyperplane arrangement to consider. By default (or by explicitly entering "mct") it supplies the standard "Modal Color Theory" arrangements of `getineqmat()`, but can be set to strings "white," "black", "gray", "roth", "infrared", "pastel", "rosy", "infrared", or "anaglyph", giving the ineqmats of `make_white_ineqmat()`, `make_black_ineqmat()`, `make_gray_ineqmat()`, `make_roth_ineqmat()`, `make_infrared_ineqmat()`, `make_pastel_ineqmat()`, `make_rosy_ineqmat()`, `make_infrared_ineqmat()`, or `make_anaglyph_ineqmat()`. For other arrangements, this parameter accepts explicit matrices.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

Single non-negative integer

Examples

```
c_natural_minor <- c(0, 2, 3, 5, 7, 8, 10)
c_melodic_minor <- c(0, 2, 3, 5, 7, 9, 11)
just_diatonic <- j(dia)
howfree(c_natural_minor)
howfree(c_melodic_minor)
howfree(just_diatonic)

howfree(c(0, 4, 7))
howfree(c(0, 4, 7), ineqmat="white")

howfree(c(0, 2, 6), ineqmat="mct")
howfree(c(0, 2, 6), ineqmat="roth")
```

`ianring` *Look up a scale at Ian Ring's* Exciting Universe of Music Theory

---

Description

Ian Ring's website *The Exciting Universe of Music Theory* is a comprehensive and useful compilation of information about pitch-class sets in twelve-tone equal temperament. It tracks many properties that musicMCT is unlikely to duplicate, so this function opens the corresponding page for a pc-set in your browser. This only works for sets in 12-edo which include pitch-class 0.

Usage

```
ianring(set)
```

Arguments

`set`	Numeric vector of pitch-classes in the set

Value

Invisibly, the integer which Ring's site uses to index the input `set`. The main purpose of the function is its side effect of opening a page of Ring's site in a browser.

Examples

```
c_major <- c(0, 2, 4, 5, 7, 9, 11)
c_major_value <- ianring(c_major)
print(c_major_value)
# And indeed you should find information about the major scale
# at https://ianring.com/musictheory/scales/2741
```

```
ianring(c(0, 2, 3, 7, 8))
```

`ifunc` *All intervals from one set to another*

Description

David Lewin's interval function (IFUNC) calculates all the intervals from some source set `x` to some goal set `y`. See Lewin, *Generalized Musical Intervals and Transformations* (New Haven, CT: Yale University Press, 1987), 88. Lewin's definition of the IFUNC depends on the GIS it applies to, but this package's `ifunc()` is less flexible. It uses only ordered pitch-class intervals as the group of IVLS to be measured. Its intervals can, however, be any continuous value and are not restricted to integers mod `edo`. The format of the result depends on whether non-integer intervals occur.

Usage

```
ifunc(
  x,
  y = NULL,
  edo = 12,
  rounder = 10,
  display_digits = 2,
  show_zeroes = TRUE
)
```

Arguments

`x`	The source set from which the intervals originate
`y`	The goal set to which the intervals lead. Defaults to `NULL`, in which case `ifunc()` gives the intervals from `x` to itself.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.
`display_digits`	Integer: how many digits to display when naming any non-integral interval sizes. Defaults to 2.
`show_zeroes`	Boolean: if `x` and `y` belong to a single mod `edo` universe, should `0` values be listed for any intervals mod edo which do not occur in their IFUNC? Defaults to `TRUE`.

Value

Numeric vector counting the number of occurrences of each interval. The `names()` of the result indicate which interval size is counted by each entry. If `x` and `y` both belong to a single mod `edo` universe (and `show_zeroes=TRUE`), the result is a vector of length `edo` and includes explicit `0` results for missing intervals. If `x` and `y` must be measured in continuous pitch-class space, no missing intervals are identified (since there would be infinitely many to list).

Examples

```
ifunc(c(0, 3, 7))
ifunc(c(0, 3, 7), c(0, 4, 7))
ifunc(c(0, 4, 7), c(0, 3, 7))

ifunc(c(0, 2, 4, 7, 9), show_zeroes=FALSE)

just_dia <- j(dia)
ifunc(just_dia)
ifunc(just_dia, display_digits=4)

# See Lewin, GMIT p. 89:
lewin_x <- c(4, 10)
lewin_y1 <- c(9, 1, 5)
lewin_y2 <- c(7, 11, 9)
isTRUE(all.equal(ifunc(lewin_x, lewin_y1), ifunc(lewin_x, lewin_y2)))
apply(cbind(lewin_y1, lewin_y2), 2, fortenum)
```

`ineqmats` *Hyperplane arrangements for MCT spaces*

Description

The data file `ineqmats` represents the hyperplane arrangements at the core of Modal Color Theory as matrices containing the hyperplanes' normal vectors. See Appendix 1.2 of Sherrill (2025) for a discussion of the format of these matrices. The matrices can be generated on the fly by `makeineqmat()`, but for large computations it's faster simply to call on precalculated data rather than to run `makeineqmat()` many thousands of times. Thus the object `ineqmats` saves the inequality matrices for scales of cardinality 1-53, to be called upon by `getineqmat()`.

Usage

```
ineqmats
```

Format

`ineqmats` A list with 53 entries. The nth entry of the list gives the inequality matrix for n-note scales. Each inequality matrix itself is an m by (n+1) matrix, where m is an element of OEIS A034828 (see Sherrill 2025, 40-42). The last column of the matrix contains an offset related to whether any of the generic intervals "wrap around the octave," as e.g. the third from 7 to 2 does in a heptachord. This column is linearly dependent on the previous n columns, which contain the coefficients of the hyperplane's normal vectors. That is, the first row of the matrix (dropping its last entry) is the normal vector for the first hyperplane of the arrangement, and so on.

Source

The data in `ineqmats` can be recreated with the command `sapply(2:53, makeineqmat)` and then appending `integer(0)` as the first element of the list (for the case of one-note scales which have no pairwise interval comparisons and therefore need a matrix of size 0).

`ineqsym` *Symmetries of hyperplane arrangements define equivalent scales*

Description

Produces scales of different colors which have equivalent scalar properties. The hyperplane arrangements of MCT have three types of symmetry, which allows us to find scales at different but equivalent points in the arrangement. Such scales will be nearly structurally identical in most senses although their specific intervals will be rather different. See details for a discussion of the symmetries involved.

Usage

```
ineqsym(set, a = 1, b = 0, involution = FALSE, edo = 12)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`a`	Integer: controls permutations of generic intervals. Must be coprime to the size of the set. Defaults to `1`.
`b`	Integer: controls modal rotation. Defaults to `0`.
`involution`	Boolean: controls involutional symmetry. Defaults to `FALSE`.
`edo`	Number of unit steps in an octave. Defaults to `12`.

Details

Two symmetries of the MCT hyperplane arrangement are familiar. One is modal "rotation": two modes of the same scale must have equivalent structures, by the defining relations of the theory. The parameter `b` controls these rotations. The second familiar symmetry is *involution* (see "Modal Color Theory," 32). Set parameter `involution` to TRUE to apply this symmetry. The more interesting symmetry of the MCT arrangements is controlled by parameter `a`. This symmetry allows us to permute the roles of the scale's generic intervals in its scalar makeup. For instance, non-degenerate well-formed scales (see `iswellformed()` are all generated by a particular generic interval. The familiar diatonic scale is generated by its generic fourths, whereas another well-formed scale like (0, 2, 3, 5, 6, 7, 9) in 10edo (with step-word LSLSSLS) is generated by its generic sixths. We can permute the hyperplanes of the heptachordal MCT arrangement so that the overall structure is preserved but the diatonic scale is mapped onto LSLSSLS. In general, the permutations of `ineqsym()` allow us to map any non-degenerate well-formed scale onto any other: they form an orbit under the symmetries of the space. This gives another sense in which "well-formedness" is a large family of scale structures. That sense generalizes to *all* scales, not just ones that are highly regular like well-formed scales.

In short, `ineqsym()` preserves many scalar properties, including:

- `countsvzeroes()` and `svzero_fingerprint()`
- `howfree()`
- `ratio()`, `delta()`, and `eps()`
- `brightnessgraph()` structure
- `evenness()`
- `isgwf()` and *a fortiori* `iswellformed()`
- Number and respective properties of adjacent colors
- `spectrumcount()` up to permutation of the values

Value

Numeric vector representing a scale of same length as `set`. Default parameters determine the identity symmetry and will return `set` itself.

Examples

```
wt_plus_1 <- sc(7,33)
equiv_scale <- ineqsym(wt_plus_1, 3, 2)
both_scales <- cbind(wt_plus_1, equiv_scale)
freedoms <- apply(both_scales, 2, howfree)
evennesses <- round(apply(both_scales, 2, evenness), 3)
svzeroes <- apply(both_scales, 2, countsvzeroes)
ratios <- round(apply(both_scales, 2, ratio), 3)
spectra <- apply(apply(both_scales, 2, spectrumcount), 2, toString)
cbind(freedoms, evennesses, svzeroes, ratios, spectra)
brightnessgraph(wt_plus_1)
brightnessgraph(equiv_scale)
```

`intervalspectrum` *Specific sizes corresponding to each generic interval*

Description

As defined by Clough and Myerson 1986 (doi:10.1080/00029890.1986.11971924), an "interval spectrum" is a list of all the specific (or "chromatic") intervals that occur as instances of a single generic (or "diatonic") interval within some reference scale. For instance, in the usual diatonic scale, the generic interval 1 (a "step" in the scale) comes in two specific sizes: 1 semitone and 2 semitones. Therefore its interval spectrum $\langle 1 \rangle = \{1, 2\}$. These functions calculates the spectrum for every generic interval within a set and return either a list of specific values in each spectrum or a summary of how many distinct values there are.

Usage

```
intervalspectrum(set, edo = 12, rounder = 10)

spectrumcount(set, edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

`intervalspectrum` returns a list of length one less than `length(set)`. The nth entry of the list represents the specific sizes of generic interval n. `spectrumcount` returns a vector that reports the length of each entry in that list (i.e. the number of distinct specific intervals for each generic interval).

Examples

```
intervalspectrum(sc(7,35))
qcm_fifth <- meantone_fifth()
qcm_dia <- sort(((0:6)*qcm_fifth)%%12)
intervalspectrum(qcm_dia)
just_dia <- 12 * log2(c(1, 9/8, 5/4, 4/3, 3/2, 5/3, 15/8))
intervalspectrum(just_dia)

spectrumcount(just_dia) # The just diatonic scale is trivalent.

# Melodic minor nearly has "Myhill's Property" except for its 3 sizes of fourth and fifth
spectrumcount(sc(7,34))
```

---

isgwf *Is a scale n-wise well formed?*

Description

Tests whether a scale has a generalized type of well formedness (pairwise or n-wise well formedness).

Usage

```
isgwf(set, stepword = NULL, allow_de = FALSE, edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`stepword`	A vector representing the ranked step sizes of a scale (e.g. `c(2, 2, 1, 2, 2, 2, 1)` for the diatonic). The distinct values of the `setword` should be consecutive integers. If you want to test a step word instead of a list of pitch classes, `set` must be entered as `NULL`.
`allow_de`	Should the function test for degenerate well-formed and distributionally even scales too? Defaults to `FALSE`.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Details

David Clampitt's 1997 dissertation ("Pairwise Well-Formed Scales: Structural and Transformational Properties," SUNY Buffalo) offers a generalization of the notion of well-formedness from 1-dimensional structures with a single generator to 2-dimensional structures that mediate between two well-formed scales. Ongoing research suggests that this can be extended further to "n-wise" or "general" well-formedness, though n-wise well-formed scales are increasingly rare as n grows larger.

Value

Boolean: is the set n-wise well formed?

Examples

```
meantone_diatonic <- c(0, 2, 4, 5, 7, 9, 11)
just_diatonic <- j(dia)
some_weird_thing <- convert(c(0, 1, 3, 6, 8, 12, 14), 17, 12)
example_scales <- cbind(meantone_diatonic, just_diatonic, some_weird_thing)

apply(example_scales, 2, howfree)
apply(example_scales, 2, isgwf)
```

`isproper` *Rothenberg propriety*

Description

Rothenberg (1978, doi:10.1007/BF01768477) identifies a potentially desirable trait for scales which he calls "propriety." Loosely speaking, a scale is proper if its specific intervals are well sorted in terms of the generic intervals they belong to. A scale is *strictly* proper if, given two generic sizes g and h such that g < h, every specific size corresponding to g is smaller than every specific size corresponding to h. A scale if improper if any specific size of g is larger than any specific size of h. An *ambiguity* occurs if any size of g equals any size of h: scales with ambiguities are weakly but not strictly proper.

Usage

```
isproper(set, strict = FALSE, edo = 12, rounder = 10)

has_contradiction(set, edo = 12, rounder = 10)

strictly_proper(set, edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`strict`	Boolean: should only strictly proper scales pass?
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

Boolean which answers whether the input satisfies the property named by the function

See Also

`make_roth_ineqmat()` creates an `ineqmat` for a hyperplane arrangement that lets you explore propriety-related issues in finer detail.

Examples

```
c_major <- c(0, 2, 4, 5, 7, 9, 11)
has_contradiction(c_major)
strictly_proper(c_major)
isproper(c_major)
isproper(c_major, strict=TRUE)

isproper(j(dia), strict=TRUE)

pythagorean_diatonic <- sort(((0:6)*z(3/2))%%12)
isproper(pythagorean_diatonic)
has_contradiction(pythagorean_diatonic)
```

`iswellformed` *Well-formedness, Myhill's property, and/or moment of symmetry*

Description

Tests whether a scale has the property of "well-formedness" or "moment of symmetry."

Usage

```
iswellformed(set, stepword = NULL, allow_de = FALSE, edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`stepword`	A vector representing the ranked step sizes of a scale (e.g. `c(2, 2, 1, 2, 2, 2, 1)` for the diatonic). The distinct values of the `setword` should be consecutive integers. If you want to test a step word instead of a list of pitch classes, `set` must be entered as `NULL`.
`allow_de`	Should the function test for degenerate well-formed and distributionally even scales too? Defaults to `FALSE`.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Details

The three concepts of "well-formedness," "Myhill's property," and "moment of symmetry" refer to nearly the same scalar property, generalizing one of the most important features of the familiar diatonic scale. See Clough, Engebretsen, and Kochavi (1999, 77; doi:10.2307/745921) for a useful discussion of their relationships. In short, except for a few edge cases, a scale possesses these properties if it is generated by copies of a single interval (as the Pythagorean diatonic is generated by the ratio 3:2) and all copies of the generator belong to the same generic interval (as the 3:2 generator of the diatonic always corresponds to a "fifth" within the scale). Such a structure typically means that all generic intervals come in 2 distinct sizes, which is the definition of "Myhill's property." An exception occurs if the generator manages to produce a perfectly even scale, e.g. when the whole tone scale is generated by 6 copies of `1/6` of the octave. Such a scale lacks Myhill's property and Carey & Clampitt (1989, 200; doi:10.2307/745935) call such cases "degenerate well-formed." Instead of Myhill's property, such scales have only 1 specific value in each `intervalspectrum()`.

Clough, Engebretsen, and Kochavi define a related concept, distributionally even scales, which include the hexatonic and octatonic scales (Forte sc6-20 and sc8-28). Such scales are in some sense halfway between "degenerate" and "non-degenerate well-formed" because some of their interval spectra have 1 element while others have 2. From another perspective, distributionally even scales are non-degenerate well formed with a period smaller than the octave (e.g. as the hexatonic scales 1-3 step pattern repeats every third of an octave).

The term "moment of symmetry" refers to the non-degenerate well-formed scales and was coined by Erv Wilson 1975 (cited in Clough, Engebretsen, and Kochavi). It tends to be more widely used in microtonal music theory, e.g. `https://en.xen.wiki/w/MOS_scale`.

Scales with this property have considerably interesting voice-leading properties and are some of the most important landmarks in the geometry of MCT. See "Modal Color Theory," pp. 14, 17, 29, 33-34, and 36-37. A substantial portion of MCT amounts to an attempt to generalize ideas developed for MOS/NDWF scales to all scale structures.

Value

Boolean answering "Is the scale MOS (with equivalence interval equal to the period)?" (if allow_de=FALSE) or "Is the scale well-formed in any sense?" (if allow_de=TRUE).

Examples

```
iswellformed(sc(7, 35))
iswellformed(c(0, 2, 4, 6))
iswellformed(c(0, 1, 6, 7))
iswellformed(c(0, 1, 6, 7), allow_de=TRUE)
iswellformed(NULL, stepword=c(2, 2, 1, 2, 1, 2, 1))
```

`isym` *Test for inversional symmetry*

Description

Is the pc-set **i**nversionally **sym**metrical? That is, does it map onto itself under T_nI for some appropriate n? `isym()` can return either `TRUE`/`FALSE` or an index of symmetry but defaults to the former. `isym_index()` is a simple wrapper for `isym()` that returns the latter. `isym_degree()` counts the total number of inversional symmetries (i.e. the number of distinct inversional axes of symmetry).

Usage

```
isym(set, return_index = FALSE, edo = 12, rounder = 10)

isym_index(set, ...)

isym_degree(set, ...)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`return_index`	Should the function return a specific index at which the set is symmetrical? Defaults to `FALSE`.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.
`...`	Arguments to be passed to `isym()`

Details

`isym()` is evaluated by asking whether, for some appropriate rotation, the step-interval series of the given set is equal to the step-interval series of the set's inversion. This is designed to work for sets in continuous pc-space, not just integers mod k. Note also that this calculates abstract pitch-class symmetry, not potential symmetry in pitch space. (See the second example.)

Value

`isym()` returns the Boolean value from testing for symmetry, unless `return_index=TRUE`, in which case isym() and `isym_index()` return a numeric value for one index of inversion at which the set is symmetrical. If the set is not inversionally symmetrical, they will return `NA`. `isym_degree()` gives the degree of inversional symmetry.

Examples

```
#### Mod 12
isym(c(0, 1, 5, 8))
isym(c(0, 2, 4, 8))

#### Continuous Values
qcm_fifth <- meantone_fifth()
qcm_dia <- sort(((0:6)*qcm_fifth)%%12)
just_dia <- j(dia)
isym(qcm_dia)
isym(just_dia)

#### Rounding matters:
isym(qcm_dia, rounder=15)

### Index and Degree
hexatonic_scale <- c(0, 1, 4, 5, 8, 9)
isym_index(hexatonic_scale) # Only returns one suitable index
isym_degree(hexatonic_scale)
```

`ivec` *Interval-class vector*

Description

The classic summary of a set's dyadic subset content from pitch-class set theory. The name `ivec` is short for **i**nterval-class **vec**tor.

Usage

```
ivec(set, edo = 12)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`edo`	Number of unit steps in an octave. Defaults to `12`.

Value

Numeric vector of length `floor(edo/2)`

Examples

```
ivec(c(0, 1, 4, 6))
ivec(c(0, 1, 3, 7))

#### Z-related sextuple in 24edo:
sextuple <- matrix(
  c(0, 1, 2, 6, 8, 10, 13, 16,
  0, 1, 3, 7, 9, 11, 12, 17,
  0, 1, 6, 8, 10, 13, 14, 16,
  0, 1, 7, 9, 11, 12, 15, 17,
  0, 1, 2, 4, 8, 10, 13, 18,
  0, 2, 3, 4, 8, 10, 15, 18), nrow=6, byrow=TRUE)
apply(sextuple, 1, ivec, edo=24) # The ic-vectors are the 6 identical columns of the output matrix
```

---

j *Convenient just-intonation intervals and scales*

Description

It's not hard to define a just interval from a frequency ratio: it only requires an input like `12*log2(freq_ratio)`. That gets pretty tiresome if you're doing this a lot, though, so for convenience `musicMCT` includes a `j()` function (not related to Clough and Douthett's J function but it wishes it was). `j()` is designed to behave a lot like base R's `c()` in the way that you'd use it to define a scale (see the examples below). The inputs that this can take are limited and hard-coded, since there's no systematic way to define short hands for every potential just interval. In general, the logic is that individual digits refer to major intervals up from the tonic in the 5-limit just diatonic scale. The prefix "m" to a number (e.g. "m3") gives the equivalent minor version of the interval. If you just want the entire 5-limit diatonic, you can enter `dia`.

Usage

```
j(..., edo = 12)
```

Arguments

`...`	One or more names that will be matched to just intervals. You can enter these as strings, but for convenience sake you needn't. Here are the currently accepted inputs, their meaning, and their return value:

- `1`: perfect 1th (0 semitones)
- `u`: unison (0 semitones)
- `synt`: syntonic comma (~.215 semitones)
- `pyth`: Pythagorean comma (~.235 semitones)
- `l`: Pythagorean limma (256:243 or ~.9 semitones)
- `s`: 5-limit just semitone (16:15 or ~1.12 semitones)
- `st`: 5-limit just semitone (16:15 or ~1.12 semitones)
- `m2`: 5-limit minor second (16:15 or ~1.12 semitones)

- `h`: half step (16:15 or ~1.12 semitones)
- `a`: Pythagorean apotome (2187:2048 or ~1.14 semitones)
- `mt`: 5-limit minor tone (10:9 or ~1.82 semitones)
- `2`: 3-limit major second (9:8 or ~2.04 semitones)
- `t`: 3-limit whole tone (9:8 or ~2.04 semitones)
- `w`: whole tone (9:8 or ~2.04 semitones)
- `wt`: whole tone (9:8 or ~2.04 semitones)
- `sept`: 7-limit (septimal) whole tone (8:7 or ~2.31 semitones)
- `sdt`: 3-limit semiditone (32/27 or ~2.94 semitones)
- `pm3`: Pythagorean minor third (32/27 or ~2.94 semitones)
- `m3`: 5-limit minor third (6:5 or ~3.16 semitones)
- `3`: 5-limit major third (5:4 or ~3.86 semitones)
- `M3`: 5-limit major third (5:4 or ~3.86 semitones)
- `dt`: 3-limit ditone (81/64 or ~4.08 semitones)
- `4`: 3-limit perfect fourth (4:3 or ~4.98 semitones)
- `utt`: 11-limit tritone (11:8 or ~5.51 semitones)
- `stt`: 7-limit tritone (7:5 or ~5.83 semitones)
- `jtt`: 5-limit tritone (45:32 or ~5.90 semitones)
- `ptt`: 3-limit tritone (729:512 or ~6.12 semitones)
- `pd5`: 3-limit diminished fifth (1024/729 or ~5.88 semitones)
- `5`: 3-limit perfect fifth (3:2 or ~7.02 semitones)
- `m6`: 5-limit minor sixth (8:5 or ~8.14 semitones)
- `6`: 5-limit major sixth (5:3 or ~8.84 semitones)
- `pm7`: Pythagorean minor seventh (16:9 or ~9.96 semitones)
- `m7`: 5-limit minor seventh (9:5 or ~10.18 semitones)
- `7`: 5-limit major seventh (15:8 or ~10.88 semitones)
- `8`: 2-limit perfect octave (2:1 or 12 semitones)
- `dia`: the complete 5-limit diatonic scale

`edo`	Number of unit steps in an octave. Defaults to `12`.

Value

Numeric vector representing the input just intervals converted to `edo` unit steps per octave

See Also

`z()` as a shortcut for 12*log2(x) when a just interval you need isn't defined for `j()`.

Examples

```
major_triad <- j(1,3,5)
isTRUE(all.equal(major_triad, j(u, M3, "5")))

isTRUE(all.equal(j(dia), j(1,2,3,4,5,6,7)))
```

```
# How far is the twelve-equal major scale from the 5-limit just diatonic?
dist(rbind(c(0,2,4,5,7,9,11), j(dia)))

# Is 53-equal temperament a good approximation of the 5-limit just diatonic?
j(dia, edo=53)
```

---

makeineqmat *Define hyperplanes for the Modal Color Theory arrangements*

Description

As described in Appendix 1.2 of "Modal Color Theory," information about the defining hyperplane arrangements is stored as a matrix containing the hyperplanes' normal vectors as rows. (Because these are **mat**rices and they correspond ultimately to the intervallic **ineq**ualities that define MCT geometry, this package refers to them as ineqmats, and sometimes to the individual hyperplanes as `ineqs`.) These have already been computed and are stored as data in this package (`ineqmats`) for cardinalities up to 53, but they can be recreated from scratch with `makeineqmat`. This might be useful if for some reason you need to deal with a huge scale and therefore want to use an arrangement whose matrix isn't already saved. Note that a call like `makeineqmat(60)` may take a dozen or more seconds to run (and at sizes that large, the arrangement is terribly complex, with ~17K distinct hyperplanes).

`getineqmat` tests whether the matrix already exists for the desired cardinality. If so, it is retrieved; if not, it is created using `makeineqmat`.

Usage

```
makeineqmat(card)

getineqmat(card)
```

Arguments

`card`	The cardinality of the scale(s) to be studied

Value

A matrix with `card+1` columns and roughly `card^(3)/8` rows

Examples

```
makeineqmat(2) # Simple: is step 1 > step 2?
makeineqmat(3) # Simple: step 1 > step 2? step 1 > step 3? step 2 > step 3?
makeineqmat(7) # Okay...
ineqmat20 <- makeineqmat(20)
dim(ineqmat20) # Yikes!
```

make_anaglyph_ineqmat *Define hyperplanes for cross-type voice leadings*

Description

Voice leadings between members of a single set class are well characterized by the Modal Color Theory arrangements of `makeineqmat()`. Those arrangements do not tell the whole story for relationships between inequivalent sets. (For instance, under what circumstances are two `brightnessgraph()` structures equivalent when `set` and `goal` belong to different set classes?) Such relationships are described by the "anaglyph" arrangements produced by this function. (The name for the arrangements alludes to those 20th-century 3D movie glasses which produce a stereoscopic effect by using lenses of different colors for each eye. Like those glasses, the anaglyph arrangements "see" two scalar colors at once.)

Usage

```
make_anaglyph_ineqmat(card)
```

Arguments

`card`	Integer: the cardinality of the two sets to be compared.

Details

Note that, unlike for most other hyperplane arrangements, for anaglyph arrangements `card` is only half the size of the data you're working with, since anaglyph arrangements compare *two* sets of size `card`. In general, when useing anaglyph ineqmats with other functions, such as `signvector()` or `howfree()`, you should enter the two sets to be compared as a single vector, i.e. `c(set, goal)`. See the use of `howfree()` in the example.

Value

A matrix with `2*card+1` columns and k rows, where k is either 4 times an entry of A050509 in the OEIS if `card` is even, or an entry of A033594 if `card` is odd.

Examples

```
min7 <- c(0, 3, 7, 10)
maj7 <- c(0, 4, 7, 11)
just_min7 <- j(1, m3, 5, m7)
just_maj7 <- j(1, 3, 5, 7)

# The 12tet and just pairs have the same anaglyph signvector:
anaglyph_tetrachords <- make_anaglyph_ineqmat(4)
signvector(c(min7, maj7), ineqmat=anaglyph_tetrachords)
signvector(c(just_min7, just_maj7), ineqmat=anaglyph_tetrachords)

# They therefore have equivalent brightness graphs:
```

```
brightnessgraph(min7, maj7)
brightnessgraph(just_min7, just_maj7)

# The pair is able to vary along two dimensions in anaglyph space:
howfree(c(min7, maj7), ineqmat="anaglyph")
```

---

make_black_ineqmat *Define hyperplanes for transposition-sensitive arrangements*

Description

The "black" hyperplane arrangement compares a set's scale degrees individually to the pitches of `edoo(card)` (where `card` is the number of notes in `set`). This primarily has the purpose of attending to the overall transposition level of a set. Most applications of Modal Color Theory assume transpositional equivalence, but occasionally it is useful to relax that assumption. Sum class (Straus 2018, doi:10.1215/002229097127694) is a natural way to track this information, but the "black" arrangements do so qualitatively in the spirit of modal color theory. `make_black_ineqmat()` returns only the inequality matrix for the "black" arrangement, while `make_gray_ineqmat()` for convenience combines the results of `make_white_ineqmat()` and `make_black_ineqmat()`.

Usage

```
make_black_ineqmat(card)

make_gray_ineqmat(card)
```

Arguments

`card`	The cardinality of the scale(s) to be studied

Value

A `card` by `card+1` inequality matrix (for `make_black_ineqmat()`) or the result of combining white and black inequality matrices (in that order) for `make_gray_ineqmat()`.

See Also

`make_white_ineqmat()`

Examples

```
# The set (1, 4, 7)'s elements are respectively below, equal to, and
# above the pitches of edoo(3).
test_set <- c(1, 4, 7)
signvector(test_set, ineqmat=make_black_ineqmat(3))

# The result changes if you transpose test_set down a semitone:
signvector(test_set - 1, ineqmat=make_black_ineqmat(3))
```

```
# The results from signvector(..., ineqmat=make_black_ineqmat) can
# also be calculated with coord_to_edo():
sign(coord_to_edo(test_set))
sign(coord_to_edo(test_set - 1))
```

---

make_infrared_ineqmat *Define hyperplanes for infrared arrangements*

Description

The "infrared" hyperplane arrangements are in some sense an extension of the "pastel" arrangements to be more like the Rothenberg arrangements. (This is the sense of the color-conceit name for the arrangments: they contain red-like colors that we don't see in ordinary use of modal color theory.) That is, the infrared arrangment for a given color contains all the pastel hyperplanes (except those filtered out when `include_wraparound=FALSE`), plus additional ones that make comparisons between generic intervals of different sizes (as the Rothenberg arrangements do). Unlike the Rothenberg arrangements, however, the infrared arrangments are central: every hyperplane passes through the "origin" generated by `edoo()`. It should be the case that every infrared hyperplane either belongs to the pastel arrangement or is parallel to one in a Rothenberg arrangement. This family of arrangements is conceieved primarily for the study of voice leadings rather than scales themselves.

Usage

```
make_infrared_ineqmat(card, include_wraparound = FALSE)
```

Arguments

`card`	The cardinality of the scale(s) to be studied
`include_wraparound`	Boolean: should hyperplanes that involve intervals that wrap around the octave be included? Defaults to `FALSE`.

Details

These arrangements are still somewhat experimental and may change. In particular, the ordering of hyperplanes currently is inconsistent between settings for `include_wraparound`.

Value

A matrix with `card+1` columns and `k` rows (where `k` is the number of hyperplanes in the arrangement). When `include_wraparound=TRUE`, `k` is the `card`th doubly triangular number.

See Also

`make_pastel_ineqmat()` and `make_roth_ineqmat()`

Examples

```
# To see the effect of "include_wraparound" param, compare to
# pastel arrangments:
make_pastel_ineqmat(3)
make_infrared_ineqmat(3)
make_infrared_ineqmat(3, include_wraparound=TRUE)

# In general, infrared arrangments are more complicated than pastel:
make_pastel_ineqmat(4)
make_infrared_ineqmat(4, include_wraparound=TRUE)
```

`make_offset_ineqmat` *Translate a hyperplane arrangement to a new center*

Description

By default, the various hyperplane arrangements of musicMCT treat the "white" perfectly even scale as their center. (It is the point where all the hyperplanes of the MCT, white, and black arrangements intersect, and although the Rothenberg arrangements do not pass through the scale by definition, it is still a center of symmetry for them.) This function let you construct hyperplane arrangements that have been shifted to treat any other set as their center. (Details on why you might want this to come.)

Usage

```
make_offset_ineqmat(set, ineqmat = NULL, edo = 12)
```

Arguments

`set`	Numeric vector of pitch-classes in the set intended to be the center of the new arrangement
`ineqmat`	Specifies which hyperplane arrangement to consider. By default (or by explicitly entering "mct") it supplies the standard "Modal Color Theory" arrangements of `getineqmat()`, but can be set to strings "white," "black", "gray", "roth", "infrared", "pastel", "rosy", "infrared", or "anaglyph", giving the `ineqmat`s of `make_white_ineqmat()`, `make_black_ineqmat()`, `make_gray_ineqmat()`, `make_roth_ineqmat()`, `make_infrared_ineqmat()`, `make_pastel_ineqmat()`, `make_rosy_ineqmat()`, `make_infrared_ineqmat()`, or `make_anaglyph_ineqmat()`. For other arrangements, this parameter accepts explicit matrices.
`edo`	Number of unit steps in an octave. Defaults to `12`.

Value

A matrix with the same shape as the ones that define the standard arrangement of type `ineqmat`

See Also

makeineqmat() for modal color theory arrangements; make_white_ineqmat(), make_black_ineqmat(), and make_roth_ineqmat() for other relevant arrangements.

Examples

```
# When used for the sign vector with any central arrangement, the
# input `set` will have a sign vector of all 0s:
viennese_trichord <- c(0, 1, 6)
signvector(viennese_trichord, ineqmat=make_offset_ineqmat(viennese_trichord))

# Where does melodic minor lie in relation to major?
major <- c(0, 2, 4, 5, 7, 9, 11)
melmin <- c(0, 2, 3, 5, 7, 9, 11)
signvector(melmin, ineqmat=make_offset_ineqmat(major, ineqmat="white"))
```

---

make_roth_ineqmat *Define hyperplanes for Rothenberg arrangements*

Description

Although the Rothenberg propriety of a single scale can be computed directly with isproper(), propriety is a scalar feature (like modal "color") which is defined by a scale's position in the geometry of continuous pc-set space. That is, propriety, contradictions, and ambiguities are all determined by a scale's relationship to a hyperplane arrangement, but the arrangements which define these properties are different from the ones of Modal Color Theory. `make_roth_ineqmat()` creates the `ineqmats` needed to study those arrangements, similar to what makeineqmat() does for MCT arrangements. `make_rosy_ineqmat()` creates the combination of Rothenberg and MCT arrangements. (The name puns on the "Roth" of Rothenberg meaning "red," lending a reddish or rosy tint to the "colors" of the MCT arrangement.)

Each row of a Rothenberg `ineqmat` represents a hyperplane, just like the rows produced by makeineqmat(). The rows are normalized so that their first non-zero entry is either `1` or `-1`, and their orientations are assigned so that a strictly proper set will return only `-1`s for its sign vector relative to the Rothenberg arrangement. A `0` in a Rothenberg sign vector represents an ambiguity. Note that the Rothenberg arrangements are never "central," which means that the hyperplanes do *not* all intersect at the perfectly even scale. (It is clear that they must not, because perfectly even scales have no ambiguities.) These arrangements also grow in complexity much faster than the MCT arrangements do: for tetrachords, MCT arrangements have 8 hyperplanes while Rothenberg arrangements have 22. For heptachords, those numbers increase to 42 and 259, respectively. Thus, this function runs slowly when called on cardinalities of only modest size (e.g. 12-24). Matrices for cardinalities up through 24 have been precomputed and are stored in `roth_ineqmats`; `get_roth_ineqmat()` attempts to access them from that file rather than generating them from scratch.

Usage

```
make_roth_ineqmat(card)

get_roth_ineqmat(card)

make_rosy_ineqmat(card)
```

Arguments

`card`	The cardinality of the scale(s) to be studied

Value

A matrix with `card+1` columns and k rows, where k is the number of hyperplanes in the arrangement.

Examples

```
c_major <- c(0, 2, 4, 5, 7, 9, 11)
hepta_roth_ineqmat <- make_roth_ineqmat(7)
isproper(c_major)
cmaj_roth_sv <- signvector(c_major, ineqmat=hepta_roth_ineqmat)
table(cmaj_roth_sv)
hepta_roth_ineqmat[which(cmaj_roth_sv==0),]
# This reveals that c_major has one ambiguity, which results from
# the interval from 4 to 7 being exactly half an octave.
```

`make_white_ineqmat` *Define hyperplanes for white arrangements*

Description

Although the hyperplane arrangements of Modal Color Theory determine most scalar properties, there are some potentially interesting questions which require different arrangements. This function makes "white" arrangements which consider how many of a scale's intervals correspond exactly to the "white" or perfectly even color for their generic size. That is, for an interval `x` belonging to generic size `g` in an `n` note scale, does $x = g \cdot \frac{edo}{n}$? This may be relevant, for instance, because two modes have identical sum brightnesses when the interval that separates their tonics is "white" in this way. Mostly you will want to use these matrices as inputs to functions with an `ineqmat` parameter.

In many cases, it is desirable to use a combination of the MCT `ineqmat` from `makeineqmat()` and the quasi-white `ineqmat` from `make_white_ineqmat()`. Generally these are distinct, but they do have some shared hyperplanes in even cardinalities related to formal tritones (intervals that divide the scale exactly in half). Therefore, the function `make_pastel_ineqmat()` exists to give the result of combining them with duplicates removed. (The moniker "pastel" is meant to suggest combining the colors of MCT arrangements with a white pigment from white arrangements.)

Just as the MCT arrangements are concretized by the files "representative_scales" and "representative_signvectors," the white and pastel arrangements are represented by offwhite_scales, offwhite_signvectors, pastel_scales, and pastel_signvectors. This data has not been as thoroughly vetted as the files for the MCT arrangements, and currently white and pastel arrangements are only represented up through cardinality 6. The files are hosted at the modalcolortheory repo like representative_scales because they are too large to include in musicMCT.

Usage

```
make_white_ineqmat(card)

make_pastel_ineqmat(card)
```

Arguments

`card`	The cardinality of the scale(s) to be studied

Value

A matrix with `card+1` columns and k rows, where k is the nth triangular number

Examples

```
major_triad <- c(0, 4, 7)
howfree(major_triad)
howfree(major_triad, ineqmat=make_white_ineqmat(3))
# Because it's now constrained to preserve its step of exactly 1/3 the octave.

just_major_triad <- j(1, 3, 5)
howfree(just_major_triad)
howfree(just_major_triad, ineqmat=make_white_ineqmat(3))
# Because this triad's major third isn't identical to 400 cents which equally
# divide the octave.

ait1 <- c(0, 1, 4, 6)
quantize_color(ait1, reconvert=TRUE)
# quantize_color() doesn't match (0146) exactly because it's only looking for
# any set in the same 3-dimensional color as 0146.

quantize_color(ait1, ineqmat=make_white_ineqmat(4), reconvert=TRUE)
# Now that it's constrained to respect ait1's minor third from 1 to 4, the set 0146
# is now the first satisfactory result that quantize_color() finds.
```

`maxeven` *Maximally even scales*

Description

Scales which are "maximally even" divisions of some equal-tempered universe have several musically interesting properties. When a maximally even scale has a number of notes (`card`) that is coprime to the size of the equal-tempered universe, the maximally even scale is called a "non-degenerate well-formed" or "moment of symmetry" scale. When its size divides the equal temperament, it is a perfectly even scale. When it is neither coprime nor a divisor, it produces a scale with a structure like the octatonic (i.e. a union of perfectly even scales, or a well-formed scale with a period smaller than the octave). The scale is generated by quantizing a perfectly even scale to the chosen chromatic cardinality. Two quantization options are offered (rounding down and rounding to the nearest value).

Usage

```
maxeven(card, edo = 12, floor = TRUE)
```

Arguments

`card`	Number of notes in the scale. Numeric.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`floor`	Boolean determining how to quantize. Defaults to `TRUE` causing the quantization to round down. If `FALSE` rounds to the nearest value.

Value

Numeric vector of length `card` representing a scale of `card` notes.

Examples

```
maxeven(7, 12)
maxeven(6, 15)
maxeven(6, 15, floor=FALSE)

diatonic_in_19 <- maxeven(7, 19)
tresillo <- maxeven(3,8)
```

`meantone_fifth` *Define a tempered fifth for various meantone scales*

Description

Creates an interval that approximates a pure 3:2 fifth which has been tempered smaller by some fraction of a syntonic comma, making it easy to construct diatonic meantone scales. The default is to create a quarter-comma meantone fifth (i.e. about 697 cents).

Usage

```
meantone_fifth(frac = 1/4)
```

Arguments

`frac`	The fraction of a syntonic comma that the fifth should be tempered by. Defaults to `1/4`. Numeric.

Value

Single numeric value of the tempered fifth measured in 12edo semitones.

Examples

```
zarlino_fifth <- meantone_fifth(2/7)
zarlino_diatonic <- sort((0:6 * zarlino_fifth) %% 12)
print(zarlino_diatonic)

fifth_in_19edo <- convert(11, 19, 12)
meantone_fifth(1/3) - fifth_in_19edo
```

`minimize_vl` *Smallest voice leading between two sets*

Description

Given a `source` set and a `goal` to move to, find the voice leading from `source` to `goal` with smallest size.

Usage

```
minimize_vl(
  source,
  goal,
  method = c("taxicab", "euclidean", "chebyshev", "hamming"),
  no_ties = FALSE,
  edo = 12,
  rounder = 10
)
```

Arguments

`source`	Numeric vector, the pitch-class set at the start of your voice leading
`goal`	Numeric vector, the pitch-class set at the end of your voice leading
`method`	What distance metric should be used? Defaults to `"taxicab"` but can be `"euclidean"`, `"chebyshev"`, or `"hamming"`.
`no_ties`	If multiple VLs are equally small, should only one be returned? Defaults to `FALSE`, which is generally what an interactive user should want.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Details

Unless method="hamming", it is assumed that the minimal voice leading should be strongly crossing-free, so you might get strange results if your `source` and `goal` are not both in ascending order.

Using method="hamming" in principle should only care about preserving common tones, with no other restrictions on how voices move. This gives a profusion of tied voice leadings, which is not generally useful. This function therefore eliminates many of the options by requiring that the voices which aren't common tones make a minimal voice leading by the taxicab metric. Nevertheless, for multisets, method="hamming" can still return many tied possibilities.

Value

Numeric array. In most cases, a vector the same length as `source`; or a vector of `NA` the same length as `source` if `goal` and `source` have different lengths. If `no_ties=FALSE` and multiple voice leadings are equivalent, the array can be a matrix with `m` rows where `m` is the number of equally small voice leadings.

Examples

```
c_major <- c(0, 4, 7)
ab_minor <- c(8, 11, 3)
minimize_vl(c_major, ab_minor)

diatonic_scale <- c(0, 2, 4, 5, 7, 9, 11)
minimize_vl(diatonic_scale, tn(diatonic_scale, 7))

d_major <- c(2, 6, 9)
minimize_vl(c_major, d_major)
minimize_vl(c_major, d_major, no_ties=TRUE)
minimize_vl(c_major, d_major, method="euclidean", no_ties=FALSE)

minimize_vl(c(0, 4, 7, 10), c(7, 7, 11, 2), method="euclidean")
minimize_vl(c(0, 4, 7, 10), c(7, 7, 11, 2), method="euclidean", no_ties=TRUE)

natural_hexachord <- c(0, 2, 4, 5, 7, 9)
hard_hexachord <- c(7, 9, 11, 0, 2, 4)
minimize_vl(natural_hexachord, hard_hexachord, method="hamming")
```

`normal_form` *Hook's OPTIC normal forms*

Description

Following Hook (2023, 416-18, ISBN: 9780190246013), calculates a normal form for the input `set` using any combination of OPTIC symmetries.

Usage

```
normal_form(set, optic = "opc", edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`optic`	String: the OPTIC symmetries to apply. Defaults to "opc".
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Details

This function is designed for flexibility in the `optic` parameter, not speed. In situations where you need to calculate a large number of OPTIC- or OPTC-normal forms, you should use `primeform()` or `tnprime()` respectively, which are considerably faster.

Value

Numeric vector with the desired normal form of `set`

See Also

`primeform()`, `tnprime()`, and `startzero()` for faster functions dedicated to specific symmetry combinations

Examples

```
# See Exercise 10.4.8 in Hook (2023, 420):
eroica <- c(-25, -13, -6, -3, 0, 3)
normal_form(eroica, optic="pti")
normal_form(eroica, optic="op")

# See Table 10.4.1 in Hook (2023, 417):
alpha <- c(-5, -11, 14, 9, 14, 14, 2)
num_symmetries <- sample(0:5, 1)
random_symmetries <- sample(c("o", "p", "t", "i", "c"), num_symmetries)
random_symmetries <- paste(random_symmetries, collapse="")
print(random_symmetries)
normal_form(alpha, optic=random_symmetries)
```

`optc_test` *Does a scale lie in the canonical fundamental domain for OPTC symmetries?*

Description

Modal Color Theory is capable of describing "scales" (perhaps "melodies" might be more accurate) which do all sorts of non-scalar things, like repeating notes, ascending and descending inconsistently, not observing octave equivalence, and so on. This function tests whether an input has a 'well-behaved' form in that it starts on `0`, only ascends, doesn't repeat pitches, and doesn't go above the octave. If you find an interesting scale structure represented by a set that *doesn't* satisfy these constraints, you can always desaturate it until it does (i.e. call something like `saturate(.1, my_scale_with_bad_OPTCs)`).

Usage

```
optc_test(set, edo = 12, rounder = 10, single_answer = TRUE)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.
`single_answer`	Should the function return a single value of `TRUE` or `FALSE`? Defaults to `TRUE`. If set to `FALSE`, returns a vector of 4 Boolean values that indicate whether the scale individually passes O, P, T, and C criteria for being in the fundamental domain.

Value

Either a single Boolean value or a vector of 4 Boolean values, depending on the `single_answer` argument.

Examples

```
major_triad_normal_form <- c(0, 4, 7)
major_triad_open_spacing <- c(0, 7, 16)
major_triad_voice_crossing <- c(0, 7, 4)
major_triad_on_des <- c(1, 5, 8)
major_triad_doubled_third_omit_5 <- c(0, 4, 4)
example_triads <- cbind(major_triad_normal_form,
   major_triad_open_spacing,
   major_triad_voice_crossing,
   major_triad_on_des,
   major_triad_doubled_third_omit_5)

apply(example_triads, 2, optc_test)
optc_test(major_triad_voice_crossing, single_answer=FALSE)
```

populate_flat *Randomly generate scales on a flat*

Description

Sometimes it's useful to explore a flat or a color by testing small differences that result from different positions within the flat. This function generates random points on the desired flat to test, similar to `surround_set()` but constrained to lie on a target flat. Requires a base `set` that serves as an "origin" around which the random scales are to be generated (before being projected onto the target flat).

Usage

```
populate_flat(
  set,
  target_scale = NULL,
  target_rows = NULL,
  start_zero = TRUE,
  ineqmat = NULL,
  edo = 12,
  rounder = 10,
  magnitude = 2,
  distance = 1
)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`target_scale`	A numeric vector which represents a scale on the target flat.
`target_rows`	An integer vector: each integer specifies a row of `ineqmat` which helps to determine the target flat. The rows must be linearly independent.
`start_zero`	Boolean: should the result be transposed so that its pitch initial is zero? Defaults to `TRUE`.
`ineqmat`	Specifies which hyperplane arrangement to consider. By default (or by explicitly entering "mct") it supplies the standard "Modal Color Theory" arrangements of `getineqmat()`, but can be set to strings "white," "black", "gray", "roth", "infrared", "pastel", "rosy", "infrared", or "anaglyph", giving the `ineqmat`s of `make_white_ineqmat()`, `make_black_ineqmat()`, `make_gray_ineqmat()`, `make_roth_ineqmat()`, `make_infrared_ineqmat()`, `make_pastel_ineqmat()`, `make_rosy_ineqmat()`, `make_infrared_ineqmat()`, or `make_anaglyph_ineqmat()`. For other arrangements, this parameter accepts explicit matrices.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.
`magnitude`	Numeric value specifying how many sets to return. Defaults to `2`.

`distance`	How far (in units of voice leading work, using the Euclidean metric) should the sampled scales be from the input `set`?

Details

The target flat can be specified by naming the `target_rows` that determine the flat (in the manner of `project_onto()`) or by naming a `target_scale` on the desired flat. Both parameters default to `NULL`, in which case the function populates the flat that `set` itself lies on.

Value

A matrix whose columns represent scales on the desired flat. The matrix has n rows (where n is the number of notes in `set`) and `n * 10^magnitude` columns.

Examples

```
# Let's sample several scales on the same flat as j(dia):
major <- c(0, 2, 4, 5, 7, 9, 11)
jdia_flat_scales <- populate_flat(major, j(dia))
unique(apply(jdia_flat_scales, 2, whichsvzeroes), MARGIN=2)

# So all the scales do lie on one flat, but they may be different colors.
# Let's plot them using different literal colors to represent the scalar "colors."
jdia_flat_svs <- apply(apply(jdia_flat_scales, 2, signvector), 2, toString)
unique_svs <- sort(unique(jdia_flat_svs))
match_sv <- function(sv) which(unique_svs == sv)
sv_colors <- grDevices::hcl.colors(length(unique_svs),
                              palette="Green-Orange")[sapply(jdia_flat_svs, match_sv)]
plot(jdia_flat_scales[2,], jdia_flat_scales[3,], pch=20, col=sv_colors,
  xlab = "Height of scale degree 2", ylab = "Height of scale degree 3",
  asp=1)
abline(0, 2, lty="dashed", lwd=2)
points(j(2), j(3), cex=2, pch="x")
points(2, 4, cex=2, pch="o")

# Most of our sampled sets belong to two colors separated by the dashed
# line on the plot. The dashed line represents the inequality that determines
# the size of a scale's second step in relation to its first step. This is
# hyperplane #1 in the space, so it corresponds to the first entry in each
# scale's sign vector. The point labeled "x" represents the just diatonic scale
# itself, which has a larger first step than second step. The point labeled
# "o" represents the 12-equal diatonic, whose whole steps are all equal and which
# therefore lies directly on hyperplane #1. Finally, note that our sampled scales
# also touch on a few other colors at the bottom & left fringes of the scatter plot.
```

`primary_hue` *Primary colors*

Description

In traditional pitch-class set theory, concepts like normal order and `primeform()` establish a canonical representative for each equivalence class of pitch-class sets. It's useful to do something similar in MCT as well: given a family of scales, such as the collection of modes or a `scale_palette()`, we can define the "primary color" of the family as the one that comes first when the scales' sign vectors are ordered lexicographically. `primary_hue()` uses `ineqsym()` to return a specific representative of the primary color which belongs to the same palette of hues as the input. Because `primary_hue()` focuses on hues rather than colors, it may not highlight the fact that two scales have the same primary color. Thus, for information about broader families, `primary_colornum()` returns the color number of the primary color, `primary_signvector()` returns the sign vector, and `primary_color()` itself uses `quantize_color()` to return a consistent representative of each color.

Usage

```
primary_hue(
  set,
  type = c("all", "half_palette", "modes"),
  ineqmat = NULL,
  edo = 12,
  rounder = 10
)

primary_colornum(set, type = "all", signvector_list = NULL, ...)

primary_signvector(set, type = "all", ...)

primary_color(set, type = "all", nmax = 12, reconvert = FALSE, ...)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`type`	How broad of an equivalence class should be considered? May be one of three options: • "all", the default, uses the full range of `scale_palette()` relationships • "half_palette" uses `scale_palette()` with `include_involution=FALSE` • "modes" uses only the n modes of `set`
`ineqmat`	Specifies which hyperplane arrangement to consider. By default (or by explicitly entering "mct") it supplies the standard "Modal Color Theory" arrangements of `getineqmat()`, but can be set to strings "white," "black", "gray", "roth", "infrared", "pastel", "rosy", "infrared", or "anaglyph", giving the ineqmats of `make_white_ineqmat()`, `make_black_ineqmat()`, `make_gray_ineqmat()`, `make_roth_ineqmat()`, `make_infrared_ineqmat()`, `make_pastel_ineqmat()`, `make_rosy_ineqmat()`, `make_infrared_ineqmat()`, or `make_anaglyph_ineqmat()`. For other arrangements, this parameter accepts explicit matrices.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

`signvector_list`	A list of signvectors to use as the reference by which `colornum` assigns a value. Defaults to `NULL` and will attempt to use `representative_signvectors`, which needs to be downloaded and assigned separately from the package musicMCT. (If a named `ineqmat` other than "mct" is chosen, the function attempts to replace a `NULL` signvector list with a corresponding object in the global environment. For instance, if `ineqmat="pastel"` then the function tries to use `pastel_signvectors` for `signvector_list`.)
`...`	Arguments to be passed to `primary_hue()`
`nmax`	Integer, essentially a limit to how far the function should search before giving up. Although every real color should have a rational representation in some mod k universe, for some colors that k must be very high. Increasing nmax makes the function run longer but might be necessary if small chromatic universes don't produce a result. Defaults to `12`.
`reconvert`	Boolean. Should the scale be converted to the input edo? Defaults to `FALSE`.

Value

A numeric vector representing a scale for `primary_hue()`; a single integer for `primary_colornum()`; a `signvector()` for `primary_signvector()`; and a list like `quantize_color()` for `primary_color()`.

Examples

```
major_64 <- c(0, 5, 9)
primary_hue(major_64)
primary_hue(major_64, type="modes")

viennese_trichord <- c(0, 6, 11)
# Same primary color as major_64:
apply(cbind(major_64, viennese_trichord), 2, primary_signvector)

# But a different primary hue:
primary_hue(viennese_trichord)

# Only works with representative_signvectors loaded:
primary_colornum(major_64) == primary_colornum(viennese_trichord)

primary_color(major_64)
primary_color(viennese_trichord)
```

`primeform` *Prime form of a set using Rahn's algorithm*

Description

Takes a set (in any order, inversion, and transposition) and returns the canonical ("prime") form that represents the T_n/T_nI-type to which the set belongs. Uses the algorithm from Rahn 1980 rather than Forte 1973.

Usage

```
primeform(set, edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Details

In principle this should work for sets in continuous pitch-class space, not just those in a mod k universe. But watch out for rounding errors: if you can manage to work with integer values, that's probably safer. Otherwise, try rounding your set to various decimal places to test for consistency of result.

Value

Numeric vector of same length as `set`

Examples

```
primeform(c(0, 3, 4, 8))
primeform(c(0, 1, 3, 7, 8))
primeform(c(0, 3, 6, 9, 12, 14), edo=16)
```

`project_onto` *Closest point on a given flat*

Description

Projects a scale onto the nearest point that lies on a target flat of the hyperplane arrangement. `project_onto()` determines the target flat from a list of linearly independent rows in `ineqmat` which define the flat. `match_flat()` determines the target by extrapolating from a given scale on that flat. Note that while the projection lies on the desired flat (i.e. it will have all of the necessary `0`s in its sign vector), it will not necessarily belong to any particular *color*. (That is, projection doesn't give you control over the `1`s and `-1`s of the sign vector.)

Usage

```
project_onto(
  set,
  target_rows,
  ineqmat = NULL,
  start_zero = TRUE,
  edo = 12,
  rounder = 10
)

match_flat(
  set,
  target_scale,
  start_zero = TRUE,
  ineqmat = NULL,
  edo = 12,
  rounder = 10
)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`target_rows`	An integer vector: each integer specifies a row of `ineqmat` which helps to determine the target flat. The rows must be linearly independent.
`ineqmat`	Specifies which hyperplane arrangement to consider. By default (or by explicitly entering "mct") it supplies the standard "Modal Color Theory" arrangements of `getineqmat()`, but can be set to strings "white," "black", "gray", "roth", "infrared", "pastel", "rosy", "infrared", or "anaglyph", giving the `ineqmat`s of `make_white_ineqmat()`, `make_black_ineqmat()`, `make_gray_ineqmat()`, `make_roth_ineqmat()`, `make_infrared_ineqmat()`, `make_pastel_ineqmat()`, `make_rosy_ineqmat()`, `make_infrared_ineqmat()`, or `make_anaglyph_ineqmat()`. For other arrangements, this parameter accepts explicit matrices.
`start_zero`	Boolean: should the result be transposed so that its pitch initial is zero? Defaults to `TRUE`.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.
`target_scale`	A numeric vector which represents a scale on the target flat.

Value

A numeric vector of same length as `set`, representing the projection of `set` onto the flat determined by `target_rows` or `target_scale`.

Examples

```
minor_triad <- c(0, 3, 7)
project_onto(minor_triad, 3)
project_onto(minor_triad, 1)
project_onto(minor_triad, c(1, 3))
# This last projection results in the perfectly even scale
# because that's the only scale on both hyperplanes 1 and 3.

major_scale <- c(0, 2, 4, 5, 7, 9, 11)
projected_just_dia <- match_flat(j(dia), major_scale)
```

```
print(projected_just_dia)

# This is very close to fifth-comma meantone:
fifth_comma_meantone <- sim(sort(((0:6) * meantone_fifth(1/5))%%12))[,5]
vl_dist(projected_just_dia, fifth_comma_meantone)
```

`quantize_color` *Find a scale mod k that matches a given color*

Description

Modal Color Theory is useful for analyzing scales in continuous pitch-class space with irrational values, but sometimes those irrational values can be inconvenient to work with. Therefore it's often quite useful to find a scale that has the same color as the one you're studying, but which can be represented by integers in some mod k universe. See "Modal Color Theory," 27.

Usage

```
quantize_color(
  set,
  nmax = 12,
  reconvert = FALSE,
  ineqmat = NULL,
  target_edo = NULL,
  edo = 12,
  rounder = 10
)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`nmax`	Integer, essentially a limit to how far the function should search before giving up. Although every real color should have a rational representation in some mod k universe, for some colors that k must be very high. Increasing nmax makes the function run longer but might be necessary if small chromatic universes don't produce a result. Defaults to `12`.
`reconvert`	Boolean. Should the scale be converted to the input edo? Defaults to `FALSE`.
`ineqmat`	Specifies which hyperplane arrangement to consider. By default (or by explicitly entering "mct") it supplies the standard "Modal Color Theory" arrangements of `getineqmat()`, but can be set to strings "white," "black", "gray", "roth", "infrared", "pastel", "rosy", "infrared", or "anaglyph", giving the `ineqmat`s of `make_white_ineqmat()`, `make_black_ineqmat()`, `make_gray_ineqmat()`, `make_roth_ineqmat()`, `make_infrared_ineqmat()`, `make_pastel_ineqmat()`, `make_rosy_ineqmat()`, `make_infrared_ineqmat()`, or `make_anaglyph_ineqmat()`. For other arrangements, this parameter accepts explicit matrices.
`target_edo`	Numeric (expected integer) determining a specific equal division of the octave to quantize to. Defaults to `NULL`, in which any potential edo will be accepted.

`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

If `reconvert=FALSE`, a list of two elements: element 1 is `set` with a vector of integers representing the quantized scale; element 2 is edo representing the number k of unit steps in the mod k universe. If `reconvert=TRUE`, returns a single numeric vector measured relative to the unit step size input as `edo`: these generally will not be integers. Values may be `NA` if no suitable quantization was found beneath the limit given by `nmax` or in `target_edo` (if specified).

Examples

```
qcm_fifth <- meantone_fifth()
qcm_lydian <- sort(((0:6)*qcm_fifth)%%12)
quantize_color(qcm_lydian)

# Let's approximate the Werckmeister III well-temperament
werck_ratios <- c(1, 256/243, 64*sqrt(2)/81, 32/27, (256/243)*2^(1/4), 4/3,
  1024/729, (8/9)*2^(3/4), 128/81, (1024/729)*2^(1/4), 16/9, (128/81)*2^(1/4))
werck3 <- z(werck_ratios)
quantize_color(werck3)
quantize_color(werck3, reconvert=TRUE)

quantize_color(j(dia))
quantize_color(j(dia), target_edo=22)
```

`quantize_hue` *Find a scale mod k that matches a given hue*

Description

Given any scale, attempts to find a scale defined as integers mod k which belongs to the same hue as the input (i.e. would return `TRUE` when `same_hue()` is applied). This function thus is similar in spirit to `quantize_color()` but seeks a more precise structural match between input and quantization. Note, though, that while `quantize_color()` should always be able to find a suitable quantization (if `nmax` is set high enough), this is not necessarily true for `quantize_hue()`. There are lines in $\mathbb{R}^n$ which pass through no rational points but the origin, so some hues (including ones of musical interest like the 5-limit just diatonic scale) may not have any quantization.

Usage

```
quantize_hue(
  set,
  nmax = 12,
  reconvert = FALSE,
  target_edo = NULL,
  edo = 12,
  rounder = 10
)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`nmax`	Integer, essentially a limit to how far the function should search before giving up. Although every real color should have a rational representation in some mod k universe, for some colors that k must be very high. Increasing nmax makes the function run longer but might be necessary if small chromatic universes don't produce a result. Defaults to `12`.
`reconvert`	Boolean. Should the scale be converted to the input edo? Defaults to `FALSE`.
`target_edo`	Numeric (expected integer) determining a specific equal division of the octave to quantize to. Defaults to `NULL`, in which any potential edo will be accepted.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

If `reconvert=FALSE`, a list of two elements: element 1 is `set` with a vector of integers representing the quantized scale; element 2 is `edo` representing the number k of unit steps in the mod k universe. If `reconvert=TRUE`, returns a single numeric vector measured relative to the unit step size input as `edo`: these generally will not be integers. Values may be NA if no suitable quantization was found beneath the limit given by nmax or in target_edo (if specified).

Examples

```
meantone_diatonic <- sort(((0:6)*meantone_fifth())%%12)
quantize_hue(meantone_diatonic) # Succeeds
quantize_hue(j(dia), nmax=15) # Fails no matter how high you set nmax.

quasi_guido <- convert(c(0, 2, 4, 5, 7, 9), 13, 12)
quantize_color(quasi_guido)
quantize_hue(quasi_guido)

quantize_hue(c(0, 1, 4, 6))
quantize_hue(c(0, 1, 4, 6), target_edo=16)
```

readSCL *Import a Scala (.scl) file as a scale*

Description

This function allows you to import scales that have been defined in the Scala tuning format (*.scl) into R to analyze with the functions of musicMCT. Scales can be defined in .scl files in different ways, some of which may lack the precision that computations in musicMCT normally assume. If you import a scale that seems to have less regularity than you expected (i.e. it's on 0 hyperplanes even though it seems to be very regular), try increasing your rounding tolerance (i.e. lower the value of rounder arguments in the functions you apply to the imported scale).

Usage

```
readSCL(filename, scaleonly = TRUE, edo = 12)
```

Arguments

filename	String with the path to the file to be imported
scaleonly	Boolean: should readSCL return only a vector of pitches, not additional information from the file? Defaults to TRUE
edo	Number of unit steps in an octave. Defaults to 12.

Value

A numeric vector with the scale's pitches if scaleonly=TRUE; else a list in which the scale's pitches are the first entry, the length of the scale is the second, and the size of the period is the third.

Examples

```
# We'll read a sample .scl file that comes with the `musicMCT` package.
demo_filepath <- system.file("extdata", "sample_pentachord.scl", package="musicMCT")
fun_pentachord <- readSCL(demo_filepath)
sim(fun_pentachord)
brightnessgraph(fun_pentachord)
```

realize_stepword *Define scale by entering its relative step sizes*

Description

Where asword() takes you from a scale to a ranked list of its step sizes, realize_stepword does the opposite: given a list of ranked step sizes, it defines a scale with those steps. It does not attempt to define a scale that exists in 12-tone equal temperament or another mod k universe, though the result will have integral values in *some* mod k setting. If you want that information, set reconvert to FALSE.

Usage

```
realize_stepword(stepword, edo = 12, reconvert = TRUE)
```

Arguments

stepword	A numeric vector (intended to be nonnegative integers) of ranked step sizes; should be the same length as desired output set.
edo	Number of unit steps in an octave. Defaults to 12.
reconvert	Boolean. Should the result be expressed measured in terms of semitones (or a different mod k step if edo is not set to 12)?

Value

Numeric vector of same length as set, if `reconvert` is `TRUE`. If `reconvert` is `FALSE`, returns a list with two elements. The first element (`set`) expresses the defined set as integer values in some edo. The second element (edo) tells you which edo (mod k universe) the set is defined in.

Examples

```
dim7 <- realize_stepword(c(1, 1, 1, 1))
four_on_the_floor <- realize_stepword(c(1, 1, 1, 1), edo=16)
my_luggage <- realize_stepword(c(1, 2, 3, 4, 5))
my_luggage_in_15edo <- realize_stepword(c(1, 2, 3, 4, 5), reconvert=FALSE)
dim7
four_on_the_floor
my_luggage
my_luggage_in_15edo

pwf_scale <- realize_stepword(c(3, 2, 1, 3, 2, 3, 1))
asword(pwf_scale)
```

`rotate` *Circular rotation of an ordered tuple*

Description

Changes which element of a circularly-ordered series is in the first position without otherwise changing the order. Used primarily to generate the modes of a scale. Single application moves one element from the beginning of a tuple to the end.

Usage

```
rotate(x, n = 1, transpose_up = FALSE, edo = 12)
```

Arguments

`x`	Vector to be rotated
`n`	Number of positions the vector should be rotated left. Defaults to `1`. May be negative.
`transpose_up`	Boolean, defaults to `FALSE` which leaves entries unchanged. If set to `TRUE`, elements moved from the head to the tail of the vector are increased in value by `edo`.
`edo`	Number of unit steps in an octave. Defaults to `12`.

Value

(Rotated) vector of same length as x

Examples

```
rotate(c(0, 2, 4, 5, 7, 9, 11), n=2)
rotate(c(0, 2, 4, 5, 7, 9, 11), n=-2)
rotate(c(0, 2, 4, 5, 7, 9, 11), n=2, transpose_up=TRUE)
rotate(c(0, 2, 4, 5, 7, 9, 11), n=2, transpose_up=TRUE, edo=15)
rotate(c("father", "charles", "goes", "down", "and", "ends", "battle"),
  n=4)
```

`roth_ineqmats` *Hyperplane arrangements for Rothenberg arrangements*

Description

The data file `roth_ineqmats` represents the Rothenberg hyperplane arrangements that `make_roth_ineqmat()` generates. Just like the file `ineqmats`, for large computations it's faster simply to call on precalculated data rather than to run `make_roth_ineqmat()` many thousands of times. Thus the object `roth_ineqmats` saves the inequality matrices for scales of cardinality 1-24, to be called upon by `get_roth_ineqmat()`.

Usage

```
roth_ineqmats
```

Format

`roth_ineqmats` A list with 24 entries. The nth entry of the list gives the inequality matrix for n-note scales. Each inequality matrix itself is an m by (n+1) matrix, where m is the number of hyperplanes in the relevant Rothenberg arrangement. (The values m are currently only empirical: so far, no principled enumeration exists.) The last column of the matrix contains constants that translate the hyperplane away from the origin.

Source

The data in `roth_ineqmats` can be recreated with the command `sapply(1:24, make_roth_ineqmat)`.

same_hue *Do two scales lie on the same ray?*

Description

Two scales which lie on the same ray from `edoo()` (the perfectly even scale) differ only in their saturation and are said to belong to the same "hue." They are not only members of a large "color" but also a much more specific structure which preserves properties such as `ratio()` and the precise shape of `brightnessgraph()`. `same_hue()` tests whether two scales have this close relationship.

Usage

```
same_hue(set_1, set_2, edo = 12, rounder = 10)
```

Arguments

`set_1, set_2`	Numeric vectors of pitch-classes in the sets. Must be of same length.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

Boolean: are the sets of the same hue? NB: `TRUE` for identical sets (even perfectly even scales); `FALSE` for scales which are related by "involution."

Examples

```
set39 <- c(0, 5, 9, 10, 14, 16, 21)
set53 <- c(0, 7, 13, 16, 22, 26, 33)
set39 <- convert(set39, 39, 12)
set53 <- convert(set53, 53, 12)
same_hue(set39, set53)
# Since they have the same hue, we can resaturate one to become the other:
relative_evenness <- evenness(set53)/evenness(set39)
set53
saturate(relative_evenness, set39)

# These two hexachords belong to the same quasi-pairwise well formed
# color (see "Modal Color Theory," p. 37), but not to the same hue:
guidonian_1 <- c(0, 2, 4, 5, 7, 9)
guidonian_2 <- convert(guidonian_1, 13, 12)
isTRUE(all.equal(signvector(guidonian_1), signvector(guidonian_2)))
same_hue(guidonian_1, guidonian_2)
```

saturate *Modify evenness without changing hue*

Description

Saturation parameterizes scale structures along a single degree of freedom which corresponds to size of the vector from the "white" perfectly even scale to the scale in question. Variation in a scale's saturation minimally affects its structural properties. The function `saturate()` takes in a scale and a saturation parameter (`r`) and returns another scale along the same line (i.e. including the scale's hue and its scalar involution–see "Modal Color Theory," 32).

Usage

```
saturate(r, set, edo = 12)
```

Arguments

`r`	Numeric: the relative proportion to (de)saturate the set by. If r is set to 0, returns white; if r = 1, returns the input set. If 0 < r < 1, the saturation is decreased. If r > 1, the saturation is increased, potentially to the point where the set moves past some OPTIC boundary. If r < 0, the result is an "involution" of the set.
`set`	Numeric vector of pitch-classes in the set
`edo`	Number of unit steps in an octave. Defaults to `12`.

Value

Numeric vector of same length as `set` (another scale on the same hue)

Examples

```
lydian <- c(0, 2, 4, 6, 7, 9, 11)
qcm_fifth <- meantone_fifth()
qcm_dia <- sort(((0:6)*qcm_fifth)%%12)
evenness_ratio <- evenness(qcm_dia) / evenness(lydian)
desaturated_lydian <- saturate(evenness_ratio, lydian)
desaturated_lydian
qcm_dia

ionian <- c(0, 2, 4, 5, 7, 9, 11)
involution_of_ionian <- saturate(-2, ionian)
convert(involution_of_ionian, 12, 42)
asword(ionian)
asword(involution_of_ionian)
```

`sc` *Set class from Forte's list*

Description

Given a cardinality and ordinal position, returns the (Rahn) prime form of the set class from Allen Forte's list in *The Structure of Atonal Music* (1973). Draws the information from hard-coded values in the package's data.

Usage

```
sc(card, num)
```

Arguments

`card`	Integer value between 1 and 12 (inclusive) that indicates the number of distinct pitch-classes in the set class.
`num`	Ordinal number of the desired set class in Forte's list

Value

Numeric vector of length `card` representing a pc-set of `card` notes.

Examples

```
ait1 <- sc(4, 15)
ait2 <- sc(4, 29)

NB_rahn_prime_form <- sc(6, 31)
print(NB_rahn_prime_form)
```

`scale_palette` *Orbit of a scale under symmetries of hyperplane arrangement*

Description

Given an input scale, return a "palette" of related scalar colors. All the returned scales are the image of the input under some `ineqsym()`.

Usage

```
scale_palette(set, include_involution = TRUE, edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`include_involution`	Should involutional symmetry be included in the applied transformation group? Defaults to `TRUE`.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

A matrix whose columns represent the colors in `set`'s palette.

Examples

```
# The palette of a minor triad is all inversions of major and minor:
minor_triad <- c(0, 3, 7)
scale_palette(minor_triad)

# But 12edo is a little too convenient. The palette of the just minor triad
# involves some less-consonant intervals:
just_minor <- j(1, m3, 5)
scale_palette(just_minor)

# The palette of the diatonic scale includes all 42 well-formed heptachord colors:
dia_palette <- scale_palette(sc(7, 35))
dim(dia_palette)
table(apply(dia_palette, 2, iswellformed))
```

`sc_comp` *Set class complement*

Description

Find the complement of a set class in a given mod k universe. Complements have long been recognized in pitch-class set theory as sharing many properties with each other. This is true to *some* extent when considering scales in continuous pc-space, but sometimes it is not! Therefore whenever you're exploring an odd property that a scale has, it can be useful to check that scale's complement (if you've come across the scale in some mod k context, of course).

Usage

```
sc_comp(set, canon = c("tni", "tn"), edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`canon`	What transformations should be considered equivalent? Defaults to "tni" (using standard set classes) but can be "tn" (using transposition classes)
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

Numeric vector representing a set class of length `edo - n` where `n` is the length of the input `set`

Examples

```
diatonic19 <- c(0, 3, 6, 9, 11, 14, 17)
chromatic19 <- sc_comp(diatonic19, edo=19)
icvecs_19 <- rbind(ivec(diatonic19, edo=19), ivec(chromatic19, edo=19))
rownames(icvecs_19) <- c("diatonic ivec", "chromatic ivec")
icvecs_19
```

`set_from_signvector` *Create a scale from a sign vector*

Description

This function attempts to take in a sign vector (and associated cardinality and `ineqmat`) and create a scale whose sign vector matches the input. This is not always possible because not all sign vectors correspond to colors that actually exist (just like there is no Fortean set class with the interval-class vector <`1 1 0 1 0 0`>). The function will do its best but may eventually time out, using a similar process as `quantize_color()`. You can increase the search time by increasing `nmax`, but in some cases you could search forever and still find nothing. I don't advise trying to use this function on many sign vectors at the same time.

Usage

```
set_from_signvector(
  signvec,
  card,
  nmax = 12,
  reconvert = FALSE,
  ineqmat = NULL,
  target_edo = NULL,
  edo = 12,
  rounder = 10
)
```

Arguments

`signvec`	Vector of `0`, `-1`, and `1`s: the sign vector that you want to realize.
`card`	Integer: the number of notes in your desired scale.
`nmax`	Integer, essentially a limit to how far the function should search before giving up. Although every real color should have a rational representation in some mod k universe, for some colors that k must be very high. Increasing nmax makes the function run longer but might be necessary if small chromatic universes don't produce a result. Defaults to `12`.
`reconvert`	Boolean. Should the scale be converted to the input edo? Defaults to `FALSE`.
`ineqmat`	Specifies which hyperplane arrangement to consider. By default (or by explicitly entering "mct") it supplies the standard "Modal Color Theory" arrangements of `getineqmat()`, but can be set to strings "white," "black", "gray", "roth", "infrared", "pastel", "rosy", "infrared", or "anaglyph", giving the `ineqmat`s of `make_white_ineqmat()`, `make_black_ineqmat()`, `make_gray_ineqmat()`, `make_roth_ineqmat()`, `make_infrared_ineqmat()`, `make_pastel_ineqmat()`, `make_rosy_ineqmat()`, `make_infrared_ineqmat()`, or `make_anaglyph_ineqmat()`. For other arrangements, this parameter accepts explicit matrices.
`target_edo`	Numeric (expected integer) determining a specific equal division of the octave to quantize to. Defaults to `NULL`, in which any potential `edo` will be accepted.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

If `reconvert=FALSE`, a list of two elements: element 1 is `set` with a vector of integers representing the realized scale; element 2 is `edo` representing the number k of unit steps in the mod k universe. If `reconvert=TRUE`, returns a single numeric vector converted to measurement relative to 12-tone equal tempered semitones. Values may be `NA` if no suitable quantization was found beneath the limit given by nmax or in target_edo (if specified).

Examples

```
# This first command produces a real tetrachord:
set_from_signvector(c(-1, 1, 1, -1, -1, -1, 0, -1), 4)

# But this one, which changes only the last entry of the previous sign vector
# has no solution so will return only `NA`s.
set_from_signvector(c(-1, 1, 1, -1, -1, -1, 0, 1), 4)
```

`set_to_distribution` *Convert between pitch-class sets and distributions*

Description

For applications of the Discrete Fourier Transform to pitch-class set theory, it's typically convenient to represent musical sets in terms of *distributions* rather than lists of their elements. (See Chapter 1 of Amiot 2016, doi:10.1007/9783319455815.) These functions convert back and forth between those representations. s2d() and d2s() are shorthands for set_to_distribution() and distribution_to_set(), respectively.

Usage

```
set_to_distribution(set, edo = 12, rounder = 10)

distribution_to_set(
  distro,
  multiset = TRUE,
  reconvert = TRUE,
  edo = 12,
  rounder = 10
)

s2d(...)

d2s(...)
```

Arguments

`set`	Numeric vector of pitch-classes in the set. May be a multiset, in which case the result is different from the corresponding set with repetitions removed. Entries must be integers.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.
`distro`	Numeric vector representing a pitch-class distribution.
`multiset`	Boolean. Should distribution_to_set() return a multiset if element weights are greater than 1? Defaults to `TRUE`.
`reconvert`	Boolean. Should the scale be converted to the input edo? Defaults to `TRUE`.
`...`	Arguments to be passed from s2d() or d2s() to unabbreviated functions.

Value

set_to_distribution() returns a numeric vector with length `edo`, whose `i`th entry represents the weight assigned to pitch-class `i` in the distribution. distribution_to_set() returns a (multi)set represented by listing its elements in a vector. (Non-integer weights are rounded *up* to the next highest integer if `multiset` is `TRUE`.)

Examples

```
set_to_distribution(c(0, 4, 7))
s2d(c(0, 4, 7)) # Same result but quicker to type
s2d(c(0, 4, 4, 7)) # The doubled third is reflected by the value 2 in the result

minor_triad_distro <- c(2, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0)
distribution_to_set(minor_triad_distro)
d2s(minor_triad_distro, multiset=FALSE)

# distribution_to_set automatically converts to 12edo, which
# can sometimes be undesirable, as in this case:
tresillo_distro <- c(1, 0, 0, 1, 0, 0, 1, 0)
d2s(tresillo_distro)
d2s(tresillo_distro, reconvert=FALSE)
```

`signed_interval_class` *Ordered pitch-class interval represented as interval class with sign*

Description

Represents an ordered interval between two pitch-classes as a value between `-edo/2` and `edo/2`, i.e. with an absolute value that matches its interval class as well as a sign (plus or minus) that disambiguates between the two OPCIs included in the interval-class. That is, C->D is `2` whereas C->B-flat is `-2`. Exactly half the octave is represented as a positive value.

Usage

```
signed_interval_class(x, edo = 12)
```

Arguments

`x`	Single numeric value, representing an ordered pitch-class interval
`edo`	Number of unit steps in an octave. Defaults to `12`.

Value

Single numeric value

Examples

```
signed_interval_class(8)
signed_interval_class(6)
signed_interval_class(-6)
signed_interval_class(3*pi)
```

signvector *Detect a scale's location relative to a hyperplane arrangement*

Description

As "Modal Color Theory" describes (pp. 25-26), each distinct scalar "color" is determined by its relationships to the hyperplanes that define the space. For any scale, this function calculates a sign vector that compares the scale to each hyperplane and returns a vector summarizing the results. If the scale lies on hyperplane 1, then the first entry of its sign vector is `0`. If it lies below hyperplane 2, then the second entry of its sign vector is `-1`. If it lies above hyperplane 3, then the third entry of its sign vector is `1`. Two scales with identical sign vectors belong to the same "color".

Usage

```
signvector(set, ineqmat = NULL, edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`ineqmat`	Specifies which hyperplane arrangement to consider. By default (or by explicitly entering "mct") it supplies the standard "Modal Color Theory" arrangements of `getineqmat()`, but can be set to strings "white," "black", "gray", "roth", "infrared", "pastel", "rosy", "infrared", or "anaglyph", giving the `ineqmat`s of `make_white_ineqmat()`, `make_black_ineqmat()`, `make_gray_ineqmat()`, `make_roth_ineqmat()`, `make_infrared_ineqmat()`, `make_pastel_ineqmat()`, `make_rosy_ineqmat()`, `make_infrared_ineqmat()`, or `make_anaglyph_ineqmat()`. For other arrangements, this parameter accepts explicit matrices.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

A vector whose entries are `0`, `-1`, or `1`. Length of vector equals the number of hyperplanes in `ineqmat`.

Examples

```
# 037 and 016 have identical sign vectors because they belong to the same trichordal color
signvector(c(0, 3, 7))
signvector(c(0, 1, 6))

# Just and equal-tempered diatonic scales have different sign vectors because they have
# different internal structures (e.g. 12edo dia is generated but just dia is not).
dia_12edo <- c(0, 2, 4, 5, 7, 9, 11)
just_dia <- j(dia)
isTRUE( all.equal( signvector(dia_12edo), signvector(just_dia) ) )
```

sim	*Scalar (and interscalar) interval matrix*

Description

As defined by Tymoczko 2008 ("Scale Theory, Serial Theory and Voice Leading") doi:10.1111/j.14682249.2008.00257.x, the **s**calar **i**nterval **m**atrix represents the "rotations" of a set, transposed to begin on 0, in its columns. Its nth row represents the specific intervals which represent its generic interval of size n. If changed from its default (`NULL`), the parameter `goal` calculates Tymoczko's *interscalar* interval matrix from `set` to `goal`.

Usage

```
sim(set, goal = NULL, edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`goal`	Numeric vector of same length as set. Defaults to `NULL`.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

Numeric n by n matrix where n is the number of notes in `set`

Examples

```
diatonic_modes <- sim(c(0, 2, 4, 5, 7, 9, 11))
print(diatonic_modes)

miyakobushi_modes <- sim(c(0, 1, 5, 7, 8)) # rows show trivalence
print(miyakobushi_modes)

# Interscalar Interval Matrix:
sim(c(0, 3, 6, 10), c(0, 4, 7, 10))

# Note that the interscalar matrices factor out transposition:
minor <- c(0, 3, 7)
major <- c(0, 4, 7)
sim(minor, major)
sim(minor-1, major)
sim(minor, major+2)

# But not permutation:
major_64 <- c(0, 5, 9)
major_open <- c(0, 7, 4)
```

```
sim(minor, major_64)
sim(minor, major_open)
```

---

simplify_scale *Best ways to regularize a scale*

Description

Given an input scale, identify which adjacent colors represent good approximations of it, in a sense consistent with "Modal Color Theory," pp. 31-32.

Usage

```
simplify_scale(
  set,
  start_zero = TRUE,
  ineqmat = NULL,
  scales = NULL,
  signvector_list = NULL,
  adjlist = NULL,
  method = c("euclidean", "taxicab", "chebyshev", "hamming"),
  display_digits = 2,
  edo = 12,
  rounder = 10
)

best_simplification(set, ...)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`start_zero`	Boolean: should the result be transposed so that its pitch initial is zero? Defaults to `TRUE`.
`ineqmat`	Specifies which hyperplane arrangement to consider. By default (or by explicitly entering "mct") it supplies the standard "Modal Color Theory" arrangements of `getineqmat()`, but can be set to strings "white," "black", "gray", "roth", "infrared", "pastel", "rosy", "infrared", or "anaglyph", giving the `ineqmat`s of `make_white_ineqmat()`, `make_black_ineqmat()`, `make_gray_ineqmat()`, `make_roth_ineqmat()`, `make_infrared_ineqmat()`, `make_pastel_ineqmat()`, `make_rosy_ineqmat()`, `make_infrared_ineqmat()`, or `make_anaglyph_ineqmat()`. For other arrangements, this parameter accepts explicit matrices.
`scales`	List of scales representing the faces of your hyperplane arrangement. Defaults to `NULL` in which case the function looks for `representative_scales` in the global environment.

`signvector_list`	A list of signvectors to use as the reference by which `colornum` assigns a value. Defaults to `NULL` and will attempt to use `representative_signvectors`, which needs to be downloaded and assigned separately from the package musicMCT. (If a named `ineqmat` other than "mct" is chosen, the function attempts to replace a `NULL` signvector list with a corresponding object in the global environment. For instance, if `ineqmat="pastel"` then the function tries to use `pastel_signvectors` for `signvector_list`.)
`adjlist`	Adjacency list structured in the same way as `color_adjacencies`. Defaults to `NULL` in which case the function looks for `color_adjacencies` in the global environment.
`method`	What distance metric should be used? Defaults to `"euclidean"` (unlike most functions with a method parameter in musicMCT) but can be `"taxicab"`, `"chebyshev"`, or `"hamming"`.
`display_digits`	Integer: how many digits to display when naming any non-integral interval sizes. Defaults to 2.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.
`...`	Other arguments to be passed from `best_simplification()` to `simplify_scale()`.

Details

Suppose that you've gathered data on how a particular instrument is tuned. Two intervals in its scale differ by about 12 cents: does it make sense to consider those intervals to be essentially the same, up to some combination of measurement error and the permissiveness of cognitive categories? `simplify_scale()` helps to answer such a question by considering whether eliding a precisely measured difference results in a significant simplification of the overall scale structure.

It accomplishes this by starting from two premises:

- Any simplification should move to an adjacent color with fewer degrees of freedom.
- There's a tradeoff between moving farther (i.e. requiring more measurement fuzziness) and achieving greater regularity. Therefore it starts by projecting the input scale onto all neighboring flats with fewer degrees of freedom. Some projections can be rejected immediately because the closest point on the flat isn't actually an adjacent color. The non-rejected projections can therefore be ranked by calculating the "cost" of each additional regularity: for every `1` or `-1` in the sign vector that is converted to a `0`, how far does one have to move in voice leading space?

To answer this question, `simplify_signvector` needs access to data about the hyperplane arrangement in question. For the basic "Modal Color Theory" arrangements, this is the data in `representative_scales.rds`, `representative_signvectors.rds`, and `color_adjacencies.rds`. The function assumes that, if you don't specify other data, you have those three files loaded into your workspace. It can't function without them.

Value

A matrix with `n+6` rows, where `n` is the number of notes in the scale. Each column represents a scale which is a potential simplification of the input `set`, together with details about that simplified scale. The first `n` entries of the column represent the pitches of the scale itself:

- The `n+1`th row indicates the color number of the simplification.
- The `n+2`th row shows how many degrees of freedom the simplification has (always between `0` and `d-1` where `d` is `set`'s degree of freedom).
- The `n+3`th row calculates the voice-leading distance from `set` to the simplified scale (according to the chosen `method`, for which Euclidean distance is the default because it corresponds to the assumption that orthogonal projection finds the closest point on a neighboring flat).
- The `n+4`th row counts how many more hyperplanes the simplified scale lies on compared to `set`.
- The `n+5`th row is a quotient of the previous two rows (distance divided by number of new regularities).
- The `n+6`th row calculates a final "score" which is used to order the columns from best (first) to worst (last) simplifications. This score is the inverse of the previous row divided by the total number of hyperplanes in the arrangement. (Without this normalization, scores for higher cardinalities quickly become much larger than scores for low cardinalities.)

If `display_digits` is a value other than `NULL`, the function prints to console a suitably rounded representation of the data, while invisibly returning the unrounded information.

`best_simplification()` returns simply a numeric vector with the scale judged optimal by `simplify_scale()` (i.e. the first `n` entries of its first column, without all the other information).

Examples

```
# For this example to run, you need the necessary data files loaded.
# Let's see what happens if we try to simplify the 5-limit just diatonic:

simplify_scale(j(dia))

# So the best option is color number 942659, which is the "well-formed"
# structure of the familiar diatonic scale. The particular saturation of
# that meantone structure is very close to 1/5-comma meantone:

simplified_jdia <- best_simplification(j(dia))
fifth_comma_dia <- sim(sort((meantone_fifth(1/5)*(0:6))%%12))[,5]
vl_dist(simplified_jdia, fifth_comma_dia)
```

`step_signvector` *Specify a scale's step pattern with a sign vector*

Description

Rather than calculate the full sign vector from the "modal color" hyperplane arrangement, sometimes it's advantageous to use a sign vector that reflects only the pairwise comparisons on a scale's steps. This function does that.

Usage

```
step_signvector(set, ineqmat = NULL, edo = 12, rounder = 10)
```

Arguments

set	Numeric vector of pitch-classes in the set
ineqmat	Specifies which hyperplane arrangement to consider. By default (or by explicitly entering "mct") it supplies the standard "Modal Color Theory" arrangements of `getineqmat()`, but can be set to strings "white," "black", "gray", "roth", "infrared", "pastel", "rosy", "infrared", or "anaglyph", giving the `ineqmat`s of `make_white_ineqmat()`, `make_black_ineqmat()`, `make_gray_ineqmat()`, `make_roth_ineqmat()`, `make_infrared_ineqmat()`, `make_pastel_ineqmat()`, `make_rosy_ineqmat()`, `make_infrared_ineqmat()`, or `make_anaglyph_ineqmat()`. For other arrangements, this parameter accepts explicit matrices.
edo	Number of unit steps in an octave. Defaults to `12`.
rounder	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

A vectors of signs, `-1`, `0`, and `1`, corresponding to the step-related hyperplanes in the defined `ineqmat`.

Examples

```
step_signvector(sc(7, 35)) # Half the length of a full sign vector for heptachords:
signvector(sc(7, 35))
```

---

subsetspectrum *Subset varieties for all subsets of a fixed size*

Description

Applies `subset_varieties()` not just to a particular subset shape but to all possible subset shapes given a fixed cardinality. For example, finds the specific varieties of *all* trichordal subsets of the major scale, not than just the varieties of the tonal triad. Comparable to `intervalspectrum()` but for subsets larger than dyads.

Usage

```
subsetspectrum(
  set,
  subsetcard,
  simplify = TRUE,
  mode = "tn",
  edo = 12,
  rounder = 10
)
```

Arguments

`set`	The scale to find subsets of, as a numeric vector
`subsetcard`	Single integer defining the cardinality of subsets to consider
`simplify`	Should "inversions" of a subset be ignored? Boolean, defaults to `TRUE`
`mode`	String `"tn"` or `"tni"`. When defining subset shapes, use transposition or transposition & inversion to reduce the number of shapes to consider? Defaults to `"tn"`.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Details

The parameter `simplify` lets you control whether to consider different "inversions" of a subset shape independently. For instance, with `simplify=TRUE`, only root position triads (0, 2, 4) would be considered; but with `simplify=FALSE`, the first inversion (0, 2, 5) and second inversion (0, 3, 5) subset shapes would also be displayed.

Value

A list whose length matches the number of distinct subset shapes (given the chosen options). Each entry of the list is a matrix displaying the varieties of some particular subset type.

Examples

```
c_major_scale <- c(0, 2, 4, 5, 7, 9, 11)
subsetspectrum(c_major_scale, 3)
subsetspectrum(c_major_scale, 3, simplify=FALSE)
subsetspectrum(c_major_scale, 3, mode="tni") # Note the absence of a "0, 2, 3" matrix
```

subset_multiplicities *Count the multiplicities of a subset-type's varieties*

Description

Given the varieties of a subset type returned by `subset_varieties()`, `subset_multiplicities()` counts how many times each one occurs in the scale. These are the multiplicities of the subsets in the sense of Clough and Myerson (1985)'s result "structure yields multiplicity" for well-formed scales.

Usage

```
subset_multiplicities(
  subsetdegrees,
  set,
  edo = 12,
  rounder = 10,
  display_digits = 2
)
```

Arguments

`subsetdegrees`	Vector of integers indicating the generic shape to use, e.g. `c(0, 2, 4)` for tertian triads in a heptachord. Expected to begin with `0` and must have length > 1.
`set`	The scale to find subsets of, as a numeric vector
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.
`display_digits`	Integer: how many digits to display when naming any non-integral interval sizes. Defaults to 2.

Value

Numeric vector whose names indicate the `k` varieties of the subset type and whose entries count how often each variety occurs.

Examples

```
subset_multiplicities(c(0, 2, 4), sc(7, 35))
subset_multiplicities(c(0, 1, 4), sc(7, 35))

subset_multiplicities(c(0, 2, 4), j(dia))
```

subset_varieties *Specific varieties of scalar subsets given a generic shape*

Description

Considered mod 7, the traditional triads of a diatonic scale are all instances of the generic shape (0, 2, 4). They come in three varieties: major, minor, and diminished. This function lists the distinct varieties of any similarly defined generic shape which occur as subsets in some specified scale (or larger set).

Usage

```
subset_varieties(subsetdegrees, set, unique = TRUE, edo = 12, rounder = 10)
```

Arguments

`subsetdegrees`	Vector of integers indicating the generic shape to use, e.g. `c(0, 2, 4)` for tertian triads in a heptachord. Expected to begin with `0` and must have length > 1.
`set`	The scale to find subsets of, as a numeric vector
`unique`	Should each variety be listed only once? Defaults to `TRUE`. If `FALSE`, each specific variety will be listed corresponding to how many times it occurs as a subset.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

A numeric matrix whose columns represent the specific varieties of the subset

Examples

```
c_major_scale <- c(0, 2, 4, 5, 7, 9, 11)
double_harmonic_scale <- c(0, 1, 4, 5, 7, 8, 11)

subset_varieties(c(0, 2, 4), c_major_scale)
subset_varieties(c(0, 2, 4), c_major_scale, unique=FALSE)
subset_varieties(c(0, 2, 4), double_harmonic_scale)
```

surround_set *Random scales uniformly distributed on a hypersphere around an input*

Description

Sometimes you want to explore what other scale structures a given scale is *close* to. This can be done by studying the network of color adjacencies in suitably low cardinalities (see "Modal Color Theory," 31-37), but it can also be rewarding simply to randomly sample scales that are suitably close to the one you started with.

The larger your starting scale, the more complicated is the geometry of the color space it lives in. Therefore this function generates a larger number of random scales for larger cardinalities: by default, if the length of the input `set` is `card`, `surround_set` gives `card * 100` output scales. The parameter `magnitude` controls the order of magnitude of your sample (i.e. if you want ~1000 scales rather than ~100, set `magnitude=3`).

The size of the hypersphere which the function samples is, by default, 1. When we're working with a unit of 12 semitones per octave, 1 semitone of voice leading work can get you pretty far away from the original set, especially in higher cardinalities. (For instance, C major to C melodic minor is just 1 semitone of motion, but there are 3 other colors that intervene between these two scales along a direct path.) Depending on your goals, you might want to try a couple different orders of magnitude for `distance`.

Usage

```
surround_set(set, magnitude = 2, distance = 1)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`magnitude`	Numeric value specifying how many sets to return. Defaults to `2`.
`distance`	How far (in units of voice leading work, using the Euclidean metric) should the sampled scales be from the input `set`?

Value

a Matrix with `length(set)` rows and `10^magnitude` columns, representing `10^magnitude` different scales

Examples

```
# First we sample 30 trichords surrounding the minor triad 037.
chords_near_minor <- surround_set(c(0,3,7), magnitude=1, distance=.5)
chords_near_minor

# The next two commands will plot the sampled trichords on an x-y plane as
# circles; the minor triad that they surround is marked with a "+" sign.
plot(chords_near_minor[2,], chords_near_minor[3,],
  xlab="Third", ylab="Fifth", asp=1)
points(3, 7, pch="+")

# The following two commands will plot the two lines (i.e. hyperplanes) that
# demarcate the boundaries of the minor triad's color. Most but not all
# of our randomly generated points should fall in the space between the
# two lines, in the same region as the "+" representing 037.
abline(0, 2)
abline(6, 1/2)
```

svzero_fingerprint *Distinguish different types of interval equalities*

Description

Not all hyperplanes are made equal. Those which represent "formal tritone" comparisons and those which are "exceptional" because they check a scale degree twice ("Modal Color Theory," 40-41) play a different role in the structure of the hyperplane arrangement than the rest. This function returns a "fingerprint" of a scale which is like `countsvzeroes()` but which counts the different types of hyperplane separately.

Usage

```
svzero_fingerprint(set, ineqmat = NULL, edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`ineqmat`	Specifies which hyperplane arrangement to consider. By default (or by explicitly entering "mct") it supplies the standard "Modal Color Theory" arrangements of `getineqmat()`, but can be set to strings "white," "black", "gray", "roth", "infrared", "pastel", "rosy", "infrared", or "anaglyph", giving the ineqmats of `make_white_ineqmat()`, `make_black_ineqmat()`, `make_gray_ineqmat()`, `make_roth_ineqmat()`, `make_infrared_ineqmat()`, `make_pastel_ineqmat()`, `make_rosy_ineqmat()`, `make_infrared_ineqmat()`, or `make_anaglyph_ineqmat()`. For other arrangements, this parameter accepts explicit matrices.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

Numeric vector with 3 entries: the number of 'normal' hyperplanes the set lies on, the number of 'exceptional' hyperplanes, and the number of hyperplanes which compare a formal tritone to itself.

Examples

```
# Two hexachords on the same number of hyperplanes but with different fingerprints
hex1 <- c(0, 1, 3, 5, 8, 9)
hex2 <- c(0, 1, 3, 5, 6, 9)
countsvzeroes(hex1) == countsvzeroes(hex2)
svzero_fingerprint(hex1)
svzero_fingerprint(hex2)

# Their brightness graphs make their difference more apparent:
brightnessgraph(hex1)
brightnessgraph(hex2)
```

`tc` *Transpositional combination & pitch multiplication*

Description

Cohn (1988, doi:10.2307/745790) defines transpositional combination as a procedure that generates a pc-set as the union of two (or more) transpositions of some smaller set. `tc()` takes the small set and a vector of transposition levels, returning the larger pc-set that results. (Pierre Boulez referred to this procedure as pitch "multiplication", which Amiot (2016, doi:10.1007/9783319455815) shows to be not at all fanciful, as a convolution of two pitch-class sets.)

Usage

```
tc(set, multiplier = NULL, edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`multiplier`	Numeric vector of transposition levels to apply to `set`. If not specified, defaults to `set`.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

Numeric vector of length $\leq$ `length(set)` $\cdot$ `length(multiplier)`

Examples

```
tc(c(0, 4), c(0, 7))
tc(c(0, 7), c(0, 4))

pyth_tetrachord <- j(1, t, dt, 4)
pyth_dia <- tc(pyth_tetrachord, j(1, 5))
same_hue(pyth_dia, c(0, 2, 4, 5, 7, 9, 11))
```

`tn` *Transposition and Inversion*

Description

Calculate the classic operations on pitch-class sets T_n and T_nI. That is, `tn` adds a constant to all elements in a set modulo the octave, and `tni` essentially multiplies a set by `-1` (modulo the octave) and then adds a constant (modulo the octave). If `sorted` is `TRUE` (as is default), the resulting set is listed in ascending order, but sometimes it can be useful to track transformational voice leadings, in which case you should set `sorted` to `FALSE`.

`startzero` transposes a set so that its first element is `0`. (Note that this is different from `tnprime()` because it doesn't attempt to find the most compact form of the set. See examples for the contrast.)

Sometimes you just want to invert a set and you don't care what the index is. `charm` is a quick way to do this, giving a name to the transposition-class of T_0I of the set. (The name `charm` is a reference to "strange" and "charm" quarks in particle physics: I like these as names for the "a" and "b" forms of a set class, i.e. the strange common triad is 3-11a = (0, 3, 7) and the charm common triad is 3-11b = (0, 4, 7). The name of the function `charm` means that if you input a strange set, you get out a charm set, but NB also vice versa.)

Usage

```
tn(
  set,
  n,
  sorted = TRUE,
  octave_equivalence = TRUE,
  optic = NULL,
  edo = 12,
  rounder = 10
)

tni(
  set,
  n = NULL,
  sorted = TRUE,
  octave_equivalence = TRUE,
  optic = NULL,
  edo = 12,
  rounder = 10
)

startzero(
  set,
  sorted = TRUE,
  octave_equivalence = TRUE,
  optic = NULL,
  edo = 12,
  rounder = 10
)

charm(set, edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`n`	Numeric value (not necessarily an integer) representing the index of transposition or inversion. For `tni()` only, defaults to `NULL`, in which case `n` is chosen automatically to fix the first and last entries of `set` as common tones.
`sorted`	Do you want the result to be in ascending order? Boolean, defaults to `TRUE`.
`octave_equivalence`	Do you want to normalize the result so that all values are between 0 and `edo`? Boolean, defaults to `TRUE`.
`optic`	String: the OPTIC symmetries to apply. Defaults to `NULL`, applying symmetries most appropriate to the given function. If specified, overrides parameters `sorted` and `octave_equivalence`.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

Numeric vector of same length as `set`

Examples

```
c_major <- c(0, 4, 7)
tn(c_major, 2)
tn(c_major, -10)
tn(c_major, -10, optic="p") # Equivalent to tn(c_major, -10, octave_equivalence=FALSE)
tni(c_major, 4)
tni(c_major, 4, sorted=FALSE)
# If no index is supplied for tni, n is chosen to fix the first and last entries of the set:
tni(c_major)

tn(c(0, 1, 6, 7), 6)
tn(c(0, 1, 6, 7), 6, sorted=FALSE)

##### Difference between startzero and tnprime
e_maj7 <- c(4, 8, 11, 3)
startzero(e_maj7)
tnprime(e_maj7)
isTRUE(all.equal(tnprime(e_maj7), charm(e_maj7))) # True because inversionally symmetrical

##### Derive minimal voice leading from ionian to lydian
ionian <- c(0, 2, 4, 5, 7, 9, 11)
lydian <- rotate(tn(ionian, 7, sorted=FALSE), 3)
lydian - ionian

##### Easy to create a 12-tone matrix
row <- c(9, 10, 6, 8, 5, 7, 1, 2, 3, 11, 0, 4)
matrix_from_0 <- sapply(row, tni, set=row, optic="o")
matrix_from_9 <- tn(matrix_from_0, 9, optic="o")
print(matrix_from_0)
print(matrix_from_9)
```

tndists *Distances between continuous transpositions of a set*

Description

One way to think about the voice-leading potential of a set is to consider the minimal voice-leadings by which it can move to transpositions of itself (or another set). For instance, the major triad's closest transpositions are T_4 and T_8 while its most distant transposition is T_6, and potentially also $T_{\pm 2}$ depending on the distance metric you use. For the major triad restricted to 12-tone equal temperament, this set of relationships is well modeled by Richard Cohn's discussion of Douthett & Steinbach's "Cube Dance" in *Audacious Euphony* (102-106). The behavior of other sets is not always what you might expect extrapolating from the case of tertian sonorities. For instance, the trichord (027) has different minimal neighbors depending on the metric chosen: its nearest neighbors are $T_{\pm 4}$ under the Euclidean metric but $T_{\pm 5}$ under the taxicab metric.

This function allows us to visualize such relationships by plotting the minimal voice leading distance from a set to transpositions of its goal in continuous pc-space. (In spirit, it is like a continuous version of `vl_rolodex()` except that it visualizes a voice-leading distance rather than reporting the specific motions of the set's individual voices.) The main intended use of the function is the plot that it produces, which represents many discrete T_ns of the set (for a sampling of each edo step divided into `subdivide` amounts) on the x axis and voice-leading distance on the y axis. Secondarily, `tndists()` invisibly returns the distance values that it plots, named according to the T_n they correspond to.

Usage

```
tndists(
  set,
  goal = NULL,
  method = c("taxicab", "euclidean", "chebyshev", "hamming"),
  subdivide = 100,
  edo = 12,
  rounder = 10
)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`goal`	Numeric vector like set: what is the tn-type of the voice leading's destination? Defaults to `NULL`, in which case the function uses `set` as the tn-type.
`method`	What distance metric should be used? Defaults to `"taxicab"` but can be `"euclidean"`, `"chebyshev"`, or `"hamming"`.
`subdivide`	Numeric: how many small amounts should each edo step be divided into? Defaults to `100`.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

Numeric vector of length `edo * subdivide` representing distances of the transpositions. Names indicate the transposition index that corresponds to each distance.

Examples

```
major_triad <- c(0, 4, 7)
taxicab_dists <- tndists(major_triad)
euclidean_dists <- tndists(major_triad, method="euclidean")
tns_to_display <- c("1.9", "1.92", "1.95", "2", "2.05", "2.08", "2.1")
taxicab_dists[tns_to_display]
euclidean_dists[tns_to_display]
```

`tnprime` *Transposition class of a given pc-set*

Description

Uses Rahn's algorithm to calculate the best normal order for the transposition class represented by a given set. Reflects transpositional but not inversional equivalence, i.e. all major triads return (0, 4, 7) and all minor triads return (0, 3, 7).

Usage

```
tnprime(set, edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

Numeric vector of same length as `set` representing the set's Tn-prime form

Examples

```
tnprime(c(2, 6, 9))
tnprime(c(0, 3, 6, 9, 14), edo=16)
```

`tsym` *Test for transpositional symmetry*

Description

Does the set map onto itself at some transposition other than T_0? That is, does it map onto itself under T_n for some appropriate n? `tsym()` can return either `TRUE`/`FALSE` or an index of symmetry but defaults to the former. `tsym_index()` is a simple wrapper for `tsym()` that returns the latter. `tsym_degree()` counts the total number of transpositional symmetries.

Usage

```
tsym(set, return_index = FALSE, edo = 12, rounder = 10)

tsym_index(set, ...)

tsym_degree(set, ...)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`return_index`	Should the function return a specific index at which the set is symmetrical? Defaults to `FALSE`.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.
`...`	Arguments to be passed to `tsym()`

Value

By default, `tsym()` returns `TRUE` if the set has non-trivial transpositional symmetry, `FALSE` otherwise. If `return_index` is `TRUE`, returns a vector of transposition levels at which the set is symmetric, including `0`. `tsym_index()` is a wrapper for `tsym()` which sets `return_index` to `TRUE`. `tsym_degree()` gives the degree of symmetry, which is simply the length of `tsym_index()`'s value.

Examples

```
tsym(sc(6, 34))
tsym(sc(6, 35))
tsym(edoo(5))

# Works for continuous values:
tsym(tc(j(dia), edoo(3)))


# Index and Degree:
tsym_index(c(0, 1, 3, 6, 7, 9))
tsym_degree(edoo(7))
```

`vlsig` *Elementary voice leadings*

Description

Calculate elementary voice leadings which represent motion by a single arrow on a `brightnessgraph()`. `vlsig()` finds "**v**oice-**l**eading **sig**nature" of a set moving to transpositions of itself, as determined by `vl_generators()`. `inter_vlsig()` finds the elementary voice leadings from a set to some other set, i.e. where the goal parameter of `brightnessgraph()` is not `NULL`. By default, `inter_vlsig()` finds voice leadings for contextual inversions of a set.

Usage

```
vlsig(set, index = NULL, display_digits = 2, edo = 12, rounder = 10)

inter_vlsig(
  set,
  goal = NULL,
  index = NULL,
  type = c("ascending", "commontone", "obverse"),
  display_digits = 2,
  edo = 12,
  rounder = 10
)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`index`	Integer: which voice-leading generator should be displayed? Defaults to `NULL`, displaying all voice leadings.
`display_digits`	Integer: how many digits to display when naming any non-integral interval sizes. Defaults to 2.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.
`goal`	For `inter_vlsig()` only, vector of the transposition type to voice lead to. Defaults to `NULL`, producing voice leadings to the inversion of `set`.
`type`	For `inter_vlsig()` only. String: "ascending", "commontone", or "obverse". Defaults to "ascending", which makes the result prefer ascending voice leadings (as for `vlsig()`). The second makes the result prefer common tones (as might be expected for contextual inversions). The third option, "obverse", gives the obverse of a voice-leading in a sense that generalizes Morris (1998, doi:10.2307/746047)'s concept for Neo-Riemannian PLR transformations. This option returns voice leadings that lead *to* `set` rather than away from it.

Details

Note that the voice leadings determined by `vlsig()` can be different from the corresponding ones at the same T_n level in `vl_rolodex()`. The latter function prioritizes minimal voice leadings, whereas `vlsig()` prioritizes *elementary* voice leadings derived from a set's `brightnessgraph()`. In particular, this means that `vlsig()` voice leadings will always be ascending, involve at least one common tone, and involve no contrary motion. See the `odd_pentachord` voice leadings in the Examples.

For `vlsig()` the value "rotation" in the result is non-arbitrary: if the rotation value is n, the voice leading takes `set` to the nth mode of `set`. For `inter_vlsig()`, there is no canonical correspondence between modes of `set` and `goal`, except to assume that the input modes are the 1st mode of each scale. If `goal` is `NULL`, finding contextual inversions of `set`, the first mode of the inversion is taken to be the one that holds the first and last pitches of `set` in common. These "rotation" values do not have a transparent relationship to the values of `inter_vlsig()`'s index parameter.

For `inter_vlsig()` results are not as symmetric between `set` and `goal` as you might expect. Since these voice-leading functions study ascending arrows on a brightness graph the possibilities for *ascending from X to Y* are in principle somewhat different from the possibilities for *ascending from Y to X*. See the examples for the "Tristan genus." Note that this is still true when `type="commontone"`, which might lead to counterintuitive results.

Value

List with three elements:

- "vl" which shows the distance (in `edo` steps) that each voice moves,
- "tn" which indicates the (chromatic) transposition achieved by the voice leading,
- "rotation" which indicates the scalar transposition caused by the voice leading.

If `index=NULL`, returns instead a matrix whose rows are all the elementary voice leadings.

See Also

`vl_generators()` and `brightnessgraph()`

Examples

```
# Hook's elementary signature transformation
major_scale <- c(0, 2, 4, 5, 7, 9, 11)
vlsig(major_scale, index=1)

pure_major_triad <- j(1, 3, 5)
vlsig(pure_major_triad, index=1)
vlsig(pure_major_triad, index=2)

odd_pentachord <- c(0, 1, 4, 9, 11) # in 15-edo
vlsig(odd_pentachord, index=2, edo=15)
vl_rolodex(odd_pentachord, edo=15)$"8"

# Contextual inversions for Tristan genus:
dom7 <- c(0, 4, 7, 10)
halfdim7 <- c(0, 3, 6, 10)
inter_vlsig(dom7, halfdim7)
inter_vlsig(halfdim7, dom7)

# Elementary voice leadings between unrelated sets:
maj7 <- c(0, 4, 7, 11)
min7 <- c(0, 3, 7, 10)
inter_vlsig(min7, maj7)
brightnessgraph(min7, maj7)

# Elementary inversional VL for just diatonic which is NOT a Q-relation:
inter_vlsig(j(dia), index=3)

# Obverse voice leadings:
# First we see the Parallel transformation which leads from minor to major:
minor <- c(0, 3, 7)
P <- inter_vlsig(minor, index=1)
print(P)
# Compare to its obverse, Slide, leading *to* minor from major:
S <- inter_vlsig(minor, index=1, type="obverse")
print(S)
# A voice-leading plus its obverse is a chromatic transposition:
P$vl + S$vl
```

`vl_dist` *How far apart are two scales?*

Description

Using the chosen `method` to measure distance, determines how far apart two scales are in voice-leading space.

Usage

```
vl_dist(
  set_1,
  set_2,
  method = c("taxicab", "euclidean", "chebyshev", "hamming"),
  rounder = 10
)
```

Arguments

`set_1, set_2`	Numeric vectors of pitch-classes in the sets. Must be of same length.
`method`	What distance metric should be used? Defaults to `"taxicab"` but can be `"euclidean"`, `"chebyshev"`, or `"hamming"`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

Numeric: distance between `set_1` and `set_2`

Examples

```
c_major <- c(0, 4, 7)
a_minor_63 <- c(0, 4, 9)
f_minor_64 <- c(0, 5, 8)
vl_dist(c_major, a_minor_63)
vl_dist(c_major, f_minor_64)
vl_dist(c_major, a_minor_63, method="euclidean")
vl_dist(c_major, f_minor_64, method="euclidean")
```

`vl_generators` *Which transpositions give elementary voice leadings?*

Description

Just as the transpositions of the diatonic scale can be generated by Hook's (2008, doi:10.1515/9781580467476008) elementary "signature transformation," the transpositional voice leadings of any set can generally be decomposed into a small number of basic motions. These motions correspond to the arrows in a set's `brightnessgraph()`. (The qualifier "generally" is needed because of certain problematic edge cases, such as the perfectly even scales of `edoo()` whose minimal voice leadings always involve entirely parallel motion, which cannot be derived from "mode shift" voice leadings represented on a brightness graph.) `vl_generators()` identifies these basic voice-leading motions.

Usage

```
vl_generators(set, edo = 12, rounder = 10)
```

Arguments

`set`	Numeric vector of pitch-classes in the set
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

2-by-m matrix whose m columns represent the m distinct voice-leading generators. The top row indicates the generic size of each interval; the bottom row indicates the specific size. Results are sorted so that the first row (generic intervals) is strictly increasing.

Examples

```
diatonic_scale <- c(0, 2, 4, 5, 7, 9, 11)
melodic_minor <- c(0, 2, 3, 5, 7, 9, 11)
vl_generators(diatonic_scale)
vl_generators(melodic_minor)
vl_generators(j(dia))

maj7 <- c(0, 4, 7, 11)
vl_generators(maj7)
```

`vl_rolodex` *Minimal voice leadings to all transpositions of some Tn-type mod k*

Description

Given a starting set (`source`) and some tn-type as a voice leading goal (`goal_type`), find the minimal voice leading to every transposition (in some mod k universe) of the goal. If a goal is not specified, the goal is assumed to be the tn-type of the `source` set. This lets you see, for example, the minimal voice leading from C7 to other dominant seventh chords mod 12. I couldn't think of a suitably serious and clear name for this information, so the metaphor behind "rolodex" is that these voice leadings are the contact information that `source` has for all its acquaintances in `goal_type`.

Usage

```
vl_rolodex(
  source,
  goal_type = NULL,
  reorder = TRUE,
  method = c("taxicab", "euclidean", "chebyshev", "hamming"),
  edo = 12,
  rounder = 10,
  no_ties = FALSE
)
```

Arguments

`source`	Numeric vector, the pitch-class set at the start of your voice leading
`goal_type`	Numeric vector, any pitch-class set representing the tn-type of your voice leading goal
`reorder`	Should the results be listed from smallest to largest voice leading size? Defaults to `TRUE`. If `FALSE` results are listed in transposition order (i.e. T_1, T_2, ..., T_{edo-1}, T_0).
`method`	What distance metric should be used? Defaults to `"taxicab"` but can be `"euclidean"`, `"chebyshev"`, or `"hamming"`.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.
`no_ties`	If multiple VLs are equally small, should only one be returned? Defaults to `FALSE`, which is generally what an interactive user should want.

Value

A list of length `edo`, each entry of which represents a voice leading (or group of tied voice leadings). List entries are named by their transposition level.

Examples

```
vl_rolodex(c(0, 4, 7))

vl_rolodex(c(0, 4, 7), reorder=FALSE)

#Multisets sort of work! Best resolutions from dom7 to triads with doubled root:
vl_rolodex(c(0, 4, 7, 10), c(0, 0, 4, 7))
```

`whichmodebest` *Smallest crossing-free voice leading between two pitch-class sets*

Description

Given source and goal pitch-class sets, which mode of the goal is closest to the source (assuming crossing-free voice leadings and the given `method` for determining distance).

Usage

```
whichmodebest(
  source,
  goal,
  method = c("taxicab", "euclidean", "chebyshev", "hamming"),
  no_ties = FALSE,
  edo = 12,
  rounder = 10
)
```

Arguments

`source`	Numeric vector, the pitch-class set at the start of your voice leading
`goal`	Numeric vector, the pitch-class set at the end of your voice leading
`method`	What distance metric should be used? Defaults to `"taxicab"` but can be `"euclidean"`, `"chebyshev"`, or `"hamming"`.
`no_ties`	If multiple VLs are equally small, should only one be returned? Defaults to `FALSE`, which is generally what an interactive user should want.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

Numeric value(s) identifying the modes of `goal`. Single value if `no_ties` is `TRUE`, otherwise n values for an n-way tie.

Examples

```
c_53 <- c(0, 4, 7)
c_64 <- c(7, 0, 4)
d_53 <- c(2, 6, 9)
e_53 <- c(4, 8, 11)

whichmodebest(c_53, c_64)
whichmodebest(c_64, c_53)
whichmodebest(c_53, e_53)
whichmodebest(c_53, d_53)
whichmodebest(c_53, d_53, method="euclidean")

# See "Modal Color Theory," p. 12, note 21
pyth_dia_modes <- sim(sort((j(5) * 0:6)%%12))
pyth_lydian <- pyth_dia_modes[,1]
pyth_locrian <- pyth_dia_modes[,4]
whichmodebest(pyth_locrian, pyth_lydian)
```

`whichsvzeroes` *Which interval-comparison equalities does a scale satisfy?*

Description

As "Modal Color Theory" (p. 26) describes, one useful measure of a scale's **regularity** is the number of zeroes in its sign vector. This indicates how many hyperplanes a scale lies *on*, a geometrical fact whose musical interpretation is, roughly speaking, how many times two generic intervals equal each other in specific size. (I say only "roughly speaking" because one hyperplane usually represents multiple comparisons: see Appendix 1.1.) Scales with a great degree of symmetry or other forms of regularity such as well-formedness tend to be on a very high number of hyperplanes compared to all sets of a given cardinality.

musicMCT offers two convenience functions that return pertinent information from signvector(). countsvzeroes returns this **count** of the number of **s**ign-**v**ector **zeroes**, while whichsvzeroes gives a list of the specific hyperplanes the scale lines on (numbered according to their position on the given ineqmat). The specific information in whichsvzeroes can be useful because it determines the "flat" of the hyperplane arrangement that the scale lies on, which is a more general kind of scalar structure than color (as determined by the entire sign vector).

Usage

```
whichsvzeroes(set, ineqmat = NULL, edo = 12, rounder = 10)

countsvzeroes(set, ineqmat = NULL, edo = 12, rounder = 10)
```

Arguments

set	Numeric vector of pitch-classes in the set
ineqmat	Specifies which hyperplane arrangement to consider. By default (or by explicitly entering "mct") it supplies the standard "Modal Color Theory" arrangements of getineqmat(), but can be set to strings "white," "black", "gray", "roth", "infrared", "pastel", "rosy", "infrared", or "anaglyph", giving the ineqmats of make_white_ineqmat(), make_black_ineqmat(), make_gray_ineqmat(), make_roth_ineqmat(), make_infrared_ineqmat(), make_pastel_ineqmat(), make_rosy_ineqmat(), make_infrared_ineqmat(), or make_anaglyph_ineqmat(). For other arrangements, this parameter accepts explicit matrices.
edo	Number of unit steps in an octave. Defaults to 12.
rounder	Numeric (expected integer), defaults to 10: number of decimal places to round to when testing for equality.

Value

Single numeric value for countsvzeroes and a numeric vector for whichsvzeroes

Examples

```
# Sort 12edo heptachords by how many sign vector zeroes they have (from high to low)
heptas12 <- unique(apply(utils::combn(12, 7), 2, primeform), MARGIN=2)
heptas12_svzeroes <- apply(heptas12, 2, countsvzeroes)
colnames(heptas12) <- apply(heptas12, 2, fortenum)
heptas12[, order(heptas12_svzeroes, decreasing=TRUE)]

# Multiple hexachords on the same flat but of different colors
hex1 <- c(0, 2, 4, 5, 7, 9)
hex2 <- convert(c(0, 1, 2, 4, 5, 6), 9, 12)
hex3 <- convert(c(0, 3, 6, 8, 11, 14), 15, 12)
hex_words <- rbind(asword(hex1), asword(hex2), asword(hex3))
rownames(hex_words) <- c("hex1", "hex2", "hex3")
c(colornum(hex1), colornum(hex2), colornum(hex3))
whichsvzeroes(hex1)
whichsvzeroes(hex2)
whichsvzeroes(hex3)
```

`hex_words`

`writeSCL` *Create a Scala tuning file from a given scale*

Description

You mean you don't want to play around in R forever? This function lets you export any scale you've defined in R as a .scl tuning file for use in Scala or by any synth that can read .scl files. Will write to your working directory.

In addition to saving the necessary tuning data, the function will attempt to add as comments extra information that can be derived from MCT functions, like the color number, degrees of freedom, number of sign-vector zeroes, etc.

Usage

```
writeSCL(x, path, filename, period = 2, ineqmat = NULL, edo = 12, rounder = 10)
```

Arguments

`x`	Numeric vector: the scale to export
`path`	String specifying path where Scala file should be saved. No default and cannot be missing.
`filename`	String (in quotation marks): what to name your Scala file. Defaults to using the name of `x` as the file name if you enter nothing.
`period`	The frequency ratio at which your scale repeats; defaults to `2` which indicates an octave-repeating scale.
`ineqmat`	Specifies which hyperplane arrangement to consider. By default (or by explicitly entering "mct") it supplies the standard "Modal Color Theory" arrangements of `getineqmat()`, but can be set to strings "white," "black", "gray", "roth", "infrared", "pastel", "rosy", "infrared", or "anaglyph", giving the `ineqmat`s of `make_white_ineqmat()`, `make_black_ineqmat()`, `make_gray_ineqmat()`, `make_roth_ineqmat()`, `make_infrared_ineqmat()`, `make_pastel_ineqmat()`, `make_rosy_ineqmat()`, `make_infrared_ineqmat()`, or `make_anaglyph_ineqmat()`. For other arrangements, this parameter accepts explicit matrices.
`edo`	Number of unit steps in an octave. Defaults to `12`.
`rounder`	Numeric (expected integer), defaults to `10`: number of decimal places to round to when testing for equality.

Value

Invisible `NULL`

Examples

```
neat_pentachord <- convert(c(0, 1, 4, 9, 11), 15, 12)

writeSCL(neat_pentachord, path=tempdir(), "neat_pentachord.scl")
```

---

z — *Frequency ratios to logarithmic pitch intervals (e.g. semitones)*

Description

Simple convenience function for converting frequency ratios to semitones. Useful to have in addition to `j()` because `j()` is only defined for specific common values. Defaults to 12-tone equal temperament but `edo` parameter allows other units.

Usage

```
z(..., edo = 12)
```

Arguments

`...`	One or more numerics values which represent frequency ratios.
`edo`	Number of unit steps in an octave. Defaults to `12`.

Details

The name `z()` doesn't make a lot of sense but has the virtue of being a letter that isn't otherwise very common. `r` (for ratio) and `q` (for the rationals) were both avoided because they're already used for other functions.

Value

Numeric vector representing the input ratios converted to `edo` unit steps per octave

See Also

`j()` is a more convenient input method for the most common frequency ratios.

Examples

```
z(81/80) == j(synt)

mod_jdia <- z(1, 10/9, 5/4, 4/3, 3/2, 5/3, 15/8)
minimize_vl(j(dia), mod_jdia)

z(1, 5/4, 3/2, edo=53)
```

`zmate` *Twin set in the Z-relation (Z mate)*

Description

For the standard 12edo sets of Fortean pitch-class set theory, given one pitch-class set, finds a set class whose interval-class vector is the same as the input set but which does not include the input set. Not all set classes participate in the Z-relation, in which case the function returns `NA`.

Usage

```
zmate(set)
```

Arguments

`set`	Numeric vector of pitch-classes in the set

Details

These values are hard-coded from Forte's list for non-hexachords and only work for subsets of the standard chromatic scale. `zmate()` doesn't even give you an option to work in a different `edo`. If it were to do so, I can't see a better solution than calculating all the set classes of a given cardinality on the spot, which can be slow for higher `edo`s.

Value

`NA` or numeric vector of same length as `set`

Examples

```
zmate(c(0, 4, 7))
zmate(c(0, 1, 4, 6))
```

Index