Package: bmetr (via r-universe)

September 18, 2024

Type Package

Title Beethoven's Metronome Analysis

Version 0.1.1

Description Supporting data and methods for Martín-Castro & Ucar (2020, <doi:10.1371/journal.pone.0243616>). Includes functions and utilities for modelling a mechanical metronome and filtering tempo signals.

License MIT + file LICENSE

Encoding UTF-8

URL https://github.com/Enchufa2/bmetr

BugReports https://github.com/Enchufa2/bmetr/issues

LazyData true

Depends R (>= 3.5.0)

Imports stats, graphics, errors, dplyr

Suggests zoo, tuneR, lme4, merTools, performance, ggplot2, ggridges, ggthemes, ggpmisc, patchwork, knitr, rmarkdown, tufte

RoxygenNote 7.1.1

Roxygen list(old_usage = TRUE)

VignetteBuilder knitr

Repository https://enchufa2.r-universe.dev

RemoteUrl https://github.com/Enchufa2/bmetr

RemoteRef HEAD

RemoteSha f369b81990fb9bf0a9e0dc808c038846d0c4ff78

Contents

`bmetr-package` **bmetr***: Beethoven's Metronome Analysis*

Description

Supporting data and methods for Martín-Castro & Ucar (2020). Includes functions and utilities for modelling a mechanical metronome and filtering tempo signals. See the vignettes for further details.

Author(s)

Almudena Martín-Castro, Iñaki Ucar

References

Martín-Castro A, Ucar I (2020). Conductors' tempo choices shed light over Beethoven's metronome. *PLOS ONE* 15(12), e0243616. doi:10.1371/journal.pone.0243616

`errors` *Data Wrangling with* **errors**

Description

Auxiliary functions to perform some common data wrangling operations in an **errors**-aware fashion, i.e., to preserve uncertainty metadata.

Usage

```
gather_errors(.data, key, value, ...)

unite_errors(.data, suffix = ".se")

separate_errors(.data, suffix = ".se")
```

Arguments

`.data`	a data frame.
`key`, `value`	names of the new key and value columns, as strings.
`...`	a selection of columns to gather.
`suffix`	suffix for the error column(s).

Details

gather_errors is the **errors**-aware equivalent to tidyr::gather.

unite_errors and separate_errors are similar to tidyr::unite and tidyr::separate, but only for errors objects. For example, for each variable var of class errors, separate_errors will store the numeric value in var and will create another variable called var.se for the errors. Similarly, for each pair of variables var and var.se, unite_errors will store an errors object in var and will remove var.se. Note that the suffix .se can be changed.

---

find_peaks *Find Local Peaks*

Description

Find local maxima in a neighbourhood. To find local minima, just provide -x instead of x.

Usage

```
find_peaks(x, m = 3)
```

Arguments

x	numeric vector.
m	size of the neighbourhood.

Value

A vector of indices.

References

Based on https://github.com/stas-g/findPeaks.

---

metr.data *Metronomes Data*

Description

Measurements of five models of metronome: Neewer, Neewer (photo), Patent, TB 06 and TB 07. The first two correspond to a contemporary metronome, a Neewer NW-707. The difference is that Neewer corresponds to precise direct measurements in a dismantled metronome, while Neewer (photo) measurements were taken from a photograph. The rest of the models follow this methodology. The third one, Patent, corresponds to the original diagram from Maelzel's 1815 English patent. Finally, TB 06 and TB 07 are metronomes number 6 and 7, respectively, from Tony Bingham's collection (see the references).

Usage

```
metr.neewer

metr.marks

metr.params
```

Format

`metr.neewer` provides oscillation frequency measurements for the `Neewer` metronome.

`mark`	metronome mark, in pulses per minute.
`bpm`	measured frequency, in pulses per minute.
`bpm.se`	standard error of `bpm`, in the same units.

`metr.marks` provides position measurements of each metronome mark for all the metronomes.

`model`	metronome model.
`mark`	metronome mark, in pulses per minute.
`r`	distance from the shaft to the mark, in mm.
`r.se`	standard error of `r`, in the same units.

`metr.params` provides measurements of fixed dimensions for all the metronomes. All the additional columns suffixed with `.se` are the standard errors of the corresponding variables in the same units.

`model`	metronome model.
`rcm`	position of the center of mass of the upper mass with respect to the metronome mark, in mm.
`l`	length of the rod over the shaft, in mm.
`R`	distance from the shaft to the center of mass of the lower mass, in mm.
`L`	length of the rod below the shaft, in mm.
`A`	oscillation amplitude, in degrees.

References

Bingham, T. and Turner, A. (2017) *Metronomes and Musical Time: Catalogue of the Tony Bingham Collection at the exhibition AUF TAKT! Held in the Museum für Musik, Basel*. London. ISBN: 978-0-946113-11-8.

`metr_fit` *Metronome Model Fit*

Description

Fit metronome's nondimensionalized masses, `M.` and `m.` (see `metr_model`), from a set of measurements and parameters.

Usage

```
metr_fit(.data, params, by = "model")

metr_predict(.fit, by = "model")
```

Arguments

.data	measurements of distance to the shaft for each metronome mark. It must contain the variables model, mark and r, as, e.g., the metr.marks data set.
params	the rest of the dimensions: correction of the center of mass of the upper mass rcm, length of the rod above the shaft l, length of the rod below the shaft L, distance of the lower mass to the shaft R and oscillation amplitude A, as, e.g., the metr.params data set.
by	variable to group by; e.g., the metronome model, to fit several metronomes.
.fit	output from metr_fit.

Value

A grouped data frame with model, M. (and error), mu. (and error) and the fit object.

---

metr_model *Metronome Model*

Description

Functions to compute a metronome's oscillation frequency. metr_model is the main function. It computes the base value (metr_model_base), which includes the contribution of the mass of the rod (mu.), and applies corrections due to the oscillation angle (metr_model_angle) and the friction (metr_model_friction).

Usage

```
metr_model(r, R = 50, M. = 5, l = 220, L = R, mu. = 0, g = 9.807,
  A = 0, ep0 = 0)

metr_model_base(r, R = 50, M. = 5, l = 220, L = R, mu. = 0, g = 9.807)

metr_model_angle(A, terms = 8)

metr_model_friction(r, R = 50, M. = 5, l = 220, L = R, mu. = 0, ep0 = 0)

metr_model_bias(mark, R = 50, M. = 5, l = 220, L = R, mu. = 0,
  g = 9.807, A = 0, ep0 = 0, shift = 0)

metr_model_r(mark, R = 50, M. = 5, l = 220, L = R, mu. = 0, g = 9.807, A = 0)
```

Arguments

r	distance from the shaft to the center of mass of the upper mass, in mm.
R	distance from the shaft to the center of mass of the lower mass, in mm.
M.	nondimensionalized lower mass (lower mass divided by upper mass).
l	length of the rod above the shaft, in mm.
L	length of the rod below the shaft, in mm.
mu.	nondimensionalized rod mass (rod mass divided by upper mass).
g	standard gravity, in m/s^2.
A	oscillation amplitude, in degrees.
ep0	nondimensional friction parameter (>= 0).
terms	number of terms to approximate the angle correction.
mark	metronome mark, in pulses per minute.
shift	include a shift of the rod with respect to the scale, in mm.

Details

Function metr_model_bias computes, for a given metronome mark, the resulting oscillation frequency for an alteration of any parameter. To this end, it uses metr_model_r to first calculate the position of the upper mass for this metronome mark.

A value of mu.=0 means that the rod is considered massless. A value of A=0 means that no correction is applied due to the oscillation amplitude. A value of ep0=0 means that the movement is frictionless. A value of ep0=0.5 means that the metronome stops due to friction at the lowest possible frequency.

To compute the frequency bias for any parameter, two values must be supplied to metr_model_bias. For example, if the original position of the lower mass is 50 mm, and we want to calculate the result of lowering it down 5 mm, then R=c(50, 55) should be supplied.

Value

metr_model, metr_model_base and metr_model_bias return the oscillation frequency in pulses per minute.

metr_model_angle and metr_model_friction return a value >= 1.

metr_model_r returns the distance from the shaft to the upper mass in mm.

sym.data *Beethoven's Symphonies Data*

Description

Six data sets with tempo measurements plus additional information about Beethoven's nine symphonies.

- Data sets `sym.marks`, `sym.recordings` and `sym.duration` provide detailed information about Beethoven's annotations as well as the recordings selected for this study (the complete symphonies from 36 different conductors).
- Data sets `sym.window`, and `sym.sample` provide tempo measurements extracted using the Marsyas framework (and the tempo estimation algorithm by Percival and Tzanetakis; see the references) and distinct methodologies.

Usage

```
sym.marks

sym.recordings

sym.duration

sym.window

sym.sample
```

Format

`sym.marks` provides complete timing information extracted from Beethoven's scores: character annotations, tempo markings, beats and bars for the nine symphonies. Note that some movements have several character annotations (more than one section).

`symphony`	symphony number.
`movement`	movement number.
`character`	character annotation in this section.
`mark`	metronome mark.
`beat`	beats per bar.
`bar.tot`	total number of bars in this section.
`bar.rep`	total number of repeated bars in this section.
`bar.1`	total number of bars in first repetition boxes.
`tsig`	time signature, as expected by `tmp_rectify_tsig`
`section`	convenience section identifier (one per character).

Therefore, the theoretical duration, in minutes, *without repetitions* for each section can be computed as follows: `(bar.tot - bar.1) * beat / mark`. And the theoretical duration, in minutes, *with repetitions* is just the previous value + `bar.rep * beat / mark`.

`sym.recordings` provides complete information about the albums analyzed in this work (the nine symphonies for 36 different conductors).

`conductor`	conductor's name.
`orchestra`	orchestra(s)'s name.
`title`	album title.
`label`	album label.
`upc`	Universal Product Code.
`date`	recording dates.
`year`	release date.
`ptype`	performance type: romantic, Historically Informed (HI) or under HI influence.

`sym.duration` provides performed duration (track length) for the nine symphonies and 36 different conductors.

`symphony`	symphony number.
`movement`	movement number.
`conductor`	conductor's name.
`duration`	track length, in seconds.

`sym.window` provides continuous tempo measurements per symphony, movement and conductor by means of a sliding window.

`symphony`	symphony number.
`movement`	movement number.
`conductor`	conductor's name.
`n`	sample index. Samples with `n=0` correspond to an estimation for the entire track.
`start`	start time for the sliding window, in seconds.
`duration`	window length, in seconds.
`tempo`	estimated tempo.

`sym.sample`, in contrast to `sym.window`, provides a single tempo estimation for each symphony, movement and conductor. The sample was collected at the end of each track (the "coda" of the movement), where the tempo is arguably more stable. It has the same variables as `sym.window`, except for `n`. Instead, `sym.sample` contains a `section` column to identify the section from which the tempo was sampled. This section identifier corresponds to the homonymous one present in `sym.marks`.

References

Tzanetakis, G. and Cook, P. (2000) "MARSYAS: a framework for audio analysis." *Organised Sound*, 4(3):169-175

Percival, G. and Tzanetakis, G. (2013) "An effective, simple tempo estimation method based on self-similarity and regularity." *IEEE International Conference on Acoustics, Speech and Signal Processing*, 241-245.

Percival, G. and Tzanetakis, G. (2014) "Streamlined Tempo Estimation Based on Autocorrelation and Cross-correlation with Pulses." *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, 22(12):1765-1776.

`tmp_rectify` *Tempo Correction and Smoothing*

Description

Functions for data cleaning. `tmp_rectify` is a general function to rectify a set of tempo harmonics in a sequence given some reference. `tmp_rectify_tsig` and `tmp_smooth` are wrappers around the latter to rectify a specific set of tempo harmonics given a time signature and to smooth a sequence of tempi respectively. The function `tmp_prevalent` is intended to find the most prevalent tempo in a sequence of tempi.

Usage

```
tmp_rectify(x, ref, harmonics, rtol, rtol2 = rtol, cond = any)

tmp_rectify_tsig(x, ref, tsig, rtol = 0.15, rtol2 = rtol)

tmp_smooth(x, ref, tsig, rtol = 0.1, rtol2 = rtol, ref2 = 1, nref = 3)

tmp_prevalent(x, breaks = 30)
```

Arguments

`x`	a sequence of tempi.
`ref`	tempo reference(s) for the harmonics.
`harmonics`	set of tempo harmonics to rectify.
`rtol`, `rtol2`	ratios of tolerance to compute the bounds and determine whether a given tempo is a harmonic of `ref`. `rtol2` can be specified to implement asymmetric bounds; otherwise, they are symmetric.
`cond`	condition to apply the correction.
`tsig`	time signature. Notably, this function distinguishes between binary time signatures (specified by `2`), time signatures with ternary subdivision (specified by `0.3`).
`ref2`	additional tempo harmonic to consider.
`nref`	number of previous samples to take into account.
`breaks`	number of breaks to compute the histogram.

Value

The rectified sequence, except for `tmp_prevalent`, which returns a single value.

Index