Package: EGM (via r-universe)

October 6, 2026

Title Intracardiac Electrograms

Version 0.3.0

Description A system for importing electrophysiological signal, based on the 'Waveform Database (WFDB)' software package, written by Moody et al 2022 <doi:10.13026/gjvw-1m31>. A R-based system to utilize 'WFDB' functions for reading and writing signal data, as well as functions for analysis, are provided. A stable and broadly compatible class for working with signal data, supporting the reading in of cardiac electrophysiological files such as intracardiac electrograms, is introduced.

License GPL (>= 3)

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

Depends R (>= 4.2.0),

Imports vctrs (>= 0.5.0), data.table (>= 1.15.0), stats, tools, utils, xml2, base64enc, stringr, signal, S7 (>= 0.2.0)

Suggests covr, knitr, rmarkdown, testthat (>= 3.0.0), withr, moments, lifecycle, quarto

LinkingTo cpp11

Config/testthat/edition 3

URL https://shah-in-boots.github.io/EGM/

BugReports https://github.com/shah-in-boots/EGM/issues

VignetteBuilder knitr, quarto

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libicu-dev libxml2-dev

Repository https://shah-in-boots.r-universe.dev

Date/Publication 2026-09-06 02:25:03 UTC

RemoteUrl https://github.com/shah-in-boots/egm

RemoteRef HEAD

RemoteSha 4b9c250ab584e727be808431c4d83965235b7f52

Contents

[.windows . . . 3
add_annotation . . . 4
analyze_atrial_signal . . . 5
annotation_table . . . 6
annotators . . . 8
as_ECG . . . 9
bard . . . 10
baseline_window . . . 12
by_beat . . . 14
by_pwave . . . 16
by_rhythm . . . 18
c.windows . . . 21
calculate_approximate_entropy . . . 21
calculate_dominant_frequency . . . 22
calculate_organization_index . . . 23
calculate_sample_entropy . . . 25
calculate_welch_spectrum . . . 26
cancel_ventricular_signal . . . 27
change_frequency . . . 29
channel_zero . . . 31
channels . . . 32
detect_QRS . . . 34
ECG . . . 35
ecg_leads . . . 36
EGM . . . 38
extract_f_waves . . . 39
f_wave_diagnostics . . . 46
feature_range . . . 48
format.windows . . . 49
frequency . . . 49
get_annotation . . . 51
get_signal . . . 52
get_windows . . . 53
header_table . . . 55
is_template . . . 57
is_window_set . . . 58
is_window_strategy . . . 58
kors . . . 59
label_waves . . . 60
landmark . . . 61
learn_template . . . 62
list_annotators . . . 63
map_windows . . . 64
median_window . . . 65
merge_annotations . . . 67
muse . . . 67

new_windows 68
normalize_window 69
pad_window 71
print.windows 74
prucka 75
signal_table 76
signal_units 77
template 78
validate_time_parameters 79
vcg 80
warp_window 84
wfdb 86
wfdb_annotation_labels 87
wfdb_annotations 88
wfdb_io 91
window_dropped 94
window_strategy 95

Index **96**

`[.windows` *Subset a* `windows` *object*

Description

Subset a `windows` object

Usage

```
## S3 method for class 'windows'
x[i, ...]
```

Arguments

`x`	A `windows` object.
`i`	Index to subset.
`...`	Additional arguments passed to methods.

Value

A `windows` object with the specified subset of elements.

add_annotation *Add an annotation table to an* EGM *object*

Description

Add an annotation_table to an EGM object, with validation to ensure compatibility. The annotation table is added to the list of annotations using the annotator name as the list name. If an annotation with the same annotator name already exists, you can choose to overwrite it or merge the annotations.

Usage

```
add_annotation(x, annotation, overwrite = FALSE)
```

Arguments

x	An EGM object to which the annotation will be added
annotation	An annotation_table object to add to the EGM object
overwrite	Logical. If an annotation with the same annotator name already exists, should it be overwritten? If FALSE (default), the annotations will be merged. If TRUE, the existing annotation will be replaced.

Value

The modified EGM object with the annotation table added

Validation

The function performs several validation checks to ensure the annotation table is compatible with the EGM object:

- **Channel validation**: A channel given as a lead name ("II") is resolved against the record's header to its signal number, so the table stored on the object always counts channels the way channels describes. Every channel other than the global channel 0 has to exist in the header; a name the record does not carry is an error.
- **Sample validation**: Checks that all sample indices in the annotation table are within the valid range of samples in the recording.

Handling Existing Annotations

When adding an annotation table with an annotator name that already exists:

- **Overwrite mode** (overwrite = TRUE): The existing annotation table is completely replaced with the new one.
- **Merge mode** (overwrite = FALSE): The new annotations are combined with the existing ones, duplicates are removed, and the result is sorted by sample number.

Empty Annotations

If the EGM object has no existing annotations (empty list or blank annotation table), the new annotation table will replace the empty annotations.

Examples

```
## Not run:
# Add a new annotation table to an `EGM` object
`EGM` <- add_annotation(`EGM`, my_annotation_table)

# Overwrite an existing annotation
`EGM` <- add_annotation(`EGM`, new_annotation, overwrite = TRUE)

# Merge with existing annotation (default)
`EGM` <- add_annotation(`EGM`, additional_annotation, overwrite = FALSE)

## End(Not run)
```

---

analyze_atrial_signal *Analyse an isolated atrial signal*

Description

Analyse an isolated atrial signal

Usage

```
analyze_atrial_signal(
  atrial_signal,
  frequency,
  characteristics = c("amplitude", "dominant_frequency", "organization",
    "sample_entropy"),
  original_signal = NULL,
  raw_signal = original_signal,
  qrs_loc = NULL,
  annotation = NULL,
  amplitude_window = c("tq", "all"),
  band = c(4, 10),
  dominant_frequency = NULL,
  entropy_rate = 256,
  entropy_input = c("broadband", "main_atrial_wave"),
  entropy_bandwidth = 3,
  entropy_tolerance = 0.2,
  tol = 0.15
)
```

Arguments

`atrial_signal`	Numeric vector of the cancelled (atrial) signal
`frequency`	Sampling frequency in Hz
`characteristics`	Character vector of features to compute
`original_signal`	The uncancelled, filtered signal for the same lead, which is what the cancellation residual and the harmonic evidence are measured against.
`raw_signal`	The same lead before bandpass filtering, used for the QRS excursion. Falls back to `original_signal`. The passband reaches only 30 Hz, and the QRS carries energy above it, so a filtered excursion is small by 4-37% across the leads of the bundled records, median 14%, and by how much depends on the lead and the QRS width – which would put a morphology-dependent factor into `f_ratio`, the one amplitude measure whose whole purpose is comparability between patients.
`qrs_loc`	Integer vector of QRS positions
`annotation`	Optional annotation table used to locate TQ segments. It must already be resolved to a single lead; see `extract_f_waves()`.
`amplitude_window`	Either `"tq"` or `"all"`
`band`	Numeric length-2 frequency band in Hz
`dominant_frequency`	Dominant frequency in Hz to use for the rate, the organisation index and the main atrial wave, or `NULL` (default) to estimate it from `atrial_signal`. `extract_f_waves()` passes the record's in, so every lead is read at one frequency.
`entropy_rate`	Rate in Hz to decimate to before computing entropy
`entropy_input, entropy_bandwidth, entropy_tolerance`	What the entropy is taken of, how wide a band around `dominant_frequency` the main atrial wave keeps, and the tolerance `r` as a multiple of the standard deviation; see `extract_f_waves()`.
`tol`	Tolerance for the harmonic coincidence test

Value

A named list of features

`annotation_table` *Annotation Table*

Description

`annotation_table()` modifies the `data.table` class to work with annotation data. The columns are of all equal length, and each row describes a single annotation (although there may be duplicate time points).

Usage

```
annotation_table(
  annotator = character(),
  time = character(),
  sample = integer(),
  frequency = integer(),
  type = character(),
  subtype = character(),
  channel = integer(),
  number = integer(),
  aux = character(),
  channel_zero = c("global", "signal"),
  ...
)

is_annotation_table(x)
```

Arguments

annotator	Character vector of WFDB-compatible annotation types. When reading, multiple annotators can be specified to read multiple annotation files simultaneously (e.g., `c("ecgpuwave", "atr")`). When writing, must be a single string serving as the file extension. See `read_annotation()` and `write_annotation()` for further details.
time	A `character` time stamp of the annotation, written in the format of **HH:MM:SS.SSS**, starting at **00:00:00.000**. This is converted to the appropriate time based on the header file (which records the actual start time and sampling frequency). This is often a missing variable and is given for compatibility with the WFDB applications.
sample	An `integer` representing the sample number of the annotation
frequency	An `integer` that represents the sampling frequency in Hertz
type	A `character` or string representing the type of the annotation
subtype	A `character` or string representing the subtype of the annotation
channel	An `integer` representing the channel number of the annotation, or a `character` representing the channel name. A name is resolved to its signal number when the table is added to a record with `add_annotation()`; `write_annotation()` refuses a table still holding names, since the file stores an integer. See channels for how the numbers count.
number	An additional `integer` value or number that classifies the annotation (allows for compatibility with multiple annotation types)
aux	A `character` auxiliary string carried with each annotation, the WFDB aux field. Free text, such as a rhythm label (`"(AFIB"`) or a note from the annotator. Defaults to empty.
channel_zero	How the `channel` values given count signals: `"global"` (default), from `1` with `0` the global channel, which is how the table holds them; or `"signal"`, from `0`

	as the WFDB tools count, in which case one is added to each so the table counts from 1 like every other, and `write_annotation()` takes it off again by default. See channels.
`...`	Additional arguments to be passed to the function
`x`	A `data.table` object that represents an annotation table

Details

The `annotation_table()` function creates a compatible table that can be used with `write_annotation()` and `read_annotation()` functions.

Value

A `data.table` that has invariant columns that are compatible with the WFDB library. The key columns include the sample index, the type of annotation (and its subtype and number qualifier), and the channel.

Annotation files

The following annotation file types are described below.

`ecgpuwave`**:**

`ecgpuwave` analyzes an ECG signal from the specified record, detecting the QRS complexes and locating the beginning, peak, and end of the P, QRS, and ST-T waveforms. The output of ecgpuwave is written as a standard WFDB-format annotation file (the extension is "*.ecgpuwave", as would be expected). This file can be converted into text format using `rdann`. Further details are given at the ECGPUWAVE page.

The **type** column can be *p*, *t*, or *N* for the peak of the P wave, T wave, and QRS (R peak) directly. The output notation also includes waveform onset `(` and waveform offset `)`. The **number** column gives further information about each of these **type** labels.

The **number** column gives modifier information. If the **type** classifier is a T wave annotation, the **number** column can be 0 (normal), 1 (inverted), 2 (positive), 3 (negative), 4 (biphasic negative-positive), 5 (biphasic positive-negative). If the **type** is an waveform onset or offset, then **number** can be 0 (P wave), 1 (QRS complex), 2 (T wave).

`annotators` *Annotator systems for WFDB objects*

Description

These functions create templates for annotation in R and extend the ability for developers to create their own annotation systems that are stable for WFDB objects. They are compatible with WFDB annotations and can be written out to a WFDB-compatible file. This also allows extensibility.

Choosing an annotator

What this package needs from a delineating annotator is narrower than it looks. Wave identity is recovered positionally by `label_waves()`, from the peak symbol enclosed by each `(`/`)` bracket pair, so an annotator that leaves the WFDB `number` column at zero throughout - which by the letter of the convention means every onset marks a P wave - is still perfectly usable. The requirement is only that each wave is bracketed around a typed peak (`p`, `N`, or `t`).

An annotator run per lead writes one such set of brackets for each lead, distinguished by the `channel` column, which is resolved by naming a guiding `channel`; see channels.

See Also

`label_waves()`, `read_annotation()`, channels

---

as_ECG *Convert an EGM object to an ECG object*

Description

Extracts the surface ECG from a record. Recognised surface leads are kept and renamed to their canonical form; every other channel is discarded. On a 12-lead recording this is a relabelling; on an electrophysiology study it is the step that separates the body-surface leads from the intracardiac ones.

Usage

```
as_ECG(x, ...)
```

Arguments

`x`	An object of class `EGM`
`...`	Additional arguments

Details

Channels are matched case-insensitively and ignoring separators, so `aVR`, `av r` and `AV-R` all resolve to `AVR`. A record with no surface leads at all is an error: an intracardiac channel cannot stand in for a surface lead, and the analyses gated on this class would otherwise return a number that looks reasonable and is not.

Annotation `channel` indices refer to positions in the original record. When channels are dropped those indices no longer address the leads they name, so a warning is raised if any channel-specific annotation is carried across.

For the same reason the kept leads stay in the order the record wrote them, rather than being put into display order. A channel index addresses a signal column by position, so permuting the columns would repoint every per-lead fiducial at a different lead - silently, since the indices remain valid. Renumbering them is not open to this function either: whether an annotator counted from zero or from one is a property of the file, and channel `0` is also how a fiducial says it belongs to no particular lead. Order at the point of use instead, with `ecg_leads()` and `lead_factor()`.

Value

An object of class ECG, holding its leads in the source record's order.

See Also

ecg_leads() and lead_factor() for the display order, ECG() for the class, extract_f_waves() and vcg() for analyses that require it.

---

bard *Read in ECG and EGM data from Bard (LabSystem Pro)*

Description

This function allows for reading in LS Pro data based on their text export of signals. Signals can be exported directly from the LS Pro system. The actual software is written by *Bard*.

The LabSystem Pro was acquired by Boston Scientific from the original company Bard. They are a common electrophysiology signal processing device for visualization and measurement of intracardiac signals.

Usage

```
read_bard(file, n = Inf)

read_bard_header(file)

read_bard_signal(file, n = Inf)
```

Arguments

file	The path to the file where the data is located. It must be a ***.txt** file. See details below about its format.
n	Number of signal values to return (this will be the same for each channel of data). Defaults to all values.

Value

An EGM class object that is a list of EP signals the format of a data.table, with an attached **header** attribute that contains additional recording data.

Data Export

The steps to data export are as follows.

1. Start LabSystem PRO
2. Open a patient record
3. Display a waveform recording in a Review Window

4. Scroll to a point of interest in a waveform recording
5. Right click on the review window to the left of the region of interest
6. Select an Export option, either a default time range or the entire visible page (which depends on the sweep speed).

Data Format

```
[Header] Recording info - contains (example):

[Header]<CR><LF>
  File Type: 1<CR><LF>
  Version: 1<CR><LF>
  Channels exported: 22<CR><LF>
  Samples per channel: 5000<CR><LF>
  Start time:  6:55:24<CR><LF>
  End time:  6:55:29<CR><LF>
  Ch. Info. Pointer: 320<CR><LF>
  Stamp Data: T<CR><LF>
  Mux format: 0<CR><LF>
  Mux Block Size: <CR><LF>
  Data Format 1<CR><LF>
  Sample Rate: 1000Hz<CR><LF>

[Header] Channel info (per channel example):

  Channel #: 1<CR><LF>
  Label: III<CR><LF>
  Range: 5mv <CR><LF>
  Low: 1Hz<CR><LF>
  High: 100Hz<CR><LF>
  Sample rate: 1000Hz<CR><LF>
  Color: 0000FF<CR><LF>
  Scale: -7<CR><LF>

[Data] As described below:

 -256,-1056,576,-256,320,-736,144,576,-592,176,608,240,176,-560,496,-
 144,0,0,-32,-48,-32,-80<CR><LF>
```

Channel Data is interleaved in the example above (sample indexed at 1):

1	2	3	...	22
Ch1:1	Ch2:1	Ch3:1	...	Ch22:1
Ch1:2	Ch2:2	Ch3:2	...	Ch22:2
Ch1:3	Ch2:3	Ch3:3	...	Ch22:3
...	...	...	...	...
Ch1:5000	Ch2:5000	Ch3:5000	...	Ch22:5000

`baseline_window` *Reference windows to their isoelectric baseline*

Description

[Experimental]

Subtracts each window's own baseline from every lead, so that the isoelectric line sits at zero. Nothing else in the windowing chain does this: reading, cutting, padding, warping and reducing all preserve whatever DC offset the recording carried, and that offset is not small next to a P wave.

Usage

```
baseline_window(
  x,
  reference = "(",
  width = 20,
  channel = NULL,
  preserve_class = TRUE,
  ...
)
```

Arguments

`x`	A `windows` object, a list of `EGM` objects, or a single `EGM`.
`reference`	What marks the isoelectric segment: a fiducial (a `character` type symbol, default `"("`, or a named list of annotation criteria), `"start"`, `"end"`, or a `numeric` level to subtract directly.
`width`	Milliseconds of signal averaged to estimate the level. Default 20. Ignored when `reference` is numeric.
`channel`	The lead whose annotations locate `reference`, given as a channel number or name. Required when the annotations span more than one channel; see the channels section.
`preserve_class`	Logical. If TRUE (default), a `windows` input returns a `windows` object; if FALSE, a plain list of `EGM` objects. A single `EGM` always returns a single `EGM`.
`...`	Additional arguments (currently unused).

Details

The offset survives every step that does not go looking for it, and it dominates the ones that do. Run a principal components analysis over a set of median beats without correcting first and the leading components are a constant vertical shift of the whole window rather than any morphology - the offset is the largest thing varying between records, so a variance-based method finds it before anything else. It is invisible in a scree table and obvious the moment a component is plotted as a waveform.

Correct *before* reducing, not after. A per-beat correction also removes the beat-to-beat baseline wander that would otherwise smear the reduction:

```
get_windows(ecg, by = by_beat(channel = 2)) |>
  baseline_window(reference = "(", channel = 2) |>
  median_window()
```

Which samples count as isoelectric depends on `reference`:

- A fiducial - given as a type symbol such as `"("`, or a named list of annotation criteria - takes the `width` milliseconds *before* it. Anchored on a wave onset this is the PR or TP segment, which is the segment a clinician would call the baseline.
- `"start"` takes the first `width` milliseconds of the window, and `"end"` the last. Use these where the window has no delineation to anchor to, or where it was cut so that its edges are known to be quiet.
- A `numeric` scalar is subtracted from every lead as given, which is the escape hatch for a baseline estimated some other way.

The level is the *median* over the reference samples rather than the mean, so a single noisy sample cannot displace the whole window. Each lead is corrected by its own level; a window whose reference fiducial is missing is returned uncorrected with a warning, since shifting it by some other window's baseline would be worse than leaving it alone.

Value

The input with every lead shifted so its baseline is zero, in the shape it was given: a `windows` object, a list, or a single `EGM`.

Guiding channel

A delineating annotator may be run over a record once, or once per lead. Run per lead it writes twelve independent copies of every fiducial, separated only by the `channel` column, and such a table does not describe beats until it is resolved to a single lead: pooled, it reports twelve times as many beats as the record contains, and taking the first match at each point silently follows whichever lead happens to sort first.

So every function that reads fiducials - `get_windows()`, `learn_template()`, `median_window()` and the other window transforms, `extract_f_waves()`, `vcg()` - takes a `channel`, and resolves it the same way:

- annotations spanning more than one channel with no `channel` given are an error, not a guess;
- a `channel` that the annotations do not carry is an error, which is what catches a numbering convention that does not match;
- the global channel `0` is not a lead, so it never counts toward the span and is always kept alongside the chosen lead.

The channel may be a number or a channel name (`"II"`), resolved against the record's own header. A number is matched literally against the `channel` column, and the numbering there belongs to the annotator that wrote the file. Read it off the record rather than assuming:

```
table(get_annotation(x)$channel)
```

Only the boundaries are restricted to one lead. Windows still carry the signal for every channel, which is what makes a windowed beat usable as a vectorcardiogram.

See Also

pad_window(), median_window(), vcg(), which references each loop to its own onset by the same rule

Examples

```
## Not run:
ecg <- read_wfdb("ecg", test_path(), "ecgpuwave")
beats <- get_windows(ecg, by = by_beat(channel = 1))

# The PR segment, taken as the 20 ms before each QRS onset
baseline_window(beats, reference = list(type = "(", wave = "QRS"), channel = 1)

# A median beat corrected after the fact
median_window(beats) |> baseline_window(reference = "start")

## End(Not run)
```

---

by_beat *Window a fixed span around each beat*

Description

[Experimental]

Builds a window_strategy that cuts the same span of signal around every occurrence of a fiducial, by default the QRS peak. Every window is the same length by construction, which is what by_rhythm() cannot promise.

Usage

```
by_beat(
  before = 300,
  after = 500,
  feature = "N",
  channel = NULL,
  adjust_sample_indices = TRUE
)
```

Arguments

before, after	Milliseconds of signal kept before and after the fiducial. The defaults span a full PQRST at ordinary rates.
feature	The fiducial each window is built around, given as a character type symbol (default "N", the QRS peak) or a named list of annotation criteria. Resolved the same way as a by_rhythm() criterion.

`channel` The lead whose annotations locate the fiducial, given as a channel number or name, as in `by_rhythm()`. See the channels section.

`adjust_sample_indices`
Logical, whether to rebase sample indices in the returned windows to be zero-based and relative to the window start.

Details

That guarantee is what a representative beat needs. Reducing ragged windows means padding them onto a common grid first, and a padded sample is a fabricated one: it pulls `median_window()` toward whatever the padding says, wherever the windows do not all reach. Cutting a fixed span out of the continuous recording leaves nothing to pad, which is how the standard representative beat is derived (Kligfield et al. 2007).

Beats too near either end of the record for the full span to be cut are dropped rather than truncated, since a short window gives up the guarantee the strategy exists for. How many were dropped is recorded on the returned collection, where printing it shows the count and `window_dropped()` reads it back. It is not also announced: a fixed span overhangs at least one end of a short strip almost every time, so the notice carried no information, and a message is invisible on a background worker regardless.

The span is given in milliseconds so that one strategy can be reused across records of differing sampling frequency.

Value

A window_strategy object.

Guiding channel

A delineating annotator may be run over a record once, or once per lead. Run per lead it writes twelve independent copies of every fiducial, separated only by the `channel` column, and such a table does not describe beats until it is resolved to a single lead: pooled, it reports twelve times as many beats as the record contains, and taking the first match at each point silently follows whichever lead happens to sort first.

So every function that reads fiducials - `get_windows()`, `learn_template()`, `median_window()` and the other window transforms, `extract_f_waves()`, `vcg()` - takes a `channel`, and resolves it the same way:

- annotations spanning more than one channel with no `channel` given are an error, not a guess;
- a `channel` that the annotations do not carry is an error, which is what catches a numbering convention that does not match;
- the global channel `0` is not a lead, so it never counts toward the span and is always kept alongside the chosen lead.

The channel may be a number or a channel name (`"II"`), resolved against the record's own header. A number is matched literally against the `channel` column, and the numbering there belongs to the annotator that wrote the file. Read it off the record rather than assuming:

```
table(get_annotation(x)$channel)
```

Only the boundaries are restricted to one lead. Windows still carry the signal for every channel, which is what makes a windowed beat usable as a vectorcardiogram.

References

Kligfield P, Gettes LS, Bailey JJ, et al. Recommendations for the standardization and interpretation of the electrocardiogram: part I. *Circulation*. 2007;115(10):1306-1324. doi:10.1161/CIRCULATIONAHA.106.180200

See Also

`get_windows()`, `median_window()`, `by_rhythm()`, `window_dropped()`

Examples

```
# A fixed PQRST window around every QRS peak
by_beat()

# Tight around the P wave instead, guided by lead II
by_beat(before = 200, after = 200, feature = "p", channel = 2)
```

---

by_pwave *Window the P wave*

Description

[Experimental]

Builds a window_strategy that cuts the atrial portion of each beat: from the P onset to either the QRS onset (default) or the P offset. Isolating the P wave is what makes atrial morphology modellable, since the QRS is an order of magnitude taller and otherwise absorbs the variance in any basis expansion fitted over a whole beat.

Usage

```
by_pwave(
  to = c("qrs_onset", "p_offset"),
  channel = NULL,
  adjust_sample_indices = TRUE
)
```

Arguments

`to`	Where the window ends: `"qrs_onset"` (default) or `"p_offset"`.
`channel`	The lead whose annotations define the window boundaries, given as a channel number or name. See the channels section.
`adjust_sample_indices`	Logical, whether to rebase sample indices in the returned windows to be zero-based and relative to the window start.

Details

Ending at the QRS onset keeps the PR segment, which costs nothing - it is isoelectric - and buys robustness, since the P offset is the least reliably placed of the two fiducials and a P wave truncated by a mis-delineated offset is a distorted one. End at the P offset instead when the window itself, rather than what it contains, is the measurement.

Windows containing a second P onset are discarded: that means a QRS onset went undetected and the window has run into the following beat.

This is `by_rhythm()` with a fixed set of criteria, so it uses the same engine and returns windows of differing length. Windows are named `pwave1`, `pwave2`, and so on.

Value

A window_strategy object.

Guiding channel

A delineating annotator may be run over a record once, or once per lead. Run per lead it writes twelve independent copies of every fiducial, separated only by the `channel` column, and such a table does not describe beats until it is resolved to a single lead: pooled, it reports twelve times as many beats as the record contains, and taking the first match at each point silently follows whichever lead happens to sort first.

So every function that reads fiducials - `get_windows()`, `learn_template()`, `median_window()` and the other window transforms, `extract_f_waves()`, `vcg()` - takes a `channel`, and resolves it the same way:

- annotations spanning more than one channel with no `channel` given are an error, not a guess;
- a `channel` that the annotations do not carry is an error, which is what catches a numbering convention that does not match;
- the global channel `0` is not a lead, so it never counts toward the span and is always kept alongside the chosen lead.

The channel may be a number or a channel name (`"II"`), resolved against the record's own header. A number is matched literally against the `channel` column, and the numbering there belongs to the annotator that wrote the file. Read it off the record rather than assuming:

```
table(get_annotation(x)$channel)
```

Only the boundaries are restricted to one lead. Windows still carry the signal for every channel, which is what makes a windowed beat usable as a vectorcardiogram.

See Also

`get_windows()`, `by_rhythm()`, `vcg()`

Examples

```
# P onset to QRS onset, guided by one lead
by_pwave(channel = 2)

# The P wave alone
by_pwave(to = "p_offset")
```

`by_rhythm` *Window by rhythm*

Description

[Experimental]

Builds a window_strategy that finds windows bounded by a pair of annotation fiducials - an onset and the next matching offset - optionally requiring a reference fiducial to fall between them. For sinus rhythm this yields one window per beat, from P-onset to T-offset with the QRS as the reference.

Usage

```
by_rhythm(
  rhythm = "sinus",
  onset = NULL,
  offset = NULL,
  reference = NULL,
  channel = NULL,
  reject_overlap = NULL,
  adjust_sample_indices = TRUE
)
```

Arguments

`rhythm`	A `character` naming the rhythm type, used to label the windows and to select defaults. `"sinus"` is the only value carrying built-in criteria; any other name requires `onset` and `offset`, and is the documented way to describe a different segmentation.
`onset`	A named list of criteria identifying window onsets. Defaults to `list(type = "(", wave = "P")` (P-wave onset) for sinus.
`offset`	A named list of criteria identifying window offsets. Defaults to `list(type = ")", wave = "T")` (T-wave offset) for sinus. Override either for alternative segmentations, e.g. `offset = list(type = "(", wave = "P")` for P-onset to next P-onset.
`reference`	A named list of criteria for a fiducial that must exist between onset and offset, or `NULL` to skip the check. Defaults to `list(type = "N")` (the QRS peak) for sinus.

`channel` The lead whose annotations define the window boundaries, given as a channel number or a channel name. An annotator run per lead (an `ecgpuwave`-style file, say) writes one copy of every fiducial for each lead, separated only by the `channel` column, and such a table has to be resolved to one lead before it describes beats: see the channels section below. Annotations on the global channel (`0`) are always retained, and the returned windows still contain the signal for every channel. `NULL` (default) uses every annotation, which is correct for a single-channel annotation file and an error for any other.

`reject_overlap` Logical, whether to discard a window that contains a second onset. `NULL` (default) means `TRUE` for sinus and `FALSE` otherwise. A second onset inside the window means a fiducial went undetected and the window has run on into the following beat.

`adjust_sample_indices`
Logical, whether to rebase signal and annotation sample indices in the returned windows to be zero-based and relative to the window start. Default `TRUE`, since each window is a new WFDB record.

Details

Criteria are named lists matched column-by-column against the annotation table, with the addition of a virtual `wave` field (`"P"`, `"QRS"`, or `"T"`) inferred positionally from the peak symbol enclosed by each `(`/`)` bracket pair (see `label_waves()`). This recovers wave identity even when the WFDB `number` column is unpopulated, as with `ecgpuwave` run per lead.

When `rhythm = "sinus"` the criteria default to a full beat and windows that contain a second onset are rejected, so that only clean non-overlapping beats are returned.

Value

A window_strategy object.

Segmentations other than a whole beat

`rhythm` is a free-form label: it names the windows and their info strings, and it selects the sinus defaults. Any other name is an extension point - give `onset` and `offset` explicitly and the same engine will find whatever pair of fiducials you describe. Windowing from one P onset to the next, for instance:

```
by_rhythm(
  rhythm = "atrial",
  onset = list(type = "(", wave = "P"),
  offset = list(type = "(", wave = "P"),
  reference = list(type = "p")
)
```

`by_pwave()` is that pattern named and given defaults, for the segmentation worth having a constructor of its own.

Guiding channel

A delineating annotator may be run over a record once, or once per lead. Run per lead it writes twelve independent copies of every fiducial, separated only by the `channel` column, and such a table does not describe beats until it is resolved to a single lead: pooled, it reports twelve times as many beats as the record contains, and taking the first match at each point silently follows whichever lead happens to sort first.

So every function that reads fiducials - `get_windows()`, `learn_template()`, `median_window()` and the other window transforms, `extract_f_waves()`, `vcg()` - takes a `channel`, and resolves it the same way:

- annotations spanning more than one channel with no `channel` given are an error, not a guess;
- a `channel` that the annotations do not carry is an error, which is what catches a numbering convention that does not match;
- the global channel `0` is not a lead, so it never counts toward the span and is always kept alongside the chosen lead.

The channel may be a number or a channel name (`"II"`), resolved against the record's own header. A number is matched literally against the `channel` column, and the numbering there belongs to the annotator that wrote the file. Read it off the record rather than assuming:

```
table(get_annotation(x)$channel)
```

Only the boundaries are restricted to one lead. Windows still carry the signal for every channel, which is what makes a windowed beat usable as a vectorcardiogram.

See Also

`get_windows()`, `by_pwave()`, `label_waves()`

Examples

```
# Sinus defaults: one clean P-onset -> T-offset beat per window
by_rhythm()

# Guided by a single lead in a multi-lead annotation file
by_rhythm(channel = 2)

# P-onset to next P-onset instead of a full beat
by_rhythm(offset = list(type = "(", wave = "P"))
```

c.windows *Concatenate* windows *objects*

Description

Concatenate windows objects

Usage

```
## S3 method for class 'windows'
c(...)
```

Arguments

...	windows objects to concatenate.

Value

A windows object containing all the elements of the inputs.

calculate_approximate_entropy *Calculate Approximate Entropy (ApEn) of a time series*

Description

Computes approximate entropy by the method of Pincus (1991), comparing vectors embedded in m and m + 1 dimensions and taking the difference of the resulting correlation integrals.

Usage

```
calculate_approximate_entropy(x, m = 2, r = NULL)
```

Arguments

x	Numeric vector of the time series
m	Embedding dimension. Default 2.
r	Tolerance. Default NULL, which uses 0.2 times the standard deviation of x.

Details

Approximate entropy counts self-matches, which biases it toward regularity and makes it dependent on record length. `calculate_sample_entropy()` avoids both and is generally preferable.

The tolerance `r` defaults to 0.2 times the standard deviation of `x`. Note that this default changed: earlier versions used 3.5 times the standard deviation, which admitted nearly every pair of vectors as a match and drove the statistic toward zero regardless of the input.

The calculation is O(n^2). Decimate before calling; see `calculate_sample_entropy()` for how far.

Value

Approximate entropy value

References

Pincus SM. Approximate entropy as a measure of system complexity. *Proceedings of the National Academy of Sciences of the USA*. 1991;88(6):2297-2301. doi:10.1073/pnas.88.6.2297

Examples

```
set.seed(123)
calculate_approximate_entropy(rnorm(500))
```

---

calculate_dominant_frequency

Calculate the dominant frequency of a time series

Description

Returns the frequency carrying the most power within a search band, estimated from a Welch-averaged periodogram.

Usage

```
calculate_dominant_frequency(x, frequency, f_min = 4, f_max = 10, ...)
```

Arguments

`x`	Numeric vector of the time series
`frequency`	Sampling frequency of the signal in Hz
`f_min`	Lower edge of the search band in Hz. Default 4.
`f_max`	Upper edge of the search band in Hz. Default 10. Sustained atrial fibrillation is usually quoted at 4-10 Hz; typical atrial flutter is 4-5.5 Hz and slow or drug-modified flutter falls below 4 Hz, so widen the band when flutter is expected.
`...`	Passed to `calculate_welch_spectrum()`

Value

Dominant frequency in Hz. NA when the band holds no finite power.

Examples

```
x <- sin(2 * pi * 6 * seq(0, 10, by = 1 / 500)) + rnorm(5001, sd = 0.1)
calculate_dominant_frequency(x, frequency = 500)
```

---

calculate_organization_index

Calculate the organisation index of an atrial signal

Description

The share of spectral power carried by the dominant frequency and its harmonics.

Usage

```
calculate_organization_index(
  x,
  frequency,
  dominant_frequency = NULL,
  n_harmonics = 4L,
  band = c(0.5, 50),
  half_width = 0.5
)
```

Arguments

x	Numeric vector of the atrial signal
frequency	Sampling frequency in Hz
dominant_frequency	Dominant frequency in Hz. Estimated from x if NULL.
n_harmonics	Number of harmonics summed with the dominant peak. Default 4, as in An et al.; Everett's four peaks are n_harmonics = 3.
band	Numeric length-2 vector for the total-power reference band. Default c(0.5, 50), following An et al., and clipped below the Nyquist frequency. Note that the band cannot put back what the signal does not carry: an atrial signal band-passed to 30 Hz contributes nothing between 30 and 50 Hz, so widening the band beyond the passband changes little.
half_width	Half-width in Hz of the window placed on the dominant frequency. The harmonic windows are 1.5 times as wide. Default 0.5.

Details

A highly organised atrium concentrates its energy in a narrow peak and its harmonics; a disorganised one spreads it across the spectrum. This is one of the few f-wave features with a reasonably direct electrophysiological reading, being related to the number of independent wavefronts the atrium is holding.

The definition follows the published one: the area under the dominant peak and its first `n_harmonics` harmonics, over the total area of the spectrum. Everett et al. introduced it on an interatrial electrogram as "the ratio of the area under the first four harmonic peaks to the total area of the spectrum" – four peaks in all, the dominant and three harmonics. An et al. give the clearest surface-ECG implementation, "the area under the highest peak and its first 4 harmonics" over the total area from 0 to 50 Hz on a QRST-cancelled lead V1 – five peaks. The defaults here are An's, and on a surface record the fifth peak is immaterial, as the next section says.

Value

Organisation index between 0 and 1

What the value can be compared with

Not a published cut-point, and the gap is not small. An et al. report a median of 0.33 (IQR 0.27-0.39) in lead V1 of 102 patients; other surface implementations report 0.26-0.32 and 0.70-0.75. This function returns about 0.13 on the bundled `muse-af` record with the same formula and the same band, so the remaining difference is in the cancellation, the peak integration and the cohort rather than in the definition. Published surface organisation indices are already implementation-dependent enough that they do not transfer between groups; treat these values as comparable within a cohort analysed this way and with nothing else.

Two further details are local choices with no published basis. The peak is integrated over a fixed window rather than out to the shoulders of the peak itself, and the harmonic windows are 1.5 times as wide as the fundamental's to allow for the harmonic drifting. On a surface record the harmonic count barely matters: at a dominant frequency of 7.5 Hz the third harmonic and above sit where the f-wave carries no energy, and moving from two peaks to five changes the value on `muse-af` by 0.007.

References

Everett TH 4th, Kok LC, Vaughn RH, Moorman JR, Haines DE. Frequency domain algorithm for quantifying atrial fibrillation organization to increase defibrillation efficacy. *IEEE Transactions on Biomedical Engineering*. 2001;48(9):969-978. doi:10.1109/10.942586

Everett TH 4th, Moorman JR, Kok LC, Akar JG, Haines DE. Assessment of global atrial fibrillation organization to optimize timing of atrial defibrillation. *Circulation*. 2001;103(23):2857-2861. doi:10.1161/01.CIR.103.23.2857

An K, Li H, Yu C, Zheng Z. Surface electrocardiogram f wave analysis in patients with atrial fibrillation undergoing thoracoscopic epicardial ablation. *Interdisciplinary CardioVascular and Thoracic Surgery*. 2024;38(5):ivae057. doi:10.1093/icvts/ivae057

Stavrakis S, Dyer JW, Koomson E, et al. Spectral analysis of baseline electrocardiogram during atrial fibrillation predicts response to antiarrhythmic drug therapy in patients with persistent atrial

fibrillation. *Journal of Cardiovascular Electrophysiology*. 2016;27(11):1312-1318. doi:10.1111/jce.13064

Examples

```
x <- sin(2 * pi * 6 * seq(0, 10, by = 1 / 500)) + rnorm(5001, sd = 0.5)
calculate_organization_index(x, frequency = 500)
```

`calculate_sample_entropy`

Calculate Sample Entropy of a time series

Description

Sample entropy measures the irregularity of a time series as the negative log conditional probability that two sequences similar for `m` points remain similar at the next point.

Usage

```
calculate_sample_entropy(x, m = 2, r = NULL)
```

Arguments

`x`	Numeric vector of the time series
`m`	Embedding dimension. Default 2.
`r`	Tolerance. Default `NULL`, which uses 0.2 times the standard deviation of `x`.

Details

Unlike approximate entropy, sample entropy excludes self-matches. Counting self-matches biases approximate entropy toward regularity and makes it depend on record length, which are the two problems sample entropy was introduced to fix. Prefer this over `calculate_approximate_entropy()`, particularly when record length varies.

The calculation is O(n^2), which is a standing temptation to decimate further than the measurement tolerates. Alcaraz et al. tuned `m`, `r` and the sampling rate for atrial fibrillation organisation specifically and found classification degraded below 256 Hz, with `m` of 1 or 2 and `r` between 0.1 and 0.25 times the standard deviation. The defaults here sit inside that, and `extract_f_waves()` decimates to 256 Hz rather than to something derived from the fibrillatory bandwidth.

They tuned them on the *main atrial wave* – lead V1's atrial signal narrowed by selective filtering to a 3 Hz band around its dominant frequency – and not on a broadband signal, and on that wave Alcaraz and Rieta (2009) tested `m` of 1 and 2 with `r` of 0.1, 0.15, 0.2 and 0.25, found the outcomes very similar, and selected `m = 2` and a tolerance of 0.25 standard deviations. `extract_f_waves()` computes on the broadband cancelled signal by default and offers the main atrial wave through `entropy_input`; the two are not on one scale.

Value

Sample entropy. NA when no matches are found at either length.

References

Richman JS, Moorman JR. Physiological time-series analysis using approximate entropy and sample entropy. *American Journal of Physiology. Heart and Circulatory Physiology*. 2000;278(6):H2039-H2049. doi:10.1152/ajpheart.2000.278.6.H2039

Alcaraz R, Abasolo D, Hornero R, Rieta JJ. Optimized assessment of atrial fibrillation organization through suitable parameters of sample entropy. *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. 2010;2010:118-121. doi:10.1109/IEMBS.2010.5627169

Alcaraz R, Rieta JJ. Sample entropy of the main atrial wave predicts spontaneous termination of paroxysmal atrial fibrillation. *Medical Engineering & Physics*. 2009;31(8):917-922. doi:10.1016/j.medengphy.2009.05.002

Examples

```
set.seed(123)
calculate_sample_entropy(rnorm(500))
```

---

calculate_welch_spectrum

Welch power spectral density estimate

Description

Averages the periodograms of overlapping, Hann-tapered segments.

Usage

```
calculate_welch_spectrum(
  x,
  frequency,
  segment_sec = 4,
  overlap = 0.5,
  resolution = 0.05
)
```

Arguments

x	Numeric vector
frequency	Sampling frequency in Hz
segment_sec	Segment length in seconds. Default 4.
overlap	Fractional overlap between segments. Default 0.5.
resolution	Target frequency resolution in Hz, achieved by zero padding. Default 0.05.

Details

A raw periodogram is an inconsistent estimator: its variance does not fall as the record lengthens, so on a ten second record the spectrum is very noisy and an argmax taken over a several-Hz band is correspondingly unstable. Averaging tapered segments trades frequency resolution for variance, and zero-padding to a fixed resolution keeps the bin edges from moving with record length.

Value

A list with `freq` and `spec`

References

Welch PD. The use of fast Fourier transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms. *IEEE Transactions on Audio and Electroacoustics*. 1967;15(2):70-73. doi:10.1109/TAU.1967.1161901

Examples

```
x <- sin(2 * pi * 6 * seq(0, 10, by = 1 / 500)) + rnorm(5001, sd = 0.1)
psd <- calculate_welch_spectrum(x, frequency = 500)
psd$freq[which.max(psd$spec)]
```

---

cancel_ventricular_signal

Cancel the ventricular contribution from a multi-lead ECG

Description

Removes the QRST complex from each lead so that the atrial signal can be measured. Two methods are available.

Usage

```
cancel_ventricular_signal(
  signals,
  frequency,
  qrs_loc = NULL,
  method = c("spatiotemporal", "average_beat"),
  min_beats = 3L,
  aberrancy_threshold = 0.9
)
```

Arguments

signals	A named list of numeric vectors, one per lead, already bandpass filtered and all of the same length.
frequency	Sampling frequency in Hz. Required.
qrs_loc	Integer vector of QRS sample positions shared by all leads. Detected from a composite if NULL.
method	Either "spatiotemporal" or "average_beat".
min_beats	Minimum number of beats required before any template-based subtraction is attempted. Default 3. Below this an "average" beat is effectively a copy of a single beat, and subtracting it returns zero.
aberrancy_threshold	Correlation below which a beat is called morphologically aberrant and excluded from template construction. Default 0.9. Aberrancy is judged on QRS morphology, never on RR interval: in atrial fibrillation the RR interval is irregular by definition, so an RR-based criterion fires on normally conducted beats in the exact rhythm this function targets.

Details

"spatiotemporal" fits each beat, in each lead, to a linear combination of the median beat templates of that lead and its *adjacent* leads plus an intercept, and subtracts the fit (Stridh and Sornmo, 2001). Adjacency runs along the frontal leads in Cabrera order and along V1 to V6; a lead that is not a surface lead takes its neighbours in the order given. The cross-lead terms give the model enough freedom to absorb beat-to-beat rotation of the electrical axis, which a single-lead template cannot represent. They are deliberately few: fitted against every lead's template, as earlier versions did, the model absorbs the atrial signal in the window as well, and on muse-sinus with a known f-wave added it kept 43-62% of that wave inside the beat windows against 82-88% for the adjacent-lead fit. The cancellation section of extract_f_waves() has the table.

"average_beat" subtracts each lead's own median template, scaled by a single least-squares coefficient. This is the Slocum average-beat predecessor and is provided for comparison.

Beats are never blanked or interpolated across. A beat that the template models badly is still fitted and subtracted, and the quality of that fit is reported through cancellation_residual rather than being hidden by replacing the segment with a straight line.

Both methods share one assumption, and it is worth stating plainly: what the template holds is whatever repeats at a fixed phase to the QRS. In atrial fibrillation the atrial signal has no such phase, which is why the method works. In atrial flutter conducting at a fixed ratio it does, so the flutter wave is built into the template and subtracted along with the QRST – around 90% of it, on a synthetic 2:1 record. The fit is *better* for having absorbed it, so cancellation_residual does not report the loss. See the cancellation section of extract_f_waves(), which flags the condition through record$rr_regular.

Value

A list with atrial (named list of cancelled signals), qrs_loc, n_beats_cancelled, n_beats_skipped, and n_beats_aberrant.

References

Stridh M, Sornmo L. Spatiotemporal QRST cancellation techniques for analysis of atrial fibrillation. *IEEE Transactions on Biomedical Engineering*. 2001;48(1):105-111. doi:10.1109/10.900266

`change_frequency` *Change the sampling frequency of signal data*

Description

[Experimental]

Converts signal data from one sampling frequency to another, up- or down-sampling as needed while preserving the recording's duration. This harmonises records captured at different rates (e.g. mixing 250 Hz and 500 Hz ECGs) so that downstream operations such as `median_window()` or `normalize_window()` compare like with like, and lets analyses that assume a fixed rate be fed at that rate.

Usage

```
change_frequency(
  x,
  to,
  from = NULL,
  method = c("linear", "spline", "step", "polyphase"),
  anti_alias = TRUE,
  preserve_class = TRUE,
  ...
)
```

Arguments

`x`	The signal data to convert. Either an `EGM` object, a `windows` object, a plain list of `EGM` objects, or a bare `numeric` vector holding a single lead.
`to`	The target sampling rate in Hz.
`from`	The rate the data is currently sampled at, in Hz. Defaults to the rate the object's own header declares, and must agree with it when given. Required for a bare `numeric` lead, which carries no header.
`method`	The resampling method, one of `"linear"` (default), `"spline"`, `"step"`, or `"polyphase"`. See details.
`anti_alias`	Logical. If `TRUE` (default), applies a low-pass filter before down-sampling with an interpolating method. Ignored when up-sampling, and by the `polyphase` method.
`preserve_class`	Logical, for `windows` input only. If `TRUE` (default), returns a `windows` object; if `FALSE`, a plain list of `EGM` objects.
`...`	Additional arguments (currently unused).

Details

Only the target rate (`to`) has to be stated. The source rate is taken from the object's own header, which already carries it; state `from` as well when you want it checked, and a disagreement with the recorded rate is an error. A bare `numeric` lead has no header, so there `from` is required - the sampling rate is the one property that cannot be recovered from the samples themselves. Read the recorded rate with `frequency()`.

The change is duration-preserving: a 0.8 second beat stays 0.8 seconds long, but its sample count scales with the frequency ratio. For `EGM` input the signal values are recomputed on the new grid, the header frequency and sample count are updated, and every annotator's sample indices are rescaled onto the new grid. Annotation `time` strings are left alone, because absolute time is unchanged by definition.

The annotations carried by the object are rescaled; a separate copy of them is not. Annotations read separately from disk, or held from before the conversion, remain on the original grid, and mixing the two halves every interval measured from them (or doubles it). Take the annotations from the converted object with `get_annotation()` rather than re-reading them:

```
slow <- change_frequency(ecg, to = 250)
get_annotation(slow)      # rescaled, on the 250 Hz grid
read_annotation(...)      # still on the 500 Hz grid
```

It is written as a pipe stage rather than an argument of the functions that need it, so the rate change is explicit and happens exactly once:

```
read_wfdb("ecg", ".", "ecgpuwave") |>
  change_frequency(to = 250) |>
  get_windows(by = "rhythm")
```

Four `method` choices are offered, covering the standard approaches:

- `linear` (default) - piecewise-linear interpolation onto the new grid. Fast, monotone, and free of overshoot, which matters when the sharp edges of a QRS complex would otherwise ring.
- `spline` - natural cubic spline interpolation. Smoother through curved segments such as T waves, at the cost of possible overshoot at steep transitions.
- `step` - nearest-preceding sample (piecewise constant). Non-interpolating, so it never invents intermediate voltages; mainly useful for step-like or categorical channels such as pacing markers.
- `polyphase` - anti-aliased polyphase FIR resampling via `signal::resample()`, the conventional DSP approach. The rate ratio is approximated by a rational `p/q`, so it is most natural for related rates (500 to 250 Hz, 250 to 1000 Hz) and applies its own anti-alias filter.

Down-sampling folds any energy above the new Nyquist frequency back into the retained band. The interpolating methods do not guard against this on their own, so `anti_alias = TRUE` (the default) low-pass filters each lead with a zero-phase Butterworth filter before the grid change. Records too short to filter stably are left unfiltered, and up-sampling needs no filter at all. The `polyphase` method ignores this argument because its own FIR stage already performs the equivalent filtering.

Value

An object of the same kind as x, sampled at to Hz.

See Also

frequency(), normalize_window() to change the number of samples without changing the physical rate.

Examples

```
## Not run:
ecg <- read_wfdb("ecg", test_path(), "ecgpuwave")

# Down-sample the whole record to 250 Hz; the source rate is on the header
slow <- change_frequency(ecg, 250)
frequency(slow)

# State `from` as well to assert what you believe you were given
change_frequency(ecg, to = 250, from = 500)

# Or harmonise a set of windowed beats after the fact
beats <- get_windows(ecg, by = "rhythm")
change_frequency(beats, to = 1000, method = "polyphase")

# A bare lead works too, for signal-processing pipelines. It carries no
# header, so `from` is required
change_frequency(ecg$signal$II, to = 1000, from = 500)

## End(Not run)
```

---

channel_zero *How the file behind an annotation table counts its signals*

Description

[Experimental]

Reports the convention of the file an annotation table was read from, and will be written back to: "global", where channels count signals from 1 and 0 marks a fiducial belonging to no lead in particular, or "signal", where they count from 0 as the WFDB tools do.

Usage

```
channel_zero(x)
```

Arguments

x	An annotation_table, or an EGM carrying one.

Details

In memory every table counts from `1`, with `0` the global channel. A file declared `"signal"` at `read_annotation()` or `read_wfdb()` has one added to each channel as it is read, so the label here says nothing about the numbers in the table; it says how `write_annotation()` should number them on the way out, which it does by default. See channels for why the WFDB specification leaves the two readings open.

Value

A single `character`, `"global"` or `"signal"`.

See Also

channels for the policy, `read_annotation()` to declare it, `write_annotation()` to override it on the way out.

Examples

```
## Not run:
channel_zero(get_annotation(read_wfdb("ecg", test_path(), "ecgpuwave")))
#> [1] "global"

## End(Not run)
```

---

channels *Per-lead annotations and the guiding channel*

Description

[Experimental]

Why several functions in this package ask which lead to follow, and how they decide when they have to.

Guiding channel

A delineating annotator may be run over a record once, or once per lead. Run per lead it writes twelve independent copies of every fiducial, separated only by the `channel` column, and such a table does not describe beats until it is resolved to a single lead: pooled, it reports twelve times as many beats as the record contains, and taking the first match at each point silently follows whichever lead happens to sort first.

So every function that reads fiducials - `get_windows()`, `learn_template()`, `median_window()` and the other window transforms, `extract_f_waves()`, `vcg()` - takes a `channel`, and resolves it the same way:

- annotations spanning more than one channel with no `channel` given are an error, not a guess;

- a `channel` that the annotations do not carry is an error, which is what catches a numbering convention that does not match;
- the global channel `0` is not a lead, so it never counts toward the span and is always kept alongside the chosen lead.

The channel may be a number or a channel name (`"II"`), resolved against the record's own header. A number is matched literally against the `channel` column, and the numbering there belongs to the annotator that wrote the file. Read it off the record rather than assuming:

```
table(get_annotation(x)$channel)
```

Only the boundaries are restricted to one lead. Windows still carry the signal for every channel, which is what makes a windowed beat usable as a vectorcardiogram.

What channel zero means

Zero is the one value the WFDB specification leaves undecided. `annot(5)` says the `chan` field starts at `0` and persists until a `CHN` record changes it; nothing reserves a value for "no particular signal", and nothing requires the field to name a real one. So on disk `0` may mean the first signal, or it may mean the annotator never said.

In memory the package settles it: `channel` **counts signals from** `1`**, as** `header$number` **and the signal columns do, and** `0` **is the global channel**, a fiducial belonging to no lead. Every function that resolves a channel relies on that one reading, so a file that counts from `0` is renumbered as it is read rather than carried as an exception:

```
ecg <- read_wfdb("record", dir, "ann", channel_zero = "signal")
```

That adds one to every channel, so lead I is `1` whichever way the file counted, and `write_annotation()` takes one off again by default, so the file goes back out the way it came in - the table remembers which, and `channel_zero()` reports it.

Nothing has to be declared for the files most annotators write. An all-zero column - `ecgpuwave`, `wqrs`, and the twelve per-lead files of the bundled LUDB record, which carry the lead in the *file extension* - is an absence of information, read as global, and such a record windows without a `channel` at all. A column running `1 .. nsig` already counts from one. The one column the reader cannot settle is `0 .. nsig-1` exactly, which fills the signals if `0` is the first of them and is a global channel plus every lead but the last otherwise; that file is refused with a message naming both declarations, since either reading taken silently would misplace a lead. A column holding `0` beside a few low channels is read as global, and a `channel` it turns out not to carry is caught when it is asked for.

See Also

`get_windows()`, `label_waves()`, `get_annotation()`, `channel_zero()`

`detect_QRS` *Detect QRS complexes in ECG signals*

Description

`detect_QRS()` implements a modified Pan-Tompkins algorithm to detect QRS complexes in ECG signals. The function applies a sequence of processing steps including bandpass filtering, differentiation, squaring, and moving window integration to identify R peaks in the signal.

Usage

```
detect_QRS(signal, frequency, window_size = 0.15)
```

Arguments

`signal`	Numeric vector representing the ECG signal
`frequency`	Sampling frequency of the signal in Hz
`window_size`	Width of the integration window in seconds, default is 0.150 seconds

Details

The Pan-Tompkins algorithm is a widely-used method for QRS detection in ECG signals. This implementation follows these steps:

1. Bandpass filtering (5-15 Hz) to reduce noise and emphasize QRS complexes
2. Differentiation to highlight the steep slopes of QRS complexes
3. Squaring to amplify high-frequency components
4. Moving window integration to consider the overall QRS morphology
5. Thresholding at the mean plus half a standard deviation of the integrated signal to identify peaks
6. Application of a refractory period to prevent multiple detections of the same QRS complex

The threshold is a single static one over the whole record, not the pair of adaptive signal and noise thresholds with searchback that Pan and Tompkins describe. That is enough on a clean ten-second strip and will drop beats on a record whose amplitude drifts.

Positions are reported at the peak of the integration window, which lags the true R peak by roughly half the integration width. `extract_f_waves()` refines them against the signal before use, and anything else that needs beat alignment should do the same.

Value

Integer vector containing the sample indices of detected QRS complexes

References

Pan, J., & Tompkins, W. J. (1985). A real-time QRS detection algorithm. IEEE Transactions on Biomedical Engineering, (3), 230-236. doi:10.1109/TBME.1985.325532

Examples

```
## Not run:
ecg_data <- read_wfdb("muse-af", system.file("extdata", package = "EGM"))
freq <- attributes(ecg_data$header)$record_line$frequency
qrs_locations <- detect_QRS(ecg_data$signal$II, freq)

## End(Not run)
```

ECG *Electrocardiogram data class for 12-lead ECG studies*

Description

A specialization of the EGM class for the standard 12-lead surface electrocardiogram. It carries the same three components - signal, header, and annotation - and adds the guarantee that every channel is a recognised surface ECG lead, named canonically.

Usage

```
ECG(
  signal = signal_table(),
  header = header_table(),
  annotation = annotation_table(),
  ...
)

is_ECG(x)
```

Arguments

signal	A signal_table object generated by the signal_table() function containing standard ECG leads
header	A header_table object generated by the header_table() function
annotation	A annotation_table object generated by the annotation_table() function
...	Additional arguments to be passed to the function
x	An ECG object to be used with support functions

Details

An electrophysiology study records surface and intracardiac channels side by side, so the 12-lead ECG is often a subset of a larger record. as_ECG() extracts that subset: it keeps the surface leads, renames them to their canonical form (I, II, III, AVF, AVL, AVR, V1 ... V6), and discards every other channel. This is the intended route from a study to an ECG analysis.

Functions that are only defined on the surface ECG - extract_f_waves(), vcg() - coerce their input through that same path and then check that the leads each one needs are actually present. A record carrying only intracardiac channels is an error rather than a silent result.

Construction is otherwise identical to EGM(). A record with a lead set that is not the standard twelve is accepted with a warning, since partial surface sets are common and several analyses tolerate them.

Leads are held in the order the source record wrote them, which is not always the display order and is not always the same order twice: a MUSE export and a PhysioNet record disagree about where the augmented limb leads sit. Index by name rather than by position, and use ecg_leads() and lead_factor() where an order is needed - for a plot, a sort, or the columns of a matrix stacked across records.

Value

An object of class ECG, which inherits from EGM, containing signal, header, and annotation components.

See Also

as_ECG() to extract the surface leads from a study, EGM() for the general class.

---

ecg_leads *The twelve surface leads, in the order they are displayed*

Description

[Experimental]

ecg_leads() returns the standard twelve leads as an ordered factor. lead_factor() puts a set of lead labels onto that order, whatever they were called and whatever sequence they arrived in.

Usage

```
ecg_leads(order = c("standard", "cabrera"))

lead_factor(x, order = c("standard", "cabrera"), drop = FALSE)
```

Arguments

order	The display sequence. "standard" (default) or "cabrera".
x	A character vector of lead labels, or a factor of them.
drop	Logical. If TRUE, levels no label uses are dropped, which is what a plot of a subset of leads usually wants. Default FALSE, so that records with different lead sets stay comparable.

Details

The display sequence is the AHA/ACCF/HRS one (Kligfield et al. 2007): the three bipolar limb leads, the three augmented limb leads, and then the precordials left to right.

I, II, III, aVR, aVL, aVF, V1, V2, V3, V4, V5, V6

It is what every ECG cart prints and what PhysioNet's twelve-lead databases are written in, so it is the order to hold a cohort in. `order = "cabrera"` gives the Cabrera sequence (aVL, I, -aVR, II, aVF, III, then the precordials), the recognised alternative, in which the frontal leads run contiguously from -30 to +120 degrees. Note that Cabrera displays *-aVR*; the lead named here is still `AVR`, and inverting its signal is the caller's to do.

A record's leads are stored in whatever order its source wrote them, and `as_ECG()` does not reorder them - annotation `channel` indices address signal columns by position, so permuting the columns would silently repoint every per-lead fiducial. Order at the point of use instead:

```
features$lead <- lead_factor(features$lead)   # then sort, facet, or split
signal <- get_signal(ecg)[, c("sample", as.character(ecg_leads()))]
```

`lead_factor()` canonicalises before ordering, so `aVR`, `av r` and `AV-R` all become `AVR`. A label that is not a surface lead becomes `NA`, which is deliberate: it shows up in a plot or a sort rather than being dropped quietly.

Value

`ecg_leads()`, an ordered `factor` of twelve lead names. `lead_factor()`, an ordered `factor` of the same levels, one element per element of `x`.

References

Kligfield P, Gettes LS, Bailey JJ, et al. Recommendations for the standardization and interpretation of the electrocardiogram: part I. *Circulation*. 2007;115(10):1306-1324. doi:10.1161/CIRCULATIONAHA.106.180200

See Also

`as_ECG()` for extracting the surface leads from a record, ECG for the class

Examples

```
ecg_leads()

ecg_leads("cabrera")

# Whatever they were called, and whatever order they came in
lead_factor(c("v2", "aVR", "II", "AV-L"))

sort(lead_factor(c("V6", "I", "aVF", "V1")))

# Not a surface lead, and said so rather than dropped
lead_factor(c("II", "CS 1-2"))
```

EGM *Electrogram data class from electrophysiology studies*

Description

This class serves as a combinatorial class to describe cardiovascular electrical signal data in R. It is based off of the formats available in WFDB, but has been formatted for ease of use within the R ecosystem. An EGM object contains three components in a list:

- signal data in multiple channels
- header information
- annotation labels at specified time points

These components help to navigate, and visualize data. The EGM class is the backbone for working with WFDB objects in R, and provides an interface for integrating or converting other raw signal data to a WFDB format.

Usage

```
EGM(
  signal = signal_table(),
  header = header_table(),
  annotation = annotation_table(),
  ...
)

is_EGM(x)
```

Arguments

signal	A signal_table object generated by the signal_table() function
header	A header_table object generated by the header_table() function
annotation	Either a single annotation_table object generated by the annotation_table() function (deprecated), or a named list of annotation_table objects for multiple annotators (recommended)
...	Additional arguments to be passed to the function
x	An EGM object, typically generated by the EGM() function, to be used with support functions (e.g. is_EGM()

Details

The individual components of the class are further defined in their respective children functions signal_table(), header_table(), annotation_table(). They are very simple classes that build upon the data.table class that allow for class safety checks when working with different data types (particularly WFDB).

IMPORTANT: The EGM class can be built from ground-up by the user, however it is primarily generated for the user using the other read/write functions, such as read_bard() or read_wfdb().

Value

An object of class EGM that is always a list of the above three components. Oftentimes, the annotation_table object may be missing, and it is replaced with an empty table as a place holder.

---

extract_f_waves *Extract fibrillatory wave features from a surface ECG*

Description

extract_f_waves() isolates atrial activity from a surface ECG by cancelling the ventricular (QRST) contribution, then summarises the residual fibrillatory signal in each lead.

Usage

```
extract_f_waves(
  object,
  lead = NULL,
  qrs_loc = NULL,
  channel = NULL,
  cancel_method = c("spatiotemporal", "average_beat"),
  f_characteristics = c("amplitude", "dominant_frequency", "organization",
    "sample_entropy"),
  amplitude_window = c("tq", "all"),
  normalize = c("none", "qrs"),
  band = c(4, 10),
  entropy_rate = 256,
  entropy_input = c("broadband", "main_atrial_wave"),
  entropy_bandwidth = 3,
  entropy_tolerance = 0.2,
  spectrum = c("V1", "pooled", "lead"),
  rhythm = NULL,
  tol = 0.15,
  min_beats = 3L,
  aberrancy_threshold = 0.9,
  keep_signal = FALSE,
  verbose = TRUE
)
```

Arguments

object	An object of class EGM or of subclass ECG. An EGM from an electrophysiology study is reduced to its surface leads first (see as_ECG()); a record with no surface leads is an error, since a fibrillatory wave cannot be read from an intracardiac channel.

`lead` Optional. A character vector of leads to analyse. If `NULL` (default), all available surface leads are processed. Cancellation always uses every available surface lead regardless of this argument, since the spatiotemporal fit draws on all of them.

`qrs_loc` Optional integer vector of QRS sample positions, at the record's own sampling frequency. If `NULL` (default), positions are taken from the object's annotation table when one is attached, and otherwise detected from a multi-lead composite.

`channel` The lead whose beat annotations give the QRS positions, given as a channel number or name. Required when the annotations span more than one channel: pooled across twelve leads they report twelve times as many beats as the record holds, and every rhythm measure derived from them follows. See the channels section.

Supplying `qrs_loc` does not make this unnecessary. The beat positions come from `qrs_loc` then, but the TQ segments are still read from annotations, and pooled across leads they overlap and cover the record several times over. A record whose annotations span more than one channel is an error either way.

`cancel_method` Ventricular cancellation method. Either `"spatiotemporal"` (default) or `"average_beat"`, the single-lead Slocum predecessor kept as the comparison baseline.

`f_characteristics`

Character vector of features to compute. Any of `"amplitude"`, `"dominant_frequency"`, `"organization"`, `"sample_entropy"`, `"approximate_entropy"`.

`amplitude_window`

Where amplitude is measured. `"tq"` (default) restricts measurement to the TQ segments, where the ventricles are electrically silent. `"all"` uses the whole record, which confounds fibrillatory amplitude with cancellation error, since whatever cancellation fails to remove is concentrated at the QRS.

`normalize` Amplitude normalisation. `"none"` (default) leaves `f_amplitude` as `f_amplitude_rms`, in raw signal units. `"qrs"` points it at `f_ratio` instead, which divides the root-mean-square amplitude by the QRS excursion in the same lead. The intent is to cancel the thoracic transfer function to first order – the ventricular signal crosses the same tissue, so it carries the same attenuation – and so make amplitudes comparable *between* patients of different body habitus. Every amplitude column is always returned, and this argument only sets which one `f_amplitude` refers to.

`f_ratio` is the package's form of the *normalised f-wave amplitude* of the Alcaraz-Rieta group, who express the root-mean-square f-wave amplitude as a percentage of the R-peak magnitude to remove recording gain, skin conductivity and electrode impedance (Escribano et al. 2024). It differs in the denominator – the peak-to-peak QRS excursion within 60 ms of the R peak on the unfiltered lead, which does not depend on the polarity of the QRS – and in being a fraction rather than a percentage, so the two are proportional and not equal. Earlier versions divided the peak-to-peak amplitude instead, which put the length confound below into the one measure meant for comparison between patients.

RMS is the default because peak-to-peak is a maximum over the segment and so grows with the segment's length: on white noise its expectation rises 58% between a 20-sample and a 400-sample window. TQ segments are as long as

the RR interval lets them be, which is irregular within an AF record by definition and shorter at higher rates between patients, so `f_amplitude_p2p` carries a heart-rate confound that `f_amplitude_rms` and `f_ratio` do not. Peak-to-peak is nonetheless the measure the coarse- versus-fine literature uses: Li et al. tied a coarse f-wave – 1 mm or more in V1 – to left atrial appendage dysfunction and thrombus.

`band` Numeric length-2 vector giving the frequency band searched for the dominant fibrillatory frequency, in Hz. Default `c(4, 10)`. Sustained atrial fibrillation is usually quoted at 4-10 Hz. Typical atrial flutter is 4-5.5 Hz, and slow or drug-modified flutter falls below 4 Hz, so widen this when flutter is expected. Alcaraz and Rieta take the largest peak in 3-9 Hz as the main atrial frequency.

`entropy_rate` Sampling rate in Hz to which the atrial signal is decimated before entropy is computed, or `NULL` to compute at the record's own rate. Default 256.

Entropy is O(n^2), so this is the argument that decides how long a batch takes. It is not free to lower it: Alcaraz et al. (2010) tuned sample entropy for this exact task and found classification degraded below 256 Hz, which is why that is the default rather than a rate chosen from the fibrillatory bandwidth. A lower value returns a number that is cheaper and outside the range the parameter set was validated on.

`entropy_input` What the entropy is computed on. `"broadband"` (default) uses the cancelled signal in its 0.5-30 Hz passband. `"main_atrial_wave"` first band-pass filters it to `entropy_bandwidth` Hz centred on the dominant frequency, which is the *main atrial wave* that Alcaraz and Rieta compute sample entropy on, and on which the `m`, `r` and 256 Hz defaults were tuned (Alcaraz and Rieta 2009; Alcaraz et al. 2010); see `entropy_bandwidth` and `entropy_tolerance` for their exact settings. The two are not on one scale: on `muse-af` lead V1 the broadband value is 0.16 and the main-atrial-wave value 0.12. The broadband default is kept because the narrow band inherits whatever is wrong with the dominant frequency – a peak sitting on a heart-rate harmonic filters out the atrial signal and keeps the residual – so read the main-atrial-wave entropy only with `on_harmonic` beside it.

`entropy_bandwidth`

Width in Hz of the band kept around the dominant frequency for `entropy_input = "main_atrial_wave"`. Default 3, the value Alcaraz and Rieta (2009, section 3.3) selected: the band has to be narrower than 6 Hz so that the first harmonic is excluded, and of the bandwidths and filter orders they tried, 3 Hz with a 768-coefficient linear-phase FIR at 1 kHz classified terminating from non-terminating episodes best. The filter here is a third-order Butterworth band-pass run forwards and backwards, zero-phase as theirs was linear-phase, and at least as selective: for a 6 Hz dominant frequency it is about 27 dB down 1.5 Hz outside the band and about 57 dB down at the first harmonic.

`entropy_tolerance`

The sample entropy tolerance `r`, as a multiple of the standard deviation of the signal the entropy is taken of. Default 0.2. Alcaraz and Rieta (2009) tested 0.1, 0.15, 0.2 and 0.25 with `m` of 1 and 2 on the main atrial wave, found the outcomes very similar, and selected `m = 2` with 0.25; pass `entropy_tolerance = 0.25` with `entropy_input = "main_atrial_wave"` to match them exactly.

`spectrum` Where the dominant frequency is read from. `"V1"` (default) takes it from the cancelled lead V1 and shares it with every lead, which is how the atrial fibrillatory rate is reported: V1 lies nearest the right atrium and carries the largest f-waves relative to the ventricular signal, so it is the lead the surface-ECG literature analyses (Bollmann et al. 2006; An et al. 2024; Escribano et al. 2024). A record without V1 is an error when something needs the frequency – pass `"pooled"` or `"lead"`.

`"pooled"` takes it from a spectrum averaged across the analysed leads, each normalised to unit power in 2.5-15 Hz first so that one high-amplitude lead does not dictate the peak. This is not a published procedure, and it weights a lead carrying no atrial signal the same as V1. It also buys less than the lead count suggests: four of the twelve leads are exact linear combinations of I and II, so at most eight can be independent, and the atrial signal is close to a single dipole, so fewer are. On the bundled `muse-af` record the correlation matrix of the twelve cancelled leads has a participation ratio of 1.7, making the variance reduction about 1.3-fold rather than the 3.5-fold twelve independent leads would give.

`"lead"` estimates it in each lead separately, so `dominant_rate` differs from row to row; on `muse-af` it spans 313 to 465 per minute across the twelve. Whichever is chosen, `organization_index` and the main atrial wave are computed at the same frequency as `dominant_rate` reports.

`rhythm` Optional character string naming the rhythm, e.g. `"af"`, `"flutter"`, `"sinus"`. Used only to decide whether to warn. If `NULL` (default), rhythm is inferred from RR irregularity.

`tol` Numeric tolerance for the harmonic test. A peak is flagged `on_harmonic` when its harmonic index falls within `tol` of an integer. Default 0.15.

This is a sensitive screen and not a specific test, and the difference is large enough to plan around. The harmonic index of an uncontaminated peak is a ratio of two unrelated numbers, so with integers spaced one apart a tolerance of 0.15 lands on one 30% of the time by arithmetic alone – simulated at 0.301 for a dominant frequency uniform on 4-10 Hz and a heart rate uniform on 60-160 bpm, and flat across rate. Excluding every flagged record therefore discards roughly a third of the good ones too. Read it with `cancellation_residual`, which says whether there was residual for the peak to have come from.

`min_beats, aberrancy_threshold` Passed to `cancel_ventricular_signal()`. See there for what each is guarding.

`keep_signal` Logical. If `TRUE`, the cancelled atrial signals are returned alongside the features. Default `FALSE`, since retaining them across a large batch is expensive.

`verbose` Logical. If `TRUE` (default), report which leads are analysed. This controls the progress message only. The rhythm warnings are raised either way: batch processing is where `verbose = FALSE` gets set, and it is also where an unnoticed flutter record does the most damage.

Value

An object of class `f_wave_analysis`, a list with:

`features` A `data.table` with one row per lead, holding `f_amplitude_p2p`, `f_amplitude_rms`, `qrs_amplitude`, `f_ratio`, `dominant_rate`, `organization_index`, `sample_entropy`, and the per-lead diagnostics `harmonic_overlap`, `on_harmonic`, `cancellation_residual`, and `tq_fraction`.

`record` A one-row `data.table` of record-level values, including `n_beats_cancelled`, `n_beats_skipped`, `n_beats_aberrant`, `heart_rate`, `rr_cv`, `af_like`, and `rr_regular` - the flag that the cancellation may have taken the atrial signal with it, described in the cancellation section.

`signal` The cancelled atrial signals, when `keep_signal = TRUE`.

Cancellation

The default `cancel_method = "spatiotemporal"` implements the *spatial* half of Stridh and Sornmo (2001). A single set of QRS positions is shared by every lead, and a template beat is formed for each lead by taking the *median* across beats. Each individual beat, in each lead, is then fitted by least squares to a linear combination of the templates of that lead and its *adjacent* leads plus an intercept, and the fit is subtracted. Adjacency runs along the two chains the leads sit on – the frontal leads in Cabrera order (aVL, I, -aVR, II, aVF, III) and the precordials V1 to V6 – so a lead is fitted against at most three templates. A lead that is not a surface lead takes its neighbours in the order it was given.

Adjacent, and not all twelve, is the published model, and the difference is not academic. A single-lead template cannot absorb the beat-to-beat rotation of the heart's electrical axis caused by respiration and by varying ventricular filling, which is the dominant reason a template fails to fit its own beat; what it leaves behind is periodic at the heart rate and lands on heart-rate harmonics, inside the band this function reads. Two neighbouring templates are enough freedom to follow that rotation. Twelve are enough to follow the atrial signal lying in the beat window as well, and to subtract it. Measured on the bundled `muse-sinus` record with a fibrillatory wave of known amplitude added at 5.3, 6.4 and 7.7 Hz (heart rate 90, beat windows covering 95% of the record), the share of that wave surviving inside the beat windows was:

Templates fitted	Surviving	Residual at the QRS
all twelve	43-62%	0.066-0.075
own and adjacent	82-88%	0.074-0.084
own only (`"average_beat"`)	91%	0.079-0.086

At fibrillation rates the beat windows cover the whole record, so the all-lead fit – which earlier versions used – was discarding about half the atrial signal for a tenth less residual. `f_amplitude` was never affected, being measured in the TQ segments; `dominant_rate`, `organization_index` and `sample_entropy` were.

Two parts of the published method are not implemented. It estimates a per-beat time shift and time scaling of the template as well, which is what the *temporal* in its name refers to; the absence shows up as a larger `cancellation_residual` on records whose QRS width varies beat to beat. And it estimates the f-wave around each beat and subtracts it before the templates are averaged, so that less of it is built into them; here the median across beats is relied on for that. Use f_wave_diagnostics to check what the cancellation left behind.

What cancellation cannot separate:

A template built by stacking beats keeps whatever repeats at a fixed phase to the QRS. That is the ventricular complex, and it is also atrial activity in a rhythm whose AV conduction ratio is fixed – typical atrial flutter. The method has no way to tell the two apart, so the flutter wave joins the template and is subtracted with the QRST.

The size of the effect is not marginal. On a synthetic 12-lead record carrying a 5 Hz (300/min) flutter wave, the fraction of that wave surviving cancellation (median over leads) and the organisation index lead V1 then reports are:

Ventricular response	Atrial cycles per RR	Surviving	Organisation index
2:1, fixed	2.00	8%	0.18
3:1, fixed	3.00	23%	0.23
4:1, fixed	4.00	24%	0.21
fixed, non-integer ratio	2.50	88%	0.96
irregular (AF-like)	–	76%	0.80

The uncancelled signal scores 0.94. So a flutter record conducting at a fixed ratio comes back with an organisation index in the range fibrillation occupies, and a dominant rate describing whatever survived rather than the flutter. **A cohort compared on** `organization_index` **will not separate flutter from fibrillation, and the failure looks like a null result rather than an artefact.**

Nothing in the fit reports this. The template models the beat *better* for having absorbed the atrial wave, so `cancellation_residual` is small and reassuring. What does report it is the ventricular response being regular: `record$rr_regular` is `TRUE` when RR CV is below 0.05, and a warning is raised. It is deliberately not silenced by `rhythm = "flutter"`, since that is the case it exists for.

There is no cancellation method here that avoids this – `"average_beat"` shares the assumption. Where flutter is the question, read the atrial wave from a segment between QRS complexes rather than from a cancelled signal.

Interpretation

Every spectral feature is returned alongside `on_harmonic`. A contaminated dominant frequency is not noisy, it is precise and wrong, and it is highly reproducible because heart rate is highly reproducible within a patient. `dominant_rate` must not be used without conditioning on `on_harmonic`.

Ten seconds is a short record for this measurement. Even with correct cancellation a material fraction of atrial fibrillation records will carry a contaminated rate.

The output is meaningful only in atrial fibrillation or flutter. In sinus rhythm there is no fibrillatory wave to find, so the estimator returns whatever is largest in the band. When `rhythm` is not supplied, irregularity is inferred from the RR series and a warning is issued if the record does not look like atrial fibrillation.

Guiding channel

A delineating annotator may be run over a record once, or once per lead. Run per lead it writes twelve independent copies of every fiducial, separated only by the `channel` column, and such a table does not describe beats until it is resolved to a single lead: pooled, it reports twelve times as many beats as the record contains, and taking the first match at each point silently follows whichever lead happens to sort first.

So every function that reads fiducials - `get_windows()`, `learn_template()`, `median_window()` and the other window transforms, `extract_f_waves()`, `vcg()` - takes a `channel`, and resolves it the same way:

- annotations spanning more than one channel with no `channel` given are an error, not a guess;
- a `channel` that the annotations do not carry is an error, which is what catches a numbering convention that does not match;
- the global channel `0` is not a lead, so it never counts toward the span and is always kept alongside the chosen lead.

The channel may be a number or a channel name (`"II"`), resolved against the record's own header. A number is matched literally against the `channel` column, and the numbering there belongs to the annotator that wrote the file. Read it off the record rather than assuming:

```
table(get_annotation(x)$channel)
```

Only the boundaries are restricted to one lead. Windows still carry the signal for every channel, which is what makes a windowed beat usable as a vectorcardiogram.

References

Stridh M, Sornmo L. Spatiotemporal QRST cancellation techniques for analysis of atrial fibrillation. *IEEE Transactions on Biomedical Engineering*. 2001;48(1):105-111. doi:10.1109/10.900266

Slocum J, Sahakian A, Swiryn S. Diagnosis of atrial fibrillation from surface electrocardiograms based on computer-detected atrial activity. *Journal of Electrocardiology*. 1992;25(1):1-8. doi:10.1016/00220736(92)90123h

Bollmann A, Husser D, Mainardi L, et al. Analysis of surface electrocardiograms in atrial fibrillation: techniques, research, and clinical applications. *Europace*. 2006;8(11):911-926. doi:10.1093/europace/eul113

Ng J, Goldberger JJ. Understanding and interpreting dominant frequency analysis of AF electrograms. *Journal of Cardiovascular Electrophysiology*. 2007;18(6):680-685. doi:10.1111/j.1540-8167.2007.00832.x

Lankveld TAR, Zeemering S, Crijns HJGM, Schotten U. The ECG as a tool to determine atrial fibrillation complexity. *Heart*. 2014;100(14):1077-1084. doi:10.1136/heartjnl2013305149

Alcaraz R, Abasolo D, Hornero R, Rieta JJ. Optimized assessment of atrial fibrillation organization through suitable parameters of sample entropy. *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. 2010;2010:118-121. doi:10.1109/IEMBS.2010.5627169

Alcaraz R, Abasolo D, Hornero R, Rieta JJ. Optimal parameters study for sample entropy-based atrial fibrillation organization analysis. *Computer Methods and Programs in Biomedicine*. 2010;99(1):124-132. doi:10.1016/j.cmpb.2010.02.009

Alcaraz R, Rieta JJ. Sample entropy of the main atrial wave predicts spontaneous termination of paroxysmal atrial fibrillation. *Medical Engineering & Physics*. 2009;31(8):917-922. doi:10.1016/j.medengphy.2009.05.002

Escribano P, Rodenas J, Garcia M, et al. Combination of frequency- and time-domain characteristics of the fibrillatory waves for enhanced prediction of persistent atrial fibrillation recurrence after catheter ablation. *Heliyon*. 2024;10(3):e25295. doi:10.1016/j.heliyon.2024.e25295

An K, Li H, Yu C, Zheng Z. Surface electrocardiogram f wave analysis in patients with atrial fibrillation undergoing thoracoscopic epicardial ablation. *Interdisciplinary CardioVascular and Thoracic Surgery*. 2024;38(5):ivae057. doi:10.1093/icvts/ivae057

Li YH, Hwang JJ, Tseng YZ, Kuan P, Lien WP. Clinical significance of fibrillatory wave amplitude. A clue to left atrial appendage function in nonrheumatic atrial fibrillation. *Chest*. 1995;108(2):359-363. doi:10.1378/chest.108.2.359

See Also

`cancel_ventricular_signal()`, `calculate_dominant_frequency()`, `calculate_sample_entropy()`, `calculate_organization_index()`

Examples

```
## Not run:
af <- read_wfdb("muse-af", system.file("extdata", package = "EGM"))
res <- extract_f_waves(af)

# Never read the rate without the diagnostics beside it
res$features[, .(lead, dominant_rate, on_harmonic, cancellation_residual)]

## End(Not run)
```

---

f_wave_diagnostics *Diagnostics returned with every fibrillatory estimate, and where they came from*

Description

Not a function. This documents the diagnostic fields that `extract_f_waves()` returns beside each spectral feature, why they must be read together with it, and – for each output of the fibrillatory analysis – whether it is a published quantity, an adaptation of one, or specific to this package.

Where each output comes from

A number with no citation should be assumed to be wrong until it has one, so the provenance is stated here rather than left to be inferred.

Published, and computed as published. `dominant_rate` is the atrial fibrillatory rate, reported in fibrillations per minute as the literature does and read from lead V1 by default as the literature does (Bollmann et al. 2006; An et al. 2024; Escribano et al. 2024). `f_amplitude_p2p` is fibrillatory wave amplitude, the measure behind the coarse-versus-fine distinction: Li et al. called atrial fibrillation coarse at a peak-to-peak amplitude of 1 mm or more in V1, and found left atrial appendage dysfunction and thrombus more common in those patients. `rr_cv` is the coefficient of variation of the RR interval, which Tateno and Glass used to detect atrial fibrillation at 86.6% sensitivity and 84.3% specificity.

Adapted from a published quantity. `sample_entropy` is Richman and Moorman's statistic with the embedding dimension, tolerance and sampling rate that Alcaraz et al. tuned for atrial fibrillation – but they tuned them on the *main atrial wave*, lead V1's atrial signal narrowed to a band around its dominant frequency, and by default it is computed here on the broadband cancelled signal, which reads higher (0.16 against 0.12 on `muse-af` V1). `entropy_input = "main_atrial_wave"` computes it on that wave through the 3 Hz band they selected, and with `entropy_tolerance = 0.25` on their exact parameter set. `f_ratio` is the Alcaraz-Rieta group's normalised f-wave amplitude, the RMS f-wave amplitude over the R-peak magnitude, with the peak-to-peak QRS excursion as the denominator and as a fraction rather than a percentage, so proportional to theirs and not equal. `organization_index` follows Everett's definition and An's surface implementation but is not on their scale; see `calculate_organization_index()`. `cancellation_residual` is a variant of the published ventricular residue index, on a different normalisation.

Specific to this package, with no published definition. `harmonic_overlap` and `on_harmonic`, described below.

Package operating points, not published cut-points. `af_like` fires at an RR coefficient of variation of 0.12 with a normalised RMSSD of 0.10, and `rr_regular` below 0.05. The indices are the published ones; these three thresholds were chosen here, and exist to decide whether to warn rather than to diagnose anything.

The diagnostic fields

`harmonic_overlap` How many fibrillatory cycles fit into one average heartbeat: the dominant frequency divided by the heart rate, both in Hz. Ventricular energy that cancellation failed to remove sits on integer multiples of the heart rate, so a contaminated peak makes this a whole number. **This is not a published quantity.** The concern behind it is established – Ng and Goldberger set out how a dominant frequency can report ventricular residual rather than atrial activity – but no paper defines this ratio, and the name should not be read as naming something from the literature. It was called `harmonic_index` in an earlier version, which implied more than it should have.

`on_harmonic` `TRUE` when `harmonic_overlap` is within `tol` of an integer. `dominant_rate` **must not be used without conditioning on this.** A contaminated estimate is precise, wrong, and highly reproducible, because heart rate is highly reproducible within a patient. Validating the feature by test-retest reliability will therefore select the artifact. It is a sensitive screen rather than a specific test: the overlap of an honest peak is a ratio of two unrelated numbers, so with integers spaced one apart a tolerance of 0.15 lands on one about three times in ten by arithmetic alone, and excluding every flagged record discards roughly a third of the good ones with the bad. Read it beside `cancellation_residual`; a flagged record with a small residual is probably a coincidence.

`cancellation_residual` Residual energy in a window around each QRS as a fraction of that window's energy before cancellation. Model-free, so it is independent of any assumption about the atrial spectrum. It reports what cancellation *failed* to remove, never what it removed and should not have; see `rr_regular` for that.

`rr_regular` `TRUE` when the ventricular response is regular (RR CV below 0.05). Cancellation subtracts whatever repeats at a fixed phase to the QRS, so on a regular rhythm it takes the atrial signal with it - flutter conducting at a fixed ratio loses most of its wave, and its `organization_index` collapses into the range fibrillation occupies. `cancellation_residual` is *small* when this happens, because the template fits better for having absorbed the atrial wave, so this is the only field that reports it. See the cancellation section of `extract_f_waves()`.

`tq_fraction` Share of the record that was electrically silent and therefore usable for amplitude measurement.

`n_beats_cancelled`, `n_beats_skipped`, `n_beats_aberrant` How much of the record was actually processed.

References

Ng J, Goldberger JJ. Understanding and interpreting dominant frequency analysis of AF electrograms. *Journal of Cardiovascular Electrophysiology*. 2007;18(6):680-685. doi:10.1111/j.1540-8167.2007.00832.x

Li YH, Hwang JJ, Tseng YZ, Kuan P, Lien WP. Clinical significance of fibrillatory wave amplitude. A clue to left atrial appendage function in nonrheumatic atrial fibrillation. *Chest*. 1995;108(2):359-363. doi:10.1378/chest.108.2.359

Tateno K, Glass L. Automatic detection of atrial fibrillation using the coefficient of variation and density histograms of RR and deltaRR intervals. *Medical & Biological Engineering & Computing*. 2001;39(6):664-671. doi:10.1007/BF02345439

Alcaraz R, Rieta JJ. Adaptive singular value cancelation of ventricular activity in single-lead atrial fibrillation electrocardiograms. *Physiological Measurement*. 2008;29(12):1351-1369. doi:10.1088/09673334/29/12/001

Alcaraz R, Sornmo L, Rieta JJ. Reference database and performance evaluation of methods for extraction of atrial fibrillatory waves in the ECG. *Physiological Measurement*. 2019;40(7):075011. doi:10.1088/13616579/ab2b17

Alcaraz R, Rieta JJ. Sample entropy of the main atrial wave predicts spontaneous termination of paroxysmal atrial fibrillation. *Medical Engineering & Physics*. 2009;31(8):917-922. doi:10.1016/j.medengphy.2009.05.002

Alcaraz R, Abasolo D, Hornero R, Rieta JJ. Optimal parameters study for sample entropy-based atrial fibrillation organization analysis. *Computer Methods and Programs in Biomedicine*. 2010;99(1):124-132. doi:10.1016/j.cmpb.2010.02.009

Escribano P, Rodenas J, Garcia M, et al. Combination of frequency- and time-domain characteristics of the fibrillatory waves for enhanced prediction of persistent atrial fibrillation recurrence after catheter ablation. *Heliyon*. 2024;10(3):e25295. doi:10.1016/j.heliyon.2024.e25295

Bollmann A, Husser D, Mainardi L, et al. Analysis of surface electrocardiograms in atrial fibrillation: techniques, research, and clinical applications. *Europace*. 2006;8(11):911-926. doi:10.1093/europace/eul113

An K, Li H, Yu C, Zheng Z. Surface electrocardiogram f wave analysis in patients with atrial fibrillation undergoing thoracoscopic epicardial ablation. *Interdisciplinary CardioVascular and Thoracic Surgery*. 2024;38(5):ivae057. doi:10.1093/icvts/ivae057

`feature_range` *Specify a range-valued annotation criterion*

Description

Creates a criterion for numeric annotation fields that can be placed in a landmark criteria list, for example `list(voltage = feature_range(-1, 1))`.

Usage

```
feature_range(lower, upper, inclusive = TRUE)
```

Arguments

lower, upper	Finite numeric interval limits.
inclusive	Whether both limits are included.

Value

A feature_range criterion object.

---

format.windows *Format a* windows *object for printing*

Description

Format a windows object for printing

Usage

```
## S3 method for class 'windows'
format(x, ...)
```

Arguments

x	A windows object.
...	Additional arguments passed to methods.

Value

Invisibly returns x.

---

frequency *Sampling frequency of signal data*

Description

[Experimental]

Reports the sampling rate, in Hz, that a record was acquired at. This is read from the header rather than inferred from the signal, and is the value that converts sample indices into physical time. Use change_frequency() to change it.

Usage

```
## S3 method for class 'EGM'
frequency(x, ...)

## S3 method for class 'header_table'
frequency(x, ...)

## S3 method for class 'windows'
frequency(x, ...)
```

Arguments

x	An EGM, header_table, or windows object.
...	Additional arguments (currently unused).

Details

These are methods for the stats::frequency() generic, defined for every object that carries a rate: an EGM (and so an ECG), a bare header_table, and a windows collection. A header_table needs its own method because the default one answers 1 for any object without a tsp attribute, and a 500 Hz record reported as 1 Hz is wrong in a way that nothing downstream can catch.

For a single record the rate is a scalar. A windows collection may in principle mix rates - windows from different source records can be concatenated with c.windows() - so its method returns the *distinct* rates present. A result of length one therefore means the collection is already harmonised, and anything longer means the windows are not directly comparable.

A record whose header carries no usable rate is an error rather than an NA. There is no arithmetic that recovers from a missing sampling rate: it propagates into every interval, every heart rate, and every duration, while leaving the analyses that do not divide by it looking healthy. Repair the header, or state the rate with change_frequency().

Value

A numeric sampling rate in Hz; for windows objects a vector of the distinct rates present (empty for an empty collection).

See Also

change_frequency()

Examples

```
## Not run:
ecg <- read_wfdb("ecg", test_path(), "ecgpuwave")
frequency(ecg)

# The header answers for itself
frequency(ecg$header)

# A harmonised collection reports a single rate
```

```
frequency(get_windows(ecg))

## End(Not run)
```

`get_annotation` *Get annotation from* `EGM` *object*

Description

Extract an annotator's annotation table from an `EGM` object, which may hold several.

Usage

```
get_annotation(x, annotator = NULL)
```

Arguments

`x`	An `EGM` object containing annotations
`annotator`	Character string specifying which annotator to extract. If NULL (default), returns every annotator's table.

Details

The shape returned matches `read_annotation()`: a single `annotation_table` when the record carries one annotator, and a named list of them when it carries several. That symmetry is the point of the function

- code can move between a record read from disk and one held in memory without changing shape, and an empty record answers with an empty `annotation_table` rather than an empty list.

It is also the way to reach annotations that only exist in memory. The sample indices carried by an `EGM` are rescaled by `change_frequency()`, while a copy read separately from disk is not, so after a rate change these are the annotations that agree with the signal.

Value

An `annotation_table` for the requested annotator; with `annotator = NULL`, the single `annotation_table` present, or a named list of them when the record carries more than one.

See Also

`read_annotation()`, `list_annotators()`, `change_frequency()`

get_signal *Get raw signal data from an* EGM *object*

Description

Raw signal data may be all that is required, particularly when storing or manipulating data, or for example, feeding it into an analytical pipeline. This means the extraneous elements, such as the *meta* information, may be unnecessary. This function retrieves just the signal data itself and channel names.

Usage

```
get_signal(object, data_format = c("data.frame", "matrix", "array"), ...)
```

Arguments

object	An EGM object that contains the signal data to be extracted
data_format	A character choice of either *data.frame* (default), *matrix*, or *array* that tells how the data should be structured. Further explanation in the details.
...	Additional arguments to be passed to the function

Details

The options to return the data vary based on need. The data can be extracted as follows:

- data.frame containing an equal number of rows to the number of samples, with each column named after the recording channel it was derived from. Data frames, as they are columnar by nature, will also include the sample index position.
- matrix containing an equal number of rows to the number of samples, with each column named after the recording channel it was derived from
- array containing individual vectors of signal, each named after the channel they were derived from

Value

An object as described by the data_format option

`get_windows` *Extract windows of signal from a record*

Description

[Experimental]

Searches a record for every span of signal that satisfies a windowing strategy, and returns each span as its own `EGM`. This is how a whole study is broken into the analysable segments - individual beats, mapping windows, intervals of interest - that the rest of the window functions operate on.

Usage

```
get_windows(object, by = "rhythm")
```

Arguments

`object`	An `EGM` object, including header, signal, and annotation information.
`by`	A window_strategy object, or the name of one as a string. Defaults to `"rhythm"`.

Details

The strategy may be given as a bare name for its defaults, or built explicitly with a `by_*()` constructor when its arguments need setting:

```
get_windows(ecg, by = "rhythm")
get_windows(ecg, by = by_rhythm(channel = 2))
```

Three strategies are built in: `by_rhythm()` cuts a whole beat between two fiducials, `by_pwave()` the atrial portion of one, and `by_beat()` a fixed span around a fiducial.

Both forms route through the same constructor, so both are validated the same way. Strategy arguments belong to the constructor rather than to this function, which is what allows a mistyped argument to be reported instead of silently ignored.

Each returned window is a complete `EGM`: its own header (named `<record>_<strategy><n>`, with an info string recording the source sample range), the signal for every channel, and the annotations falling inside it. Windows are generally *not* the same length, since they follow the signal rather than a fixed duration; see `pad_window()`, `normalize_window()`, and `warp_window()` for the ways to bring them onto a common basis.

Value

A windows object: a collection of `EGM` objects, one per window.

Guiding channel

A delineating annotator may be run over a record once, or once per lead. Run per lead it writes twelve independent copies of every fiducial, separated only by the `channel` column, and such a table does not describe beats until it is resolved to a single lead: pooled, it reports twelve times as many beats as the record contains, and taking the first match at each point silently follows whichever lead happens to sort first.

So every function that reads fiducials - `get_windows()`, `learn_template()`, `median_window()` and the other window transforms, `extract_f_waves()`, `vcg()` - takes a `channel`, and resolves it the same way:

- annotations spanning more than one channel with no `channel` given are an error, not a guess;
- a `channel` that the annotations do not carry is an error, which is what catches a numbering convention that does not match;
- the global channel `0` is not a lead, so it never counts toward the span and is always kept alongside the chosen lead.

The channel may be a number or a channel name (`"II"`), resolved against the record's own header. A number is matched literally against the `channel` column, and the numbering there belongs to the annotator that wrote the file. Read it off the record rather than assuming:

```
table(get_annotation(x)$channel)
```

Only the boundaries are restricted to one lead. Windows still carry the signal for every channel, which is what makes a windowed beat usable as a vectorcardiogram.

See Also

`by_rhythm()`, `by_pwave()` and `by_beat()` for the available strategies, `window_dropped()` for the candidates a strategy did not return, `pad_window()`, `normalize_window()` and `warp_window()` to bring windows onto a common length, and `change_frequency()` to harmonise sampling rates either before windowing or on the returned collection.

Examples

```
## Not run:
ecg <- read_wfdb("ecg", test_path(), "ecgpuwave")

# Sinus beats, using the strategy defaults
beats <- get_windows(ecg, by = "rhythm")

# Guided by a single lead, then collapsed to a median beat
get_windows(ecg, by = by_rhythm(channel = 2)) |>
  median_window(align_feature = "N", channel = 2)

## End(Not run)
```

header_table *Header Table*

Description

header_table() modifies the data.table class to work with header data. The header data is read in from a similar format as to that of WFDB files and should be compatible/interchangeable when writing out to disk. The details extensively cover the type of data that is input. Generally, this function is called by read_*_header() functions and will generally not be called by the end-user.

Usage

```
header_table(
  record_name = character(),
  number_of_channels = integer(),
  frequency = integer(),
  samples = integer(),
  start_time = as.POSIXct(NA),
  ADC_saturation = integer(),
  file_name = character(),
  storage_format = integer(),
  ADC_gain = 200L,
  ADC_baseline = ADC_zero,
  ADC_units = "mV",
  ADC_resolution = 12L,
  ADC_zero = 0L,
  initial_value = ADC_zero,
  checksum = 0L,
  blocksize = 0L,
  label = character(),
  info_strings = list(),
  additional_gain = 1,
  low_pass = integer(),
  high_pass = integer(),
  color = "#000000",
  scale = integer()
)

is_header_table(x)
```

Arguments

record_name	A character vector of record line information
number_of_channels	An integer describing number of signals
frequency	A numeric value of sampling frequency, 250 Hz default

samples	An integer for the number of samples
start_time	The POSIXct time of recording, with miliseconds included. For example, strptime(start_time, "%Y-%m-%d %H:%M%:%OSn") where as described in base::strptime()
ADC_saturation	An integer representing ADC saturation
file_name	A character for the signal specific information
storage_format	An integer of the bits for the storage format, 16-bit default
ADC_gain	An integer of ADC gain, default of 200
ADC_baseline	An integer of ADC baseline, defaults to **ADC_zero**
ADC_units	A character to describe ADC units, "mV" is default
ADC_resolution	An integer for ADC resolution, default is 12
ADC_zero	An integer for ADC zero, defaults to 0
initial_value	An integer for the initial value, defaults to **ADC_zero** value
checksum	An integer that serves as the checksum
blocksize	An integer of the block size
label	A character description of the signal
info_strings	A list of strings that will be written as an appendix to the header file, usually containing information about the channels, (e.g. list of colors, extra labels, etc).
additional_gain	A numeric Additional gain, defaults to 1.0
low_pass	An integer Low pass filter
high_pass	An integer High pass filter
color	A character Color as hexadecimal format, defaults to black
scale	An integer Scale
x	A data.table object that serves as the header table

Details

The header_table object is relatively complex in that it directly deals with properties of the signal, and allows compatibility with WFDB files and other raw header files for other signal objects. It can be written out using write_wfdb().

Value

A header_table object that is an extension of the data.table class. This contains an adaptation of the function arguments, allowing for compatibility with the WFDB class.

Header file structure

There are three components to the header file:

1. **Record line** that contains the following information, in the order documented, however pieces may be missing based on different parameters. From left to right...
    - Record name

- Number of signals: represents number of segments/channels
- Sampling frequency (optional)
- Number of samples (optional)
- Time: in HH:MM:SS format (optional)
- Date: in DD/MM/YYYY (optional)

2. **Signal specification lines** contains specifications for individual signals, and there must be as many signal lines as there are reported by the above record line. From left to right....
   - File name: usually *.dat
   - Format `integer`: represents storage type, e.g. 8-bit or 16-bit
   - ADC gain: ADC units per physical unit (optional)
     - Baseline: corresponds to 0 physical units, sep = '*(0)" (optional)
     - Units: with '/' as a field separator e.g '*/mV' (optional)
   - ADC resolution `integer`: bits, usually 8 or 16 (optional)
   - ADC zero: represents middle of ADC input range (optional)
   - Initial value (optional)
   - Checksum (optional)
   - Block size (optional)
   - Description: text or label information (optional)
3. **Info strings** are unstructured lines that contains information about the record. Usually are descriptive. Starts with initial '#' without preceding white space at beginning of line.

`is_template` *Test whether an object is a template*

Description

Test whether an object is a template

Usage

```
is_template(x)
```

Arguments

`x`	An object to test.

Value

A single logical value.

is_window_set *Test if an object is a* windows *collection*

Description

Named `is_window_set()` rather than the shorter `is_windows()` so it is not mistaken for the operating-system predicate of that name that many packages define (a test of `.Platform$OS.type`). This tests the EGM `windows` class, not the platform.

Usage

```
is_window_set(x)
```

Arguments

`x`	An object to test.

Value

`TRUE` if `x` is a `windows` object, `FALSE` otherwise.

is_window_strategy *Test whether an object is a window strategy*

Description

Test whether an object is a window strategy

Usage

```
is_window_strategy(x)
```

Arguments

`x`	An object to test.

Value

A single logical value.

`kors` *Kors regression transformation matrix*

Description

The linear map from the 12-lead ECG to the orthogonal X, Y, Z leads of the Frank vectorcardiographic system, used by `vcg()`.

Usage

```
kors
```

Format

A 3 by 8 `numeric` matrix. Rows are the orthogonal axes `X` (positive to the left), `Y` (positive inferiorly) and `Z` (positive posteriorly); columns are the source leads `I`, `II` and `V1` through `V6`.

Details

Only eight leads enter the transformation. III, aVR, aVL and aVF are exact linear combinations of I and II, so a matrix over all twelve would be rank deficient and the extra columns would carry nothing.

Kors' matrix is preferred to the inverse Dower matrix: it reproduces the recorded Frank leads more closely, and measures derived from it carry more prognostic information (Man et al. 2011; Kück et al. 2018).

References

Kors JA, van Herpen G, Sittig AC, van Bemmel JH. Reconstruction of the Frank vectorcardiogram from standard electrocardiographic leads: diagnostic comparison of different methods. *European Heart Journal*. 1990;11(12):1083-1092. doi:10.1093/oxfordjournals.eurheartj.a059647

Man S, Algra AM, Schreurs CA, et al. Influence of the vectorcardiogram synthesis matrix on the power of the electrocardiogram-derived spatial QRS-T angle to predict arrhythmias in patients with ischemic heart disease and systolic left ventricular dysfunction. *Journal of Electrocardiology*. 2011;44(4):410-415. doi:10.1016/j.jelectrocard.2011.04.007

Kück K, Isaksen JL, Graff C, et al. Spatial QRS-T angle variants for prediction of all-cause mortality. *Journal of Electrocardiology*. 2018;51(5):768-775. doi:10.1016/j.jelectrocard.2018.05.011

See Also

`vcg()`

Examples

```
# The orthogonal leads are row combinations of the eight source leads
kors["Z", ]
```

label_waves *Label annotation waves positionally*

Description

[Experimental]

Adds a wave column (one of "P", "QRS", "T", or NA) to an annotation table by recovering wave identity from the peak symbol enclosed within each (/) waveform bracket.

Usage

```
label_waves(ann)
```

Arguments

ann	An annotation_table (or compatible data.table).

Details

Peaks are mapped directly by their type symbol (p -> "P", N -> "QRS", t -> "T"). Brackets are labelled per channel, in sample order: each onset (and its matching offset) inherit the wave of the single peak that falls between them (the first peak if several are present, NA if none). The returned table is a copy; the input is not modified.

Value

A copy of ann with an additional wave column.

Which annotators are usable

This positional inference is what decides whether a delineating annotator can be used with this package, so it is worth stating plainly. The WFDB convention is that a waveform onset or offset carries its wave identity in the number column - 0 for P, 1 for QRS, 2 for T - but plenty of annotators leave number at zero throughout, which looks disqualifying and is not. As long as the file brackets each wave with (and) around a typed peak, wave identity is recovered here from the position of that peak, and the number column is never consulted.

So the requirement on an annotator is only this: (,), and at least one of p, N, t between each pair. Anything meeting it can drive by_rhythm(), by_pwave(), and the wave criteria used throughout, whatever it writes in number.

Wave identity is available anywhere a criteria list is accepted, as a virtual wave field alongside the real columns:

```
by_rhythm(onset = list(type = "(", wave = "P"))
```

See Also

by_rhythm(), get_windows(), annotation_table()

Examples

```
## Not run:
ecg <- read_wfdb("ecg", test_path(), "ecgpuwave")

# `number` is uninformative here, `wave` is not
labelled <- label_waves(get_annotation(ecg))
table(labelled$type, labelled$wave, useNA = "ifany")

## End(Not run)
```

---

landmark *A single template landmark*

Description

[Experimental]

Usage

```
landmark(
  name,
  criteria,
  channel = NA_integer_,
  position = NA_real_,
  required = TRUE
)
```

Arguments

name	A single non-empty landmark name.
criteria	A non-empty named list of annotation matching criteria, such as `list(type = "N")`.
channel	One non-negative channel number, a stable channel name, or `NA` to search without a channel restriction.
position	Target sample position, or `NA` for an unfitted landmark.
required	Whether this landmark must be found for a complete match.

Details

A landmark identifies one annotation-derived fiducial point on one channel. Its `position` is `NA` until learned, or a sample coordinate on a template's target grid when supplied manually or returned by `learn_template()`.

Value

A `landmark` S7 object.

`learn_template` *Learn a landmark template from EGM examples*

Description

Locates each requested landmark in multiple annotated EGM examples, estimates its median or mean fractional position, and returns a fitted template. For manual templates, construct positioned landmark objects and pass them to `template()` directly.

Usage

```
learn_template(
  x,
  landmarks = list(P_onset = list(type = "(", wave = "P"), QRS = list(type = "N"),
    T_offset = list(type = ")", wave = "T")),
  target_samples = 500L,
  channel = NULL,
  frequency = NULL,
  position_estimator = c("median", "mean"),
  missing = c("complete", "available", "error"),
  ambiguous = c("error", "first"),
  order_policy = c("drop", "error"),
  ...
)
```

Arguments

`x`	An annotated `EGM`, list of `EGM` objects, or `windows` object.
`landmarks`	A named list of concise landmark specifications or a list of unfitted landmark objects. A concise specification may be a type string or a named criteria list with optional `channel` and `required` fields.
`target_samples`	Number of samples in the template target grid.
`channel`	Default channel for landmarks without their own one, given as a channel number or name. Required when the examples' annotations span more than one channel; see the channels section.
`frequency`	Optional source frequency stored as provenance. By default it is read from the first example.
`position_estimator`	Use the median or mean landmark phase.
`missing`	Retain only complete examples, use all available observations, or error on incomplete examples.
`ambiguous`	Error when a landmark has multiple matches, or use the first.
`order_policy`	Drop examples with crossed landmarks, or error.
`...`	Additional arguments, currently unused.

Value

A learned template S7 object.

Guiding channel

A delineating annotator may be run over a record once, or once per lead. Run per lead it writes twelve independent copies of every fiducial, separated only by the `channel` column, and such a table does not describe beats until it is resolved to a single lead: pooled, it reports twelve times as many beats as the record contains, and taking the first match at each point silently follows whichever lead happens to sort first.

So every function that reads fiducials - `get_windows()`, `learn_template()`, `median_window()` and the other window transforms, `extract_f_waves()`, `vcg()` - takes a `channel`, and resolves it the same way:

- annotations spanning more than one channel with no `channel` given are an error, not a guess;
- a `channel` that the annotations do not carry is an error, which is what catches a numbering convention that does not match;
- the global channel `0` is not a lead, so it never counts toward the span and is always kept alongside the chosen lead.

The channel may be a number or a channel name (`"II"`), resolved against the record's own header. A number is matched literally against the `channel` column, and the numbering there belongs to the annotator that wrote the file. Read it off the record rather than assuming:

```
table(get_annotation(x)$channel)
```

Only the boundaries are restricted to one lead. Windows still carry the signal for every channel, which is what makes a windowed beat usable as a vectorcardiogram.

`list_annotators` *List annotators in* `EGM` *object*

Description

Get the names of all annotators present in an `EGM` object.

Usage

```
list_annotators(x)
```

Arguments

`x`	An `EGM` object containing annotations

Value

A character vector of annotator names

map_windows *Apply a function across a collection of windows*

Description

[Experimental]

Applies `f` to each `EGM` in a collection. When every result is itself an `EGM` the collection is rebuilt as a `windows` object with its provenance carried forward; otherwise a plain list of results is returned, which is the natural shape for extracting per-window measurements.

Usage

```
map_windows(x, f, ...)
```

Arguments

`x`	A `windows` object or list of `EGM` objects.
`f`	A function applied to each window.
`...`	Additional arguments passed to `f`.

Value

A `windows` object when all results are `EGM` objects, otherwise a list.

Examples

```
## Not run:
beats <- get_windows(read_wfdb("ecg", test_path(), "ecgpuwave"))

# Per-window measurement -> plain list
map_windows(beats, function(w) nrow(w$signal))

# EGM-valued transform -> windows object, provenance preserved
map_windows(beats, function(w) w)

## End(Not run)
```

median_window *Collapse windows to a single median beat*

Description

[Experimental]

Reduces a windows collection to a single representative EGM by taking the element-wise median across all windows, lead by lead. Given an ECG windowed into its individual beats, this returns the "median beat" - a robust template that suppresses beat-to-beat noise and ectopy while preserving the dominant morphology.

Usage

```
median_window(x, align_feature = NULL, channel = NULL, na.rm = TRUE, ...)
```

Arguments

x	A windows object (or list of EGM objects) to reduce.
align_feature	Optional feature used to pad-align windows of unequal length before averaging, given as a character type symbol (e.g. "N", the QRS peak) or a named list of annotation criteria. If NULL (default), the windows must already share a common length.
channel	The lead whose align_feature anchors the alignment, given as a channel number or name. Required when the annotations span more than one channel; see the channels section.
na.rm	Logical passed to stats::median(); if TRUE (default), missing values are ignored when averaging.
...	Additional arguments (currently unused).

Details

The median is computed per sample and per lead, so every window must share the same length and the same set of leads. Beats windowed from raw rhythm are generally *not* the same length (see get_windows()), so this function offers to align them first: when align_feature is supplied and the windows differ in length, they are padded and anchored on that feature via pad_window() before the median is taken. If the windows differ in length and no align_feature is given, an error is raised pointing to pad_window() or normalize_window() so the caller chooses the alignment explicitly.

The returned beat carries the fiducials that produced it, each placed at the median of its own positions across the aligned windows. Annotations are matched between windows by channel, type, and how many of that pair came before them in their own window, so the first QRS onset of one beat lines up with the first of every other. A fiducial that most windows do not carry is dropped, since it is not part of what the median describes. The header is named <source>_median and carries an info string recording how many windows it came from.

Value

A single EGM object representing the median beat.

Its fiducials live where every EGM keeps them, under $annotation, which is a *named list* of annotation_tables - one per annotator - and not a table. Reach them with get_annotation(x), which unwraps the single annotator case; nrow(x$annotation) is NULL here for the same reason it is NULL on a record straight from read_wfdb().

Amplitudes are not baseline-corrected. The median of a set of windows that all share a DC offset still carries that offset, and it will be the first thing any variance-based method finds; baseline_window() removes it, best applied to the windows before they are reduced.

Guiding channel

A delineating annotator may be run over a record once, or once per lead. Run per lead it writes twelve independent copies of every fiducial, separated only by the channel column, and such a table does not describe beats until it is resolved to a single lead: pooled, it reports twelve times as many beats as the record contains, and taking the first match at each point silently follows whichever lead happens to sort first.

So every function that reads fiducials - get_windows(), learn_template(), median_window() and the other window transforms, extract_f_waves(), vcg() - takes a channel, and resolves it the same way:

- annotations spanning more than one channel with no channel given are an error, not a guess;
- a channel that the annotations do not carry is an error, which is what catches a numbering convention that does not match;
- the global channel 0 is not a lead, so it never counts toward the span and is always kept alongside the chosen lead.

The channel may be a number or a channel name ("II"), resolved against the record's own header. A number is matched literally against the channel column, and the numbering there belongs to the annotator that wrote the file. Read it off the record rather than assuming:

```
table(get_annotation(x)$channel)
```

Only the boundaries are restricted to one lead. Windows still carry the signal for every channel, which is what makes a windowed beat usable as a vectorcardiogram.

See Also

get_annotation() to read the fiducials, baseline_window(), pad_window(), by_beat()

merge_annotations *Merge multiple annotations into a single table*

Description

Combine multiple annotation tables from an EGM object into a single annotation table with an additional annotator column to identify the source of each annotation.

Usage

```
merge_annotations(x, annotators = NULL)
```

Arguments

x	An EGM object containing annotations, or a named list of annotation_table objects
annotators	Optional character vector specifying which annotators to merge. If NULL (default), merges all available annotators.

Value

A single annotation_table object with all annotations combined and an additional annotator column

muse *Read in ECG data from MUSE*

Description

This function serves to read/convert XML based files from the MUSE system to digital signal. This can subsequently be written into other formats. The MUSE system is somewhat proprietary, and each version may or may not allow export options into XML.

Usage

```
read_muse(file)
```

Arguments

file	An ECG file from MUSE in XML format

Details

GE Healthcare MUSE v9 is currently the model that is being used. These functions have not been tested in older versions.

Value

An EGM class object that is a list of surface ECG signals the format of a data.table, with an attached **header** attribute that contains additional recording data.

Units and gain

MUSE stores each sample as an ADC count together with the scale that converts it, LeadAmplitudeUnitsPerBit, and the units that scale is in, LeadAmplitudeUnits. The scale is applied while reading, so the returned signal holds those units — microvolts on every export seen — and the header gets the ADC_gain that carries them back to millivolts, 1000 for a microvolt payload. A record whose units are unrecognised, or that names different units on different leads, is refused rather than given a gain that would put its amplitudes on an unknown scale.

The samples are deliberately left on the microvolt scale rather than returned to raw counts. The augmented limb leads are derived here as halves of the recorded ones, so they fall between counts, and rounding them back onto the count grid would throw that precision away.

---

new_windows *Create a* windows *object containing a collection of EGM windows*

Description

[Experimental]

windows objects are lists of EGM objects that represent segments of one or more source signals. This allows specialized methods to be applied to a whole collection of segments at once. This function is the class constructor and only applies class attributes; collections are normally produced by get_windows() rather than built by hand.

Usage

```
new_windows(
  x = list(),
  method = character(),
  source_record = character(),
  history = method,
  dropped = integer(),
  ...
)
```

Arguments

x	A list of EGM objects.
method	The extraction strategy that produced the collection, e.g. "rhythm". Recorded once and carried through every transform.
source_record	The name of the original record.
history	A character vector of the steps applied so far, in order. Defaults to method, which is correct at the point of extraction.

dropped	A named integer vector counting candidate beats the strategy found but did not return, by reason. Read it back with window_dropped().
...	Additional arguments passed to methods.

Value

An object of class windows which inherits from list.

See Also

get_windows() to create one from an EGM, window_dropped() for the candidates it did not return.

---

normalize_window *Time-normalize windows to a fixed length*

Description

[Experimental]

Resamples each window so that every window spans the same number of samples. Without align_feature this stretches or compresses each window as a whole, mapping its left and right borders (e.g. P-onset and T-offset for a sinus beat) onto a common [0, target_samples) grid. With align_feature the windows are instead resampled at native resolution around that fiducial, so the feature lands at the centre of every output window and the true morphology timing is preserved.

This is a *destructive* standardization - in the border-to-border mode the time axis is warped, so absolute intervals are no longer comparable - but it is a fast, simple way to bring ragged beats onto a shared basis for averaging or matrix operations.

Usage

```
normalize_window(
  x,
  target_samples = 500L,
  target_ms = NULL,
  interpolation_method = c("linear", "spline", "step"),
  align_feature = NULL,
  channel = NULL,
  preserve_amplitude = FALSE,
  preserve_class = TRUE,
  ...
)
```

Arguments

x	A windows object (or list of EGM objects) to normalize.
target_samples	Desired output length in samples. Default 500.

`target_ms` Alternative target length in milliseconds, converted using the first window's sampling frequency. Used only when `target_samples` is `NULL`.

`interpolation_method`
Interpolation method for the stretch. One of "linear" (default), "spline", or "step".

`align_feature` Optional feature to centre each window on, given as a `character` type symbol (e.g. "N", the QRS peak) or a named list of annotation criteria. `NULL` (default) resamples border-to-border.

`channel` The lead whose `align_feature` the windows are centred on, given as a channel number or name. Multi-lead annotation files (e.g. an `ecgpuwave`-style run per lead) carry one fiducial per lead at slightly different samples, so a bare `align_feature = "N"` would otherwise centre on whichever lead sorts first. Ignored when `align_feature` is `NULL` or already specifies a `channel`; see the channels section.

`preserve_amplitude`
Logical. If TRUE, rescales each lead back to its original amplitude range after interpolation. Defaults to FALSE, since a pure time stretch should leave amplitudes to the interpolation.

`preserve_class` Logical. If TRUE (default), returns a `windows` object; if FALSE, returns a plain list of `EGM` objects.

`...` Additional arguments (currently unused).

Details

For alignment on a fiducial without warping the time axis at all, use `pad_window()` instead; to align several landmarks at once, use `warp_window()`; to change the physical sampling rate while preserving duration, use `change_frequency()`.

When `align_feature` is given but not found in a window, that window falls back to border-to-border resampling with a warning.

Value

A `windows` object (or list) of time-normalized `EGM` objects, each exactly `target_samples` long. Each carries a resampled `signal_table`, the window's own `header_table` (with `samples` updated and the per-beat record/file name preserved), and its annotations remapped onto the resampled time base.

Guiding channel

A delineating annotator may be run over a record once, or once per lead. Run per lead it writes twelve independent copies of every fiducial, separated only by the `channel` column, and such a table does not describe beats until it is resolved to a single lead: pooled, it reports twelve times as many beats as the record contains, and taking the first match at each point silently follows whichever lead happens to sort first.

So every function that reads fiducials - `get_windows()`, `learn_template()`, `median_window()` and the other window transforms, `extract_f_waves()`, `vcg()` - takes a `channel`, and resolves it the same way:

- annotations spanning more than one channel with no `channel` given are an error, not a guess;
- a `channel` that the annotations do not carry is an error, which is what catches a numbering convention that does not match;
- the global channel `0` is not a lead, so it never counts toward the span and is always kept alongside the chosen lead.

The channel may be a number or a channel name (`"II"`), resolved against the record's own header. A number is matched literally against the `channel` column, and the numbering there belongs to the annotator that wrote the file. Read it off the record rather than assuming:

```
table(get_annotation(x)$channel)
```

Only the boundaries are restricted to one lead. Windows still carry the signal for every channel, which is what makes a windowed beat usable as a vectorcardiogram.

Examples

```
## Not run:
ecg <- read_wfdb("ecg", test_path(), "ecgpuwave")
beats <- get_windows(ecg, by = by_rhythm(channel = 2))

# Stretch every beat border-to-border onto 500 samples
normalize_window(beats, target_samples = 500)

# Or 500 milliseconds, using the record's sampling frequency
normalize_window(beats, target_samples = NULL, target_ms = 500)

# Centre each beat on its QRS instead of stretching the borders
normalize_window(beats, align_feature = "N", channel = 2)

## End(Not run)
```

`pad_window` *Pad windows to a common length*

Description

[Experimental]

Extends each window with `pad_value` so that every window shares a common length. Unlike time normalization, padding is *non-destructive*: the original samples are preserved and only the edges are added to. Use it to anchor a fiducial - typically the QRS peak - at a fixed index across windows, or to make ragged windows rectangular for stacking.

Where the windows can be cut equal in the first place, `by_beat()` is the better route: padding marks the added samples absent, but absent samples still thin out the reduction that follows.

Usage

```
pad_window(
  x,
  target_samples = NULL,
  target_ms = NULL,
  align = c("center", "left", "right", "feature"),
  align_feature = "N",
  feature_position = NULL,
  channel = NULL,
  pad_value = NA,
  preserve_class = TRUE,
  ...
)
```

Arguments

x	A windows object (or list of EGM objects) to pad.
target_samples	Desired output length in samples. If NULL (default), the length is derived from the data (the longest window, or the feature-span for align = "feature").
target_ms	Alternative target length in milliseconds, converted to samples using the first window's sampling frequency. Ignored when target_samples is supplied.
align	One of "center" (default), "left", "right", or "feature". See details.
align_feature	Feature used for align = "feature", given as a character type symbol (default "N", the QRS peak) or a named list of annotation criteria. Ignored for other alignments.
feature_position	Target index (0-based) for the aligned feature when align = "feature" and target_samples is supplied. Defaults to the window centre.
channel	The lead whose align_feature anchors the alignment, given as a channel number or name. Required when the annotations span more than one channel; see the channels section.
pad_value	The value used for padding: NA (default), a single number, or the string "edge". NA marks the added samples as absent rather than as a measurement. Padding with 0 states that the potential there was zero, which is a fabricated observation: it biases median_window() toward the origin at the edges of the beat, and with it every measure read off the loop tails. "edge" extends the nearest observed sample outward instead. See the fill section for choosing between them.
preserve_class	Logical. If TRUE (default), returns a windows object; if FALSE, returns a plain list of EGM objects.
...	Additional arguments (currently unused).

Details

The align argument controls where the real signal sits within the padded window:

- center - equal padding on both edges.

- `left` - signal at the start, padding appended after ("pad after").
- `right` - padding prepended before, signal at the end ("pad before").
- `feature` - each window is shifted so that `align_feature` lands on a shared anchor index, padding the remainder. This is the alignment used to stack beats on the R wave.

For `feature` alignment with no explicit `target_samples`, the output length is derived so that no window is truncated: it spans the largest pre-feature extent plus the largest post-feature extent seen across the collection, and the feature is anchored at that common pre-feature offset. When `target_samples` is supplied, the feature is anchored at `feature_position` (default: the window centre) and any samples that fall outside the target are truncated.

Value

A `windows` object (or list) of padded `EGM` objects, all sharing the same sample length. Amplitudes are untouched, so whatever DC offset the recording carried is still there; `baseline_window()` removes it, and variance-based methods find it first if it is left in.

What the fill claims

Every padded sample is a claim about signal that was never recorded, and the three fills make different ones.

`NA` claims nothing, which is why it is the default. The cost is that a padded collection cannot go straight into a matrix method, and the obvious handling is the wrong one: where the padded length exceeds the median wave span - as it must, or nothing would be padded - *every* window carries some `NA`, so dropping incomplete cases empties the matrix rather than trimming it.

`"edge"` claims the signal held its last observed value. For a window whose pad region is isoelectric - the TP and PR segments either side of a P wave, say - that is close to true and is the fill to reach for. Across a wave it is not, and it flattens the tails it invents.

`0` claims the potential was zero there. Because the windows are not baseline-corrected (see the return value), zero is not even the isoelectric line; it is wherever the recording's DC offset happens to put it.

Guiding channel

A delineating annotator may be run over a record once, or once per lead. Run per lead it writes twelve independent copies of every fiducial, separated only by the `channel` column, and such a table does not describe beats until it is resolved to a single lead: pooled, it reports twelve times as many beats as the record contains, and taking the first match at each point silently follows whichever lead happens to sort first.

So every function that reads fiducials - `get_windows()`, `learn_template()`, `median_window()` and the other window transforms, `extract_f_waves()`, `vcg()` - takes a `channel`, and resolves it the same way:

- annotations spanning more than one channel with no `channel` given are an error, not a guess;
- a `channel` that the annotations do not carry is an error, which is what catches a numbering convention that does not match;
- the global channel `0` is not a lead, so it never counts toward the span and is always kept alongside the chosen lead.

The channel may be a number or a channel name (`"II"`), resolved against the record's own header. A number is matched literally against the `channel` column, and the numbering there belongs to the annotator that wrote the file. Read it off the record rather than assuming:

```
table(get_annotation(x)$channel)
```

Only the boundaries are restricted to one lead. Windows still carry the signal for every channel, which is what makes a windowed beat usable as a vectorcardiogram.

See Also

`baseline_window()`, `median_window()`, `normalize_window()`, `by_beat()` for cutting windows equal in the first place

---

`print.windows` *Print a* `windows` *object*

Description

Print a `windows` object

Usage

```
## S3 method for class 'windows'
print(x, ...)
```

Arguments

`x`	A `windows` object.
`...`	Additional arguments passed to methods.

Value

Invisibly returns `x`.

prucka *Read Prucka System Files*

Description

`read_prucka()` reads both the signal data (*.txt) and header information (*.inf) exported from the Prucka cardiac electrophysiology system, which is the underlying recording software used in GE Healthcare's CardioLab EP system.

Usage

```
read_prucka(signal_file, header_file = NULL, n = Inf)

read_prucka_header(header_file)

read_prucka_signal(signal_file, n = Inf)
```

Arguments

`signal_file`	Path to the *.txt signal data file
`header_file`	Path to the *.inf header file. If NULL, will look for a file with the same base name as signal_file but with .inf extension.
`n`	Number of signal values to return (this will be the same for each channel of data). Defaults to all values.

Details

Exporting from GE CardioLab/Prucka:

To export data from the GE CardioLab system:

1. Open the study/recording in CardioLab
2. Select the time sEGMent you want to export
3. Navigate to **File > Export** or **Tools > Export**
4. Choose **ASCII Export** or **Text Export** format
5. Select the channels to export
6. Choose export location and filename
7. The system will create two files:
   - **X####.txt**: Space-delimited signal data
   - **X####.inf**: Header file with metadata

File Format Details:

Signal file (*.txt):

- Space-delimited numeric data
- Each row represents one time point
- First column: sample index/time marker

- Subsequent columns: channel data in mV
- All channels sampled at the same rate

Header file (*.inf):

- Key-value pairs with "=" delimiter
- Patient information (name, date, description)
- Recording parameters (sampling rate, duration, channel count)
- Channel mapping section listing channel numbers and labels
- Channel numbers may be non-sequential (e.g., 1-12, 49-50, 75-76)

Both files must have the same base name (e.g., X001.txt and X001.inf).

Notes:

- Default units are mV for electrical signals and mmHg for pressure
- The system typically uses 16-bit ADC resolution
- Channel labels may include surface ECG leads (I, II, III, aVR, aVL, aVF, V1-V6) and intracardiac catheters (ABL, His, CS, RV, etc.)
- Export may be limited by system memory for very long recordings

Value

An `EGM` class object that is a list of EP signals the format of a `data.table`, with an attached **header** attribute that contains additional recording data.

`signal_table` *Signal tables*

Description

The `signal_table()` function modifies the `data.table` class to work with electrical signal data. The input should be a data set of equal number of rows. It will add a column of index positions called `sample` if it does not already exist.

Usage

```
signal_table(..., units = c("digital", "physical"))

is_signal_table(x)
```

Arguments

`...`	A `list` of equal lengths
`units`	A `character` naming the units the values are in, either `"digital"` (raw ADC counts, the default) or `"physical"` (the units the header names, usually mV). Must be given by name.
`x`	`data.frame` A data frame of signal data

Details

Every signal table is labelled with the units its values are in, `"digital"` or `"physical"`, readable with `signal_units()`. The label is carried through windowing, padding, medians and resampling, so a table many steps from the file it was read out of still says what its numbers mean.

The distinction is a factor of the ADC gain - commonly 200 - and nothing in the values themselves reveals which one is in hand. Since the same amplitude is a plausible number in either, a mislabelled signal is not detectable downstream; only the label makes it checkable. `write_wfdb()` refuses to write a table whose label contradicts the units it was told to expect.

Value

An object of class `signal_table`, which is an extension of the `data.table` class. The `sample` column is *invariant* and will always be present. The other columns represent additional channels.

See Also

`signal_units()`, `read_wfdb()`

`signal_units` *Units a signal is recorded in*

Description

[Experimental]

Reports whether a signal holds raw ADC counts (`"digital"`) or the physical units its header names (`"physical"`, usually mV). The two differ by the ADC gain - 200 in a great many records - and nothing in the values themselves says which is which, so the answer is carried as a label rather than inferred.

Usage

```
signal_units(x)
```

Arguments

`x`	An `EGM` object or a `signal_table`.

Value

A single `character`, `"digital"` or `"physical"`.

See Also

`read_wfdb()`, `signal_table()`

Examples

```
## Not run:
signal_units(read_wfdb("ecg", test_path()))
#> [1] "digital"

signal_units(read_wfdb("ecg", test_path(), units = "physical"))
#> [1] "physical"

## End(Not run)
```

`template` *A template of ordered landmarks*

Description

[Experimental]

Usage

```
template(
  landmarks,
  target_samples = 500L,
  method = "manual",
  frequency = NA_real_,
  statistics = NULL
)
```

Arguments

`landmarks`	A position-ordered list of fitted landmark objects.
`target_samples`	Length of the target sample grid.
`method`	Either `"manual"` or `"learned"`.
`frequency`	Source sampling frequency retained as provenance, or `NA`.
`statistics`	Optional landmark learning statistics.

Details

A template is an ordered group of positioned landmark objects on a common target sample grid. Create one manually with `template()` or learn landmark positions from multiple EGM examples with `learn_template()`. It is passed directly to `warp_window()`.

Value

A `template` S7 object.

Examples

```
manual_template <- template(
  landmarks = list(
    landmark("onset", list(type = "("), channel = 2, position = 0),
    landmark("QRS", list(type = "N"), channel = 2, position = 250),
    landmark("offset", list(type = ")"), channel = 2, position = 499)
  ),
  target_samples = 500
)
is_template(manual_template)
```

---

validate_time_parameters

Validate WFDB time parameters

Description

`validate_time_parameters()` validates and normalizes the time window used by the WFDB readers. Time-only `begin` and `end` values are elapsed from the record start, consistent with WFDB standard time. Dated values are absolute. `interval` is a duration measured from `begin`.

Usage

```
validate_time_parameters(
  begin = NULL,
  end = NULL,
  interval = NULL,
  start_time = as.POSIXct(NA),
  study_duration = NA_real_
)
```

Arguments

`begin`, `end`	A single time value. Character values may use `"HH:MM:SS"`, `"YYYY-MM-DD HH:MM:SS"`, or the WFDB-style `"HH:MM:SS DD/MM/YYYY"` format, with optional fractional seconds and optional square brackets around a dated WFDB timestamp. `POSIXt`, `hms`, `ITime`, and `difftime` objects are also supported. A `difftime` value is interpreted as elapsed time from the record start.
`interval`	A single duration. Numeric values are seconds. Character values combine a number and unit, such as `"30s"`, `"100 ms"`, `"2 min"`, or `"1.5 hours"`. A `difftime` value is also accepted.
`start_time`	The record start as a `POSIXt` value. This is required for absolute `begin` and `end` values, but not for elapsed time-only values.
`study_duration`	The available record duration in seconds. When known, requested endpoints are clamped to this duration. Records longer than one day require dated `begin` and `end` values instead of time-only clocks.

Value

A list containing normalized `begin`, `end`, and `interval` values in elapsed seconds from the beginning of the record.

`vcg` *Reconstruct a vectorcardiogram from the 12-lead ECG*

Description

`vcg()` applies the kors transformation to the surface ECG and cuts the result at the annotated wave boundaries, tracing the ventricular loop by default and the atrial loop where `method = "atrial"`. It returns the orthogonal X, Y, Z signal together with the components extracted from it.

Usage

```
vcg(
  object,
  method = c("ventricular", "atrial"),
  beats = c("median", "all"),
  channel = NULL,
  baseline = TRUE
)
```

Arguments

`object`	An ECG: a whole record, a single windowed beat, or a median beat. An `EGM` is reduced to its surface leads first (see `as_ECG()`), which is how a 12-lead record read straight off disk, or the surface leads of an electrophysiology study, become usable. All eight leads of the kors transformation must be present.
`method`	Which loop to trace. `"ventricular"` (default) cuts the QRS and T waves and reports the global electric heterogeneity components alongside the geometric ones; `"atrial"` cuts the P wave alone and reports the geometric components only. Each needs the record's own delineation of the waves it cuts.
`beats`	Which beats to trace when the object holds more than one. `"median"` (default) reduces them with `median_window()` and returns a single loop; `"all"` returns one loop per beat, preserving the beat-to-beat variability a signal average is designed to remove (Tachmatzidis et al. 2022). An object that already holds one beat is unaffected by either.
`channel`	The lead whose annotations delineate the waves, given as a channel number or name. Required when the annotations span more than one channel, as they do when an annotator has been run per lead; see the channels section.
`baseline`	Logical. If `TRUE` (default), each beat is referenced to its own onset, taken as the median of the 10 ms following it. Orientation is measured from the origin, so an offset baseline rotates every angle.

Value

A `list` of two `data.tables`:

`loop` The orthogonal signal, with columns `beat`, `sample`, `X`, `Y` and `Z`, in the units of the source signal.

`components` One row per loop, as tabulated below.

Azimuth is measured in the transverse plane from `+X` toward `+Z`, and elevation out of that plane toward `+Y`, both in degrees.

Not every beat reaches the result, and `window_dropped()` reads back how many did not and why: `incomplete_span`, beats too near an end of the record for the fixed window to be cut, and `no_delineation`, beats the annotator did not mark the wave in. Both are counted rather than announced, since the first is unavoidable on a short strip and neither is visible from a background worker.

Reconstruction

The transformation is linear, so it commutes with the segmentation: a beat cut from the transformed record and a beat transformed after cutting are the same signal, and it makes no difference that each beat is transformed separately here.

It does not commute with the median, which is not a linear operator. The median is taken lead by lead, by `median_window()`, so `beats = "median"` returns the loop of the median beat rather than the median of the beats' loops.

Segmentation

The unit of analysis is one beat. Wave boundaries come from the record's own delineation annotations, since neither loop can be delimited without them; a record without them is an error rather than a guess.

A record is cut into beats by `by_beat()`, a fixed span around each wave peak. Equal-length windows are what let `median_window()` reduce them without padding, and a padded sample is a fabricated one - it would drag the median toward whatever the padding says at exactly the loop tails the components are read from.

An object that already holds one beat - a window, or a median of them - is taken as it stands, since there is no room in it for a span and nothing left to reduce. So

```
vcg(ecg)
```

and

```
ecg |>
  get_windows() |>
  median_window(align_feature = "N") |>
  vcg()
```

both describe one beat, the first taking `beats = "median"` from its default and the second building the beat itself. Reach for the second when the windowing or the alignment needs to be something other than the default; `map_windows()` will run `vcg()` over every window of a collection.

Components

Both loops yield the same geometric components: the peak and mean spatial vectors with their orientation, the area the loop encloses, and how far it departs from a plane.

The ventricular loop additionally yields the global electric heterogeneity (GEH) components, which describe the discordance between depolarization and repolarization and so need the T wave to also be delineated: the spatial QRS-T angle in its peak and mean forms, the spatial ventricular gradient (magnitude, azimuth and elevation), and the sum absolute QRST integral (Waks et al. 2016). Where the T wave is not delineated they are returned as `NA` rather than dropped.

`magnitude_peak` on the atrial loop is the spatial P-wave vector magnitude, which falls as the left atrium remodels and tracks low-voltage area at electroanatomic mapping (Yano et al. 2023). A P loop only exists in an organised atrial rhythm; in atrial fibrillation the atrial signal is characterised by `extract_f_waves()` instead.

Components and their units

Half of these components are scale-free and half are not, which matters because the source signal may be in millivolts or in raw ADC counts depending on how the record was read (see the `units` argument of `read_wfdb()`, and `signal_units()` to ask an object which it holds). A gain of 200 changes every value in the "signal units" rows by 200, and none of the others. Between-record comparisons of those rows are only meaningful on signals read the same way.

Component	Units	Scale-free
`duration`	seconds	yes
`magnitude_peak`, `magnitude_mean`	signal units	no
`azimuth_peak`, `elevation_peak`	degrees	yes
`azimuth_mean`, `elevation_mean`	degrees	yes
`area`	signal units squared	no
`planarity`	proportion (0-1)	yes
`qrst_angle_peak`, `qrst_angle_mean`	degrees	yes
`svg_magnitude`	signal units x seconds	no
`svg_azimuth`, `svg_elevation`	degrees	yes
`sai_qrst`	signal units x seconds	no

The last five are returned under `method = "ventricular"` only; they are the global electric heterogeneity components, and the atrial loop declines to report a QRS-T relationship for a P loop.

Guiding channel

A delineating annotator may be run over a record once, or once per lead. Run per lead it writes twelve independent copies of every fiducial, separated only by the `channel` column, and such a table does not describe beats until it is resolved to a single lead: pooled, it reports twelve times as many beats as the record contains, and taking the first match at each point silently follows whichever lead happens to sort first.

So every function that reads fiducials - `get_windows()`, `learn_template()`, `median_window()` and the other window transforms, `extract_f_waves()`, `vcg()` - takes a `channel`, and resolves it the same way:

- annotations spanning more than one channel with no `channel` given are an error, not a guess;
- a `channel` that the annotations do not carry is an error, which is what catches a numbering convention that does not match;
- the global channel `0` is not a lead, so it never counts toward the span and is always kept alongside the chosen lead.

The channel may be a number or a channel name (`"II"`), resolved against the record's own header. A number is matched literally against the `channel` column, and the numbering there belongs to the annotator that wrote the file. Read it off the record rather than assuming:

```
table(get_annotation(x)$channel)
```

Only the boundaries are restricted to one lead. Windows still carry the signal for every channel, which is what makes a windowed beat usable as a vectorcardiogram.

References

Kors JA, van Herpen G, Sittig AC, van Bemmel JH. Reconstruction of the Frank vectorcardiogram from standard electrocardiographic leads: diagnostic comparison of different methods. *European Heart Journal*. 1990;11(12):1083-1092. doi:10.1093/oxfordjournals.eurheartj.a059647

Waks JW, Sitlani CM, Soliman EZ, et al. Global electric heterogeneity risk score for prediction of sudden cardiac death in the general population: the Atherosclerosis Risk in Communities (ARIC) and Cardiovascular Health (CHS) studies. *Circulation*. 2016;133(23):2222-2234. doi:10.1161/CIRCULATIONAHA.116.021306

Havmöller R, Carlson J, Holmqvist F, et al. Age-related changes in P wave morphology in healthy subjects. *BMC Cardiovascular Disorders*. 2007;7:22. doi:10.1186/14712261722

Holmqvist F, Platonov PG, Carlson J, et al. Variable interatrial conduction illustrated in a hypertrophic cardiomyopathy population. *Annals of Noninvasive Electrocardiology*. 2007;12(3):227-236. doi:10.1111/j.1542474X.2007.00166.x

Yano M, Egami Y, Kawanami S, et al. Ratio of P-wave duration to P-wave amplitude and left atrial remodeling. *The American Journal of Cardiology*. 2023;212:109-117. doi:10.1016/j.amjcard.2023.11.046

Tachmatzidis D, Tsarouchas A, Mouselimis D, et al. P-wave beat-to-beat analysis to predict atrial fibrillation recurrence after catheter ablation. *Diagnostics*. 2022;12(4):830. doi:10.3390/diagnostics12040830

See Also

kors for the transformation itself, `as_ECG()` for the surface lead contract, `get_windows()` and `median_window()` for building a beat by hand, `extract_f_waves()` for the atrial signal when there is no P wave to trace.

Examples

```
## Not run:
ecg <- read_wfdb("muse-sinus", system.file("extdata", package = "EGM"),
                 annotator = "ecgpuwave")

vcg(ecg)$components
vcg(ecg, method = "atrial", beats = "all")$components
```

```
# The same loop, with the windowing and alignment chosen explicitly
ecg |>
  get_windows(by = by_rhythm(channel = 2)) |>
  median_window(align_feature = "N", channel = 2) |>
  vcg()

## End(Not run)
```

---

warp_window *Landmark-warp windows onto a template*

Description

[Experimental]

Warps each window so that its landmarks (located from annotations) align to the positions held in a template, typically created by `learn_template()`. Between consecutive landmarks the time axis is stretched or compressed piecewise-linearly, giving a landmark-aligned beat on the template's common time base. This is the most flexible - and most destructive - of the standardization methods, aligning multiple fiducials at once rather than a single anchor.

Usage

```
warp_window(
  x,
  template,
  interpolation_method = c("linear", "spline", "step"),
  channel = NULL,
  preserve_amplitude = FALSE,
  preserve_class = TRUE,
  missing = c("partial", "drop", "error"),
  ambiguous = c("error", "first", "drop"),
  order_policy = c("error", "drop"),
  ...
)
```

Arguments

x	A windows object (or list of EGM objects) to warp.
template	A template created manually with `template()` or learned with `learn_template()`. Plain lists are not accepted.
interpolation_method	Interpolation method for the warp. One of "linear" (default), "spline", or "step".
channel	Fallback guiding channel used when locating a landmark whose own spec does not name one, given as a channel number or name. A landmark's own channel always takes precedence; see the channels section.

`preserve_amplitude`
: Logical. If TRUE, rescales each lead back to its original amplitude range after warping. Defaults to FALSE.

`preserve_class`
: Logical. If TRUE (default), returns a `windows` object; if FALSE, returns a plain list of `EGM` objects.

`missing`
: What to do when a required landmark is absent: warp with the remaining landmarks, drop that window, or error.

`ambiguous`
: What to do when a landmark matches multiple annotations: error, use the first, or drop that window.

`order_policy`
: What to do when landmarks are crossed or duplicated: error or drop that window.

`...`
: Additional arguments (currently unused).

Details

Before warping, each landmark defined by the template is confirmed against the windows; landmarks missing from some windows raise a warning (those windows warp on their remaining anchors) and landmarks absent from every window are dropped. For every window the landmark samples are located - each on its own channel, so a multi-channel template is honoured - and combined with the window's own start/end as edge anchors, forming a set of `(source -> target)` breakpoints. Output samples are mapped back to source positions by inverting this piecewise-linear correspondence, and each lead is interpolated there; annotations are carried forward through the same map. Any breakpoints that are non-increasing in either axis are collapsed to keep the mapping monotonic. All leads are warped regardless of which channel a landmark was detected on.

Value

A `windows` object (or list) of landmark-warped `EGM` objects, each `target_samples` long (as defined by the template).

Guiding channel

A delineating annotator may be run over a record once, or once per lead. Run per lead it writes twelve independent copies of every fiducial, separated only by the `channel` column, and such a table does not describe beats until it is resolved to a single lead: pooled, it reports twelve times as many beats as the record contains, and taking the first match at each point silently follows whichever lead happens to sort first.

So every function that reads fiducials - `get_windows()`, `learn_template()`, `median_window()` and the other window transforms, `extract_f_waves()`, `vcg()` - takes a `channel`, and resolves it the same way:

- annotations spanning more than one channel with no `channel` given are an error, not a guess;
- a `channel` that the annotations do not carry is an error, which is what catches a numbering convention that does not match;
- the global channel `0` is not a lead, so it never counts toward the span and is always kept alongside the chosen lead.

The channel may be a number or a channel name (`"II"`), resolved against the record's own header. A number is matched literally against the `channel` column, and the numbering there belongs to the annotator that wrote the file. Read it off the record rather than assuming:

```
table(get_annotation(x)$channel)
```

Only the boundaries are restricted to one lead. Windows still carry the signal for every channel, which is what makes a windowed beat usable as a vectorcardiogram.

See Also

`learn_template()`, `template()`, `landmark()`

`wfdb` *Waveform Database (WFDB) Software Package*

Description

This implementation of WFDB is a back-end for the WFDB using a combination of *python*, *C++*, and *C* language. The related functions are documented separately. This serves as an overview of the conversion of WFDB formats to R formats. In this documentation, the specific WFDB generated files will be described.

Arguments

`record`	String that will be used to name the WFDB record. Cannot include extensions, and is not a filepath. alphanumeric characters are acceptable, as well as hyphens (-) and underscores (_)
`record_dir`	File path of directory that should be used read and write files. Defaults to current directory.
`annotator`	Character vector of WFDB-compatible annotation types. When reading, multiple annotators can be specified to read multiple annotation files simultaneously (e.g., `c("ecgpuwave", "atr")`). When writing, must be a single string serving as the file extension. See `read_annotation()` and `write_annotation()` for further details.
`...`	Additional arguments to be passed to the function

WFDB

The WFDB (Waveform Database) Software Package has been developed over the past thirty years, providing a large collection of software for processing and analyzing physiological waveforms. The package is written in highly portable C and can be used on all popular platforms, including GNU/Linux, MacOS X, MS-Windows, and all versions of Unix.

The foundation of the WFDB Software Package is the WFDB library, consisting of a set of functions for reading and writing digitized signals and annotations. These functions can be used by programs written in C, C++, or Fortran, running under any operating system for which an ANSI/ISO C compiler is available, including all versions of Unix, MS-DOS, MS-Windows, the Macintosh OS, and VMS.

Data format

The records that the WFDB uses have three components...

1. Signals: integer values that are at equal intervals at a certain sampling frequency
2. Header attributes: recording information such as sample number, gain, sampling frequency
3. Annotations: information about the record such as a beat labels or alarm triggers

Author(s)

Original software: George Moody, Tom Pollard, Benjamin Moody
R implementation: Anish S. Shah
Last updated: 2026-09-05

---

wfdb_annotation_labels

Standard WFDB annotation nomenclature

Description

Provides the standard label definitions used by WFDB annotation files. These helper functions make it easier to interpret the contents of the `annotation_table()` object by exposing the symbol, mnemonic, and description that correspond to each label store value defined in the WFDB Applications Guide (Moody and collaborators).

Usage

```
wfdb_annotation_labels(symbol = NULL, label_store = NULL)

wfdb_annotation_decode(annotation, column = "type")
```

Arguments

`symbol`	Optional character vector of WFDB annotation symbols to filter the results.
`label_store`	Optional integer vector of WFDB label store values to filter the results.
`annotation`	An `annotation_table()` or compatible data frame whose annotation symbols or label store values should be augmented with the standard WFDB nomenclature.
`column`	Name of the column within `annotation` that contains either the WFDB symbol (default, for the `type` column) or the label store values. If the column is numeric it is matched on `label_store`, otherwise a symbol lookup is performed.

Details

The returned table is derived from the WFDB Application Guide and matches the canonical label store values used by the WFDB software distribution. Entries that are not currently defined by the specification are omitted.

Value

`wfdb_annotation_labels()` returns a data frame with columns `label_store`, `symbol`, `mnemonic`, and `description`. `wfdb_annotation_decode()` returns the input annotation table with the WFDB nomenclature columns appended.

References

Moody GB. *WFDB Applications Guide*. PhysioNet. Available at https://www.physionet.org/physiotools/wag/.

Examples

```
wfdb_annotation_labels()

wfdb_annotation_labels(symbol = c("N", "V"))

ann <- annotation_table(
  annotator = "example",
  sample = c(100L, 200L),
  type = c("N", "V")
)

wfdb_annotation_decode(ann)
```

`wfdb_annotations` *Read WFDB-compatible annotation file*

Description

Individual annotation types are described as both a command-line tool for annotating WFDB-files, as well as the extension that is appended to the `record` name to notate the type. Generally, the types of annotations that are supported are described below:

- atr = manually reviewed and corrected reference annotation files
- ann = general annotator file
- ecgpuwave = files contain surface ECG demarcation (P, QRS, and T waves)
- sqrs/wqrs/gqrs = standard WFDB peak detection for R waves

A more thorough explanation is given in the details. Additionally, files when being read in are converted from a binary format to a textual format. The raw data however may be inadequate, as the original annotation may be erroneous. In these cases, an empty `annotation_table` object will be returned.

Usage

```
read_annotation(
  record,
  annotator,
  record_dir = ".",
  begin = NULL,
  end = NULL,
  header = NULL,
  interval = NULL,
  channel_zero = c("auto", "global", "signal")
)

write_annotation(
  data,
  annotator,
  record,
  record_dir = ".",
  channel_zero = NULL
)
```

Arguments

record	String that will be used to name the WFDB record. Cannot include extensions, and is not a filepath. alphanumeric characters are acceptable, as well as hyphens (-) and underscores (_)
annotator	Character vector of WFDB-compatible annotation types. When reading, multiple annotators can be specified to read multiple annotation files simultaneously (e.g., `c("ecgpuwave", "atr")`). When writing, must be a single string serving as the file extension. See `read_annotation()` and `write_annotation()` for further details.
record_dir	File path of directory that should be used read and write files. Defaults to current directory.
begin, end	Times delimiting a half-open range. Time-only character values such as `"00:00:01.5"` are elapsed from the record start; dated strings and `POSIXt` values are absolute timestamps. A date is required for studies longer than one day. See `validate_time_parameters()` for formats.
header	A header file is an optional named list of parameters that will be used to organize and describe the signal input from the **data** argument. If the **type** is given, specific additional elements will be searched for, such as the low or high pass filters, colors, or other signal attributes. At minimum, the following elements are required (as cannot be calculated): • frequency = sample frequency in Hertz as `integer` • label = vector of names for each channel as `character` • start_time = date/time object
interval	A duration after `begin` that takes precedence over `end`. Numeric values are seconds; character shortcuts such as `"30s"`, `"100 ms"`, and `"2 min"` are supported.

channel_zero — How the annotation file counts its signals. In memory a channel always counts them from 1, as header$number does, with 0 the global channel for a fiducial belonging to no lead; this argument says how the file does. On reading, "auto" (default) takes a column as counting from 1 - which an all-zero column and a 1 .. nsig column both do - and refuses the one column that could count either way, running 0 .. nsig-1 exactly, with a message naming the two declarations. "global" reads every column as counting from 1; "signal" reads it as counting from 0, as the WFDB tools do, and adds one to each channel so the table counts from 1 like every other. The table remembers which, readable with channel_zero(), and write_annotation() uses that by default (NULL) to number the file the way it came in; give "global" or "signal" there to choose. A table holding a global annotation cannot be written under "signal", since the value it would take is the first signal's, and that is an error. See channels.

data — An annotation_table containing the 6 invariant columns required by the annotation_table() function

Value

read_annotation() returns either a single annotation_table object (when one annotator is specified) or a named list of annotation_table objects (when multiple annotators are specified). write_annotation() writes an annotation_table to a WFDB-compatible annotation file.

IMPORTANT: as annotation files are created by annotators that were developed independently, there is a higher chance of an erroneous file being created on disk. As such, this function will note an error and return an empty annotation_table at times.

Annotation files

The following annotation file types are described below.

ecgpuwave**:**

ecgpuwave analyzes an ECG signal from the specified record, detecting the QRS complexes and locating the beginning, peak, and end of the P, QRS, and ST-T waveforms. The output of ecgpuwave is written as a standard WFDB-format annotation file (the extension is "*.ecgpuwave", as would be expected). This file can be converted into text format using rdann. Further details are given at the ECGPUWAVE page.

The **type** column can be *p*, *t*, or *N* for the peak of the P wave, T wave, and QRS (R peak) directly. The output notation also includes waveform onset (and waveform offset). The **number** column gives further information about each of these **type** labels.

The **number** column gives modifier information. If the **type** classifier is a T wave annotation, the **number** column can be 0 (normal), 1 (inverted), 2 (positive), 3 (negative), 4 (biphasic negative-positive), 5 (biphasic positive-negative). If the **type** is an waveform onset or offset, then **number** can be 0 (P wave), 1 (QRS complex), 2 (T wave).

wfdb_io *I/O of WFDB-compatible signal & header files from EP recording systems*

Description

This function allows for WFDB files to be read from any WFDB-compatible system, and also allows writing out WFDB-compatible files from specific EP recording systems, as indicated in the details section. Writing WFDB leads to creation of both a **dat** (signal) and **hea** (header) file. These are both required for reading in files as well.

Usage

```
write_wfdb(
  data,
  record,
  record_dir = ".",
  header = NULL,
  info_strings = list(),
  units = c("digital", "physical"),
  ...
)

read_wfdb(
  record,
  record_dir = ".",
  annotator = NULL,
  begin = NULL,
  end = NULL,
  interval = NULL,
  units = c("digital", "physical"),
  channels = character(),
  channel_zero = c("auto", "global", "signal"),
  ...
)

read_signal(
  record,
  record_dir = ".",
  header = NULL,
  begin = NULL,
  end = NULL,
  interval = NULL,
  units = c("digital", "physical"),
  channels = character(),
  ...
)
```

```
read_header(record, record_dir = ".", ...)
```

Arguments

data	Can either be an EGM object, or a data.frame (or similar) object. The function will appropriately set defaults based on the type. • EGM = Will extract signal and header data directly from object, and thus is simplest to convert to a WFDB format • signal_table = This is a customized data.table class that has an invariant column containing sample information. • data.frame or data.table = Must have a column that represents a time point or index, and columns that represent signal values (preferably integers)
record	String that will be used to name the WFDB record. Cannot include extensions, and is not a filepath. alphanumeric characters are acceptable, as well as hyphens (-) and underscores (_)
record_dir	File path of directory that should be used read and write files. Defaults to current directory.
header	A header file is an optional named list of parameters that will be used to organize and describe the signal input from the **data** argument. If the **type** is given, specific additional elements will be searched for, such as the low or high pass filters, colors, or other signal attributes. At minimum, the following elements are required (as cannot be calculated): • frequency = sample frequency in Hertz as integer • label = vector of names for each channel as character • start_time = date/time object
info_strings	A list of strings that will be written as an appendix to the header file, usually containing information about the channels, (e.g. list of colors, extra labels, etc).
units	A character string representing either *digital* (DEFAULT) or *physical* units that should be used for signal values. • "digital" = Returns raw ADC (analog-to-digital converter) counts as stored in the .dat file. These are integer values representing the digitized signal without any scaling applied. Use this to preserve exact round-trip fidelity. • "physical" = Returns signal values converted to physical units (e.g., mV) using the formula: physical = (digital - baseline) / gain, where baseline and gain are specified in the header file. This is the human-readable format for analysis. The two differ by the ADC gain, which is commonly 200, so reading digital counts as though they were millivolts is a 200-fold error - and one that no downstream check can catch on its own, since the same number is plausible in either. The signal table returned by read_wfdb() and read_signal() therefore carries its units as a label, readable with signal_units() and preserved through windowing, padding, medians and resampling. write_wfdb() refuses to write a signal whose label contradicts the units it was told to expect.

...	Additional arguments to be passed to the function
annotator	Character vector of WFDB-compatible annotation types. When reading, multiple annotators can be specified to read multiple annotation files simultaneously (e.g., c("ecgpuwave", "atr")). When writing, must be a single string serving as the file extension. See read_annotation() and write_annotation() for further details.
begin, end	Times delimiting a half-open range. Time-only character values such as "00:00:01.5" are elapsed from the record start; dated strings and POSIXt values are absolute timestamps. A date is required for studies longer than one day. See validate_time_parameters() for formats.
interval	A duration after begin that takes precedence over end. Numeric values are seconds; character shortcuts such as "30s", "100 ms", and "2 min" are supported.
channels	Either the signal/channel in a character vector as a name or number. Allows for duplication of signal or to re-order signal if needed. If nothing is given, will default to all channels available.
channel_zero	How the annotation file counts its signals. In memory a channel always counts them from 1, as header$number does, with 0 the global channel for a fiducial belonging to no lead; this argument says how the file does. On reading, "auto" (default) takes a column as counting from 1 - which an all-zero column and a 1 .. nsig column both do - and refuses the one column that could count either way, running 0 .. nsig-1 exactly, with a message naming the two declarations. "global" reads every column as counting from 1; "signal" reads it as counting from 0, as the WFDB tools do, and adds one to each channel so the table counts from 1 like every other. The table remembers which, readable with channel_zero(), and write_annotation() uses that by default (NULL) to number the file the way it came in; give "global" or "signal" there to choose. A table holding a global annotation cannot be written under "signal", since the value it would take is the first signal's, and that is an error. See channels.

Details

The begin, end, and interval arguments are validated as time values and converted into sample positions using the sampling frequency declared in the WFDB header. interval takes precedence over end. The resulting range includes begin and excludes end; requests beyond the recorded duration are clamped to the final timestamp.

Value

Depends on if it is a reading or writing function. For writing, will output an WFDB-based object reflecting the function. For reading, will output an extension of a data.table object reflecting the underlying function (e.g. signal_table() will return an object of class).

Functions

- write_wfdb(): Writes out signal and header data into a WFDB-compatible format from R. The units parameter indicates whether the input signal data is in digital (raw ADC counts) or physical units. When units="physical", the function automatically converts to digital units using the inverse formula: digital = (physical * gain) + baseline before writing to disk.

- `read_wfdb()`: Reads a multicomponent WFDB-formatted set of files directly into an `EGM` object. This serves to pull together `read_signal()`, `read_header()`, and `read_annotation()` for simplicity.
- `read_signal()`: Specifically reads the signal data from the WFDB binary format, returning a `signal_table` object for evaluation in the R environment
- `read_header()`: Specifically reads the header data from the WFDB header text format, returning a `header_table` object for evaluation in the R environment

Recording systems

Type of signal data, as specified by the recording system, that are currently supported.

- *bard* = Bard (LabSystem Pro), e.g. `read_bard()`
- *muse* = MUSE (GE), e.g. `read_muse()`
- *prucka* = Prucka (CardioLab), e.g. `read_prucka()`

`window_dropped` *Candidate beats a windowing strategy did not return*

Description

[Experimental]

How many candidate beats an analysis found but did not use, and why. The counts are recorded on the result rather than reported to the console, because a message is invisible on a background worker and the drop rate across a study is exactly what an audit needs.

Usage

```
window_dropped(x)
```

Arguments

`x`	A windows collection, or a `vcg()` result.

Details

The reasons are specific to whatever did the dropping. `by_beat()` reports `incomplete_span`, beats lying too near either end of the record for the full span to be cut. `by_rhythm()` reports `no_offset` (an onset with no matching offset before the record ends), `no_reference` (no reference fiducial inside the window), and `overlapping` (a second onset inside the window, which for sinus means the beat was not clean). `vcg()` reports `incomplete_span` from its own windowing and `no_delineation` for beats the annotator did not mark the wave in.

Counts are of candidate onsets, so `length(x) + sum(window_dropped(x))` is the number of candidates the strategy considered.

Value

A named integer vector of counts, one per reason; empty for an object that carries no drop accounting.

See Also

get_windows(), vcg()

Examples

```
## Not run:
beats <- get_windows(ecg, by = by_beat(channel = 2))
window_dropped(beats)

window_dropped(vcg(ecg, channel = 2))

## End(Not run)
```

---

window_strategy *A window extraction strategy*

Description

[Experimental]

Describes *how* windows should be found within a record, without performing the search. Strategy objects are values: build one once and reuse it across every record in a study. Pass one to get_windows().

Strategies are created by the by_*() constructors, which validate their own arguments; this class only carries the resolved result. by_rhythm() follows the signal and returns windows of differing length; by_beat() cuts a fixed span around a fiducial and returns windows that are all the same length.

Usage

```
window_strategy(method = character(0), params = list())
```

Arguments

method	The registry key naming the extraction engine, e.g. "rhythm".
params	A named list of resolved, strategy-specific parameters.

Value

A window_strategy S7 object.

See Also

by_rhythm(), by_beat(), get_windows()

Index

∗ **datasets**
    kors, 59
[.windows, 3

add_annotation, 4
add_annotation(), *7*
analyze_atrial_signal, 5
annotation_table, 6
annotation_table(), *35*, *38*, *60*, *87*, *90*
annotators, 8
as_ECG, 9
as_ECG(), *35–37*, *39*, *80*, *83*

bard, 10
base::strptime(), *56*
baseline_window, 12
baseline_window(), *66*, *73, 74*
by_beat, 14
by_beat(), *53, 54*, *66*, *71*, *74*, *81*, *94, 95*
by_pwave, 16
by_pwave(), *19, 20*, *53, 54*, *60*
by_rhythm, 18
by_rhythm(), *14–17*, *53, 54*, *60*, *94, 95*

c.windows, 21
c.windows(), *50*
calculate_approximate_entropy, 21
calculate_approximate_entropy(), *25*
calculate_dominant_frequency, 22
calculate_dominant_frequency(), *46*
calculate_organization_index, 23
calculate_organization_index(), *46, 47*
calculate_sample_entropy, 25
calculate_sample_entropy(), *22*, *46*
calculate_welch_spectrum, 26
calculate_welch_spectrum(), *22*
cancel_ventricular_signal, 27
cancel_ventricular_signal(), *42*, *46*
change_frequency, 29
change_frequency(), *49–51*, *54*, *70*
channel_zero, 31
channel_zero(), *33*, *90*, *93*
channels, *4*, *7–9*, 32, *32*, *90*, *93*

detect_QRS, 34

ECG, 35, *37*, *39*, *50*, *80*
ECG(), *10*
ecg_leads, 36
ecg_leads(), *9, 10*, *36*
EGM, *35*, 38
EGM(), *36*, *38*
extract_f_waves, 39
extract_f_waves(), *6*, *10*, *13*, *15*, *17*, *20*, *25*, *28*, *32*, *34, 35*, *45–47*, *54*, *63*, *66*, *70*, *73*, *82, 83*, *85*

f_wave_diagnostics, *43*, 46
feature_range, 48
format.windows, 49
frequency, 49
frequency(), *30, 31*

get_annotation, 51
get_annotation(), *30*, *33*, *66*
get_signal, 52
get_windows, 53
get_windows(), *13*, *15–17*, *20*, *32, 33*, *45*, *54*, *60*, *63*, *65, 66*, *68–70*, *73*, *82, 83*, *85*, *95*

header_table, 55
header_table(), *35*, *38*

is_annotation_table *(*annotation_table*)*, 6
is_ECG *(*ECG*)*, 35
is_EGM *(*EGM*)*, 38
is_EGM(), *38*
is_header_table *(*header_table*)*, 55
is_signal_table *(*signal_table*)*, 76

is_template, 57
is_window_set, 58
is_window_strategy, 58

kors, 59, *80*, *83*

label_waves, 60
label_waves(), *9*, *19*, *20*, *33*
landmark, 61, *62*, *78*
landmark(), *86*
lead_factor *(*ecg_leads*)*, 36
lead_factor(), *9*, *10*, *36*
learn_template, 62
learn_template(), *13*, *15*, *17*, *20*, *32*, *45*, *54*, *61*, *63*, *66*, *70*, *73*, *78*, *82*, *84–86*
list_annotators, 63
list_annotators(), *51*

map_windows, 64
map_windows(), *81*
median_window, 65
median_window(), *13–17*, *20*, *29*, *32*, *45*, *54*, *63*, *66*, *70*, *72–74*, *80–83*, *85*
merge_annotations, 67
muse, 67

new_windows, 68
normalize_window, 69
normalize_window(), *29*, *31*, *53*, *54*, *65*, *74*

pad_window, 71
pad_window(), *14*, *53*, *54*, *65*, *66*, *70*
print.windows, 74
prucka, 75

read_annotation *(*wfdb_annotations*)*, 88
read_annotation(), *7–9*, *32*, *51*, *86*, *89*, *90*, *93*, *94*
read_bard *(*bard*)*, 10
read_bard(), *38*, *94*
read_bard_header *(*bard*)*, 10
read_bard_signal *(*bard*)*, 10
read_header *(*wfdb_io*)*, 91
read_header(), *94*
read_muse *(*muse*)*, 67
read_muse(), *94*
read_prucka *(*prucka*)*, 75
read_prucka(), *94*
read_prucka_header *(*prucka*)*, 75
read_prucka_signal *(*prucka*)*, 75
read_signal *(*wfdb_io*)*, 91
read_signal(), *92*, *94*
read_wfdb *(*wfdb_io*)*, 91
read_wfdb(), *32*, *38*, *66*, *77*, *82*, *92*

signal::resample(), *30*
signal_table, 76
signal_table(), *35*, *38*, *77*
signal_units, 77
signal_units(), *77*, *82*, *92*
stats::frequency(), *50*
stats::median(), *65*

template, *62*, *63*, 78, *84*
template(), *62*, *84*, *86*

validate_time_parameters, 79
validate_time_parameters(), *89*, *93*
vcg, 80
vcg(), *10*, *13–15*, *17*, *20*, *32*, *35*, *45*, *54*, *59*, *63*, *66*, *70*, *73*, *82*, *85*, *94*, *95*

warp_window, 84
warp_window(), *53*, *54*, *70*, *78*
wfdb, 86
wfdb_annotation_decode *(*wfdb_annotation_labels*)*, 87
wfdb_annotation_labels, 87
wfdb_annotations, 88
wfdb_io, 91
window_dropped, 94
window_dropped(), *15*, *16*, *54*, *69*, *81*
window_strategy, *14–19*, *53*, 95
windows, *53*, *94*
write_annotation *(*wfdb_annotations*)*, 88
write_annotation(), *7*, *8*, *32*, *33*, *86*, *89*, *90*, *93*
write_wfdb *(*wfdb_io*)*, 91
write_wfdb(), *56*, *77*, *92*