# Package: detectPVC (via r-universe)

June 6, 2026

**Version** 0.3.2

**Date** 2024-10-12

**Title** Detect Premature Venticular Complexes

**Description** Detect premature ventricular complexes (PVCs) in data from a Polar H10 chest-strap heart rate sensor.

**Author** Karl W Broman [aut, cre] (<https://orcid.org/0000-0002-4914-6671>)

**Maintainer** Karl W Broman <broman@wisc.edu>

**Depends** R (>= 3.5.0)

**Imports** lubridate, stringr, broman (>= 0.84), graphics, parallel, utils, stats, signal, R.utils

**Suggests** testthat, devtools, roxygen2, knitr, rmarkdown

**License** MIT + file LICENSE

**URL** https://github.com/kbroman/detectPVC

**BugReports** https://github.com/kbroman/detectPVC/issues

**LazyData** true

**Encoding** UTF-8

**ByteCompile** true

**VignetteBuilder** knitr

**Roxygen** list(markdown=TRUE)

**Config/roxygen2/version** 8.0.0

**Config/pak/sysreqs** libicu-dev

**Repository** https://kbroman.r-universe.dev

**Date/Publication** 2026-06-06 11:49:38 UTC

**RemoteUrl** https://github.com/kbroman/detectPVC

**RemoteRef** HEAD

**RemoteSha** 1d263bdd215c53a20755b833704adc3a186c3448

# Contents



---

`adjust_peaks` *Adjust R peaks to hit local max*

---

## Description

Adjust the estimated R peaks from `rsleep::detect_rpeaks()` to hit the local maximum.

## Usage

```
adjust_peaks(peaks, signal, window = 10)
```

## Arguments

| | |
|---|---|
| `peaks` | Estimated R peaks (as index) |
| `signal` | ECG signal vector |
| `window` | Window (+/-) to look for the local peak |

## Value

Vector of peaks, adjusted to hit the local max

## See Also

detect_peaks()

## Examples

```
data(polar_h10)
bad_segs <- find_bad_segments(polar_h10$time, polar_h10$ecg)
peaks <- detect_peaks(polar_h10$time, polar_h10$ecg, adjust=FALSE, omit_segments=bad_segs)
peaks_adj <- adjust_peaks(peaks, polar_h10$ecg)
```

---

# calc_peak_stats *Calculate summary statistics for each R peak*

---

## Description

Calculate summary statistics for each R peak from ECG data, to help characterize which are normal and which are PVCs

## Usage

```
calc_peak_stats(
  times,
  signal,
  peaks,
  window = 10,
  qtmax = 60,
  rsmax = 20,
  omit_segments = NULL
)
```

## Arguments

| | |
|---|---|
| times | Vector of times for ECG signals |
| signal | Vector of ECG signal |
| peaks | Location of R peaks as numeric indexes (from detect_peaks() with return_index=TRUE) |
| window | Tight window around each peak to look for local max and min |
| qtmax | Maximum width of QT interval to consider |
| rsmax | Maximum width of RS interval to consider |
| omit_segments | Segments that were omitted: a data frame with two columns: the start and end index for each interval |

## Value

A data.frame with local max and min of ECG around peak, height of T (local max before the next peak), RR intervals to left and right of this R peak (in seconds), the ratio leftRR/rightRR, and the time from the R peak to the S trough (in milliseconds).

## Examples

```
data(polar_h10)
bad_segs <- find_bad_segments(polar_h10$time, polar_h10$ecg)
peaks <- detect_peaks(polar_h10$time, polar_h10$ecg, omit_segments=bad_segs)
peak_stats <- calc_peak_stats(polar_h10$time, polar_h10$ecg, peaks)
```

---

`convert_timestamp` *Convert date/time stamp for Polar H10*

---

## Description

Convert the date/time stamp in the Polar H10 data to two columns: date/time and fractional seconds

## Usage

```
convert_timestamp(timestamp, tz = Sys.timezone())
```

## Arguments

| | |
|---|---|
| `timestamp` | A vector of "epoch"-based integers as returned in the Polar H10 data. Alternatively, a vector of character strings in format like "2024-05-02 08:30:00" or just "2024-05-02 08:30" |
| `tz` | Timezone |

## Value

A vector of Date objects

## Examples

```
data(polar_h10)
polar_h10$datetime <- convert_timestamp(polar_h10$time)
```

---

`detect_peaks` *Detect R peaks in a raw ECG signal, using a sliding window*

---

## Description

Detect R peaks in a raw ECG signal, using a modified version of `rsleep::detect_rpeaks()` but using a sliding window.

## Usage

```
detect_peaks(
  time,
  signal,
  window = 80000,
  pad = window/4,
  sRate = 1000000000/7682304,
  peak_limit = 1.5,
  max_dist = 60/220,
  omit_segments = NULL,
  ...,
  return_index = TRUE,
  adjust = TRUE,
  cores = 1
)
```

## Arguments

| | |
|---|---|
| `time` | Vector of times at which ECG signal was measured (integers as from Polar H10, datetimes, or character strings) |
| `signal` | Numerical vector of ECG signal |
| `window` | Integer indicating the number of values to consider at one time |
| `pad` | Integer indicating the number of values to consider on each side, surrounding the window |
| `sRate` | ECG signal rate |
| `peak_limit` | Limit factor on size of peaks relative to mean convolved signal. Larger values will ignore low-valued peaks. |
| `max_dist` | Maximum distance allowed between adjacent peaks |
| `omit_segments` | Segments to be omitted: a data frame with two columns: the start and end index for each interval |
| `...` | Passed to a modified version of `rsleep::detect_rpeaks()` |
| `return_index` | If TRUE, the index for each R peak is returned instead of the timing |
| `adjust` | If TRUE, adjust the results to hit the nearby local maxima |
| `cores` | Number of CPU cores to use, for parallel calculations. (If `0`, use `parallel::detectCores()`.) Alternatively, this can be links to a set of cluster sockets, as produced by `parallel::makeCluster()`. |

## See Also

`adjust_peaks()`

## Examples

```
data(polar_h10)
bad_segs <- find_bad_segments(polar_h10$time, polar_h10$ecg)
peaks <- detect_peaks(polar_h10$time, polar_h10$ecg, omit_segments=bad_segs)
```

`find_bad_segments` *Find bad segments in ECG signal*

## Description

Find particularly noisy regions in ECG signal, with large values far from 0, plus also regions with excessive missing data.

## Usage

```
find_bad_segments(
  time,
  signal,
  absval_thresh = 2,
  runmean_thresh = 0.7,
  missing_thresh = 0.1,
  window = 0.2,
  min_gap = 2000,
  pad = 400,
  return_index = TRUE,
  tz = Sys.timezone(),
  sRate = 1000000000/7682304
)
```

## Arguments

| | |
|---|---|
| `time` | Vector of times (integers as from Polar H10, datetimes, or character strings) |
| `signal` | Numeric vector of ECG signal |
| `absval_thresh` | Threshold on absolute value of signal |
| `runmean_thresh` | Threshold on running mean of absolute value of signal |
| `missing_thresh` | Maximum amount of missing data within 1 second (as a proportion) |
| `window` | Window for running mean (in seconds) |
| `min_gap` | Minimum gap between bad segments (otherwise merged) |
| `pad` | Pad the bad segments by this many values on each end (in index values) |
| `return_index` | If TRUE, return indexes to `time`; otherwise return actual date/times. |
| `tz` | Timezone used by `convert_timestamp()` |
| `sRate` | Signal rate, as expected number of data points per second; needed if `return_prop=TRUE`. |

## Value

Data frame with two columns: the start and end index for each identified bad segment.

## Examples

```
data(polar_h10)
bad_segs <- find_bad_segments(polar_h10$time, polar_h10$ecg)
```

## find_closest_time *Find closest time*

### Description

Find the closest time in a vector of values to a specified target time

### Usage

```
find_closest_time(target, times, tz = Sys.timezone())
```

### Arguments

| | |
|---|---|
| target | The target time, for which we want to find the closest value |
| times | A vector of times |
| tz | Time zone |

### Value

Numeric index of the closest time

### Examples

```
times <- seq(convert_timestamp("2024-05-01 11:00"),
             convert_timestamp("2024-05-01 14:00"), length=300)
find_closest_time("2024-05-01 12:00", times)
```

## find_pvc_pattern *Find PVC patterns*

### Description

In a sequence of ECG peaks classified as PVC or not, find patterns like bigeminy, trigeminy, couplets and triplets.

### Usage

```
find_pvc_pattern(
  times,
  peaks,
  pvc,
  omit_segments = NULL,
  pattern,
  min_length = 0,
  return_index = TRUE,
  tz = Sys.timezone()
)
```

## Arguments

| | |
|---|---|
| times | Vector of times (integers as from Polar H10, datetimes, or character strings) |
| peaks | Vector of peak indices as integers from 1 to length(times) |
| pvc | Vector of boolean indicators of whether the peaks are PVC or not. Should be the same length as peaks. |
| omit_segments | Segments to be ignored in analysis, as a data frame with two columns: the start and end index for each interval to be ignored |
| pattern | The character string with the regular expression to look for, with N=normal, P=PVC, so for example "(NNP)+" for trigeminy. |
| min_length | Minimum length (as number of beats) for a pattern instance. |
| return_index | If TRUE, return indexes to times, not the actual times. |
| tz | Time zone for converting time stamps |

## Value

Data frame with start and end times + length of match (as number of peaks)

## See Also

plot_states()

## Examples

```
data(polar_h10)
bad_segs <- find_bad_segments(polar_h10$time, polar_h10$ecg)
peaks <- detect_peaks(polar_h10$time, polar_h10$ecg, omit_segments=bad_segs)
peak_stats <- calc_peak_stats(polar_h10$time, polar_h10$ecg, peaks, omit_segments=bad_segs)
pvc <- (peak_stats$RStime > 50)

trigeminy <- find_pvc_pattern(polar_h10$time, peaks, pvc, omit_segments=bad_segs,
                              pattern="(NNP)+")
bigeminy <- find_pvc_pattern(polar_h10$time, peaks, pvc, omit_segments=bad_segs,
                             pattern="(NP)+", min_length=4)
couplets <- find_pvc_pattern(polar_h10$time, peaks, pvc, omit_segments=bad_segs,
                             pattern="PP+")
triplets <- find_pvc_pattern(polar_h10$time, peaks, pvc, omit_segments=bad_segs,
                             pattern="PP+", min_length=3)
normal <- find_pvc_pattern(polar_h10$time, peaks, pvc, omit_segments=bad_segs,
                           pattern="N+", min_length=10)
```

`get_time_interval` *Get indexes for a interval in time*

## Description

For a vector of times, get the indexes that correspond to a specified interval

## Usage

```
get_time_interval(
  times,
  start = NULL,
  end = NULL,
  length = NULL,
  tz = Sys.timezone()
)
```

## Arguments

| | |
|---|---|
| `times` | A vector of times, either as integers (Polar H10 times in 1e-9 seconds), as date-times, or as character strings of the form "2024-05-02 08:56" or "2024-05-02 08:56:00" |
| `start` | Start time of interval |
| `end` | End time of interval |
| `length` | Length of interval (in seconds) |
| `tz` | Time zone |

## Details

Two of `start`, `end`, and `length`.

## Value

A vector of numeric indexes, for values in `times` that are within the specified interval.

## See Also

`convert_timestamp()`

## Examples

```
data(polar_h10)
time_int1 <- get_time_interval(polar_h10$time, start="2024-05-05 09:52:00",
                               length=4)
time_int2 <- get_time_interval(polar_h10$time, start="2024-05-05 09:52:00",
                               end="2024-05-05 09:54:20")
```

## most_common_date *Most common date in a vector of date/times*

### Description

Return the most common date in a vector of date/times

### Usage

```
most_common_date(times)
```

### Arguments

| | |
|---|---|
| times | Vector of date/times |

### Value

The most common data, as a character string in ISO format like 2024-05-26

### See Also

convert_timestamp()

## plot.pvc_stats *Plot running PVC statistics*

### Description

Plot running PVC statistics from running_pvc_stats()

### Usage

```
## S3 method for class 'pvc_stats'
plot(
  x,
  ylab_pvc = "Percent PVC",
  ylab_hr = "Heart rate (BPM)",
  ylim_pvc = c(0, max(x$pvc, na.rm = TRUE) * 1.02),
  ylim_hr = c(range(x$hr, na.rm = TRUE)) * c(0.98, 1.02),
  ...
)
```

### Arguments

| | |
|---|---|
| x | Data frame of running PVC stats from running_pvc_stats() |
| ylab_pvc | Y-axis label for the plot of percent PVC |
| ylab_hr | Y-axis label for the plot of heart rate |
| ylim_pvc | Y-axis limits for the plot of percent PVC |
| ylim_hr | Y-axis limits for the plot of heart rate |
| ... | Passed to broman::grayplot() |

### Details

Makes a plot with two panels, of percent PVC vs time and heart rate vs time.

### Value

None.

### Examples

```
data(polar_h10)
bad_segs <- find_bad_segments(polar_h10$time, polar_h10$ecg)
peaks <- detect_peaks(polar_h10$time, polar_h10$ecg, omit_segments=bad_segs)
peak_stats <- calc_peak_stats(polar_h10$time, polar_h10$ecg, peaks, omit_segments=bad_segs)
pvc <- (peak_stats$RStime > 50)

pvc_stats <- running_pvc_stats(polar_h10$time, peaks, pvc, omit_segments=bad_segs,
                               window=60, n_at=100)
plot(pvc_stats)
```

---

## plot_ecg *Plot ECG signal*

---

### Description

Plot of ECG signal approximately matching traditional ECG graphs

### Usage

```
plot_ecg(
  times,
  signal,
  vlines.col = "gray70",
  vlines.minor.col = "gray80",
  bgcolor = "gray98",
  tz = Sys.timezone(),
  ...
)
```

### Arguments

| | |
|---|---|
| `times` | Vector of times (integers as from Polar H10, datetimes, or character strings) |
| `signal` | ECG signal. If missing or NULL, we take `times` as the signal and then fill in with times assuming Polar H10 signal rate. |
| `vlines.col` | Color of vertical grid lines at seconds |
| `vlines.minor.col` | Color of vertical grid lines at 1/5 seconds |
| `bgcolor` | Background color |
| `tz` | Timezone (in converting times) |
| `...` | Passed to `broman::grayplot()` |

### Value

None.

### Examples

```
data(polar_h10)
v <- 1:(30*130)
plot_ecg(polar_h10$ecg[v])
plot_ecg(polar_h10$time[v], polar_h10$ecg[v])
```

---

## `plot_ecg_mult` *Plot ECG signal across multiple panels*

---

### Description

Plot a longer window of ECG signals across multiple panels arranged in one column.

### Usage

```
plot_ecg_mult(
  times,
  signal,
  start = NULL,
  length = 30,
  n_panel = 4,
  peaks = NULL,
  pvc = NULL,
  hilit_segments = NULL,
  col_peak = c("slateblue", "violetred"),
  pch_peak = 16,
  cex_peak = 1,
  bg_peak = c("slateblue", "violetred"),
  col_segments = "#cc00dd33",
  tz = Sys.timezone(),
  ...
)
```

## Arguments

| | |
|---|---|
| times | Vector of times (integers as from Polar H10, datetimes, or character strings) |
| signal | ECG signal. If missing or NULL, we take times as the signal and then fill in with times assuming Polar H10 signal rate. |
| start | Start time for beginning of first panel, as integer as from Polar H10, datetimes, or character strings like "2024-05-03 13:27" or "2024-05-03 13:27:00" |
| length | Length of time for each panel in seconds |
| n_panel | Number of panels. |
| peaks | Optional vector of peak indices; if provided, dots will be plotted at each peak location |
| pvc | Optional vector of boolean indicators of whether the peaks are PVC or not. If provided, should be the same length as peaks and will be used to color the points at the peaks. |
| hilit_segments | Optional matrix indicating segments to highlight, as returned from find_bad_segments(), with rows corresponding to intervals and two columns of numeric indexes of the start and end of each interval |
| col_peak | Vector of two colors to use to color the dots that indicate non-PVC and PVC peaks, respectively |
| pch_peak | Vector of one or two point types to plot at peaks. |
| cex_peak | Vector of one or two values to indicate size of points that indicate non-PVC and PVC peaks, respectively |
| bg_peak | Vector of one or two colors to use for background color for the dots that indicate non-PVC and PVC peaks, respectively. |
| col_segments | Color to highlight the segments in hilit_segments |
| tz | Timezone (in converting times) |
| ... | Passed plot_ecg() |

## Details

We use par(mfrow=c(n_panel, 1)) before creating the set of panels. However, in the case n_panel==1, we don't run par(mfrow) and so this can be used in a more general way.

## Value

None.

## Examples

```
data(polar_h10)
plot_ecg_mult(polar_h10$time, polar_h10$ecg, start="2024-05-05 09:52", length=20, n_panel=2)

bad_segs <- find_bad_segments(polar_h10$time, polar_h10$ecg)
peaks <- detect_peaks(polar_h10$time, polar_h10$ecg, omit_segments=bad_segs)
peak_stats <- calc_peak_stats(polar_h10$time, polar_h10$ecg, peaks, omit_segments=bad_segs)
pvc <- (peak_stats$RStime > 50)
plot_ecg_mult(polar_h10$time, polar_h10$ecg, start="2024-05-05 09:50:30", length=20, n_panel=2,
              peaks=peaks, pvc=pvc, hilit_segments=bad_segs)
```

plot_states *Plot PVC states*

## Description

In a sequence of ECG peaks classified as PVC or not, find patterns normal, bigeminy, trigeminy, and make a plot of these

## Usage

```
plot_states(
  times,
  peaks,
  pvc,
  omit_segments = NULL,
  min_length = 12,
  rect_col = c(blue = "#0074d9", orange = "#ff851b", green = "#2ecc40", purple =
    "#cc00dd"),
  tz = Sys.timezone(),
  draw = TRUE,
  ...
)

## S3 method for class 'states'
plot(
  x,
  peaks,
  pvc,
  omit_segments = NULL,
  min_length = 12,
  rect_col = c(blue = "#0074d9", orange = "#ff851b", green = "#2ecc40", purple =
    "#cc00dd"),
  tz = Sys.timezone(),
  draw = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| times | Vector of times (integers as from Polar H10, datetimes, or character strings) (Can also be on object of class "states", that is the output of the present function, in which case the arguments peaks, pvc, omit_segments, and min_length, and tz are ignored. |
| peaks | Vector of peak indices as integers from 1 to length(times) |
| pvc | Vector of boolean indicators of whether the peaks are PVC or not. Should be the same length as peaks. |

| | |
|---|---|
| omit_segments | Segments to be ignored in analysis, as a data frame with two columns: the start and end index for each interval to be ignored |
| min_length | Minimum length (as number of beats) for a pattern instance. |
| rect_col | Vector of four rectangle colors: normal, bigeminy, trigeminy, omitted |
| tz | Time zone for converting time stamps |
| draw | If FALSE, don't actually make the plot |
| ... | Passed to base::plot() |
| x | Object of class "states", that is the output of the present function |

## Value

Data frame with start and end times and type of state (normal, bigeminy, trigeminy, omitted). Given class "states"

## See Also

find_pvc_pattern(), summary_states()

## Examples

```
data(polar_h10)
bad_segs <- find_bad_segments(polar_h10$time, polar_h10$ecg)
peaks <- detect_peaks(polar_h10$time, polar_h10$ecg, omit_segments=bad_segs)
peak_stats <- calc_peak_stats(polar_h10$time, polar_h10$ecg, peaks, omit_segments=bad_segs)
pvc <- (peak_stats$RStime > 50)

plot_states(polar_h10$time, peaks, pvc, omit_segments=bad_segs, xlim=c(50, 60))
```

---

# polar_h10 *Example Polar H10 data*

---

## Description

Example data from a Polar H10 chest-strap heart rate sensor

## Usage

```
data(polar_h10)
```

## Format

A data frame with four columns: time (an epoch-based integer time stamp in 1e-9 seconds), ecg (the ECG values), hr (estimated heart rate in beats per minute), and rr (estimated R-R interval in milliseconds).

There is about 10 minutes of data at about 130 Hz.

## Source

Personal data derived from a Polar H10 monitor and extracted using the ECGLogger app.

## Examples

```
data(polar_h10)
```

---

`print.summary.states` *Print summary of states object*

---

## Description

Print summary of states object

## Usage

```
## S3 method for class 'summary.states'
print(x, digits = 1, ...)
```

## Arguments

| | |
|---|---|
| `x` | Object of class `"summary.states"`, as produced by `summary_states()`. |
| `digits` | Number of digits in printing; passed to `base::print()` |
| `...` | Ignored |

## Value

Invisibly returns the input, `x`.

## See Also

`plot_states()`, `summary_states()`

## Examples

```
data(polar_h10)
bad_segs <- find_bad_segments(polar_h10$time, polar_h10$ecg)
peaks <- detect_peaks(polar_h10$time, polar_h10$ecg, omit_segments=bad_segs)
peak_stats <- calc_peak_stats(polar_h10$time, polar_h10$ecg, peaks, omit_segments=bad_segs)
pvc <- (peak_stats$RStime > 50)

st <- plot_states(polar_h10$time, peaks, pvc, omit_segments=bad_segs, xlim=c(50, 60), draw=FALSE)
summary(st)
```

`read_multcsv` *Read multiple related CSV files*

## Description

Read multiple CSV files all with the same columns and rbind them together

## Usage

```
read_multcsv(
  dir = ".",
  files = NULL,
  dont_modify = FALSE,
  omit_incomplete = TRUE,
  tz = Sys.timezone(),
  cores = 1
)
```

## Arguments

| | |
|---|---|
| `dir` | Subdirectory containing the files. |
| `files` | Optional character vector of file names. If not provided, we use `list.files(dir)` to grab all CSV files in the directory `dir`. |
| `dont_modify` | If TRUE, just rbind the file contents together. If FALSE, look for a `time` column and reorder the rows by that column, and then add a converted `datetime` column using `convert_timestamp()`. |
| `omit_incomplete` | If TRUE, omit any rows where the time stamp looks to be incomplete (having < 19 characters) |
| `tz` | Time zone, used if `dont_modify=FALSE` and there is a time column to convert. |
| `cores` | Number of CPU cores to use, for parallel calculations. (If `0`, use `parallel::detectCores()`.) Alternatively, this can be links to a set of cluster sockets, as produced by `parallel::makeCluster()`. |

## Details

At least one of `files` or `dir` must be provided. The files should all have the same set of columns.

Files can be gzipped: If a file has name like 'file.csv.gz' it is gunzipped to a temporary directory and then read.

If `dont_modify=FALSE` and there is a column `time`, the rows are reordered using this column and a `datetime` column is added, converting `time` with `convert_timestamp()`

## Value

A data.frame with the contents of all files row-binded together.

running_datacount *Get running count of amount of data*

## Description

Get running sum of number of observed data points in sliding windows

## Usage

```
running_datacount(
  times,
  window = 1,
  at = NULL,
  n_at = NULL,
  tz = Sys.timezone(),
  return_prop = FALSE,
  sRate = 1000000000/7682304
)
```

## Arguments

| | |
|---|---|
| times | Vector of times |
| window | Length of sliding window in seconds |
| at | Times at which to calculate the amount of data |
| n_at | Number of times at which to calculate the amount of data |
| tz | Time zone, to convert times using convert_timestamp() |
| return_prop | If true, return the expected proportion of observed data points within each window. Values will be slightly <1 even with complete data, due to discrete nature of time points. |
| sRate | Signal rate, as expected number of data points per second; needed if return_prop=TRUE |

## Details

If n_at and at are both missing, we calculate amount of data at times. If at is missing but n_at is provided, we use n_at equally-spaced times across the observed range.

## Value

Vector of same length as times, with the number of data points in a sliding window centered at each time point.

## Examples

```
data(polar_h10)
polar_h10$datetime <- convert_timestamp(polar_h10$time)
n_pts <- running_datacount(polar_h10$datetime)

broman::timeplot(polar_h10$datetime, n_pts,
                 xlab="Time", ylab="No. data points")
```

---

`running_pvc_stats` *Calculate running PVC burden and HR*

---

## Description

Calculate running PVC burden percent and heart rate in sliding windows

## Usage

```
running_pvc_stats(
  times,
  peaks,
  pvc,
  omit_segments = NULL,
  window = 480,
  at = NULL,
  n_at = 240,
  cores = 1,
  tz = Sys.timezone()
)
```

## Arguments

| | |
|---|---|
| `times` | Vector of times (integers as from Polar H10, datetimes, or character strings) |
| `peaks` | Vector of peak indices as integers from 1 to `length(times)` |
| `pvc` | Vector of boolean indicators of whether the peaks are PVC or not. Should be the same length as `peaks`. |
| `omit_segments` | Segments to be ignored in analysis, as a data frame with two columns: the start and end index for each interval to be ignored |
| `window` | Window in seconds to calculate the running statistics |
| `at` | Times at which to calculate the running statistics. |
| `n_at` | If `at` is not provided, we use this number of equally-spaced values across the range of `times`. |
| `cores` | Number of CPU cores to use, for parallel calculations. (If `0`, use `parallel::detectCores()`.) Alternatively, this can be links to a set of cluster sockets, as produced by `parallel::makeCluster()`. |
| `tz` | Timezone (in converting times) |

## Value

Data frame with window centers, PVC percent, heart rate (in beats per minute), and window length (in seconds). The window centers may be different from `at` in cases where windows is missing data.

## Examples

```
data(polar_h10)
bad_segs <- find_bad_segments(polar_h10$time, polar_h10$ecg)
peaks <- detect_peaks(polar_h10$time, polar_h10$ecg, omit_segments=bad_segs)
peak_stats <- calc_peak_stats(polar_h10$time, polar_h10$ecg, peaks, omit_segments=bad_segs)
pvc <- (peak_stats$RStime > 50)

pvc_stats <- running_pvc_stats(polar_h10$time, peaks, pvc, window=60, n_at=15,
                               omit_segments=bad_segs)
plot(pvc_stats)
```

---

# summary_states *Summarize output of plot_states*

---

## Description

Summarize output of plot_states, with amount of time in different PVC states

## Usage

```
summary_states(object)

## S3 method for class 'states'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| `object` | An object of class `"states"`, as output by `plot_states()` |
| `...` | Ignored |

## Value

A data frame with the different possible PVC states (normal, bigeminy, trigeminy, other, omitted, and total time) and amount of time (in min) spent in each, as well as the percent of the overall time.

## See Also

`plot_states()`

## Examples

```
data(polar_h10)
bad_segs <- find_bad_segments(polar_h10$time, polar_h10$ecg)
peaks <- detect_peaks(polar_h10$time, polar_h10$ecg, omit_segments=bad_segs)
peak_stats <- calc_peak_stats(polar_h10$time, polar_h10$ecg, peaks, omit_segments=bad_segs)
pvc <- (peak_stats$RStime > 50)

st <- plot_states(polar_h10$time, peaks, pvc, omit_segments=bad_segs, xlim=c(50, 60), draw=FALSE)
summary(st)
```

---

# totlength *Total length of a set of time intervals*

---

## Description

Total length of a set of time intervals, in seconds

## Usage

```
totlength(segments, times = NULL, tz = Sys.timezone())
```

## Arguments

| | |
|---|---|
| segments | A data frame with two columns, the start and end of a set of time intervals, either as integer indexes to a provided vector of times, or as a set of date/times. |
| times | A vector of times, needed if segments is a set of integer indexes |
| tz | Time zone, used to convert times with convert_timestamp |

## Value

Total length of the set of time segments, in seconds

## Examples

```
data(polar_h10)
badsegs_i <- find_bad_segments(polar_h10$time, polar_h10$ecg)
totlength(badsegs_i, polar_h10$time)

badsegs_t <- find_bad_segments(polar_h10$time, polar_h10$ecg, return_index=FALSE)
totlength(badsegs_t)
```

`zero_segments` *Zero out segments of a ECG signal*

## Description

Replace ECG signal in a set of intervals with 0

## Usage

```
zero_segments(signal, segments)
```

## Arguments

| | |
|---|---|
| `signal` | Numeric vector of ECG signal |
| `segments` | Data frame with two columns: the start and end index for each interval that is to have signal replaced by 0 |

## Value

The input `signal` vector, but with the specified intervals replaced with 0.

## Examples

```
data(polar_h10)
bad_segs <- find_bad_segments(polar_h10$time, polar_h10$ecg)
polar_h10$ecg_z <- zero_segments(polar_h10$ecg, bad_segs)
plot_ecg_mult(polar_h10$time, polar_h10$ecg_z, hilit_segments=bad_segs)
```

# Index