Package 'cgmguru'

August 21, 2026

Type Package

Title Advanced Continuous Glucose Monitoring Analysis with High-Performance C++ Backend

Version 1.3.0

Maintainer Sang Ho Park <shstat1729@gmail.com>

Description Tools for advanced analysis of continuous glucose monitoring (CGM) time-series, implementing GRID (Glucose Rate Increase Detector) and GRID-based algorithms for postprandial peak detection, and detection of hypoglycemic and hyperglycemic episodes (Levels 1/2/Extended) aligned with international consensus CGM metrics. Core algorithms are implemented in optimized C++ using 'Rcpp' to provide accurate and fast analysis on large datasets.

License GPL-2

Encoding UTF-8

RoxygenNote 7.3.3

LinkingTo Rcpp

Imports Rcpp

Suggests testthat (>= 3.0.0), knitr, rmarkdown, iglu, dplyr, covr, ggplot2, microbenchmark

VignetteBuilder knitr

URL https://github.com/shstat1729/cgmguru, https://shstat1729.github.io/cgmguru/

BugReports https://github.com/shstat1729/cgmguru/issues

Config/testthat/edition 3

NeedsCompilation yes

Author Sang Ho Park [aut, cre], Rosa Oh [aut, ctb], Sang-Man Jin [aut, ctb]

Repository CRAN

Date/Publication 2026-08-21 05:44:51 UTC

Contents

conga_rcpp . . . 2
detect_all_events . . . 3
detect_between_maxima . . . 7
detect_hyperglycemic_events . . . 8
detect_hypoglycemic_events . . . 12
excursion . . . 15
find_local_maxima . . . 17
find_max_after_hours . . . 18
find_max_before_hours . . . 20
find_min_after_hours . . . 21
find_min_before_hours . . . 23
find_new_maxima . . . 24
grid . . . 25
interpolate_cgm . . . 27
interval_down . . . 29
mage_rcpp . . . 30
maxima_grid . . . 31
modd_rcpp . . . 33
mod_grid . . . 34
orderfast . . . 36
rebound_events . . . 37
sensor_wear . . . 39
start_finder . . . 40
transform_df . . . 41

Index **44**

`conga_rcpp` *Rcpp CONGA Calculation*

Description

Calculates Continuous Overall Net Glycemic Action (CONGA) with an Rcpp backend.

Usage

```
conga_rcpp(data, n = 24, tz = "", inter_gap = 45)
```

Arguments

`data`	A dataframe containing CGM data with columns `id`, `time`, and `gl`.
`n`	Whole number of hours separating paired glucose observations. Defaults to 24.
`tz`	Time zone used for day-grid alignment when supplied.
`inter_gap`	Maximum gap, in minutes, over which linear interpolation is allowed. Defaults to 45, matching iglu's internal default.

Details

The implementation follows the CONGA calculation approach used by `iglu::conga`: after interpolation to a regular day-aligned CGM grid, CONGA is the sample standard deviation of glucose differences separated by `n` hours.

Value

A tibble with columns `id` and `CONGA`.

References

McDonnell, C. M., et al. (2005). A novel approach to continuous glucose analysis utilizing glycemic variation. *Diabetes Technology and Therapeutics*, 7(2), 253-263. doi:10.1089/dia.2005.7.253

See Also

`iglu::conga`, `mage_rcpp`

Examples

```
library(iglu)
data(example_data_5_subject)
conga_rcpp(example_data_5_subject)
```

`detect_all_events` *Detect All Glycemic Events*

Description

Comprehensive function to detect all types of glycemic events aligned with international consensus CGM metrics (Battelino et al., 2023). This function provides a unified interface for detecting multiple event types including Level 1/2/Extended hypo- and hyperglycemia, Level 1 excluded events, and rebound hypo-/hyperglycemia summaries. Rebound hypoglycemia and hyperglycemia definitions follow Hansen and Bibby (2024). Events are counted only after the required recovery condition is confirmed; duration summaries use the event boundary immediately before recovery starts. Event preprocessing uses cgmguru's C++ port and structural adaptation of iglu-compatible day-based grid semantics: each subject is interpolated from the first observed day's midnight plus one reading interval, rather than from the first observed timestamp. Larger gaps are masked and removed before event classification, preserving gap-based segment boundaries. This preprocessing is specific to event calculation and does not affect `grid`, `maxima_grid`, or `excursion`. CGM summary metrics in `subject_summary` are calculated from the original raw glucose values by default. Set `summary_metrics_source = "preprocessed"` to calculate them from the internal event-preprocessed grid. Numeric summary outputs are rounded to `summary_digits` decimal places; set `summary_digits = NULL` or `summary_digits = "none"` to return unrounded values.

Usage

```
detect_all_events(df, reading_minutes = NULL, sort_time = FALSE,
 inter_gap = 45, return_interpolated = FALSE,
 summary_metrics_source = c("raw", "preprocessed"),
 sensor_wear_ndays = NULL, summary_digits = 2)
```

Arguments

df	A dataframe containing continuous glucose monitoring (CGM) data. Must include columns: • id: Subject identifier (string or factor) • time: Time of measurement (POSIXct) • gl: Glucose value (integer or numeric, mg/dL)
reading_minutes	Time interval between readings in minutes (optional). Can be a single integer/numeric value (applied to all subjects), a vector matching data length, or NULL. If omitted or NULL, the interval is calculated automatically per id as the median positive time difference in the data.
sort_time	Logical. If TRUE, sort rows within each id by time in C++ before interpolation. Defaults to FALSE.
inter_gap	Maximum gap in minutes to interpolate across. Defaults to 45; larger gaps split event-detection segments.
return_interpolated	Logical. If TRUE, include the internal event-preprocessed grid as interpolated_data. Defaults to FALSE.
summary_metrics_source	Character. Source glucose values for CGM summary metrics. Defaults to "raw" for original observed data; use "preprocessed" for the internal event-preprocessed grid after interpolation and gap masking. sensor_wear_percent is always calculated from the original timestamps and glucose readings.
sensor_wear_ndays	Number of days for fixed-window sensor_wear_percent calculation. Defaults to NULL, which uses the original timestamp span. Set to a positive number such as 90 to calculate observed readings over the last 90 days for each subject divided by the expected number of readings in 90 days.
summary_digits	Number of decimal places for numeric summary outputs in subject_summary and rate/duration columns in glycemic_event_summary. Defaults to 2. Use NULL or "none" to return unrounded values.

Value

A list containing:

- subject_summary: One row per subject. CGM summary metric columns are calculated on the original raw glucose values by default; set summary_metrics_source = "preprocessed" to use the event-preprocessed glucose grid. Event summaries are included as wide *_total_episodes columns only. Numeric summary columns are rounded according to summary_digits.

- `glycemic_event_summary`: One row per subject, event type, and event level. Contains the full event summary: `id`, `type`, `level`, `total_episodes`, `avg_ep_per_day`, and `avg_minutes_below_54_per_episode`.
- `interpolated_data`: Included when `return_interpolated = TRUE`, with columns `id`, `time`, and `gl`.

`subject_summary` includes:

- `id`: Subject identifier
- `TIR`: Percent of glucose readings in range 70-180 mg/dL
- `TITR`: Percent of glucose readings in tight range 70-140 mg/dL
- `TBR70`: Percent of glucose readings below 70 mg/dL
- `TBR54`: Percent of glucose readings below 54 mg/dL
- `TAR180`: Percent of glucose readings above 180 mg/dL
- `TAR250`: Percent of glucose readings above 250 mg/dL
- `CV`: Coefficient of variation in percent, `100 * SD / mean_glucose`
- `SD`: Sample standard deviation of glucose, mg/dL
- `mean_glucose`: Mean glucose, mg/dL
- `GMI`: Glucose Management Indicator, `3.31 + 0.02392 * mean_glucose`
- `uGMI`: Updated Glucose Management Indicator, `1 / (15.36 / mean_glucose + 0.0425)`
- `GRI`: Glycemia Risk Index, $3.0 * VLow + 2.4 * Low + 1.6 * VHigh + 0.8 * High$, where `VLow` is percent time $<$54 mg/dL, `Low` is 54-$<$70 mg/dL, `VHigh` is $>$250 mg/dL, and `High` is $>$180-$\leq$250 mg/dL
- `sensor_wear_percent`: Percent of expected CGM readings observed, calculated from the original timestamps using the same automatic range method as `iglu::active_percent()` by default. If `sensor_wear_ndays` is supplied, this is calculated over the last N days for each subject.
- `hypo_lv1_total_episodes`: Number of Level 1 hypoglycemia episodes
- `hypo_lv2_total_episodes`: Number of Level 2 hypoglycemia episodes
- `hypo_extended_total_episodes`: Number of extended hypoglycemia episodes
- `hypo_lv1_excl_total_episodes`: Number of Level 1 hypoglycemia episodes that do not overlap a Level 2 episode
- `hypo_rebound_total_episodes`: Number of rebound hypoglycemia episodes
- `hyper_lv1_total_episodes`: Number of Level 1 hyperglycemia episodes
- `hyper_lv2_total_episodes`: Number of Level 2 hyperglycemia episodes
- `hyper_extended_total_episodes`: Number of extended hyperglycemia episodes
- `hyper_lv1_excl_total_episodes`: Number of Level 1 hyperglycemia episodes that do not overlap a Level 2 episode
- `hyper_rebound_total_episodes`: Number of rebound hyperglycemia episodes

`glycemic_event_summary` includes:

- `id`: Subject identifier

- `type`: Event direction, either `"hypo"` or `"hyper"`
- `level`: Event level, one of `"lv1"`, `"lv2"`, `"extended"`, `"lv1_excl"`, or `"rebound"`
- `total_episodes`: Number of episodes for the subject, event direction, and event level
- `avg_ep_per_day`: Average episodes per day for the subject, event direction, and event level, rounded according to `summary_digits`
- `avg_minutes_below_54_per_episode`: For hypoglycemia rows, average minutes below 54 mg/dL per episode, rounded according to `summary_digits`; for hyperglycemia rows, 0

Event types

- Hypoglycemia: lv1 ($<$ 70 mg/dL, $\geq$ 15 min), lv2 ($<$ 54 mg/dL, $\geq$ 15 min), extended ($<$ 70 mg/dL, $\geq$ 120 min). - Hyperglycemia: lv1 ($>$ 180 mg/dL, $\geq$ 15 min), lv2 ($>$ 250 mg/dL, $\geq$ 15 min), extended ($>$ 250 mg/dL, $\geq$ 90 min in 120 min, end $\leq$ 180 mg/dL for $\geq$ 15 min). - Rebound: hypo rebound is Level 1 hyperglycemia followed by $<$70 mg/dL within 120 minutes; hyper rebound is Level 1 hypoglycemia followed by $>$180 mg/dL within 120 minutes.

References

Battelino, T., et al. (2023). Continuous glucose monitoring and metrics for clinical trials: an international consensus statement. The Lancet Diabetes & Endocrinology, 11(1), 42-57.

Bergenstal, Richard M., et al. (2026). Updated glucose management indicator (GMI) better aligns with HbA1c than current GMI: implications for clinical practice and reporting. Diabetologia, 69(8), 2182-2188. https://doi.org/10.1007/s00125-026-06739-w

Hansen, K. W., and Bibby, B. M. (2024). Rebound hypoglycemia and hyperglycemia in type 1 diabetes. Journal of Diabetes Science and Technology, 18(6), 1392-1398.

See Also

detect_hyperglycemic_events, detect_hypoglycemic_events, rebound_events

Examples

```
# Load sample data
library(iglu)
data(example_data_5_subject)
data(example_data_hall)

# Detect all glycemic events; reading_minutes is calculated automatically
# from the timestamp spacing when omitted
all_outputs <- detect_all_events(example_data_5_subject)
print(all_outputs$subject_summary)
print(all_outputs$glycemic_event_summary)

# Detect all events on larger dataset
large_outputs <- detect_all_events(example_data_hall)
print(paste("Total subjects analyzed:", nrow(large_outputs$subject_summary)))
```

`detect_between_maxima` *Detect Events Between Maxima*

Description

Identifies and analyzes events occurring between detected maxima points, providing detailed episode information for GRID analysis. This function helps characterize the glucose dynamics between identified peaks.

Usage

```
detect_between_maxima(df, transform_df)
```

Arguments

`df`	A dataframe containing continuous glucose monitoring (CGM) data. Must include columns: • `id`: Subject identifier (string or factor) • `time`: Time of measurement (POSIXct) • `gl`: Glucose value (integer or numeric, mg/dL)
`transform_df`	A dataframe containing summary information from previous transformations

Value

A list containing:

- `results`: Tibble with events between maxima (`id`, `grid_time`, `grid_gl`, `maxima_time`, `maxima_glucose`, `time_to_peak`)
- `episode_counts`: Tibble with episode counts per subject (`id`, `episode_counts`)

See Also

grid, mod_grid, find_new_maxima, transform_df

Other GRID pipeline: `find_local_maxima()`, `find_max_after_hours()`, `find_max_before_hours()`, `find_min_after_hours()`, `find_min_before_hours()`, `find_new_maxima()`, `grid()`, `maxima_grid()`, `mod_grid()`, `start_finder()`, `transform_df()`

Examples

```
# Load sample data
library(iglu)
data(example_data_5_subject)
data(example_data_hall)

# Complete pipeline to get transform_df
grid_result <- grid(example_data_5_subject, gap = 60, threshold = 130)
maxima_result <- find_local_maxima(example_data_5_subject)
```

```
mod_result <- mod_grid(example_data_5_subject, grid_result$grid_vector, hours = 2, gap = 60)
max_after <- find_max_after_hours(example_data_5_subject, mod_result$mod_grid_vector, hours = 2)
new_maxima <- find_new_maxima(example_data_5_subject,
                              max_after$max_index,
                              maxima_result$local_maxima_vector)
transformed <- transform_df(grid_result$episode_start, new_maxima)

# Detect events between maxima
between_events <- detect_between_maxima(example_data_5_subject, transformed)
print(paste("Events between maxima:", length(between_events)))

# Analysis on larger dataset
large_grid <- grid(example_data_hall, gap = 60, threshold = 130)
large_maxima <- find_local_maxima(example_data_hall)
large_mod <- mod_grid(example_data_hall, large_grid$grid_vector, hours = 2, gap = 60)
large_max_after <- find_max_after_hours(example_data_hall, large_mod$mod_grid_vector, hours = 2)
large_new_maxima <- find_new_maxima(example_data_hall,
                                    large_max_after$max_index,
                                    large_maxima$local_maxima_vector)
large_transformed <- transform_df(large_grid$episode_start, large_new_maxima)
large_between <- detect_between_maxima(example_data_hall, large_transformed)
print(paste("Events between maxima in larger dataset:", length(large_between)))
```

`detect_hyperglycemic_events` *Detect Hyperglycemic Events*

Description

Identifies and segments hyperglycemic events in CGM data based on international consensus CGM metrics (Battelino et al., 2023). Use `type` to select one of three event definitions:

- **Level 1**: $\geq$ 15 consecutive min of $>$ 180 mg/dL, ends with $\geq$ 15 consecutive min $\leq$ 180 mg/dL
- **Level 2**: $\geq$ 15 consecutive min of $>$ 250 mg/dL, ends with $\geq$ 15 consecutive min $\leq$ 250 mg/dL
- **Extended**: $>$ 250 mg/dL lasting $\geq$ 90 cumulative min within a 120-min period, ends when glucose returns to $\leq$ 180 mg/dL for $\geq$ 15 consecutive min after

Events are counted only after glucose remains at or below the end threshold for the specified end length. In `events_detailed`, `end_time`, `end_glucose`, and `end_index` report the last hyperglycemic reading immediately before that confirmed recovery period starts.

Usage

```
detect_hyperglycemic_events(df, ..., type = "extended",
 reading_minutes = NULL, sort_time = FALSE, inter_gap = 45,
 return_interpolated = TRUE)
```

Arguments

df	A dataframe containing continuous glucose monitoring (CGM) data. Must include columns: • id: Subject identifier (string or factor) • time: Time of measurement (POSIXct) • gl: Glucose value (integer or numeric, mg/dL)
...	Custom event criteria supplied by name. Prefer type for standard Level 1, Level 2, and Extended events. Supported custom criteria are: • dur_length: Minimum event duration in minutes required for an event to qualify. • end_length: Required recovery duration in minutes before event termination is confirmed. • start_gl: Glucose threshold in mg/dL used to qualify hyperglycemic readings. Hyperglycemic readings are above this value. • end_gl: Glucose threshold in mg/dL used to confirm recovery. Hyperglycemic events end after glucose remains at or below this value for end_length minutes.
type	Hyperglycemia event definition. One of "extended" (default), "lv1", "lv2", or "lv1_excl".
reading_minutes	Time interval between readings in minutes (optional). If omitted or NULL, the interval is calculated automatically per id as the median positive time difference in the data.
sort_time	Logical. If TRUE, sort rows within each id by time in C++ before interpolation. Defaults to FALSE.
inter_gap	Maximum gap in minutes to interpolate across. Defaults to 45; larger gaps split event-detection segments.
return_interpolated	Logical. If TRUE, include the interpolated grid data used for event detection in the returned list. Defaults to TRUE.

Value

A list containing:

- events_total: Tibble with summary statistics per subject (id, total_episodes, avg_ep_per_day)
- events_detailed: Tibble with detailed event information (id, start_time, start_glucose, end_time, end_glucose, start_index, end_index). End fields report the last dysglycemic reading before confirmed recovery starts. start_index and end_index are 1-based row positions in the internal interpolated event grid, returned as interpolated_data when return_interpolated = TRUE.
- interpolated_data: Included when return_interpolated = TRUE, with columns id, time, and gl.

Methods

Hyperglycemic events can be detected using either the recommended type argument or named custom threshold and duration criteria.

1. Preset method using type **(recommended):** Use type when you want the standard Level 1, Level 2, or Extended hyperglycemia definitions without manually entering thresholds.

- type = "lv1" uses start_gl = 180, dur_length = 15, end_length = 15, and end_gl = 180.
- type = "lv2" uses start_gl = 250, dur_length = 15, end_length = 15, and end_gl = 250.
- type = "extended" uses start_gl = 250, dur_length = 120, end_length = 15, and end_gl = 180.
- type = "lv1_excl" returns Level 1 episodes that do not overlap Level 2 episodes.

2. Custom criteria method: Supply start_gl, dur_length, end_length, and end_gl directly when using a custom definition, for example detect_hyperglycemic_events(df, start_gl = 180, dur_length = 15,end_length = 15, end_gl = 180) for Level 1 hyperglycemia. If an explicit type is supplied together with custom numeric criteria, the function returns results based on type; the custom criteria are ignored and a warning is issued.

Units and sampling

- reading_minutes can be a scalar (all rows) or a vector per-row. - If reading_minutes is omitted or NULL, it is calculated automatically per id from timestamp spacing. - Event classification uses cgmguru's C++ port and structural adaptation of iglu-compatible, midnight-aligned full-day grid semantics. Data are linearly interpolated at the id-specific interval up to inter_gap; larger gaps are masked, removed from the event-classification data, and split segments. - This preprocessing is specific to event calculation and does not affect grid, maxima_grid, or excursion.

References

Battelino, T., et al. (2023). Continuous glucose monitoring and metrics for clinical trials: an international consensus statement. The Lancet Diabetes & Endocrinology, 11(1), 42-57.

See Also

detect_all_events

Examples

```
# Load sample data
library(iglu)
data(example_data_5_subject)
data(example_data_hall)

# Level 1 Hyperglycemia Event (>=15 consecutive min of >180 mg/dL and event
# ends when there is >=15 consecutive min with a CGM sensor value of <=180 mg/dL)
hyper_lv1 <- detect_hyperglycemic_events(example_data_5_subject, type = "lv1")
print(hyper_lv1$events_total)

# Level 2 Hyperglycemia Event (>=15 consecutive min of >250 mg/dL and event
```

```
# ends when there is >=15 consecutive min with a CGM sensor value of <=250 mg/dL)
hyper_lv2 <- detect_hyperglycemic_events(example_data_5_subject, type = "lv2")
print(hyper_lv2$events_total)

# Extended Hyperglycemia Event (>250 mg/dL lasting >=90 cumulative min within a
# 120-min period, ends when glucose returns to <=180 mg/dL for >=15 consecutive
# min after)
hyper_extended <- detect_hyperglycemic_events(example_data_5_subject, type = "extended")
print(hyper_extended$events_total)

# Custom criteria method for the same standard definitions
hyper_lv1_custom <- detect_hyperglycemic_events(
  example_data_5_subject,
  start_gl = 180,
  dur_length = 15,
  end_length = 15,
  end_gl = 180
)
hyper_lv2_custom <- detect_hyperglycemic_events(
  example_data_5_subject,
  start_gl = 250,
  dur_length = 15,
  end_length = 15,
  end_gl = 250
)
hyper_extended_custom <- detect_hyperglycemic_events(
  example_data_5_subject,
  start_gl = 250,
  dur_length = 120,
  end_length = 15,
  end_gl = 180
)

# Compare event rates across levels
cat("Level 1 episodes:", sum(hyper_lv1$events_total$total_episodes), "\n")
cat("Level 2 episodes:", sum(hyper_lv2$events_total$total_episodes), "\n")
cat("Extended episodes:", sum(hyper_extended$events_total$total_episodes), "\n")

# Analysis on larger dataset with Level 1 criteria
large_hyper <- detect_hyperglycemic_events(example_data_hall, type = "lv1")
print(large_hyper$events_total)

# Analysis on larger dataset with Level 2 criteria
large_hyper_lv2 <- detect_hyperglycemic_events(example_data_hall, type = "lv2")
print(large_hyper_lv2$events_total)

# Analysis on larger dataset with Extended criteria
large_hyper_extended <- detect_hyperglycemic_events(example_data_hall, type = "extended")
print(large_hyper_extended$events_total)

# View detailed events for specific subject
if(nrow(hyper_lv1$events_detailed) > 0) {
  first_subject <- hyper_lv1$events_detailed$id[1]
```

```
 subject_events <- hyper_lv1$events_detailed[hyper_lv1$events_detailed$id == first_subject, ]
  head(subject_events)
}
```

`detect_hypoglycemic_events` *Detect Hypoglycemic Events*

Description

Identifies and segments hypoglycemic events in CGM data based on international consensus CGM metrics (Battelino et al., 2023). Use `type` to select one of three event definitions:

- **Level 1**: $\geq$ 15 consecutive min of $<$ 70 mg/dL, ends with $\geq$ 15 consecutive min $\geq$ 70 mg/dL
- **Level 2**: $\geq$ 15 consecutive min of $<$ 54 mg/dL, ends with $\geq$ 15 consecutive min $\geq$ 54 mg/dL
- **Extended**: $>$ 120 consecutive min of $<$ 70 mg/dL, ends with $\geq$ 15 consecutive min $\geq$ 70 mg/dL

Events are counted only after glucose remains at or above the recovery threshold for the specified end length. In `events_detailed`, `end_time`, `end_glucose`, and `end_index` report the last hypoglycemic reading immediately before that confirmed recovery period starts.

Usage

```
detect_hypoglycemic_events(df, ..., type = "extended",
 reading_minutes = NULL, sort_time = FALSE, inter_gap = 45,
 return_interpolated = TRUE)
```

Arguments

`df`	A dataframe containing continuous glucose monitoring (CGM) data. Must include columns: • `id`: Subject identifier (string or factor) • `time`: Time of measurement (POSIXct) • `gl`: Glucose value (integer or numeric, mg/dL)
`...`	Custom event criteria supplied by name. Prefer `type` for standard Level 1, Level 2, and Extended events. Supported custom criteria are: • `dur_length`: Minimum event duration in minutes required for an event to qualify. • `end_length`: Required recovery duration in minutes before event termination is confirmed. • `start_gl`: Glucose threshold in mg/dL used to qualify hypoglycemic readings. Hypoglycemic readings are below this value, and recovery is confirmed after glucose remains at or above this value for `end_length` minutes.

`type` Hypoglycemia event definition. One of `"extended"` (default), `"lv1"`, `"lv2"`, or `"lv1_excl"`.

`reading_minutes` Time interval between readings in minutes (optional). If omitted or `NULL`, the interval is calculated automatically per id as the median positive time difference in the data.

`sort_time` Logical. If `TRUE`, sort rows within each id by `time` in C++ before interpolation. Defaults to `FALSE`.

`inter_gap` Maximum gap in minutes to interpolate across. Defaults to 45; larger gaps split event-detection segments.

`return_interpolated` Logical. If `TRUE`, include the interpolated grid data used for event detection in the returned list. Defaults to `TRUE`.

Value

A list containing:

- `events_total`: Tibble with summary statistics per subject (id, total_episodes, avg_ep_per_day)
- `events_detailed`: Tibble with detailed event information (id, start_time, start_glucose, end_time, end_glucose, start_index, end_index, duration_below_54_minutes). End fields report the last dysglycemic reading before confirmed recovery starts. `start_index` and `end_index` are 1-based row positions in the internal interpolated event grid, returned as `interpolated_data` when `return_interpolated = TRUE`.
- `interpolated_data`: Included when `return_interpolated = TRUE`, with columns `id`, `time`, and `gl`.

Methods

Hypoglycemic events can be detected using either the recommended `type` argument or named custom threshold and duration criteria.

1. Preset method using `type` (recommended): Use `type` when you want the standard Level 1, Level 2, or Extended hypoglycemia definitions without manually entering thresholds.

- `type = "lv1"` uses `start_gl = 70`, `dur_length = 15`, and `end_length = 15`.
- `type = "lv2"` uses `start_gl = 54`, `dur_length = 15`, and `end_length = 15`.
- `type = "extended"` uses `start_gl = 70`, `dur_length = 120`, and `end_length = 15`.
- `type = "lv1_excl"` returns Level 1 episodes that do not overlap Level 2 episodes.

2. Custom criteria method: Supply `start_gl`, `dur_length`, and `end_length` directly when using a custom definition, for example `detect_hypoglycemic_events(df, start_gl = 70, dur_length = 15, end_length = 15)` for Level 1 hypoglycemia. If an explicit `type` is supplied together with custom numeric criteria, the function returns results based on `type`; the custom criteria are ignored and a warning is issued.

Units and sampling

- `reading_minutes` can be a scalar (all rows) or a vector per-row. - If `reading_minutes` is omitted or `NULL`, it is calculated automatically per id from timestamp spacing. - Event classification uses cgmguru's C++ port and structural adaptation of iglu-compatible, midnight-aligned full-day grid semantics. Data are linearly interpolated at the id-specific interval up to `inter_gap`; larger gaps are masked, removed from the event-classification data, and split segments. - This preprocessing is specific to event calculation and does not affect `grid`, `maxima_grid`, or `excursion`.

References

Battelino, T., et al. (2023). Continuous glucose monitoring and metrics for clinical trials: an international consensus statement. The Lancet Diabetes & Endocrinology, 11(1), 42-57.

See Also

detect_all_events

Examples

```
# Load sample data
library(iglu)
data(example_data_5_subject)
data(example_data_hall)

# Level 1 Hypoglycemia Event (>=15 consecutive min of <70 mg/dL and event
# ends when there is >=15 consecutive min with a CGM sensor value of >=70 mg/dL)
hypo_lv1 <- detect_hypoglycemic_events(example_data_5_subject, type = "lv1")
print(hypo_lv1$events_total)

# Level 2 Hypoglycemia Event (>=15 consecutive min of <54 mg/dL and event
# ends when there is >=15 consecutive min with a CGM sensor value of >=54 mg/dL)
hypo_lv2 <- detect_hypoglycemic_events(example_data_5_subject, type = "lv2")

# Extended Hypoglycemia Event (>120 consecutive min of <70 mg/dL and event
# ends when there is >=15 consecutive min with a CGM sensor value of >=70 mg/dL)
hypo_extended <- detect_hypoglycemic_events(example_data_5_subject, type = "extended")
print(hypo_extended$events_total)

# Custom criteria method for the same standard definitions
hypo_lv1_custom <- detect_hypoglycemic_events(
  example_data_5_subject,
  start_gl = 70,
  dur_length = 15,
  end_length = 15
)
hypo_lv2_custom <- detect_hypoglycemic_events(
  example_data_5_subject,
  start_gl = 54,
  dur_length = 15,
  end_length = 15
)
```

```
hypo_extended_custom <- detect_hypoglycemic_events(
  example_data_5_subject,
  start_gl = 70,
  dur_length = 120,
  end_length = 15
)

# Compare event rates across levels
cat("Level 1 episodes:", sum(hypo_lv1$events_total$total_episodes), "\n")
cat("Level 2 episodes:", sum(hypo_lv2$events_total$total_episodes), "\n")
cat("Extended episodes:", sum(hypo_extended$events_total$total_episodes), "\n")

# Analysis on larger dataset with Level 1 criteria
large_hypo <- detect_hypoglycemic_events(example_data_hall, type = "lv1")
print(large_hypo$events_total)

# Analysis on larger dataset with Level 2 criteria
large_hypo_lv2 <- detect_hypoglycemic_events(example_data_hall, type = "lv2")
print(large_hypo_lv2$events_total)

# Analysis on larger dataset with Extended criteria
large_hypo_extended <- detect_hypoglycemic_events(example_data_hall, type = "extended")
print(large_hypo_extended$events_total)
```

`excursion` *Calculate Glucose Excursions*

Description

Calculates glucose excursions in CGM data. An excursion is defined as a $>$ 70 mg/dL ($>$ 3.9 mmol/L) rise within 2 hours from a starting glucose value $\geq$ 70 mg/dL ($\geq$ 3.9 mmol/L), not preceded by a value $<$ 70 mg/dL ($<$ 3.9 mmol/L).

Usage

```
excursion(df, gap = 15)
```

Arguments

`df`	A dataframe containing continuous glucose monitoring (CGM) data. Must include columns: • `id`: Subject identifier (string or factor) • `time`: Time of measurement (POSIXct) • `gl`: Glucose value (integer or numeric, mg/dL)
`gap`	Gap threshold in minutes for excursion calculation (default: 15). This parameter defines the minimum time interval between consecutive GRID events.

Value

A list containing:

- `excursion_vector`: Tibble with excursion results (`excursion`)
- `episode_counts`: Tibble with episode counts per subject (`id`, `episode_counts`)
- `episode_start`: Tibble with all episode starts with columns:
  - `id`: Subject identifier
  - `time`: Timestamp at which the event occurs; equivalent to `df$time[index]`
  - `gl`: Glucose value at the event; equivalent to `df$gl[index]`
  - `maxima_time`: Timestamp of the maximum glucose value within 2 hours after the event start
  - `maxima_glucose`: Maximum glucose value within 2 hours after the event start
  - `time_to_peak_min`: Minutes from the event start to `maxima_time`
  - `index`: R-based (1-indexed) row number(s) in `df` denoting where the event occurs
  - `maxima_index`: R-based (1-indexed) row number(s) in `df` denoting where the maximum occurs

Notes

- `gap` is minutes; change to enforce minimum separation between excursions. - This function operates on the rows supplied in `df`. It does not use `interpolate_cgm` or the full-day event preprocessing grid.

References

Edwards, S., et al. (2022). Use of connected pen as a diagnostic tool to evaluate missed bolus dosing behavior in people with type 1 and type 2 diabetes. Diabetes Technology & Therapeutics, 24(1), 61-66.

See Also

grid

Examples

```
# Load sample data
library(iglu)
data(example_data_5_subject)
data(example_data_hall)

# Calculate glucose excursions
excursion_result <- excursion(example_data_5_subject, gap = 15)
print(paste("Excursion vector length:", length(excursion_result$excursion_vector)))
print(excursion_result$episode_counts)

# Excursion analysis with different gap
excursion_30min <- excursion(example_data_5_subject, gap = 30)
```

```
# Analysis on larger dataset
large_excursion <- excursion(example_data_hall, gap = 15)
print(paste("Excursion vector length in larger dataset:", length(large_excursion$excursion_vector)))
print(paste("Total episodes:", sum(large_excursion$episode_counts$episode_counts)))
```

`find_local_maxima` *Find Local Maxima in Glucose Time Series*

Description

Identifies local maxima (peaks) in glucose concentration time series data. Uses a difference-based algorithm to detect peaks where glucose values increase or remain constant for two consecutive points before the peak point, and decrease or remain constant for two consecutive points after the peak point.

Usage

```
find_local_maxima(df)
```

Arguments

`df`	A dataframe containing continuous glucose monitoring (CGM) data. Must include columns: • `id`: Subject identifier (string or factor) • `time`: Time of measurement (POSIXct) • `gl`: Glucose value (integer or numeric, mg/dL)

Value

A list containing:

- `local_maxima_vector`: Tibble with R-based (1-indexed) row numbers of local maxima (`local_maxima`). The corresponding occurrence time is `df$time[local_maxima]` and glucose is `df$gl[local_maxima]`.
- `merged_results`: Tibble with local maxima details (`id`, `time`, `gl`)

See Also

grid, mod_grid, find_new_maxima

Other GRID pipeline: `detect_between_maxima()`, `find_max_after_hours()`, `find_max_before_hours()`, `find_min_after_hours()`, `find_min_before_hours()`, `find_new_maxima()`, `grid()`, `maxima_grid()`, `mod_grid()`, `start_finder()`, `transform_df()`

Examples

```
# Load sample data
library(iglu)
data(example_data_5_subject)
data(example_data_hall)

# Find local maxima
maxima_result <- find_local_maxima(example_data_5_subject)
print(paste("Found", nrow(maxima_result$local_maxima_vector), "local maxima"))

# Find maxima on larger dataset
large_maxima <- find_local_maxima(example_data_hall)
print(paste("Found", nrow(large_maxima$local_maxima_vector), "local maxima in larger dataset"))

# View first few maxima
head(maxima_result$local_maxima_vector)

# View merged results
head(maxima_result$merged_results)
```

---

find_max_after_hours *Find Maximum Glucose After Specified Hours*

Description

Identifies the maximum glucose value occurring within a specified time window after a given start point. This function is useful for analyzing glucose patterns following specific events or time points.

Usage

```
find_max_after_hours(df, start_point_df, hours)
```

Arguments

df	A dataframe containing continuous glucose monitoring (CGM) data. Must include columns: • id: Subject identifier (string or factor) • time: Time of measurement (POSIXct) • gl: Glucose value (integer or numeric, mg/dL)
start_point_df	A dataframe with column start_index (R-based index into df)
hours	Number of hours to look ahead from the start point

Value

A list containing:

- max_index: Tibble with R-based (1-indexed) row numbers of maximum glucose (max_index). The corresponding occurrence time is df$time[max_index] and glucose is df$gl[max_index].

- episode_counts: Tibble with episode counts per subject (id, episode_counts)
- episode_start: Tibble with all episode starts with columns:
  - id: Subject identifier
  - time: Timestamp at which the maximum occurs; equivalent to df$time[index]
  - gl: Glucose value at the maximum; equivalent to df$gl[index]
  - index: R-based (1-indexed) row number(s) in df denoting where the maximum occurs

Notes

- start_index must be valid row numbers in df (1-indexed). - The search window is (0, hours] hours after each start index.

See Also

mod_grid, find_local_maxima, find_new_maxima, transform_df

Other GRID pipeline: detect_between_maxima(), find_local_maxima(), find_max_before_hours(), find_min_after_hours(), find_min_before_hours(), find_new_maxima(), grid(), maxima_grid(), mod_grid(), start_finder(), transform_df()

Examples

```
# Load sample data
library(iglu)
data(example_data_5_subject)
data(example_data_hall)

# Create start points for demonstration (using row index)
start_index <- seq(1, nrow(example_data_5_subject), by = 100)
start_points <- data.frame(start_index = start_index)

# Find maximum glucose in next 2 hours
max_after <- find_max_after_hours(example_data_5_subject, start_points, hours = 2)
print(paste("Found", length(max_after$max_index), "maximum points"))

# Find maximum glucose in next 1 hour
max_after_1h <- find_max_after_hours(example_data_5_subject, start_points, hours = 1)

# Analysis on larger dataset
large_start_index <- seq(1, nrow(example_data_hall), by = 200)
large_start_points <- data.frame(start_index = large_start_index)
large_max_after <- find_max_after_hours(example_data_hall, large_start_points, hours = 2)
print(paste("Found", length(large_max_after$max_index), "maximum points in larger dataset"))
```

`find_max_before_hours` *Find Maximum Glucose Before Specified Hours*

Description

Identifies the maximum glucose value occurring within a specified time window before a given start point. This function is useful for analyzing glucose patterns preceding specific events or time points.

Usage

```
find_max_before_hours(df, start_point_df, hours)
```

Arguments

`df`	A dataframe containing continuous glucose monitoring (CGM) data. Must include columns: • `id`: Subject identifier (string or factor) • `time`: Time of measurement (POSIXct) • `gl`: Glucose value (integer or numeric, mg/dL)
`start_point_df`	A dataframe with column `start_index` (R-based index into `df`)
`hours`	Number of hours to look back from the start point

Value

A list containing:

- `max_index`: Tibble with R-based (1-indexed) row numbers of maximum glucose (`max_index`). The corresponding occurrence time is `df$time[max_index]` and glucose is `df$gl[max_index]`.
- `episode_counts`: Tibble with episode counts per subject (`id`, `episode_counts`)
- `episode_start`: Tibble with all episode starts with columns:
  - `id`: Subject identifier
  - `time`: Timestamp at which the maximum occurs; equivalent to `df$time[index]`
  - `gl`: Glucose value at the maximum; equivalent to `df$gl[index]`
  - `index`: R-based (1-indexed) row number(s) in `df` denoting where the maximum occurs

Notes

- The search window is [`hours`, 0) hours before each start index.

See Also

mod_grid, find_local_maxima, find_new_maxima

Other GRID pipeline: `detect_between_maxima()`, `find_local_maxima()`, `find_max_after_hours()`, `find_min_after_hours()`, `find_min_before_hours()`, `find_new_maxima()`, `grid()`, `maxima_grid()`, `mod_grid()`, `start_finder()`, `transform_df()`

Examples

```
# Load sample data
library(iglu)
data(example_data_5_subject)
data(example_data_hall)

# Create start points for demonstration (using row index)
start_index <- seq(1, nrow(example_data_5_subject), by = 100)
start_points <- data.frame(start_index = start_index)

# Find maximum glucose in previous 2 hours
max_before <- find_max_before_hours(example_data_5_subject, start_points, hours = 2)
print(paste("Found", length(max_before$max_index), "maximum points"))

# Find maximum glucose in previous 1 hour
max_before_1h <- find_max_before_hours(example_data_5_subject, start_points, hours = 1)

# Analysis on larger dataset
large_start_index <- seq(1, nrow(example_data_hall), by = 200)
large_start_points <- data.frame(start_index = large_start_index)
large_max_before <- find_max_before_hours(example_data_hall, large_start_points, hours = 2)
print(paste("Found", length(large_max_before$max_index), "maximum points in larger dataset"))
```

---

find_min_after_hours *Find Minimum Glucose After Specified Hours*

Description

Identifies the minimum glucose value occurring within a specified time window after a given start point. This function is useful for analyzing glucose patterns following specific events or time points.

Usage

```
find_min_after_hours(df, start_point_df, hours)
```

Arguments

df	A dataframe containing continuous glucose monitoring (CGM) data. Must include columns: • id: Subject identifier (string or factor) • time: Time of measurement (POSIXct) • gl: Glucose value (integer or numeric, mg/dL)
start_point_df	A dataframe with column start_index (R-based index into df)
hours	Number of hours to look ahead from the start point

Value

A list containing:

- `min_index`: Tibble with R-based (1-indexed) row numbers of minimum glucose (`min_index`). The corresponding occurrence time is `df$time[min_index]` and glucose is `df$gl[min_index]`.
- `episode_counts`: Tibble with episode counts per subject (`id`, `episode_counts`)
- `episode_start`: Tibble with all episode starts with columns:
  - `id`: Subject identifier
  - `time`: Timestamp at which the minimum occurs; equivalent to `df$time[index]`
  - `gl`: Glucose value at the minimum; equivalent to `df$gl[index]`
  - `index`: R-based (1-indexed) row number(s) in `df` denoting where the minimum occurs

See Also

mod_grid, find_local_maxima

Other GRID pipeline: `detect_between_maxima()`, `find_local_maxima()`, `find_max_after_hours()`, `find_max_before_hours()`, `find_min_before_hours()`, `find_new_maxima()`, `grid()`, `maxima_grid()`, `mod_grid()`, `start_finder()`, `transform_df()`

Examples

```
# Load sample data
library(iglu)
data(example_data_5_subject)
data(example_data_hall)

# Create start points for demonstration (using row index)
start_index <- seq(1, nrow(example_data_5_subject), by = 100)
start_points <- data.frame(start_index = start_index)

# Find minimum glucose in next 2 hours
min_after <- find_min_after_hours(example_data_5_subject, start_points, hours = 2)
print(paste("Found", length(min_after$min_index), "minimum points"))

# Find minimum glucose in next 1 hour
min_after_1h <- find_min_after_hours(example_data_5_subject, start_points, hours = 1)

# Analysis on larger dataset
large_start_index <- seq(1, nrow(example_data_hall), by = 200)
large_start_points <- data.frame(start_index = large_start_index)
large_min_after <- find_min_after_hours(example_data_hall, large_start_points, hours = 2)
print(paste("Found", length(large_min_after$min_index), "minimum points in larger dataset"))
```

`find_min_before_hours` *Find Minimum Glucose Before Specified Hours*

Description

Identifies the minimum glucose value occurring within a specified time window before a given start point. This function is useful for analyzing glucose patterns preceding specific events or time points.

Usage

```
find_min_before_hours(df, start_point_df, hours)
```

Arguments

`df`	A dataframe containing continuous glucose monitoring (CGM) data. Must include columns: • `id`: Subject identifier (string or factor) • `time`: Time of measurement (POSIXct) • `gl`: Glucose value (integer or numeric, mg/dL)
`start_point_df`	A dataframe with column `start_index` (R-based index into `df`)
`hours`	Number of hours to look back from the start point

Value

A list containing:

- `min_index`: Tibble with R-based (1-indexed) row numbers of minimum glucose (`min_index`). The corresponding occurrence time is `df$time[min_index]` and glucose is `df$gl[min_index]`.
- `episode_counts`: Tibble with episode counts per subject (`id`, `episode_counts`)
- `episode_start`: Tibble with all episode starts with columns:
  - `id`: Subject identifier
  - `time`: Timestamp at which the minimum occurs; equivalent to `df$time[index]`
  - `gl`: Glucose value at the minimum; equivalent to `df$gl[index]`
  - `index`: R-based (1-indexed) row number(s) in `df` denoting where the minimum occurs

See Also

mod_grid, find_local_maxima

Other GRID pipeline: `detect_between_maxima()`, `find_local_maxima()`, `find_max_after_hours()`, `find_max_before_hours()`, `find_min_after_hours()`, `find_new_maxima()`, `grid()`, `maxima_grid()`, `mod_grid()`, `start_finder()`, `transform_df()`

Examples

```
# Load sample data
library(iglu)
data(example_data_5_subject)
data(example_data_hall)

# Create start points for demonstration (using row index)
start_index <- seq(1, nrow(example_data_5_subject), by = 100)
start_points <- data.frame(start_index = start_index)

# Find minimum glucose in previous 2 hours
min_before <- find_min_before_hours(example_data_5_subject, start_points, hours = 2)
print(paste("Found", length(min_before$min_index), "minimum points"))

# Find minimum glucose in previous 1 hour
min_before_1h <- find_min_before_hours(example_data_5_subject, start_points, hours = 1)

# Analysis on larger dataset
large_start_index <- seq(1, nrow(example_data_hall), by = 200)
large_start_points <- data.frame(start_index = large_start_index)
large_min_before <- find_min_before_hours(example_data_hall, large_start_points, hours = 2)
print(paste("Found", length(large_min_before$min_index), "minimum points in larger dataset"))
```

---

find_new_maxima *Find New Maxima Around Grid Points*

Description

Identifies new maxima in the vicinity of previously identified grid points, useful for refining maxima detection in GRID analysis. This function helps improve the accuracy of peak detection by searching around known event points.

Usage

```
find_new_maxima(df, mod_grid_max_point_df, local_maxima_df)
```

Arguments

`df`	A dataframe containing continuous glucose monitoring (CGM) data. Must include columns: • `id`: Subject identifier (string or factor) • `time`: Time of measurement (POSIXct) • `gl`: Glucose value (integer or numeric, mg/dL)
`mod_grid_max_point_df`	A dataframe with column `index` (candidate maxima index)
`local_maxima_df`	A dataframe with column `local_maxima` (index of local peaks)

Value

A tibble with updated maxima information containing columns (id, time, gl, index) The index column contains R-based (1-indexed) row number(s) in df; thus, time == df$time[index] and gl == df$gl[index].

See Also

find_local_maxima, find_max_after_hours, transform_df

Other GRID pipeline: detect_between_maxima(), find_local_maxima(), find_max_after_hours(), find_max_before_hours(), find_min_after_hours(), find_min_before_hours(), grid(), maxima_grid(), mod_grid(), start_finder(), transform_df()

Examples

```
# Load sample data
library(iglu)
data(example_data_5_subject)
data(example_data_hall)

# First, get grid points and local maxima
grid_result <- grid(example_data_5_subject, gap = 15, threshold = 130)
maxima_result <- find_local_maxima(example_data_5_subject)

# Create modified grid points (simplified for example)
mod_grid_indices <- data.frame(index = grid_result$episode_start$index[1:10])

# Find new maxima around grid points
new_maxima <- find_new_maxima(example_data_5_subject,
                              mod_grid_indices,
                              maxima_result$local_maxima_vector)
print(paste("Found", nrow(new_maxima), "new maxima"))

# Analysis on larger dataset
large_grid <- grid(example_data_hall, gap = 15, threshold = 130)
large_maxima <- find_local_maxima(example_data_hall)
large_mod_grid <- data.frame(index = large_grid$episode_start$index[1:20])
large_new_maxima <- find_new_maxima(example_data_hall,
                                    large_mod_grid,
                                    large_maxima$local_maxima_vector)
print(paste("Found", nrow(large_new_maxima), "new maxima in larger dataset"))
```

---

grid *GRID Algorithm for Glycemic Event Detection*

Description

Implements the GRID (Glucose Rate Increase Detector) algorithm for detecting rapid glucose rate increases in continuous glucose monitoring (CGM) data. This algorithm identifies rapid glucose

changes using specific rate-based criteria, and is commonly applied for meal detection. Meals are detected when the CGM value is $\geq$ 7.2 mmol/L ($\geq$ 130 mg/dL) and the rate-of-change is $\geq$ 5.3 mmol/L/h [$\geq$ 95 mg/dL/h] for the last two consecutive readings, or $\geq$ 5.0 mmol/L/h [$\geq$ 90 mg/dL/h] for two of the last three readings.

Usage

```
grid(df, gap = 15, threshold = 130)
```

Arguments

`df`	A dataframe containing continuous glucose monitoring (CGM) data. Must include columns: • `id`: Subject identifier (string or factor) • `time`: Time of measurement (POSIXct) • `gl`: Glucose value (integer or numeric, mg/dL)
`gap`	Gap threshold in minutes for event detection (default: 15). This parameter defines the minimum time interval between consecutive GRID events. For example, if gap is set to 60, only one GRID event can be detected within any one-hour window; subsequent events within the gap interval are not counted as new events.
`threshold`	GRID slope threshold in mg/dL/hour for event classification (default: 130)

Value

A list containing:

- `grid_vector`: A tibble with the results of the GRID analysis. Contains a `grid` column (0/1 values; 1 denotes a detected GRID event) and all relevant input columns.
- `episode_counts`: A tibble summarizing the number of GRID events per subject (`id`) as `episode_counts`.
- `episode_start`: A tibble listing the start of each GRID episode, with columns:
  - `id`: Subject ID.
  - `time`: The timestamp (POSIXct) at which the GRID event was detected.
  - `gl`: The glucose value (mg/dL; integer or numeric) at the GRID event.
  - `index`: R-based (1-indexed) row number(s) in the original dataframe where the GRID event occurs. The occurrence time equals `df$time[index]` and glucose equals `df$gl[index]`.

Algorithm

- Flags points where `gl >= 130 mg/dL` and rate-of-change meets the GRID criteria (see references).
- Enforces a minimum `gap` in minutes between detected events to avoid duplicates.

Units and sampling

- `gl` is mg/dL; `time` is POSIXct; `gap` is minutes. - The effective sampling interval is derived from `time` deltas. - This function operates on the rows supplied in `df`. It does not use `interpolate_cgm` or the full-day event preprocessing grid.

References

Harvey, R. A., et al. (2014). Design of the glucose rate increase detector: a meal detection module for the health monitoring system. Journal of Diabetes Science and Technology, 8(2), 307-320.

Adolfsson, Peter, et al. "Increased time in range and fewer missed bolus injections after introduction of a smart connected insulin pen." Diabetes technology & therapeutics 22.10 (2020): 709-718.

See Also

mod_grid, maxima_grid, find_local_maxima, detect_between_maxima

Other GRID pipeline: `detect_between_maxima()`, `find_local_maxima()`, `find_max_after_hours()`, `find_max_before_hours()`, `find_min_after_hours()`, `find_min_before_hours()`, `find_new_maxima()`, `maxima_grid()`, `mod_grid()`, `start_finder()`, `transform_df()`

Examples

```
# Load sample data
library(iglu)
data(example_data_5_subject)
data(example_data_hall)

# Basic GRID analysis on smaller dataset
grid_result <- grid(example_data_5_subject, gap = 15, threshold = 130)
print(grid_result$episode_counts)
print(grid_result$episode_start)
print(grid_result$grid_vector)

# More sensitive GRID analysis
sensitive_result <- grid(example_data_5_subject, gap = 10, threshold = 120)

# GRID analysis on larger dataset
large_grid <- grid(example_data_hall, gap = 15, threshold = 130)
print(paste("Detected", sum(large_grid$episode_counts$episode_counts), "episodes"))
print(large_grid$episode_start)
print(large_grid$grid_vector)
```

---

interpolate_cgm *Interpolate CGM Data*

Description

Interpolates continuous glucose monitoring (CGM) data onto the same iglu-compatible, midnight-aligned full-day grid used internally by cgmguru's event-detection functions. The interpolation is implemented in C++ and is intended for users who want to inspect or reuse the preprocessed grid behind `detect_all_events`, `detect_hyperglycemic_events`, and `detect_hypoglycemic_events`.

For each subject, `interpolate_cgm()` builds an equally spaced grid at `reading_minutes` intervals. If `reading_minutes` is omitted, it is inferred per subject from the median positive timestamp

difference. Glucose values are linearly interpolated only across gaps up to `inter_gap`; larger gaps are treated as missing and removed from the returned data, preserving segment boundaries used by event calculation.

The GRID-family functions `grid`, `maxima_grid`, and `excursion` do not call this helper automatically; they operate on the rows supplied by the user unless the caller explicitly passes an interpolated dataset.

Usage

```
interpolate_cgm(df, reading_minutes = NULL, sort_time = FALSE,
 inter_gap = 45)
```

Arguments

`df`	A dataframe containing CGM data with columns: • `id`: Subject identifier • `time`: POSIXct measurement timestamp • `gl`: Glucose value in mg/dL
`reading_minutes`	Time interval for the interpolation grid in minutes. If omitted or `NULL`, it is calculated automatically per id as the median positive time difference in the data.
`sort_time`	Logical. If `TRUE`, sort rows within each id by `time` in C++ before interpolation. Defaults to `FALSE`.
`inter_gap`	Maximum gap in minutes to interpolate across. Defaults to 45; larger gaps split the event-detection grid.

Value

A tibble with columns `id`, interpolated `time`, and interpolated `gl`. Rows inside gaps larger than `inter_gap` are omitted.

See Also

detect_all_events, detect_hyperglycemic_events, detect_hypoglycemic_events

Examples

```
df <- data.frame(
  id = "A",
  time = as.POSIXct(c("2026-01-01 00:15:00", "2026-01-01 00:25:00"),
                    tz = "UTC"),
  gl = c(100, 120)
)
interpolate_cgm(df)
```

`interval_down` *Downsample 5-Minute CGM Data to 15-Minute Intervals*

Description

Converts Dexcom-style 5-minute CGM data to clock-aligned 15-minute intervals. Each subject is grouped by actual time, not by row position, so a missing 5-minute record does not shift later intervals. Each aggregate is labeled at the right boundary of its 15-minute interval: `:15`, `:30`, `:45`, or `:00` past the hour.

Usage

```
interval_down(df, n_observed = FALSE)
```

Arguments

`df`	A data frame with columns `id`, `time` (POSIXct), and `gl` (numeric glucose values).
`n_observed`	Logical. If `TRUE`, include an `n_observed` column with the number of glucose readings contributing to each result. Defaults to `FALSE`.

Details

The glucose value is the mean of all non-missing readings in the interval. A single available value is retained and assigned to the interval-end output time. Intervals with no available glucose values are omitted. `n_observed = TRUE` adds a column recording how many glucose readings contributed to each result. Rows must be ordered by subject and time before calling this function.

Value

A data frame with `id`, clock-aligned `time`, and averaged `gl` columns. If `n_observed = TRUE`, it also includes `n_observed`. The `time` column retains the input POSIXct time zone.

Examples

```
dexcom <- data.frame(
  id = "participant_1",
  time = as.POSIXct(
    c("2024-01-01 00:00:00", "2024-01-01 00:05:00",
      "2024-01-01 00:10:00", "2024-01-01 00:15:00",
      "2024-01-01 00:20:00", "2024-01-01 00:25:00"),
    tz = "UTC"
  ),
  gl = c(100, 110, 120, 130, 140, 150)
)

interval_down(dexcom)
```

mage_rcpp *Rcpp MAGE Calculation*

Description

Calculates Mean Amplitude of Glycemic Excursions (MAGE) with an Rcpp backend.

Usage

```
mage_rcpp(
  data,
  version = c("ma", "naive"),
  sd_multiplier = 1,
  short_ma = 5,
  long_ma = 32,
  return_type = c("num", "df"),
  direction = c("avg", "service", "max", "plus", "minus"),
  tz = "",
  inter_gap = 45,
  max_gap = 180
)
```

Arguments

data	A dataframe containing CGM data with columns id, time, and gl.
version	Either "ma" or "naive". Defaults to "ma".
sd_multiplier	Multiplier for the standard deviation threshold in version = "naive".
short_ma	Short moving-average window length. Defaults to 5.
long_ma	Long moving-average window length. Defaults to 32.
return_type	Either "num" for one metric per id or "df" for a list-column of segment-level MAGE rows.
direction	One of "avg", "service", "max", "plus", or "minus".
tz	Time zone used for day-grid alignment when supplied.
inter_gap	Maximum gap, in minutes, over which linear interpolation is allowed. Defaults to 45.
max_gap	Gap length, in minutes, above which MAGE is calculated on separate trace segments. Defaults to 180.

Details

The implementation follows the MAGE calculation approaches used by iglu::mage. The default version = "ma" follows iglu's moving-average approach: CGM is interpolated to 5-minute intervals, short and long moving-average crossings identify candidate peak/nadir intervals, and countable excursions are those whose peak-to-nadir or nadir-to-peak amplitude is at least one glucose standard deviation.

The `version = "naive"` option matches iglu's older standard-deviation based calculation. This function is calculation-only and does not implement iglu's plotting options.

Value

A tibble with columns `id` and `MAGE`. With `return_type = "df"`, `MAGE` is a list-column of tibbles with `start`, `end`, `mage`, `plus_or_minus`, and `first_excursion`.

References

Service, F. John, et al. (1970). Mean amplitude of glycemic excursions, a measure of diabetic instability. *Diabetes*, 19(9), 644-655. doi:10.2337/diab.19.9.644

Fernandes, Nathaniel J., et al. (2022). Open-source algorithm to calculate mean amplitude of glycemic excursions using short and long moving averages. *Journal of Diabetes Science and Technology*, 16(2), 576-577. doi:10.1177/19322968211061165

See Also

`iglu::mage`, `conga_rcpp`

Examples

```
library(iglu)
data(example_data_5_subject)
mage_rcpp(example_data_5_subject)
mage_rcpp(example_data_5_subject, version = "naive")
```

---

maxima_grid *Combined Maxima Detection and GRID Analysis*

Description

Fast method for postprandial glucose peak detection combining GRID algorithm with local maxima analysis. Detects meal-induced glucose peaks by identifying GRID events (rapid glucose increases) and mapping them to corresponding local maxima within a search window. Local maxima are defined as points where glucose values increase or remain constant for two consecutive points before the peak, and decrease or remain constant for two consecutive points after the peak.

The 7-step algorithm: (1) finds GRID points indicating meal starts (2) identifies modified GRID points after minimum duration (3) locates maximum glucose within the subsequent time window (4) detects all local maxima using the two-consecutive-point criteria (5) refines peaks from local maxima candidates (6) maps GRID points to peaks within 4-hour constraint (7) redistributes overlapping peaks.

Usage

```
maxima_grid(df, threshold = 130, gap = 60, hours = 2)
```

Arguments

df	A dataframe containing continuous glucose monitoring (CGM) data. Must include columns: • id: Subject identifier (string or factor) • time: Time of measurement (POSIXct) • gl: Glucose value (integer or numeric, mg/dL)
threshold	GRID slope threshold in mg/dL/hour for event classification (default: 130)
gap	Gap threshold in minutes for event detection (default: 60). This parameter defines the minimum time interval between consecutive GRID events.
hours	Time window in hours for maxima analysis (default: 2)

Value

A list containing:

- results: Tibble with combined maxima and GRID analysis results, with columns:
  - id: Subject identifier
  - grid_time: Timestamp of GRID event detection (POSIXct)
  - grid_gl: Glucose value at GRID event (mg/dL)
  - maxima_time: Timestamp of peak glucose (POSIXct)
  - maxima_glucose: Peak glucose value (mg/dL)
  - time_to_peak_min: Time from GRID event to peak in minutes
  - grid_index: R-based (1-indexed) row number of GRID event; grid_time == df$time[grid_index], grid_gl == df$gl[grid_index]
  - maxima_index: R-based (1-indexed) row number of peak; maxima_time == df$time[maxima_index], maxima_glucose == df$gl[maxima_index]
- episode_counts: Tibble with episode counts per subject (id, episode_counts)

Algorithm (7 steps)

1) GRID -> 2) modified GRID -> 3) window maxima -> 4) local maxima -> 5) refine peaks -> 6) map GRID to peaks ($\leq$ 4h) -> 7) redistribute overlapping peaks.

Input grid

This function operates on the rows supplied in df. It does not use interpolate_cgm or the full-day event preprocessing grid unless the caller explicitly supplies interpolated data.

References

Park, Sang Ho, et al. "Identification of clinically meaningful automatically detected postprandial glucose excursions in individuals with type 1 diabetes using personal continuous glucose monitoring." Diabetes Research and Clinical Practice (2025): 112951.

Park, Soojin, et al. "High-Amplitude and Prolonged Glucose Excursions as a Key Determinant of Discordance Between Glucose Management Indicator and Glycated Hemoglobin in Type 1 Diabetes." Diabetes Care (2026): dc252820. https://doi.org/10.2337/dc25-2820

See Also

grid, mod_grid, find_local_maxima, find_new_maxima, transform_df

Other GRID pipeline: `detect_between_maxima()`, `find_local_maxima()`, `find_max_after_hours()`, `find_max_before_hours()`, `find_min_after_hours()`, `find_min_before_hours()`, `find_new_maxima()`, `grid()`, `mod_grid()`, `start_finder()`, `transform_df()`

Examples

```
# Load sample data
library(iglu)
data(example_data_5_subject)
data(example_data_hall)

# Combined analysis on smaller dataset
maxima_result <- maxima_grid(example_data_5_subject, threshold = 130, gap = 60, hours = 2)
print(maxima_result$episode_counts)
print(maxima_result$results)

# More sensitive analysis
sensitive_maxima <- maxima_grid(example_data_5_subject, threshold = 120, gap = 30, hours = 1)
print(sensitive_maxima$episode_counts)
print(sensitive_maxima$results)

# Analysis on larger dataset
large_maxima <- maxima_grid(example_data_hall, threshold = 130, gap = 60, hours = 2)
print(large_maxima$episode_counts)
print(large_maxima$results)
```

`modd_rcpp` *Rcpp MODD Calculation*

Description

Calculates Mean of Daily Differences (MODD) with an Rcpp backend. The preprocessing follows the iglu day-grid approach used by `iglu::modd`: CGM data are linearly interpolated to a regular midnight-aligned full-day grid, respecting `inter_gap`, before same-time-of-day differences are calculated.

Usage

```
modd_rcpp(data, lag = 1, tz = "", inter_gap = 45)
```

Arguments

`data`	A dataframe containing CGM data with columns: • `id`: Subject identifier • `time`: POSIXct measurement timestamp

- gl: Glucose value in mg/dL

lag	Whole number of days separating paired same-time-of-day glucose observations. Defaults to 1.
tz	Time zone used for day-grid alignment when supplied.
inter_gap	Maximum gap, in minutes, over which linear interpolation is allowed. Defaults to 45, matching iglu's internal default.

Value

A tibble with columns id and MODD.

References

Service, F. John, and Roger L. Nelson. "Characteristics of glycemic stability." Diabetes Care 3.1 (1980): 58-62.

See Also

iglu::modd, conga_rcpp, mage_rcpp

Examples

```
library(iglu)
data(example_data_5_subject)
modd_rcpp(example_data_5_subject)
modd_rcpp(example_data_5_subject, lag = 2)
```

---

mod_grid *Modified GRID Analysis*

Description

Constructs a modified GRID series by reapplying the GRID logic with a designated gap (e.g., 60 minutes) and analysis window in hours (e.g., 2 hours). It reassigns GRID events under these constraints to produce a modified grid suitable for downstream maxima mapping and episode analysis.

Usage

```
mod_grid(df, grid_point_df, hours = 2, gap = 15)
```

Arguments

df	A dataframe containing continuous glucose monitoring (CGM) data. Must include columns:

- id: Subject identifier (string or factor)
- time: Time of measurement (POSIXct)
- gl: Glucose value (integer or numeric, mg/dL)

grid_point_df	A dataframe with column start_index (start points for re-applied GRID)
hours	Time window in hours for analysis (default: 2)
gap	Gap threshold in minutes for event detection (default: 15). This parameter defines the minimum time interval between consecutive GRID events.

Value

A list containing:

- mod_grid_vector: Tibble with modified GRID results (mod_grid)
- episode_counts: Tibble with episode counts per subject (id, episode_counts)
- episode_start: Tibble with all episode starts with columns:
  - id: Subject identifier
  - time: Timestamp at which the event occurs; equivalent to df$time[index]
  - gl: Glucose value at the event; equivalent to df$gl[index]
  - index: R-based (1-indexed) row number(s) in df denoting where the event occurs

Units and sampling

- gap is minutes; hours is hours; time is POSIXct.

References

Park, Sang Ho, et al. "Identification of clinically meaningful automatically detected postprandial glucose excursions in individuals with type 1 diabetes using personal continuous glucose monitoring." Diabetes Research and Clinical Practice (2025): 112951.

Park, Soojin, et al. "High-Amplitude and Prolonged Glucose Excursions as a Key Determinant of Discordance Between Glucose Management Indicator and Glycated Hemoglobin in Type 1 Diabetes." Diabetes Care (2026): dc252820. https://doi.org/10.2337/dc25-2820

See Also

grid, find_max_after_hours, find_new_maxima

Other GRID pipeline: detect_between_maxima(), find_local_maxima(), find_max_after_hours(), find_max_before_hours(), find_min_after_hours(), find_min_before_hours(), find_new_maxima(), grid(), maxima_grid(), start_finder(), transform_df()

Examples

```
# Load sample data
library(iglu)
data(example_data_5_subject)
data(example_data_hall)

# First, get grid points
grid_result <- grid(example_data_5_subject, gap = 60, threshold = 130)

# Perform modified GRID analysis
```

```
mod_result <- mod_grid(example_data_5_subject, grid_result$grid_vector, hours = 2, gap = 60)
print(paste("Modified grid points:", nrow(mod_result$mod_grid_vector)))

# Modified analysis with different parameters
mod_result_1h <- mod_grid(example_data_5_subject, grid_result$grid_vector, hours = 1, gap = 40)

# Analysis on larger dataset
large_grid <- grid(example_data_hall, gap = 60, threshold = 130)
large_mod_result <- mod_grid(example_data_hall, large_grid$grid_vector, hours = 2, gap = 60)
print(paste("Modified grid points in larger dataset:", nrow(large_mod_result$mod_grid_vector)))
```

`orderfast` *Fast Ordering Function*

Description

Orders a dataframe by `id` and `time` columns using a C++ `std::sort` backend. Optimized for large CGM datasets, it returns the input with rows sorted by subject then timestamp while preserving all columns.

Orders a dataframe by id and time columns using the C++ backend

Usage

```
orderfast(df)
```

Arguments

`df`	A dataframe with 'id' and 'time' columns

Value

A dataframe ordered by id and time

A dataframe ordered by id and time

Examples

```
# Load sample data
library(iglu)
data(example_data_5_subject)
data(example_data_hall)

# Shuffle without replacement, then order and compare to baseline
set.seed(123)
shuffled <- example_data_5_subject[sample(seq_len(nrow(example_data_5_subject)),
                                          replace = FALSE), ]
baseline <- orderfast(example_data_5_subject)
ordered_shuffled <- orderfast(shuffled)

# Compare results
```

```
print(paste("Identical after ordering:", identical(baseline, ordered_shuffled)))
head(baseline[, c("id", "time", "gl")])
head(ordered_shuffled[, c("id", "time", "gl")])

# Order larger dataset
ordered_large <- orderfast(example_data_hall)
print(paste("Ordered", nrow(ordered_large), "rows in larger dataset"))
df <- data.frame(id = c("b", "a", "a"), time = as.POSIXct(
  c("2024-01-01 01:00:00", "2024-01-01 00:00:00", "2024-01-01 01:00:00"), tz = "UTC"
))
orderfast(df)
```

`rebound_events` *Detect Rebound Hypoglycemia and Hyperglycemia*

Description

Detects rebound hypoglycemia (Rhypo) and rebound hyperglycemia (Rhyper) as derived events built from cgmguru Level 1 event starts and a later opposite-threshold crossing within a configurable time window.

Rhypo is a Level 1 hyperglycemic event (>180 mg/dL for at least 15 minutes) followed by any glucose value <70 mg/dL within `rebound_minutes`. Rhyper is a Level 1 hypoglycemic event (<70 mg/dL for at least 15 minutes) followed by any glucose value >180 mg/dL within `rebound_minutes`. The later rebound side only needs a single qualifying threshold crossing.

These rebound hypoglycemia and hyperglycemia definitions are documented by Hansen and Bibby (2024); cgmguru applies them on its own event-preprocessed CGM grid.

Usage

```
rebound_events(df, type = c("all", "hypo", "hyper"),
 data_source = c("raw", "preprocessed"), reading_minutes = NULL,
 sort_time = FALSE, inter_gap = 45, rebound_minutes = 120,
 return_interpolated = TRUE)
```

Arguments

`df`	A dataframe containing continuous glucose monitoring (CGM) data with columns `id`, `time`, and `gl`.
`type`	Rebound direction to return. `"hypo"` returns Rhypo, `"hyper"` returns Rhyper, and `"all"` returns both.
`data_source`	Source interpretation for `df`. `"raw"` applies cgmguru event preprocessing first. `"preprocessed"` treats `df` as an already-preprocessed event grid and does not interpolate again.
`reading_minutes`	Time interval between readings in minutes. Can be a scalar, a vector matching `nrow(df)`, or `NULL`. If omitted, it is inferred per id from positive timestamp differences.

`sort_time` Logical. If `TRUE`, sort rows within each id by `time`. Defaults to `FALSE`.

`inter_gap` Maximum gap in minutes to interpolate across when `data_source = "raw"`. Defaults to 45.

`rebound_minutes` Maximum bridge interval in minutes from the initial Level 1 event end to the later rebound threshold crossing. Defaults to 120.

`return_interpolated` Logical. If `TRUE`, include the preprocessed event grid used for rebound detection as `interpolated_data`. Defaults to `TRUE`, so `rebound_events()` returns the preprocessed data by default.

Value

A list containing:

- `events_total`: Tibble with `id`, `type`, `total_episodes`, and `avg_ep_per_day`.
- `events_detailed`: Tibble with bridge boundaries (`start_time`, `end_time`, indices, and glucose values), the initial Level 1 event boundaries, rebound threshold crossing fields, and `minutes_to_rebound`.
- `interpolated_data`: The preprocessed event grid used for rebound detection, included by default when `return_interpolated = TRUE`, with columns `id`, `time`, and `gl`.

References

Hansen, K. W., and Bibby, B. M. (2024). Rebound hypoglycemia and hyperglycemia in type 1 diabetes. *Journal of Diabetes Science and Technology*, 18(6), 1392-1398.

See Also

detect_all_events, detect_hyperglycemic_events, detect_hypoglycemic_events, interpolate_cgm

Examples

```
df <- data.frame(
  id = "A",
  time = as.POSIXct("2026-01-01 00:00:00", tz = "UTC") + 0:8 * 5 * 60,
  gl = c(190, 195, 200, 170, 165, 160, 65, 100, 110)
)
rebound_events(df, type = "hypo", reading_minutes = 5)
```

sensor_wear *Calculate Sensor Wear*

Description

Calculates the percent of expected CGM readings observed. By default, the calculation uses each subject's original timestamp span from first valid reading to last valid reading. If ndays is supplied, valid readings in [end_date - ndays, end_date] are divided by the expected number of readings over that fixed retrospective window.

For fixed-window calculations, if end_date = NULL, each subject's last valid timestamp defines that subject's retrospective window. If end_date is supplied, the same endpoint is used for all subjects, which is useful for a common study cutoff or report date. Duplicate timestamps within a subject are de-duplicated after sorting, and rows with missing time or gl do not count as observed readings.

Usage

```
sensor_wear(df, end_date = NULL, ndays = NULL,
 reading_minutes = NULL)
```

Arguments

df	A dataframe containing CGM data with columns: • id: Subject identifier • time: POSIXct measurement timestamp • gl: Glucose value in mg/dL
end_date	End timestamp for a fixed-window calculation. Requires ndays. If NULL, each subject's last valid timestamp is used. Date values are converted with as.POSIXct(), matching iglu's manual active-percent behavior.
ndays	Number of days in the fixed retrospective window. Defaults to NULL, which uses the original timestamp span.
reading_minutes	Reading interval in minutes. If NULL, it is inferred per id from the median positive difference between valid readings.

Value

A tibble with columns id, sensor_wear_percent, sensor_wear, ndays, start_date, and end_date. sensor_wear is retained as a backward-compatible alias.

See Also

detect_all_events, iglu::active_percent

Examples

```
library(iglu)
data(example_data_5_subject)
sensor_wear(example_data_5_subject, reading_minutes = 5)
sensor_wear(example_data_5_subject, ndays = 90, reading_minutes = 5)
```

`start_finder` *Find Start Points for Event Analysis*

Description

Finds R-based (1-indexed) positions where the value is 1 in an integer vector of 0s and 1s, specifically identifying episode start points. This function looks for positions where a 1 follows a 0 or is at the beginning of the vector, which is useful for identifying the start of glycemic events or episodes.

Usage

```
start_finder(df)
```

Arguments

`df`	A dataframe with the first column containing an integer vector of 0s and 1s

Value

A tibble containing start_index with R-based (1-indexed) positions where episodes start Note: These index refer to positions in the provided input vector/dataframe, not necessarily rows of the original CGM `df` unless that vector was derived directly from `df` in row order.

Notes

- Returns R-based `start_index` positions relative to the provided input vector/dataframe. - If used on vectors derived from a CGM `df`, index map directly to `df` rows.

See Also

grid, mod_grid

Other GRID pipeline: `detect_between_maxima()`, `find_local_maxima()`, `find_max_after_hours()`, `find_max_before_hours()`, `find_min_after_hours()`, `find_min_before_hours()`, `find_new_maxima()`, `grid()`, `maxima_grid()`, `mod_grid()`, `transform_df()`

Examples

```
# Load sample data
library(iglu)
data(example_data_5_subject)
data(example_data_hall)

# Create a binary vector indicating episode starts
binary_vector <- c(0, 0, 1, 1, 0, 1, 0, 0, 1, 1)
df <- data.frame(episode_starts = binary_vector)

# Find R-based index where episodes start
start_points <- start_finder(df)
print(paste("Start index:", paste(start_points$start_index, collapse = ", ")))

# Use with actual GRID results
grid_result <- grid(example_data_5_subject, gap = 15, threshold = 130)
grid_starts <- start_finder(grid_result$grid_vector)
print(paste("GRID episode starts:", length(grid_starts$start_index)))

# Analysis on larger dataset
large_grid <- grid(example_data_hall, gap = 15, threshold = 130)
large_starts <- start_finder(large_grid$grid_vector)
print(paste("GRID episode starts in larger dataset:", length(large_starts$start_index)))
```

`transform_df` *Transform Dataframe for Analysis*

Description

Performs data transformations required for GRID analysis, including mapping GRID episode starts to maxima within a 4-hour window and merging grid and maxima information. This function prepares data for downstream analysis by combining these results.

Usage

```
transform_df(grid_df, maxima_df)
```

Arguments

`grid_df`	A dataframe containing grid analysis results
`maxima_df`	A dataframe containing maxima detection results

Value

A tibble with transformed data containing columns (`id`, `grid_time`, `grid_gl`, `maxima_time`, `maxima_gl`)

See Also

grid, find_new_maxima, detect_between_maxima

Other GRID pipeline: detect_between_maxima(), find_local_maxima(), find_max_after_hours(), find_max_before_hours(), find_min_after_hours(), find_min_before_hours(), find_new_maxima(), grid(), maxima_grid(), mod_grid(), start_finder()

Examples

```
# Load sample data
library(iglu)
data(example_data_5_subject)
data(example_data_hall)

# Complete pipeline example with smaller dataset
threshold <- 130
gap <- 60
hours <- 2
# 1) Find GRID points
grid_result <- grid(example_data_5_subject, gap = gap, threshold = threshold)
# 2) Find modified GRID points before 2 hours minimum
mod_grid <- mod_grid(example_data_5_subject,
                     start_finder(grid_result$grid_vector),
                     hours = hours,
                     gap = gap)

# 3) Find maximum point 2 hours after mod_grid point
mod_grid_maxima <- find_max_after_hours(example_data_5_subject,
                                        start_finder(mod_grid$mod_grid_vector),
                                        hours = hours)

# 4) Identify local maxima around episodes/windows
local_maxima <- find_local_maxima(example_data_5_subject)

# 5) Among local maxima, find maximum point after two hours
final_maxima <- find_new_maxima(example_data_5_subject,
                                mod_grid_maxima$max_index,
                                local_maxima$local_maxima_vector)

# 6) Map GRID points to maximum points (within 4 hours)
transform_maxima <- transform_df(grid_result$episode_start, final_maxima)

# 7) Redistribute overlapping maxima between GRID points
final_between_maxima <- detect_between_maxima(example_data_5_subject, transform_maxima)
# Complete pipeline example with larger dataset (example_data_hall)
# This demonstrates the same workflow on a more comprehensive dataset
hall_threshold <- 130
hall_gap <- 60
hall_hours <- 2

# 1) Find GRID points on larger dataset
hall_grid_result <- grid(example_data_hall, gap = hall_gap, threshold = hall_threshold)
```

```
# 2) Find modified GRID points
hall_mod_grid <- mod_grid(example_data_hall,
                          start_finder(hall_grid_result$grid_vector),
                          hours = hall_hours,
                          gap = hall_gap)

# 3) Find maximum points after mod_grid
hall_mod_grid_maxima <- find_max_after_hours(example_data_hall,
                                             start_finder(hall_mod_grid$mod_grid_vector),
                                             hours = hall_hours)

# 4) Identify local maxima
hall_local_maxima <- find_local_maxima(example_data_hall)

# 5) Find new maxima
hall_final_maxima <- find_new_maxima(example_data_hall,
                                     hall_mod_grid_maxima$max_index,
                                     hall_local_maxima$local_maxima_vector)

# 6) Transform data
hall_transform_maxima <- transform_df(hall_grid_result$episode_start, hall_final_maxima)

# 7) Detect between maxima
hall_final_between_maxima <- detect_between_maxima(example_data_hall, hall_transform_maxima)
```

Index

∗ **GRID pipeline**
detect_between_maxima, 7
find_local_maxima, 17
find_max_after_hours, 18
find_max_before_hours, 20
find_min_after_hours, 21
find_min_before_hours, 23
find_new_maxima, 24
grid, 25
maxima_grid, 31
mod_grid, 34
start_finder, 40
transform_df, 41

conga_rcpp, 2, *31*, *34*

detect_all_events, 3, *10*, *14*, *27, 28*, *38, 39*
detect_between_maxima, 7, *17*, *19, 20*, *22, 23*, *25*, *27*, *33*, *35*, *40*, *42*
detect_hyperglycemic_events, *6*, 8, *27, 28*, *38*
detect_hypoglycemic_events, *6*, 12, *27, 28*, *38*

excursion, *3*, *10*, *14*, 15, *28*

find_local_maxima, *7*, 17, *19, 20*, *22, 23*, *25*, *27*, *33*, *35*, *40*, *42*
find_max_after_hours, *7*, *17*, 18, *20*, *22, 23*, *25*, *27*, *33*, *35*, *40*, *42*
find_max_before_hours, *7*, *17*, *19*, 20, *22, 23*, *25*, *27*, *33*, *35*, *40*, *42*
find_min_after_hours, *7*, *17*, *19, 20*, 21, *23*, *25*, *27*, *33*, *35*, *40*, *42*
find_min_before_hours, *7*, *17*, *19, 20*, *22*, 23, *25*, *27*, *33*, *35*, *40*, *42*
find_new_maxima, *7*, *17*, *19, 20*, *22, 23*, 24, *27*, *33*, *35*, *40*, *42*

grid, *3*, *7*, *10*, *14*, *16, 17*, *19, 20*, *22, 23*, *25*, 25, *28*, *33*, *35*, *40*, *42*

iglu::active_percent, *39*
iglu::conga, *3*
iglu::mage, *30, 31*
iglu::modd, *33, 34*
interpolate_cgm, *16*, *26*, 27, *32*, *38*
interval_down, 29

mage_rcpp, *3*, 30, *34*
maxima_grid, *3*, *7*, *10*, *14*, *17*, *19, 20*, *22, 23*, *25*, *27, 28*, 31, *35*, *40*, *42*
mod_grid, *7*, *17*, *19, 20*, *22, 23*, *25*, *27*, *33*, 34, *40*, *42*
modd_rcpp, 33

orderfast, 36

rebound_events, *6*, 37

sensor_wear, 39
start_finder, *7*, *17*, *19, 20*, *22, 23*, *25*, *27*, *33*, *35*, 40, *42*

transform_df, *7*, *17*, *19, 20*, *22, 23*, *25*, *27*, *33*, *35*, *40*, 41