Package 'cgmguru'

November 5, 2025

Type Package

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

Version 0.1.0

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 MIT + file LICENSE

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>

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 2025-11-05 20:30:07 UTC

Contents

---

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, and Level 1 excluded events.

Usage

```
detect_all_events(df, reading_minutes = NULL)
```

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) or a vector matching data length (different intervals per subject)

Value

A tibble containing comprehensive event analysis with columns:

- id: Subject identifier
- type: Event type (hypo/hyper)
- level: Event level (lv1/lv2/extended/lv1_excl)
- total_episodes: Total number of episodes
- avg_ep_per_day: Average episodes per day
- avg_episode_duration_below_54: Average episode duration below 54 mg/dL in minutes (hypoglycemic events only)

Event types

- Hypoglycemia: lv1 (< 70 mg/dL, ≥ 15 min), lv2 (< 54 mg/dL, ≥ 15 min), extended (< 70 mg/dL, ≥ 120 min). - Hyperglycemia: lv1 (> 180 mg/dL, ≥ 15 min), lv2 (> 250 mg/dL, ≥ 15 min), extended (> 250 mg/dL, ≥ 90 min in 120 min, end ≤ 180 mg/dL for ≥ 15 min).

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_hyperglycemic_events](#), [detect_hypoglycemic_events](#)

Examples

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

# Detect all glycemic events with 5-minute reading intervals
all_events <- detect_all_events(example_data_5_subject, reading_minutes = 5)
print(all_events)

# Detect all events on larger dataset
large_all_events <- detect_all_events(example_data_hall, reading_minutes = 5)
print(paste("Total event types analyzed:", nrow(large_all_events)))

# Filter for specific event types
hyperglycemia_events <- all_events[all_events$type == "hyper", ]
hypoglycemia_events <- all_events[all_events$type == "hypo", ]

print("Hyperglycemia events:")
print(hyperglycemia_events)
print("Hypoglycemia events:")
print(hypoglycemia_events)
```

---

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$max_indices,
                               maxima_result$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$max_indices,
                                     large_maxima$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). Supports three event types:

- **Level 1**: ≥ 15 consecutive min of > 180 mg/dL, ends with ≥ 15 consecutive min ≤ 180 mg/dL
- **Level 2**: ≥ 15 consecutive min of > 250 mg/dL, ends with ≥ 15 consecutive min ≤ 250 mg/dL
- **Extended**: > 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

Events are detected when glucose exceeds the start threshold for the minimum duration and ends when glucose falls below the end threshold for the specified end length.

Usage

```
detect_hyperglycemic_events(df,
 reading_minutes = NULL,
 dur_length = 120,
 end_length = 15,
 start_gl = 250,
 end_gl = 180)
```

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)
`dur_length`	Minimum duration in minutes for event classification (default: 120)
`end_length`	End length criteria in minutes (default: 15)
`start_gl`	Starting glucose threshold in mg/dL (default: 250)
`end_gl`	Ending glucose threshold in mg/dL (default: 180)

Value

A list containing:

- `events_total`: Tibble with summary statistics per subject (id, total_events, avg_ep_per_day)
- `events_detailed`: Tibble with detailed event information (id, start_time, start_glucose, end_time, end_glucose, start_indices, end_indices)

Units and sampling

- `reading_minutes` can be a scalar (all rows) or a vector per-row. - If `reading_minutes` is NULL, duration is computed from `time` deltas.

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: \eqn{\geq} 15 min \eqn{>} 180 mg/dL,
# ends \eqn{\leq} 180 \eqn{\geq} 15 min
hyper_lv1 <- detect_hyperglycemic_events(
  example_data_5_subject,
  start_gl = 180,
  dur_length = 15,
```

```
    end_length = 15,
    end_gl = 180
)
print(hyper_lv1$events_total)

# Level 2: \eqn{\geq} 15 min \eqn{>} 250 mg/dL,
# ends \eqn{\leq} 250 \eqn{\geq} 15 min
hyper_lv2 <- detect_hyperglycemic_events(
  example_data_5_subject,
  start_gl = 250,
  dur_length = 15,
  end_length = 15,
  end_gl = 250
)
print(hyper_lv2$events_total)

# Extended Hyperglycemia (\eqn{>} 250 mg/dL \eqn{\geq} 90 cumulative min within 120-min period,
# ends \eqn{\leq} 180 mg/dL \eqn{\geq} 15 min after)
hyper_extended <- detect_hyperglycemic_events(example_data_5_subject)
print(hyper_extended$events_total)

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

# Analysis on larger dataset with Level 1 criteria
large_hyper <- detect_hyperglycemic_events(example_data_hall,
                                           start_gl = 180,
                                           dur_length = 15,
                                           end_length = 15,
                                           end_gl = 180)
print(large_hyper$events_total)

# Analysis on larger dataset with Level 2 criteria
large_hyper_lv2 <- detect_hyperglycemic_events(example_data_hall,
                                               start_gl = 250,
                                               dur_length = 15,
                                               end_length = 15,
                                               end_gl = 250)
print(large_hyper_lv2$events_total)

# Analysis on larger dataset with Extended criteria
large_hyper_extended <- detect_hyperglycemic_events(example_data_hall)
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). Supports three event types:

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

Events are detected when glucose falls below the start threshold for the minimum duration and ends when glucose rises above the end threshold for the specified end length.

Usage

```
detect_hypoglycemic_events(df,
                            reading_minutes = NULL,
                            dur_length = 120,
                            end_length = 15,
                            start_gl = 70)
```

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)
dur_length	Minimum duration in minutes for event classification (default: 120)
end_length	End length criteria in minutes (default: 15)
start_gl	Starting glucose threshold in mg/dL (default: 70)

Value

A list containing:

- events_total: Tibble with summary statistics per subject (id, total_events, avg_ep_per_day)
- events_detailed: Tibble with detailed event information (id, start_time, start_glucose, end_time, end_glucose, start_indices, end_indices, duration_below_54_minutes)

Units and sampling

- `reading_minutes` can be a scalar (all rows) or a vector per-row. - If `reading_minutes` is NULL, duration is computed from `time` deltas.

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: \eqn{<} 70 \eqn{\geq} 15 min,
# ends \eqn{\geq} 70 \eqn{\geq} 15 min
hypo_lv1 <- detect_hypoglycemic_events(
  example_data_5_subject,
  start_gl = 70,
  dur_length = 15,
  end_length = 15
)
print(hypo_lv1$events_total)

# Level 2: \eqn{<} 54 \eqn{\geq} 15 min,
# ends \eqn{\geq} 54 \eqn{\geq} 15 min
hypo_lv2 <- detect_hypoglycemic_events(
  example_data_5_subject,
  start_gl = 54,
  dur_length = 15,
  end_length = 15
)

# Extended: \eqn{<} 70 \eqn{\geq} 120 min,
# ends \eqn{\geq} 70 \eqn{\geq} 15 min
hypo_extended <- detect_hypoglycemic_events(example_data_5_subject)
print(hypo_extended$events_total)

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

# Analysis on larger dataset with Level 1 criteria
large_hypo <- detect_hypoglycemic_events(example_data_hall,
                                    start_gl = 70,
```

```
                                                        dur_length = 15,
                                                        end_length = 15)
print(large_hypo$events_total)

# Analysis on larger dataset with Level 2 criteria
large_hypo_lv2 <- detect_hypoglycemic_events(example_data_hall,
                                             start_gl = 54,
                                             dur_length = 15,
                                             end_length = 15)
print(large_hypo_lv2$events_total)

# Analysis on larger dataset with Extended criteria
large_hypo_extended <- detect_hypoglycemic_events(example_data_hall)
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, 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$time[indices]
  - gl: Glucose value at the event; equivalent to df$gl[indices]
  - indices: R-based (1-indexed) row number(s) in df denoting where the event occurs

Notes

- gap is minutes; change to enforce minimum separation between excursions.

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 in-
                      clude 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_indices (R-based indices into df)
hours	Number of hours to look ahead from the start point

Value

A list containing:

- max_indices: Tibble with R-based (1-indexed) row numbers of maximum glucose (max_indices). The corresponding occurrence time is df$time[max_indices] and glucose is df$gl[max_indices].
- 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$time[indices]
  - gl: Glucose value at the maximum; equivalent to df$gl[indices]
  - indices: R-based (1-indexed) row number(s) in df denoting where the maximum occurs

Notes

- start_indices 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 indices)
start_indices <- seq(1, nrow(example_data_5_subject), by = 100)
start_points <- data.frame(start_indices = start_indices)

# 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_indices), "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_indices <- seq(1, nrow(example_data_hall), by = 200)
large_start_points <- data.frame(start_indices = large_start_indices)
large_max_after <- find_max_after_hours(example_data_hall, large_start_points, hours = 2)
print(paste("Found", length(large_max_after$max_indices), "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_indices (R-based indices into df)
hours	Number of hours to look back from the start point

Value

A list containing:

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

- 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$time[indices]
    - gl: Glucose value at the maximum; equivalent to df$gl[indices]
    - indices: 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 indices)
start_indices <- seq(1, nrow(example_data_5_subject), by = 100)
start_points <- data.frame(start_indices = start_indices)

# 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_indices), "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_indices <- seq(1, nrow(example_data_hall), by = 200)
large_start_points <- data.frame(start_indices = large_start_indices)
large_max_before <- find_max_before_hours(example_data_hall, large_start_points, hours = 2)
print(paste("Found", length(large_max_before$max_indices), "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_indices (R-based indices into df)

hours   Number of hours to look ahead from the start point

Value

A list containing:

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

- 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$time[indices]
    - gl: Glucose value at the minimum; equivalent to df$gl[indices]
    - indices: 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 indices)
start_indices <- seq(1, nrow(example_data_5_subject), by = 100)
start_points <- data.frame(start_indices = start_indices)

# 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_indices), "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_indices <- seq(1, nrow(example_data_hall), by = 200)
large_start_points <- data.frame(start_indices = large_start_indices)
large_min_after <- find_min_after_hours(example_data_hall, large_start_points, hours = 2)
print(paste("Found", length(large_min_after$min_indices), "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_indices (R-based indices into df)
hours	Number of hours to look back from the start point

Value

A list containing:

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

- 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$time[indices]
  - gl: Glucose value at the minimum; equivalent to df$gl[indices]
  - indices: 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 indices)
start_indices <- seq(1, nrow(example_data_5_subject), by = 100)
start_points <- data.frame(start_indices = start_indices)

# 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_indices), "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_indices <- seq(1, nrow(example_data_hall), by = 200)
large_start_points <- data.frame(start_indices = large_start_indices)
large_min_before <- find_min_before_hours(example_data_hall, large_start_points, hours = 2)
print(paste("Found", length(large_min_before$min_indices), "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 indices (candidate maxima indices)

local_maxima_df
: A dataframe with column local_maxima (indices of local peaks)

Value

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

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(indices = grid_result$episode_start$indices[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(indices = large_grid$episode_start$indices[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 ≥ 7.2 mmol/L (≥ 130 mg/dL) and the rate-of-change is ≥ 5.3 mmol/L/h [≥ 95 mg/dL/h] for the last two consecutive readings, or ≥ 5.0 mmol/L/h [≥ 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.
  - indices: R-based (1-indexed) row number(s) in the original dataframe where the GRID event occurs. The occurrence time equals df$time[indices] and glucose equals df$gl[indices].

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.

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)
```

---

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

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)
```

---

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_indices (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$time[indices]
    - gl: Glucose value at the event; equivalent to df$gl[indices]
    - indices: 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.

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 efficiently using base R's `order`. 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

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)
```

---

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_indices with R-based (1-indexed) positions where episodes start Note: These indices 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_indices` positions relative to the provided input vector/dataframe. - If used on vectors derived from a CGM `df`, indices 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 indices where episodes start
start_points <- start_finder(df)
print(paste("Start indices:", paste(start_points$start_indices, 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_indices)))

# 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_indices)))
```

---

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$max_indices,
                                local_maxima$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$max_indices,
                                     hall_local_maxima$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