# Package 'gsDesignNB'

October 22, 2026

**Version** 0.4.0

**Title** Sample Size and Simulation for Negative Binomial Outcomes

**Description** Provides tools for planning and simulating recurrent event trials with overdispersed count endpoints analyzed using negative binomial (or Poisson) rate models. Implements sample size and power calculations for fixed designs with variable accrual, dropout, maximum follow-up, and event gaps, including methods of Zhu and Lakkis (2014) <doi:10.1002/sim.5947> and Friede and Schmidli (2010) <doi:10.3414/ME09-02-0060> as well as extensions for score-test sizing and gaps between events. Supports group sequential monitoring by building on the 'gsDesign' package. Includes recurrent-event simulation utilities (including seasonal rates), interim data truncation, Wald and score-test inference for rate ratios, and information estimation and sample size re-estimation with or without treatment-group labels.

**License** GPL (>= 3)

**URL** https://keaven.github.io/gsDesignNB/, https://github.com/keaven/gsDesignNB

**BugReports** https://github.com/keaven/gsDesignNB/issues

**Encoding** UTF-8

**Depends** R (>= 4.1.0)

**Imports** data.table, gsDesign (>= 3.10.0), simtrial, stats, MASS

**Suggests** glmmTMB, testthat (>= 3.0.0), knitr, rmarkdown, ggplot2, dplyr, lt (>= 0.5), scales, shiny, foreach, doFuture, future, future.apply, gridExtra

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Config/roxygen2/version** 8.0.0

**NeedsCompilation** no

**Author** Keaven Anderson [aut, cre], Hongtao Zhang [aut], Andrea Maes [aut],

Nan Xiao [ctb],
Merck & Co., Inc., Rahway, NJ, USA and its affiliates [cph] (ROR:
<https://ror.org/02891sr49>)

**Maintainer** Keaven Anderson <keaven_anderson@merck.com>

**Repository** CRAN

**Date/Publication** 2026-10-22 13:40:02 UTC

# Contents

blinded_ssr . . . 3
calculate_blinded_info . . . 5
check_gs_bound . . . 6
compute_info_at_time . . . 7
cut_completers . . . 9
cut_data_by_date . . . 10
cut_date_for_completers . . . 11
estimate_nb_mom . . . 12
fit_nb_glmm . . . 13
get_analysis_date . . . 14
get_cut_date . . . 15
gsNBCalendar . . . 16
impute_nb . . . 19
impute_nb_composite . . . 22
impute_nb_mar . . . 23
impute_nb_mnar_ref . . . 24
mutze_test . . . 26
nb_sim . . . 28
nb_sim_seasonal . . . 30
print.gsNBsummary . . . 31
print.sample_size_nbinom_result . . . 32
print.sample_size_nbinom_summary . . . 32
run_ssr_shiny . . . 33
sample_size_nbinom . . . 34
sim_gs_nbinom . . . 38
sim_ssr_nbinom . . . 41
summarize_gs_sim . . . 45
summarize_ssr_sim . . . 46
summary.gsNB . . . 47
summary.sample_size_nbinom_result . . . 48
toInteger . . . 48
unblinded_ssr . . . 50
update_gsNB . . . 52

**Index** **54**

blinded_ssr *Blinded sample size re-estimation for recurrent events*

## Description

Estimates the blinded dispersion and event rate from aggregated interim data and calculates the required sample size to maintain power, assuming the planned treatment effect holds. This function supports constant rates (Friede & Schmidli 2010) and accommodates future extensions for time-varying rates (Schneider et al. 2013) by using the exposure-adjusted rate.

## Usage

```
blinded_ssr(
  data,
  ratio = 1,
  lambda1_planning,
  lambda2_planning,
  rr0 = 1,
  power = 0.8,
  alpha = 0.025,
  method = "friede",
  accrual_rate,
  accrual_duration,
  trial_duration,
  dropout_rate = 0,
  max_followup = NULL,
  event_gap = NULL
)
```

## Arguments

| | |
|---|---|
| data | A data frame containing the blinded interim data. Must include columns events (number of events) and tte (total exposure/follow-up time). This is typically the output of cut_data_by_date(). |
| ratio | Planned allocation ratio (experimental / control). Default is 1. |
| lambda1_planning | Planned event rate for the control group used in original calculation. |
| lambda2_planning | Planned event rate for the experimental group used in original calculation. |
| rr0 | Rate ratio under the null hypothesis (lambda2/lambda1). Default is 1. |
| power | Target power (1 - beta). Default is 0.8. |
| alpha | One-sided significance level. Default is 0.025. |
| method | Method for sample size recalculation. Currently "friede" (Friede & Schmidli 2010) is implemented, which uses the blinded nuisance parameter estimates. |
| accrual_rate | Vector of accrual rates (patients per unit time). |

`accrual_duration`
: Vector of durations for each accrual rate. Must be same length as `accrual_rate`.

`trial_duration`
: Total planned duration of the trial.

`dropout_rate`
: Dropout rate (hazard rate). Default is 0.

`max_followup`
: Maximum follow-up time for any patient. Default is NULL (infinite).

`event_gap`
: Gap duration after each event during which no new events are counted. Default is NULL (no gap).

## Value

A list containing:

**n_total_blinded** Re-estimated total sample size using blinded estimates.

**dispersion_blinded** Estimated dispersion parameter (k) from blinded data.

**lambda_blinded** Estimated overall event rate from blinded data.

**info_fraction** Estimated information fraction at interim (blinded information / target information).

**blinded_info** Estimated statistical information from the blinded interim data.

**target_info** Target statistical information required for the planned power.

## References

Friede, T., & Schmidli, H. (2010). Blinded sample size reestimation with count data: methods and applications in multiple sclerosis. *Statistics in Medicine*, 29(10), 1145–1156. doi:10.1002/sim.3861

Schneider, S., Schmidli, H., & Friede, T. (2013). Blinded sample size re-estimation for recurrent event data with time trends. *Statistics in Medicine*, 32(30), 5448–5457. doi:10.1002/sim.5977

## Examples

```
interim <- data.frame(events = c(1, 2, 1, 3), tte = c(0.8, 1.0, 1.2, 0.9))
blinded_ssr(
  interim,
  ratio = 1,
  lambda1_planning = 0.5,
  lambda2_planning = 0.3,
  power = 0.8,
  alpha = 0.025,
  accrual_rate = 10,
  accrual_duration = 12,
  trial_duration = 18
)
```

`calculate_blinded_info` *Calculate blinded statistical information*

## Description

Estimates the blinded dispersion $k$ and event rate $\lambda$ from pooled (blinded) interim data and calculates the observed statistical information $\mathcal{I}$ for the log rate ratio. The estimation follows the approach of Friede & Schmidli (2010): a single negative binomial model is fit to the pooled data, then the estimated overall rate is split into group-specific rates using the *planned* rate ratio.

## Usage

```
calculate_blinded_info(
  data,
  ratio = 1,
  lambda1_planning,
  lambda2_planning,
  event_gap = NULL
)
```

## Arguments

| | |
|---|---|
| `data` | A data frame containing the blinded interim data. Must include columns `events` (number of events) and `tte` (total exposure/follow-up time per subject). |
| `ratio` | Planned allocation ratio $r = n_2/n_1$. Default is 1. |
| `lambda1_planning` | Planned event rate $\lambda_1$ for the control group (used to determine the rate split). |
| `lambda2_planning` | Planned event rate $\lambda_2$ for the experimental group (used to determine the rate split). |
| `event_gap` | Optional gap duration (numeric). If provided, planning rates are adjusted to effective rates $\lambda_{\text{eff}} = \lambda/(1 + \lambda \cdot \text{gap})$ before computing the rate split. |

## Details

If the ML negative binomial fit fails to converge or produces an unreliable shape estimate, the function falls back to method-of-moments (MoM) estimation via `estimate_nb_mom()` rather than silently assuming $k = 0$. This avoids the anti-conservative behaviour that would result from treating overdispersed data as Poisson.

The statistical information is computed as:

$$\mathcal{I} = \frac{1}{1/W_1 + 1/W_2}$$

where $W_g = p_g \sum_i \mu_{g,i}/(1 + k\,\mu_{g,i})$ and $\mu_{g,i} = \lambda_g t_i$ is the expected count for subject $i$ if they belonged to group $g$.

## Value

A list containing:

**blinded_info** Estimated statistical information $\mathcal{I}$.

**dispersion_blinded** Estimated dispersion parameter $k$.

**lambda_blinded** Estimated overall (pooled) event rate.

**lambda1_adjusted** Re-estimated control rate $\hat{\lambda}_1$.

**lambda2_adjusted** Re-estimated experimental rate $\hat{\lambda}_2$.

**fallback** Character label describing which estimator was used (`"ml"` or `"mom"`).

## References

Friede, T., & Schmidli, H. (2010). Blinded sample size reestimation with negative binomial counts in superiority and non-inferiority trials. *Methods of Information in Medicine*, 49(06), 618–624. doi:10.3414/ME09020060

## See Also

`blinded_ssr()` for blinded sample size reestimation; `sample_size_nbinom()` for the underlying sample size formula.

## Examples

```
interim <- data.frame(events = c(1, 2, 1, 3), tte = c(0.8, 1.0, 1.2, 0.9))
calculate_blinded_info(
  interim,
  ratio = 1,
  lambda1_planning = 0.5,
  lambda2_planning = 0.3
)
```

---

# check_gs_bound *Check group sequential bounds*

---

## Description

Updates the group sequential design boundaries based on observed information and checks if boundaries have been crossed.

## Usage

```
check_gs_bound(
  sim_results,
  design,
  info_scale = c("blinded", "unblinded"),
  info_col = NULL
)
```

### Arguments

| | |
|---|---|
| sim_results | Data frame of simulation results (from sim_gs_nbinom()). |
| design | The planning gsNB object. |
| info_scale | Character. Legacy selector for "blinded" (default) or "unblinded" information. Ignored when info_col is supplied. |
| info_col | Optional explicit column name containing the information metric to use for bounds, e.g. "info_unblinded_ml" or "info_blinded_mom". |

### Value

A data frame with added columns:

**cross_upper** Logical, true if upper bound crossed (efficacy)

**cross_lower** Logical, true if lower bound crossed (futility)

**cross_harm** Logical, true if harm bound crossed (test.type 7 or 8)

### Examples

```
design <- gsDesign::gsDesign(k = 2, n.fix = 100, test.type = 2, timing = c(0.5, 1))
sim_df <- data.frame(
  sim = c(1, 1, 2, 2),
  analysis = c(1, 2, 1, 2),
  z_stat = c(2.5, NA, -0.2, 2.2),
  blinded_info = c(50, 100, 50, 100),
  unblinded_info = c(50, 100, 50, 100)
)
check_gs_bound(sim_df, design)
check_gs_bound(sim_df, design, info_col = "unblinded_info")
```

---

## compute_info_at_time *Compute statistical information at analysis time*

---

### Description

Computes the statistical information $\mathcal{I}$ for the log rate ratio $\theta = \log(\lambda_2/\lambda_1)$ at a given calendar analysis time, accounting for staggered enrollment, dropout, maximum follow-up, and event gaps.

### Usage

```
compute_info_at_time(
  analysis_time,
  accrual_rate,
  accrual_duration,
  lambda1,
  lambda2,
  dispersion,
```

```
  ratio = 1,
  dropout_rate = 0,
  event_gap = 0,
  max_followup = Inf
)
```

## Arguments

| | |
|---|---|
| `analysis_time` | Calendar time of the analysis. |
| `accrual_rate` | Enrollment rate (subjects per time unit). |
| `accrual_duration` | Duration of the enrollment period. |
| `lambda1` | Event rate $\lambda_1$ for group 1 (control). |
| `lambda2` | Event rate $\lambda_2$ for group 2 (treatment). |
| `dispersion` | Dispersion parameter $k$ such that $\mathrm{Var}(Y) = \mu + k\mu^2$. Can be a vector of length 2. |
| `ratio` | Allocation ratio $r = n_2/n_1$. Default is 1. |
| `dropout_rate` | Dropout hazard rate. Default is 0. Can be a vector of length 2 for group-specific rates (control, treatment). |
| `event_gap` | Gap duration after each event. Default is 0. |
| `max_followup` | Maximum follow-up time per subject. Default is `Inf`. Can be a vector of length 2. |

## Details

This function delegates to `sample_size_nbinom()` with `power = NULL` and returns $\mathcal{I} = 1/\mathrm{Var}(\hat{\theta})$ from the resulting variance. This ensures full consistency with package design calculations, including piecewise accrual, dropout, max follow-up truncation, event-gap correction, and follow-up variability inflation ($Q_g$).

## Value

The statistical information $\mathcal{I}$ (inverse of variance) at the analysis time.

## Examples

```
compute_info_at_time(
  analysis_time = 12,
  accrual_rate = 10,
  accrual_duration = 10,
  lambda1 = 0.5,
  lambda2 = 0.3,
  dispersion = 0.1
)
```

`cut_completers` *Cut data for completers analysis*

## Description

Subsets the data to all subjects randomized by the specified date, and prepares the data for analysis. This is a wrapper for `cut_data_by_date()` typically used with a date determined by `cut_date_for_completers()`.

## Usage

```
cut_completers(data, cut_date, event_gap = 0)
```

## Arguments

| | |
|---|---|
| `data` | Data generated by `nb_sim()`. |
| `cut_date` | Calendar time (relative to trial start) at which to cut the data. |
| `event_gap` | Gap duration after each event during which no new events are counted. Can be a numeric value (default `0`) or a function returning a numeric value. The time at risk is reduced by the sum of these gaps (truncated by the cut date). |

## Value

A data frame with one row per subject randomized prior to `cut_date`. Contains the truncated follow-up time (`tte`) and total number of observed events (`events`).

## Examples

```
enroll_rate <- data.frame(rate = 20 / (5 / 12), duration = 5 / 12)
fail_rate <- data.frame(treatment = c("Control", "Experimental"), rate = c(0.5, 0.3))
dropout_rate <- data.frame(
  treatment = c("Control", "Experimental"),
  rate = c(0.1, 0.05), duration = c(100, 100)
)
sim <- nb_sim(enroll_rate, fail_rate, dropout_rate, max_followup = 2, n = 20)
# Find date when 5 subjects have completed
date_5 <- cut_date_for_completers(sim, 5)
# Get analysis dataset for this cut date (includes partial follow-up)
cut_completers(sim, date_5)
```

`cut_data_by_date` *Cut simulated trial data at a calendar date*

## Description

Censors follow-up at a specified calendar time and aggregates events per subject. Returns one row per subject randomized before the cut date, with the total number of observed events and follow-up times.

## Usage

```
cut_data_by_date(data, cut_date, event_gap = 0, ...)

## Default S3 method:
cut_data_by_date(data, cut_date, event_gap = 0, ...)

## S3 method for class 'nb_sim_data'
cut_data_by_date(data, cut_date, event_gap = 0, ...)

## S3 method for class 'nb_sim_seasonal'
cut_data_by_date(data, cut_date, event_gap = 0, ...)
```

## Arguments

| | |
|---|---|
| `data` | Data generated by `nb_sim()`. |
| `cut_date` | Calendar time (relative to trial start) at which to censor follow-up. |
| `event_gap` | Gap duration after each event during which no new events are counted. Can be a numeric value (default `0`) or a function returning a numeric value. The time at risk is reduced by the sum of these gaps (truncated by the cut date). |
| `...` | Additional arguments passed to methods. |

## Value

A data frame with one row per subject randomized prior to `cut_date` containing:

**id** Subject identifier

**treatment** Treatment group

**enroll_time** Time of enrollment relative to trial start

**tte** Time at risk (total follow-up minus event gap periods)

**tte_total** Total follow-up time (calendar time, not adjusted for gaps)

**events** Number of observed events

A data frame with one row per subject randomized prior to `cut_date`. This method stops with an error for unsupported classes.

A data frame with one row per subject randomized prior to `cut_date`. Includes total events and follow-up time within the cut window.

A data frame with one row per subject randomized prior to `cut_date`. Includes season and follow-up time within the cut window.

### Methods (by class)

- `cut_data_by_date(default)`: Default method.
- `cut_data_by_date(nb_sim_data)`: Method for `nb_sim` data.
- `cut_data_by_date(nb_sim_seasonal)`: Method for `nb_sim_seasonal` data.

### Examples

```
enroll_rate <- data.frame(rate = 20 / (5 / 12), duration = 5 / 12)
fail_rate <- data.frame(treatment = c("Control", "Experimental"), rate = c(0.5, 0.3))
dropout_rate <- data.frame(
  treatment = c("Control", "Experimental"),
  rate = c(0.1, 0.05), duration = c(100, 100)
)
sim <- nb_sim(enroll_rate, fail_rate, dropout_rate, max_followup = 2, n = 20)
cut_data_by_date(sim, cut_date = 1)
```

---

## `cut_date_for_completers` *Find calendar date for target completer count*

---

### Description

Finds the calendar time (since start of randomization) at which a specified number of subjects have completed their follow-up.

### Usage

```
cut_date_for_completers(data, target_completers)
```

### Arguments

| | |
|---|---|
| `data` | A data frame of simulated data, typically from `nb_sim()` or `nb_sim_seasonal()`. |
| `target_completers` | Integer. The target number of completers. |

### Value

Numeric. The calendar date when `target_completers` is achieved. If the dataset contains fewer than `target_completers` completers, returns the maximum calendar time in the dataset and prints a message.

## Examples

```
enroll_rate <- data.frame(rate = 20 / (5 / 12), duration = 5 / 12)
fail_rate <- data.frame(treatment = c("Control", "Experimental"), rate = c(0.5, 0.3))
dropout_rate <- data.frame(
  treatment = c("Control", "Experimental"),
  rate = c(0.1, 0.05), duration = c(100, 100)
)
sim <- nb_sim(enroll_rate, fail_rate, dropout_rate, max_followup = 2, n = 20)
cut_date_for_completers(sim, target_completers = 5)
```

---

# estimate_nb_mom *Method of Moments Estimation for Negative Binomial Parameters*

---

## Description

Estimates the event rate(s) and common dispersion parameter (k) for negative binomial count data using the method of moments. This is a robust alternative to Maximum Likelihood Estimation (MLE), especially when MLE fails to converge or produces boundary estimates.

## Usage

```
estimate_nb_mom(data, group = NULL)
```

## Arguments

| | |
|---|---|
| `data` | A data frame containing the data. Must include columns `events` (number of events) and `tte` (total exposure/follow-up time). |
| `group` | Optional character string specifying the grouping column name (e.g., "treatment"). If provided, rates are estimated separately for each group, while a common dispersion parameter is estimated across groups. If NULL (default), a single rate and dispersion are estimated (blinded case). |

## Details

The method of moments estimator for the dispersion parameter $k$ is derived by equating the theoretical variance to the observed second central moment, accounting for varying exposure times.

For a given group with rate $\lambda$, the expected count for subject $i$ is $\mu_i = \lambda t_i$. The variance is $V_i = \mu_i + k\mu_i^2$. The estimator is calculated as:

$$\hat{k} = \max\left(0, \frac{\sum(y_i - \hat{\mu}_i)^2 - \sum y_i}{\sum \hat{\mu}_i^2}\right)$$

where $y_i$ is the number of events, $t_i$ is the exposure time, and $\hat{\mu}_i = \hat{\lambda} t_i$ is the estimated expected count.

When multiple groups are present, the numerator and denominator are summed across all groups to estimate a common $k$.

## Value

A list containing:

| | |
|---|---|
| lambda | Estimated event rate(s). A single numeric value if group is NULL, or a named vector if group is provided. |
| dispersion | Estimated common dispersion parameter (k). |

## Examples

```
# Blinded estimation (single group)
df <- data.frame(events = c(1, 2, 0, 3), tte = c(1, 1.2, 0.5, 1.5))
estimate_nb_mom(df)

# Unblinded estimation (two groups)
df_group <- df
df_group$group <- c("A", "A", "B", "B")
estimate_nb_mom(df_group, group = "group")
```

---

# fit_nb_glmm *Fit a negative binomial GLMM for count imputation*

---

## Description

Fits a negative binomial generalized linear mixed model (GLMM) to longitudinal count data using `glmmTMB::glmmTMB()` with `family = nbinom2(link = "log")`. The fitted model and the estimated dispersion parameter $k$ (where Var(Y) = $\mu + k\mu^2$) are returned for use in subsequent imputation steps. This mirrors PROC GLIMMIX with `dist=negbin link=log` in SAS.

## Usage

```
fit_nb_glmm(data, formula, replicate_col = NULL)
```

## Arguments

| | |
|---|---|
| data | Data frame of **observed** (non-missing) records. |
| formula | A two-sided formula specifying fixed and random effects, e.g. `count ~ base + trt + visit + (1 \| id)`. The left-hand side should be the outcome variable; rows with a missing outcome should be excluded before calling this function (see `impute_nb()`). |
| replicate_col | Character or `NULL`. Column in `data` identifying bootstrap replicates. The model is fitted separately within each unique value. If `NULL` (default), all rows form a single fit. |

## Details

The dispersion parameter follows the parameterisation used throughout **gsDesignNB**: $k$ such that $\text{Var}(Y) = \mu + k\mu^2$. This matches `glmmTMB`'s `nbinom2` family, where `glmmTMB::sigma()` returns $k$ directly.

## Value

A named list—one element per unique replicate—where each element contains:

`model` The fitted `glmmTMB` object.

`k` Estimated NB dispersion $k$ (= `glmmTMB::sigma(model)`).

When `replicate_col = NULL`, the list has a single element named `"1"`.

## Examples

```
## Not run:
# Requires glmmTMB
obs_data <- long_data[!is.na(long_data$count), ]
fits <- fit_nb_glmm(
  data    = obs_data,
  formula = count ~ baseline + trt + visit + (1 | id)
)
fits[["1"]]$k  # estimated dispersion

## End(Not run)
```

---

# get_analysis_date *Find calendar date for target event count*

---

## Description

Finds the calendar time (since start of randomization) at which a specified total number of events is reached in the simulated dataset.

## Usage

```
get_analysis_date(data, planned_events, event_gap = 5/365.25)
```

## Arguments

| | |
|---|---|
| `data` | A data frame of simulated data, typically from `nb_sim()`. |
| `planned_events` | Integer. The target number of events. |
| `event_gap` | Gap duration after each event during which no new events are counted. Can be a numeric value (default `5 / 365.25`) or a function returning a numeric value. |

## Value

Numeric. The calendar date when `planned_events` is achieved. If the dataset contains fewer than `planned_events`, returns the maximum calendar time in the dataset and prints a message.

## Examples

```
enroll_rate <- data.frame(rate = 20 / (5 / 12), duration = 5 / 12)
fail_rate <- data.frame(treatment = c("Control", "Experimental"), rate = c(0.5, 0.3))
dropout_rate <- data.frame(
  treatment = c("Control", "Experimental"),
  rate = c(0.1, 0.05), duration = c(100, 100)
)
sim <- nb_sim(enroll_rate, fail_rate, dropout_rate, max_followup = 2, n = 40)
get_analysis_date(sim, planned_events = 15)
```

---

# get_cut_date *Determine analysis date based on criteria*

---

## Description

Finds the earliest calendar date at which all specified criteria are met. Criteria can include a specific calendar date, a target number of events, a target number of completers, or a target amount of blinded information.

## Usage

```
get_cut_date(
  data,
  planned_calendar = NULL,
  target_events = NULL,
  target_completers = NULL,
  target_info = NULL,
  event_gap = 0,
  ratio = 1,
  lambda1 = NULL,
  lambda2 = NULL,
  min_date = 0,
  max_date = Inf
)
```

## Arguments

| | |
|---|---|
| `data` | A data frame of simulated data (from `nb_sim()`). |
| `planned_calendar` | Numeric. Target calendar time. |
| `target_events` | Integer. Target number of observed events. |
| `target_completers` | Integer. Target number of subjects with complete follow-up. |
| `target_info` | Numeric. Target blinded information. |
| `event_gap` | Numeric. Gap duration for event counting and info calculation. |
| `ratio` | Numeric. Randomization ratio (experimental/control) for info calculation. |

| | |
|---|---|
| lambda1 | Numeric. Planned control rate for info calculation. |
| lambda2 | Numeric. Planned experimental rate for info calculation. |
| min_date | Numeric. Minimum possible date (e.g., 0 or previous analysis time). |
| max_date | Numeric. Maximum possible date (e.g., trial duration). |

## Value

Numeric. The calendar date satisfying the criteria. If criteria cannot be met within max_date (or data limits), returns max_date (or max data time).

## Examples

```
set.seed(456)
enroll_rate <- data.frame(rate = 15, duration = 1)
fail_rate <- data.frame(
  treatment = c("Control", "Experimental"),
  rate = c(0.6, 0.4)
)
sim_data <- nb_sim(enroll_rate, fail_rate, max_followup = 1, n = 20)
get_cut_date(sim_data, planned_calendar = 0.5, target_events = 5, event_gap = 0)
```

---

# gsNBCalendar *Group sequential design for negative binomial outcomes*

---

## Description

Creates a group sequential design for negative binomial outcomes based on sample size calculations from sample_size_nbinom().

## Usage

```
gsNBCalendar(
  x,
  k = 3,
  test.type = 4,
  alpha = 0.025,
  beta = 0.1,
  astar = 0,
  delta = 0,
  sfu = gsDesign::sfHSD,
  sfupar = -4,
  sfl = gsDesign::sfHSD,
  sflpar = -2,
  sfharm = gsDesign::sfHSD,
  sfharmparam = -2,
  testUpper = TRUE,
  testLower = TRUE,
```

```
  testHarm = TRUE,
  tol = 1e-06,
  r = 18,
  usTime = NULL,
  lsTime = NULL,
  analysis_times = NULL
)
```

## Arguments

| | |
|---|---|
| x | An object of class `sample_size_nbinom_result` from `sample_size_nbinom()`. |
| k | Number of analyses (interim + final). Default is 3. |
| test.type | Test type as in `gsDesign::gsDesign()`:<br>**1** One-sided<br>**2** Two-sided symmetric<br>**3** Two-sided, asymmetric, binding futility bound, beta-spending<br>**4** Two-sided, asymmetric, non-binding futility bound, beta-spending<br>**5** Two-sided, asymmetric, binding futility bound, lower spending<br>**6** Two-sided, asymmetric, non-binding futility bound, lower spending<br>**7** Two-sided, asymmetric, binding futility and binding harm bounds<br>**8** Two-sided, asymmetric, non-binding futility and non-binding harm bounds<br>Default is 4. |
| alpha | Type I error (one-sided). Default is 0.025. |
| beta | Type II error (1 - power). Default is 0.1. |
| astar | For test.type 5 or 6, allocated Type I error for the lower bound. For test.type 7 or 8, total probability of crossing the harm bound under the null. Default is 0 (set to `1 - alpha` internally when needed). |
| delta | Standardized effect size. Default is 0 (computed from design). |
| sfu | Spending function for upper bound. Default is `gsDesign::sfHSD`. |
| sfupar | Parameter for upper spending function. Default is -4. |
| sfl | Spending function for lower bound. Default is `gsDesign::sfHSD`. |
| sflpar | Parameter for lower spending function. Default is -2. |
| sfharm | Spending function for the harm bound (test.type 7 or 8). Default is `gsDesign::sfHSD`. |
| sfharmparam | Parameter for the harm spending function. Default is -2. |
| testUpper | Logical scalar or vector of length `k` specifying which analyses include an upper (efficacy) bound. `TRUE` (default) means all analyses. Where `FALSE`, the bound is inactive and displayed as `NA`. Must be `TRUE` at the final analysis. |
| testLower | Logical scalar or vector of length `k` specifying which analyses include a lower (futility) bound. `TRUE` (default) means all analyses. Where `FALSE`, the bound is inactive and displayed as `NA`. Ignored for test.type 1. |
| testHarm | Logical scalar or vector of length `k` specifying which analyses include a harm bound (test.type 7 or 8 only). `TRUE` (default) means all analyses. Where `FALSE`, the bound is inactive and displayed as `NA`. |

| | |
|---|---|
| tol | Tolerance for convergence. Default is 1e-06. |
| r | Integer controlling grid size for numerical integration. Default is 18. |
| usTime | Spending time for upper bound (optional). |
| lsTime | Spending time for lower bound (optional). |
| analysis_times | Vector of calendar times for each analysis. Must have length k. These times are stored in the T element and displayed by gsDesign::gsBoundSummary(). |

## Value

An object of class gsNB which inherits from gsDesign and sample_size_nbinom_result. While the final sample size would be planned total enrollment, interim analysis sample sizes are the expected number enrolled at the times specified in analysis_times. Output value contains all elements from gsDesign::gsDesign() plus:

**nb_design** The original sample_size_nbinom_result object

**n1** A vector with sample size per analysis for group 1

**n2** A vector with sample size per analysis for group 2

**n_total** A vector with total sample size per analysis

**events** A vector with expected total events per analysis

**events1** A vector with expected events per analysis for group 1

**events2** A vector with expected events per analysis for group 2

**exposure** A vector with expected average calendar exposure per analysis

**exposure_at_risk1** A vector with expected at-risk exposure per analysis for group 1

**exposure_at_risk2** A vector with expected at-risk exposure per analysis for group 2

**variance** A vector with variance of log rate ratio per analysis

**T** Calendar time at each analysis (if analysis_times provided)

**testUpper** Logical vector indicating which analyses have an efficacy bound

**testLower** Logical vector indicating which analyses have a futility bound

**testHarm** Logical vector indicating which analyses have a harm bound (test.type 7 or 8 only)

Note that n.I in the returned object represents the statistical information at each analysis.

## References

Jennison, C. and Turnbull, B.W. (2000), *Group Sequential Methods with Applications to Clinical Trials*. Boca Raton: Chapman and Hall.

## Examples

```
# First create a sample size calculation
nb_ss <- sample_size_nbinom(
  lambda1 = 0.5, lambda2 = 0.3, dispersion = 0.1, power = 0.9,
  accrual_rate = 10, accrual_duration = 20, trial_duration = 24
)
```

```
# Then create a group sequential design with analysis times
gs_design <- gsNBCalendar(nb_ss,
  k = 3, test.type = 4,
  analysis_times = c(10, 18, 24)
)

# Selective bound testing: futility only at IA1, efficacy deferred to IA2+
gs_selective <- gsNBCalendar(nb_ss,
  k = 3, test.type = 4,
  analysis_times = c(10, 18, 24),
  testUpper = c(FALSE, TRUE, TRUE),
  testLower = c(TRUE, TRUE, FALSE)
)
gs_selective
```

---

`impute_nb` *Multiple imputation for longitudinal negative binomial counts*

---

## Description

Orchestrates the full multiple imputation (MI) pipeline for longitudinal recurrent-event count data with negative binomial overdispersion:

## Usage

```
impute_nb(
  data,
  formula,
  outcome_col,
  miss_flag_col,
  baseline_col,
  trt_col,
  reference_trt,
  subject_col,
  strata_cols = NULL,
  mar_values = "MAR",
  mnar_value = "MNAR",
  composite_value = "Comp",
  n_imp = 5L,
  n_boot = 1L,
  seed = NULL
)
```

## Arguments

| | |
|---|---|
| `data` | Data frame in long format (one row per subject × visit). |
| `formula` | Two-sided formula passed to `fit_nb_glmm()`, specifying fixed and random effects. The left-hand side should be the outcome variable (with `NA` for missing observations). Example: `count ~ baseline + trt + visit + (1 \| id)`. |

| | |
|---|---|
| outcome_col | Character. Column name of the count outcome. |
| miss_flag_col | Character. Column name of the missingness mechanism flag. Values in this column control which imputation strategy is applied: mar_values, mnar_value, or composite_value. Rows with NA in this column are treated as complete (observed). |
| baseline_col | Character. Column name of the baseline count used by the composite strategy. |
| trt_col | Character. Column name of the treatment group. |
| reference_trt | Value in trt_col identifying the reference (comparator) arm. |
| subject_col | Character. Column name of the subject identifier (cluster unit for bootstrap resampling). |
| strata_cols | Character vector of column names used to stratify the bootstrap resampling. Default NULL (no stratification). |
| mar_values | Character vector. Values of miss_flag_col treated as MAR. Default "MAR". |
| mnar_value | Character. Value of miss_flag_col treated as MNAR (triggers reference-based imputation for non-reference arms). Default "MNAR". |
| composite_value | Character. Value of miss_flag_col that triggers the composite strategy (baseline carry-forward for missing rows). Default "Comp". |
| n_imp | Integer. Number of imputations per bootstrap replicate. Default 5L. |
| n_boot | Integer. Number of bootstrap replicates. Default 1L (no resampling; a single GLMM is fitted to the original data). |
| seed | Integer or NULL. Random seed for reproducibility. Default NULL. |

## Details

1. **Bootstrap resampling** (optional): cluster-level (subject-level) stratified resampling with replacement, creating n_boot replicates. This propagates estimation uncertainty into the imputed values, mirroring the PROC SURVEYSELECT method=urs cluster=USUBJID step in the SAS macro.
2. **GLMM fitting**: a negative binomial GLMM is fitted to the observed (non-missing) rows of each replicate via fit_nb_glmm().
3. **Imputation by mechanism**:
   - *MAR* rows: predicted mean with subject BLUPs → Gamma–Poisson draw.
   - *MNAR reference-arm* rows: same as MAR (reference arm has no "better" treatment to copy from).
   - *MNAR non-reference-arm* rows: reference-based (copy-reference) imputation. The counterfactual mean is the fixed-effects-only prediction under the reference arm multiplied by the subject's random-effect ratio (BLUP prediction / FE prediction on the response scale). See impute_nb_mnar_ref().
   - *Composite ICE* rows: missing value set to baseline count. See impute_nb_composite().
4. Returns a long-format data frame with one row per original observation × bootstrap replicate × imputation.

**Relationship between bootstrap and MI:**

Setting `n_boot > 1` combines bootstrap and MI ("boot-MI"), which yields a valid variance estimator without requiring Rubin's rules. Setting `n_boot = 1` produces conventional MI; apply Rubin's rules to the `n_imp` imputed datasets when pooling.

**Formula and GLMM specification:**

The formula is passed directly to `glmmTMB::glmmTMB()`. A typical formula mirrors the PROC GLIMMIX model:

```
outcome ~ baseline + strat1 + strat2 + trt + visit + param + (1 | id)
```

The original SAS model also included an unstructured residual covariance across visits within `id:param`:

```
+ (0 + visit | id:param)
```

Complex random-effect structures may cause convergence issues; start with a random intercept only and add complexity as needed.

**Composite strategy:**

The composite strategy applies only to **missing** post-ICE rows (`is.na(outcome_col)` must be `TRUE`). Observed rows with `miss_flag_col == composite_value` are left unchanged.

## Value

A data frame with all columns from `data` plus:

`replicate` Bootstrap replicate index (1 to `n_boot`).

`imputation` Imputation index (1 to `n_imp`).

`imputed_value` Imputed count. Equals the observed value for non-missing rows; contains imputed draws for missing rows.

The total number of rows is `nrow(data) * n_boot * n_imp`.

## Examples

```
## Not run:
# Requires glmmTMB
result <- impute_nb(
  data          = long_data,
  formula       = count ~ baseline + trt + visit + (1 | id),
  outcome_col   = "count",
  miss_flag_col = "miss_flag",
  baseline_col  = "baseline",
  trt_col       = "trt",
  reference_trt = 0L,
  subject_col   = "id",
  strata_cols   = c("trt", "strat1"),
  mar_values    = "MAR",
  mnar_value    = "MNAR",
  composite_value = "Comp",
  n_imp         = 5L,
```

```
  n_boot        = 10L,
  seed          = 42L
)
head(result[!is.na(result$miss_flag), ])

## End(Not run)
```

---

`impute_nb_composite` *Apply the composite ICE strategy: replace post-ICE outcomes with baseline*

---

## Description

For subjects whose missingness flag matches `composite_value`, all missing post-ICE count observations are set to the subject's baseline count. This implements the composite estimand strategy for intercurrent events such as death or treatment discontinuation due to disease worsening, where the event itself is incorporated into the outcome (e.g., baseline carried forward as a "worst case" placeholder).

## Usage

```
impute_nb_composite(
  data,
  outcome_col,
  imputed_value_col = "imputed_value",
  miss_flag_col,
  composite_value = "Comp",
  baseline_col
)
```

## Arguments

| | |
|---|---|
| `data` | Data frame. |
| `outcome_col` | Character. Column with the original count outcome; used to identify which rows are missing (`NA`). |
| `imputed_value_col` | Character. Column to update. If absent, it is created as a copy of `outcome_col` before the composite fill is applied. Default `"imputed_value"`. |
| `miss_flag_col` | Character. Column with the missingness flag. |
| `composite_value` | Character. Flag value triggering the composite strategy. Default `"Comp"`. |
| `baseline_col` | Character. Column with the baseline count used as the fill value. |

## Details

The function is intentionally simple and requires no model. It can be applied to a dataset already containing `imputed_value` from a prior MAR or MNAR imputation step, or directly to the original data.

## Value

Data frame with `imputed_value_col` updated for composite rows.

## Examples

```
df <- data.frame(
  count         = c(3L, NA,   NA,    5L),
  imputed_value = c(3L,  7L,  NA,    5L),
  miss_flag     = c(NA, "MAR", "Comp", NA),
  baseline      = c(4L,  4L,   4L,    6L)
)
impute_nb_composite(
  df,
  outcome_col     = "count",
  miss_flag_col   = "miss_flag",
  composite_value = "Comp",
  baseline_col    = "baseline"
)
```

---

`impute_nb_mar` *Impute missing counts under Missing at Random (MAR)*

---

## Description

For observations whose missingness flag matches `mar_values`, generates `n_imp` imputed counts using the GLMM predicted mean **including subject-level BLUPs**. Draws use the Gamma–Poisson compound distribution:

$$\lambda_i \sim \text{Gamma}(1/k,\ \hat{\mu}_i^{\text{BLUP}} k), \quad Y_i^{(m)} \mid \lambda_i \sim \text{Poisson}(\lambda_i).$$

## Usage

```
impute_nb_mar(
  data,
  fits,
  outcome_col,
  miss_flag_col,
  mar_values = "MAR",
  n_imp = 5L,
  replicate_col = NULL
)
```

## Arguments

| | |
|---|---|
| `data` | Data frame including all rows (observed and missing). |
| `fits` | Named list of fits as returned by `fit_nb_glmm()`. |
| `outcome_col` | Character. Column with the count outcome (may have `NA`). |

| | |
|---|---|
| miss_flag_col | Character. Column with the missingness flag. |
| mar_values | Character vector. Flag values treated as MAR. Default "MAR". In the reference-based framework, reference-arm MNAR subjects are also imputed as MAR (pass c("MAR", "MNAR") if desired). |
| n_imp | Integer. Number of imputations per replicate. Default 5. |
| replicate_col | Character or NULL. Replicate identifier column. |

## Details

This function handles one or more bootstrap replicates when a replicate_col is provided and fits contains one model per replicate. Observed rows (non-missing outcome) are passed through unchanged (imputed_value = observed value).

## Value

Data frame in long format with all original columns plus imputation (integer 1 to n_imp) and imputed_value (imputed count; equals the observed value for non-missing rows).

## Examples

```
## Not run:
fits    <- fit_nb_glmm(obs_data, count ~ base + trt + visit + (1 | id))
imp_mar <- impute_nb_mar(
  data          = long_data,
  fits          = fits,
  outcome_col   = "count",
  miss_flag_col = "miss_flag",
  mar_values    = "MAR",
  n_imp         = 5L
)

## End(Not run)
```

---

# impute_nb_mnar_ref *Impute missing counts under a reference-based MNAR assumption*

---

## Description

Implements the **copy-reference** (CR) strategy for observations whose missingness flag matches mnar_value in non-reference treatment arms. The imputation mean is the fixed-effects-only prediction under the reference arm, adjusted upward (or downward) by the subject's estimated random effect:

$$\hat{\mu}_i^{\text{cf}} = \hat{\mu}_i^{\text{FE, ref}} \times \frac{\hat{\mu}_i^{\text{BLUP}}}{\hat{\mu}_i^{\text{FE}}}.$$

This mirrors the SAS PROC PLM approach that re-predicts under the counterfactual treatment and then multiplies by the BLUP ratio on the response scale.

## Usage

```
impute_nb_mnar_ref(
  data,
  fits,
  outcome_col,
  miss_flag_col,
  mnar_value = "MNAR",
  trt_col,
  reference_trt,
  n_imp = 5L,
  replicate_col = NULL
)
```

## Arguments

| | |
|---|---|
| data | Data frame including all rows (observed and missing). |
| fits | Named list of fits as returned by fit_nb_glmm(). |
| outcome_col | Character. Column with the count outcome. |
| miss_flag_col | Character. Column with the missingness flag. |
| mnar_value | Character. Flag value identifying MNAR rows. Default "MNAR". |
| trt_col | Character. Column with the treatment assignment. |
| reference_trt | Value in trt_col that identifies the reference arm. |
| n_imp | Integer. Number of imputations per replicate. Default 5. |
| replicate_col | Character or NULL. Replicate identifier column. |

## Details

MNAR subjects already in the reference arm should be handled by impute_nb_mar() (MAR imputation is appropriate for the reference arm because there is no better arm to "copy from").

## Value

Data frame in long format with all original columns plus imputation and imputed_value. Only MNAR non-reference rows have counterfactual imputations; all other rows pass through unchanged.

## Examples

```
## Not run:
fits     <- fit_nb_glmm(obs_data, count ~ base + trt + visit + (1 | id))
imp_mnar <- impute_nb_mnar_ref(
  data          = long_data,
  fits          = fits,
  outcome_col   = "count",
  miss_flag_col = "miss_flag",
  mnar_value    = "MNAR",
  trt_col       = "trt",
  reference_trt = 0L,
```

```
  n_imp         = 5L
)

## End(Not run)
```

---

`mutze_test` *Wald or score test for treatment effect using negative binomial model*

---

## Description

Fits a negative binomial (or Poisson) log-rate model to the aggregated subject-level data produced by `cut_data_by_date()`. With `test_type = "wald"` (default), the method matches the Wald test described by Mutze et al. (2019). With `test_type = "score"`, the function fits only the null (no treatment effect) model and computes the score statistic, which evaluates all quantities under $H_0$ and avoids the finite-sample anti-conservatism of the Wald test.

## Usage

```
mutze_test(
  data,
  method = c("nb", "poisson"),
  test_type = c("wald", "score"),
  conf_level = 0.95,
  sided = 1,
  poisson_threshold = 50,
  mom_threshold = 20
)

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

## Arguments

| | |
|---|---|
| `data` | A data frame with at least the columns `treatment`, `events`, and `tte` (follow-up time). Typically output from `cut_data_by_date()`. |
| `method` | Type of model to fit: "nb" (default) uses a negative binomial GLM via `MASS::glm.nb()`, "poisson" fits a Poisson GLM. |
| `test_type` | Type of test statistic: `"wald"` (default) or `"score"`. The Wald test estimates the treatment effect under the alternative and divides by its standard error. The score test fits only the null model and evaluates the derivative of the log-likelihood at $\theta = 0$, avoiding estimation under the alternative. The score test typically has better finite-sample Type I error control and is faster because it only fits a one-parameter null model. |
| `conf_level` | Confidence level for the rate ratio interval. Default 0.95. |
| `sided` | Number of sides for the test: 1 (default) or 2. |

poisson_threshold
: Upper threshold (in units of `fit$theta`, the `MASS::glm.nb()` shape parameter $\theta_{\text{NB}} = 1/k$) above which the data are treated as essentially Poisson. Default is 50, corresponding to $\hat{k} < 0.02$.

mom_threshold
: Lower threshold on `fit$theta` below which the NB ML fit is considered unreliable (extreme overdispersion). Default is 20, corresponding to $\hat{k} > 20$.

x
: An object of class `mutze_test`.

...
: Additional arguments (currently ignored).

## Details

When the maximum likelihood negative binomial fit is unreliable, the test automatically switches to one of two statistically sensible fallbacks: a Poisson test when the data are essentially Poisson, or a method-of-moments (MoM) variance estimate plugged into the same negative binomial information formula when the data are extremely overdispersed or the ML fit fails to converge.

## Value

An object of class `mutze_test` containing:

- `method`: A string indicating the test method used.
- `estimate`: log rate ratio (experimental vs control). For `test_type = "score"`, this is a plug-in estimate.
- `se`: standard error for the log rate ratio.
- `z`: test statistic (Wald or score).
- `p_value`: one-sided or two-sided p-value.
- `rate_ratio`: estimated rate ratio and its confidence interval.
- `dispersion`: estimated dispersion on the $\theta = 1/k$ scale.
- `group_summary`: observed subjects/events/exposure per treatment.
- `fallback`: character label (`"ml"`, `"poisson"`, or `"mom"`).
- `test_type`: character label (`"wald"` or `"score"`).

Invisibly returns the input object.

## Methods (by generic)

- `print(mutze_test)`: Print method for `mutze_test` objects.

## Examples

```
enroll_rate <- data.frame(rate = 20 / (5 / 12), duration = 5 / 12)
fail_rate <- data.frame(treatment = c("Control", "Experimental"), rate = c(0.5, 0.3))
dropout_rate <- data.frame(
  treatment = c("Control", "Experimental"),
  rate = c(0.1, 0.05), duration = c(100, 100)
)
sim <- nb_sim(enroll_rate, fail_rate, dropout_rate, max_followup = 2, n = 40)
```

```
cut <- cut_data_by_date(sim, cut_date = 1.5)
mutze_test(cut)
mutze_test(cut, test_type = "score")
```

---

nb_sim *Simulate recurrent events with fixed follow-up*

---

## Description

Simulates recurrent events for a clinical trial with piecewise constant enrollment, exponential failure rates (Poisson process), and piecewise exponential dropout.

## Usage

```
nb_sim(
  enroll_rate,
  fail_rate,
  dropout_rate = NULL,
  max_followup = NULL,
  n = NULL,
  block = c(rep("Control", 2), rep("Experimental", 2)),
  event_gap = 0
)
```

## Arguments

| | |
|---|---|
| enroll_rate | A data frame with columns rate and duration defining the piecewise constant enrollment rates. |
| fail_rate | A data frame with columns treatment and rate defining the exponential failure rate for each treatment group. Optionally, a dispersion column can be provided to generate data from a negative binomial distribution. The dispersion parameter k is such that $\mathrm{Var}(Y) = \mu + k\mu^2$. |
| dropout_rate | A data frame with columns treatment, rate, and duration defining the piecewise constant dropout rates. |
| max_followup | Numeric. Maximum duration of follow-up for each individual (relative to their randomization time). |
| n | Total sample size. If NULL, it is estimated from enroll_rate. If provided, enrollment stops when n subjects are recruited. |
| block | Block vector for treatment allocation. Default is c(rep("Control", 2), rep("Experimental", 2)). If NULL, simple randomization is used (treatments are assigned with equal probability). If provided, it specifies the block structure, for example, c(rep("A", 2), rep("B", 2)) assigns 2 to group A and 2 to group B in each block. |
| event_gap | Numeric. Gap duration after each event during which no new events are counted. Default is 0. |

## Details

The simulation generates data consistent with the negative binomial models described by Friede and Schmidli (2010) and Mütze et al. (2019). Specifically, it simulates a Gamma-distributed frailty variable for each individual (if dispersion > 0), which acts as a multiplier for that individual's event rate. Events are then generated according to a Poisson process with this subject-specific rate.

More explicitly, for a subject with baseline rate $\lambda$ and exposure time $t$, the model used here is a Gamma–Poisson mixture:

$$\Lambda_i \sim \text{Gamma}(\text{shape} = 1/k,\ \text{scale} = k\lambda), \quad Y_i \mid \Lambda_i \sim \text{Poisson}(\Lambda_i t).$$

Marginally, $Y_i$ follows a negative binomial distribution with $\text{E}[Y_i] = \mu = \lambda t$ and $\text{Var}(Y_i) = \mu + k\mu^2$. This $k$ is the package dispersion parameter (and corresponds to $1/\theta$ in `MASS::glm.nb()` terminology).

## Value

A data frame (tibble) with columns:

**id** Subject identifier

**treatment** Treatment group

**enroll_time** Time of enrollment relative to trial start

**tte** Time to event or censoring relative to randomization

**calendar_time** Calendar time of event or censoring (enroll_time + tte)

**event** Binary indicator: 1 for event, 0 for censoring

Multiple rows per subject are returned (one for each event, plus one for the final censoring time).

## References

Friede, T., & Schmidli, H. (2010). Blinded sample size reestimation with count data: methods and applications in multiple sclerosis. *Statistics in Medicine*, 29(10), 1145–1156. doi:10.1002/sim.3861

Mütze, T., Glimm, E., Schmidli, H., & Friede, T. (2019). Group sequential designs for negative binomial outcomes. *Statistical Methods in Medical Research*, 28(8), 2326–2347. doi:10.1177/0962280218773115

## Examples

```
enroll_rate <- data.frame(rate = 20 / (5 / 12), duration = 5 / 12)
fail_rate <- data.frame(treatment = c("Control", "Experimental"), rate = c(0.5, 0.3))
dropout_rate <- data.frame(
  treatment = c("Control", "Experimental"),
  rate = c(0.1, 0.05), duration = c(100, 100)
)
sim <- nb_sim(enroll_rate, fail_rate, dropout_rate, max_followup = 2, n = 20)
head(sim)
```

nb_sim_seasonal *Simulate recurrent events with seasonal rates*

## Description

Simulates recurrent events where event rates depend on the season.

## Usage

```
nb_sim_seasonal(
  enroll_rate,
  fail_rate,
  dropout_rate = NULL,
  max_followup = NULL,
  randomization_start_date = NULL,
  n = NULL,
  block = c(rep("Control", 2), rep("Experimental", 2))
)
```

## Arguments

| | |
|---|---|
| enroll_rate | A data frame with columns rate and duration. |
| fail_rate | A data frame with columns treatment, season, rate, and optionally dispersion. Seasons should be "Spring", "Summer", "Fall", "Winter". |
| dropout_rate | A data frame with columns treatment, rate, duration. |
| max_followup | Numeric. Max follow-up duration (years). |
| randomization_start_date | Date. Start of randomization. |
| n | Integer. Total sample size. |
| block | Character vector for block randomization. |

## Value

A data frame of class nb_sim_seasonal with columns: id, treatment, season, enroll_time, start, end, event, calendar_start, calendar_end. Rows represent intervals of risk or events. event=1 indicates an event at end. event=0 indicates censoring or end of a seasonal interval at end.

## Examples

```
enroll_rate <- data.frame(rate = 20 / (5 / 12), duration = 5 / 12)
fail_rate <- data.frame(
  treatment = rep(c("Control", "Experimental"), each = 4),
  season = rep(c("Winter", "Spring", "Summer", "Fall"), times = 2),
  rate = c(0.6, 0.5, 0.4, 0.5, 0.4, 0.3, 0.2, 0.3)
)
sim <- nb_sim_seasonal(
```

```
  enroll_rate = enroll_rate,
  fail_rate = fail_rate,
  max_followup = 1,
  randomization_start_date = as.Date("2020-01-01"),
  n = 20
)
head(sim)
```

---

`print.gsNBsummary` *Print method for gsNBsummary objects*

---

## Description

Print method for gsNBsummary objects

## Usage

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

## Arguments

| | |
|---|---|
| `x` | An object of class `gsNBsummary`. |
| `...` | Additional arguments (currently ignored). |

## Value

Invisibly returns the input object.

## Examples

```
nb_ss <- sample_size_nbinom(
  lambda1 = 0.5, lambda2 = 0.3, dispersion = 0.1, power = 0.9,
  accrual_rate = 10, accrual_duration = 20, trial_duration = 24
)
gs_design <- gsNBCalendar(nb_ss, k = 3, analysis_times = c(12, 18, 24))
s <- summary(gs_design)
print(s)
```

## print.sample_size_nbinom_result

*Print method for sample_size_nbinom_result objects*

### Description

Prints a concise summary of the sample size calculation results.

### Usage

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

### Arguments

| | |
|---|---|
| x | An object of class sample_size_nbinom_result. |
| ... | Additional arguments (currently ignored). |

### Value

Invisibly returns the input object.

### Examples

```
x <- sample_size_nbinom(
  lambda1 = 0.5, lambda2 = 0.3, dispersion = 0.1, power = 0.8,
  accrual_rate = 10, accrual_duration = 20, trial_duration = 24
)
print(x)
```

## print.sample_size_nbinom_summary

*Print method for sample_size_nbinom_summary objects*

### Description

Print method for sample_size_nbinom_summary objects

### Usage

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

### Arguments

| | |
|---|---|
| x | An object of class sample_size_nbinom_summary. |
| ... | Additional arguments (currently ignored). |

### Value

Invisibly returns the input object.

### Examples

```
x <- sample_size_nbinom(
  lambda1 = 0.5, lambda2 = 0.3, dispersion = 0.1, power = 0.8,
  accrual_rate = 10, accrual_duration = 20, trial_duration = 24
)
s <- summary(x)
print(s)
```

---

## run_ssr_shiny *Launch the SSR Shiny prototype*

---

### Description

Opens a lightweight Shiny interface for interactive exploration of adaptive sample size re-estimation (SSR) scenarios. The app is a thin wrapper over sample_size_nbinom(), gsNBCalendar(), sim_ssr_nbinom(), and summarize_ssr_sim(), so the statistical computations remain in the package.

### Usage

```
run_ssr_shiny(
  display.mode = c("normal", "showcase"),
  launch.browser = interactive()
)
```

### Arguments

| | |
|---|---|
| display.mode | Character; passed to shiny::runApp(). Defaults to "normal". |
| launch.browser | Logical; passed to shiny::runApp(). Defaults to interactive(). |

### Value

Invisibly returns the Shiny app object.

## Examples

```
## Not run:
run_ssr_shiny()

## End(Not run)
```

---

`sample_size_nbinom` *Sample size calculation for negative binomial outcomes*

---

## Description

Computes the sample size (or power) for comparing two treatment groups assuming negative binomial distributed event counts. When `test_type = "wald"` (default), the formula uses a single variance evaluated under the alternative, corresponding to Method 3 of Zhu & Lakkis (2014) and the formulas of Friede & Schmidli (2010) and Mutze et al. (2019). When `test_type = "score"`, separate null and alternative variances are used (Farrington & Manning style), aligning the calculation with the null-variance scale of the score test. In practice, the final test statistic affects Type I error more than the small difference between Wald and score sizing, so score-test designs should be checked by simulation for both Type I error and power.

## Usage

```
sample_size_nbinom(
  lambda1,
  lambda2,
  dispersion,
  power = NULL,
  alpha = 0.025,
  sided = 1,
  ratio = 1,
  rr0 = 1,
  accrual_rate,
  accrual_duration,
  trial_duration,
  dropout_rate = 0,
  max_followup = NULL,
  test_type = c("wald", "score"),
  event_gap = NULL
)
```

## Arguments

| | |
|---|---|
| `lambda1` | Event rate for group 1 (control), in events per unit time. |
| `lambda2` | Event rate for group 2 (treatment), in events per unit time. |

`dispersion` Dispersion parameter $k$ such that $\mathrm{Var}(Y) = \mu + k\mu^2$. Equivalent to `1/size` in `stats::rnbinom()`. Can be a scalar (common dispersion) or a vector of length 2 (group-specific: control, treatment).

`power` Target power ($1 - \beta$). If `NULL`, power is computed for the given accrual rates (no sample size scaling). Default is 0.9.

`alpha` Significance level. Default is 0.025.

`sided` Number of sides for the test: 1 (one-sided) or 2 (two-sided). Default is 1.

`ratio` Allocation ratio $r = n_2/n_1$. Default is 1 (equal allocation).

`rr0` Rate ratio under the null hypothesis ($\lambda_2/\lambda_1$). Default is 1 (superiority). For non-inferiority, use a value > 1 (e.g., 1.1). For super-superiority, use a value < 1 (e.g., 0.8).

`accrual_rate` Vector of accrual rates (patients per unit time) for each recruitment segment.

`accrual_duration` Vector of durations for each accrual segment. Must be the same length as `accrual_rate`.

`trial_duration` Total planned duration of the trial. If `trial_duration` is less than the sum of `accrual_duration`, accrual is truncated at `trial_duration`.

`dropout_rate` Dropout hazard rate. Can be:

- A scalar (common constant rate for both groups). Default is 0.
- A vector of length 2 (group-specific constant rates: control, treatment).
- A data frame with columns `rate` and `duration` (and optionally `treatment`) defining piecewise constant dropout hazards. When a `treatment` column is present, use 1 for control and 2 for treatment. Without a `treatment` column, the same piecewise schedule applies to both groups. The last `duration` may be `Inf` to extend the final rate indefinitely.

`max_followup` Maximum follow-up time for any patient. Default is `NULL` (infinite). Can be a vector of length 2 for group-specific caps.

`test_type` Type of test for which to size the study: `"wald"` (default) uses a single variance under the alternative; `"score"` uses separate null and alternative variances ($z_\alpha\sqrt{V_0} + z_\beta\sqrt{V_1}$).

`event_gap` Gap duration after each event during which no new events are counted (e.g., a recovery period). Default is `NULL` (no gap). When specified, the effective rate is reduced to $\lambda_{\text{eff}} = \lambda/(1 + \lambda \cdot \text{gap})$.

## Details

**Sample size formula:**

**Wald test** (`test_type = "wald"`):

$$n_1 = \frac{(z_{\alpha/s} + z_\beta)^2 V_1}{(\theta - \theta_0)^2}$$

**Score test** (`test_type = "score"`):

$$n_1 = \frac{(z_{\alpha/s}\sqrt{V_0} + z_\beta\sqrt{V_1})^2}{(\theta - \theta_0)^2}$$

where $\theta = \log(\lambda_2/\lambda_1)$, $\theta_0 = \log(\mathrm{rr}_0)$, and:

$$V_1 = \left(\frac{1}{\mu_1} + k_1\right) + \frac{1}{r}\left(\frac{1}{\mu_2} + k_2\right)$$

is the variance under $H_1$. Under $H_0$ (pooled rate $\lambda_0 = (\lambda_1 + r\lambda_2\mathrm{rr}_0)/(1+r)$):

$$V_0 = \left(\frac{1}{\mu_0} + k_0\right)\left(1 + \frac{1}{r}\right)$$

with $\mu_g = \lambda_g \bar{t}_g$ the expected event count and $\bar{t}_g$ the average exposure for group $g$.
In superiority settings, the traditional Wald/Zhu-Lakkis sample size may be slightly larger than score sizing and can provide a useful power margin when the final analysis uses the score test. Compare both sizing rules and verify the chosen design with simulation when finite-sample calibration matters.

**Average exposure:**
The average exposure $\bar{t}_g$ accounts for piecewise accrual, piecewise exponential dropout, and maximum follow-up truncation. With piecewise constant dropout hazards $\delta_1, \delta_2, \ldots$ over successive intervals, the survival function is $S(t) = \exp(-\sum_j \delta_j \ell_j)$ where $\ell_j$ is the time spent in interval $j$. The expected exposure for a patient with potential follow-up $u$ is $m(u) = \int_0^u S(t)\,dt$, computed as a sum of exponential integrals over each piece. For a single constant rate $\delta > 0$ this simplifies to $m(u) = (1 - e^{-\delta u})/\delta$. The overall average is a weighted mean across accrual segments.

**Variance inflation:**
When follow-up times are variable, the dispersion is inflated by a factor $Q_g = \mathrm{E}[t_g^2]/(\mathrm{E}[t_g])^2 \geq 1$ (Zhu & Lakkis, 2014) to account for the non-linear dependence of the NB variance on exposure.

**Event gap correction (Jensen's inequality):**
When `event_gap` > 0, the naive effective rate $\lambda/(1+\lambda g)$ overestimates the true population-level effective rate because of subject-level heterogeneity (frailty). In the Gamma-Poisson mixture, each subject's rate $\Lambda_i \sim \mathrm{Gamma}(1/k, k\lambda)$ is random. Since $f(x) = x/(1+xg)$ is concave, Jensen's inequality gives $\mathrm{E}[f(\Lambda)] < f(\mathrm{E}[\Lambda])$.
A second-order Taylor correction is applied:

$$\lambda_{\mathrm{eff}} \approx \frac{\lambda}{1+\lambda g}\left(1 - \frac{k\lambda g}{(1+\lambda g)^2}\right)$$

This uses $f''(\lambda) = -2g/(1+\lambda g)^3$ and $\mathrm{Var}(\Lambda) = k\lambda^2$.

## Value

An object of class `sample_size_nbinom_result`, which is a list containing:

**inputs** Named list of the original function arguments.

**n1** Sample size for group 1 (control).

**n2** Sample size for group 2 (treatment).

**n_total** Total sample size ($n_1 + n_2$).

**alpha** Significance level used.

**sided** One-sided or two-sided test.

**power** Power of the test.

**exposure** Average calendar exposure $\bar{t}_g$ (vector of length 2 for control and treatment).

**exposure_at_risk_n1** Average at-risk exposure for group 1 (adjusted for event gap).

**exposure_at_risk_n2** Average at-risk exposure for group 2 (adjusted for event gap).

**events_n1** Expected number of events in group 1.

**events_n2** Expected number of events in group 2.

**total_events** Total expected number of events.

**variance** Variance of the log rate ratio $\mathrm{Var}(\hat{\theta})$.

**variance_null** Null variance of the log rate ratio used for score-test sizing, on the same final-analysis scale as `variance`.

**accrual_rate** Accrual rate(s) used (possibly scaled to achieve target power).

**accrual_duration** Accrual duration(s) used.

## References

Zhu, H., & Lakkis, H. (2014). Sample size calculation for comparing two negative binomial rates. *Statistics in Medicine*, 33(3), 376–387. doi:10.1002/sim.5947

Friede, T., & Schmidli, H. (2010). Blinded sample size reestimation with negative binomial counts in superiority and non-inferiority trials. *Methods of Information in Medicine*, 49(06), 618–624. doi:10.3414/ME09020060

Mutze, T., Glimm, E., Schmidli, H., & Friede, T. (2019). Group sequential designs for negative binomial outcomes. *Statistical Methods in Medical Research*, 28(8), 2326–2347. doi:10.1177/0962280218773115

## See Also

`compute_info_at_time()` for computing statistical information at a given analysis time; `blinded_ssr()` for blinded sample size reestimation; `gsNBCalendar()` for group sequential designs; `vignette("sample-size-nbinom", package = "gsDesignNB")` for detailed methodology.

## Examples

```
# Basic sample size calculation
x <- sample_size_nbinom(
  lambda1 = 0.5, lambda2 = 0.3, dispersion = 0.1, power = 0.8,
  accrual_rate = 10, accrual_duration = 20, trial_duration = 24
)
class(x)
summary(x)

# With piecewise accrual
x2 <- sample_size_nbinom(
  lambda1 = 0.5, lambda2 = 0.3, dispersion = 0.1, power = 0.8,
  accrual_rate = c(5, 10), accrual_duration = c(3, 3),
  trial_duration = 12
```

```
)
summary(x2)

# Compute power for a fixed design (power = NULL)
sample_size_nbinom(
  lambda1 = 0.5, lambda2 = 0.3, dispersion = 0.1, power = NULL,
  accrual_rate = 10, accrual_duration = 20, trial_duration = 24
)
```

---

`sim_gs_nbinom` *Simulate group sequential clinical trial for negative binomial outcomes*

---

## Description

Simulates multiple replicates of a group sequential clinical trial with negative binomial outcomes, performing interim analyses at specified calendar times. Supports parallel execution via the **future** framework for faster simulation with reproducible random number generation.

## Usage

```
sim_gs_nbinom(
  n_sims,
  enroll_rate,
  fail_rate,
  dropout_rate = NULL,
  max_followup,
  event_gap = NULL,
  analysis_times = NULL,
  n_target = NULL,
  design = NULL,
  data_cut = cut_data_by_date,
  cuts = NULL,
  test_type = c("wald", "score"),
  seed = TRUE
)
```

## Arguments

| | |
|---|---|
| `n_sims` | Number of simulations to run. |
| `enroll_rate` | Enrollment rates (data frame with `rate` and `duration`). |
| `fail_rate` | Failure rates (data frame with `treatment`, `rate`, `dispersion`). |
| `dropout_rate` | Dropout rates (data frame with `treatment`, `rate`, `duration`). |
| `max_followup` | Maximum follow-up time. |
| `event_gap` | Event gap duration. If `NULL`, inherits `design$inputs$event_gap` when available; otherwise defaults to `0`. |

`analysis_times` Vector of calendar times for interim and final analyses. Optional if `cuts` is provided.

`n_target` Total sample size to enroll (optional, if not defined by `enroll_rate`).

`design` An object of class `gsNB` or `sample_size_nbinom_result`. Used to extract planning parameters (`lambda1`, `lambda2`, `ratio`) for blinded information estimation.

`data_cut` Function to cut data for analysis. Defaults to `cut_data_by_date()`. The function must accept `sim_data`, `cut_date`, and `event_gap` as arguments.

`cuts` A list of cutting criteria for each analysis. Each element of the list should be a list of arguments for `get_cut_date()` (e.g., `planned_calendar`, `target_events`, `target_info`). If provided, `analysis_times` is ignored (or used as a fallback if `planned_calendar` is missing in a cut).

`test_type` Type of test statistic passed to `mutze_test()`: `"wald"` (default) or `"score"`. See `mutze_test()` for details.

`seed` Random seed for reproducible simulations. Controls the `future.seed` argument of `future.apply::future_lapply()`:

- `TRUE` (default): Automatically generates parallel-safe L'Ecuyer-CMRG random number streams. Results are reproducible when preceded by `set.seed()` regardless of the number of workers.
- An integer: Used as the seed for L'Ecuyer-CMRG streams directly (equivalent to calling `set.seed()` with this value before the run).
- `FALSE` or `NULL`: No special RNG handling (not recommended; results may not be reproducible in parallel).

When **future.apply** is not installed, `seed` is used with `set.seed()` for sequential execution. See **Details** for parallel usage.

## Details

### Parallel execution:

This function uses `future.apply::future_lapply()` to distribute simulation replicates across workers. By default, simulations run sequentially (equivalent to `lapply()`). To enable parallel execution, set a **future** plan before calling this function:

```
library(future)
plan(multisession, workers = 4)   # use 4 parallel workers
results <- sim_gs_nbinom(...)
plan(sequential)                  # restore default
```

### Reproducibility:

The default `seed = TRUE` ensures that results are fully reproducible regardless of the **future** plan (sequential or parallel) and regardless of the number of workers. This is achieved via the L'Ecuyer-CMRG algorithm which generates statistically independent random number streams for each simulation replicate. To obtain the same results across runs:

```
set.seed(42)
res1 <- sim_gs_nbinom(n_sims = 100, ..., seed = TRUE)
```

```
set.seed(42)
res2 <- sim_gs_nbinom(n_sims = 100, ..., seed = TRUE)

identical(res1, res2)  # TRUE, even with different plan()
```

## Value

A data frame containing simulation results for each analysis of each trial. Columns include:

**sim** Simulation ID

**analysis** Analysis index

**analysis_time** Calendar time of analysis

**n_enrolled** Number of subjects enrolled

**n_ctrl** Number of subjects in control group

**n_exp** Number of subjects in experimental group

**events_total** Total events observed

**events_ctrl** Events in control group

**events_exp** Events in experimental group

**exposure_at_risk_ctrl** Exposure at risk in control group (adjusted for event gaps)

**exposure_at_risk_exp** Exposure at risk in experimental group (adjusted for event gaps)

**exposure_total_ctrl** Total exposure in control group (calendar follow-up)

**exposure_total_exp** Total exposure in experimental group (calendar follow-up)

**z_stat** Z-statistic from the Wald test (positive favors experimental if rate ratio < 1)

**estimate** Estimated log rate ratio from the model

**se** Standard error of the estimate

**method_used** Method used for inference ("nb" or "poisson")

**dispersion** Estimated dispersion parameter from the model

**blinded_info** Estimated blinded statistical information (ML)

**unblinded_info** Observed unblinded statistical information (ML)

**info_unblinded_ml** Observed unblinded statistical information (ML)

**info_blinded_ml** Estimated blinded statistical information (ML)

**info_unblinded_mom** Observed unblinded statistical information (Method of Moments)

**info_blinded_mom** Estimated blinded statistical information (Method of Moments)

## Examples

```
# Basic sequential usage with reproducible seed
set.seed(123)
enroll_rate <- data.frame(rate = 10, duration = 3)
fail_rate <- data.frame(
  treatment = c("Control", "Experimental"),
  rate = c(0.6, 0.4),
  dispersion = 0.2
```

```
)
dropout_rate <- data.frame(
  treatment = c("Control", "Experimental"),
  rate = c(0.05, 0.05),
  duration = c(6, 6)
)
design <- sample_size_nbinom(
  lambda1 = 0.6, lambda2 = 0.4, dispersion = 0.2, power = 0.8,
  accrual_rate = enroll_rate$rate, accrual_duration = enroll_rate$duration,
  trial_duration = 6
)
cuts <- list(
  list(planned_calendar = 2),
  list(planned_calendar = 4)
)
sim_results <- sim_gs_nbinom(
  n_sims = 2,
  enroll_rate = enroll_rate,
  fail_rate = fail_rate,
  dropout_rate = dropout_rate,
  max_followup = 4,
  n_target = 30,
  design = design,
  cuts = cuts,
  seed = TRUE
)
head(sim_results)

## Not run:
# Parallel execution (requires future and future.apply)
library(future)
plan(multisession, workers = 4)
set.seed(42)
sim_results <- sim_gs_nbinom(
  n_sims = 1000,
  enroll_rate = enroll_rate,
  fail_rate = fail_rate,
  dropout_rate = dropout_rate,
  max_followup = 4,
  n_target = 30,
  design = design,
  cuts = cuts,
  seed = TRUE
)
plan(sequential)

## End(Not run)
```

---

`sim_ssr_nbinom` *Simulate adaptive group sequential trials with sample size re-estimation*

---

## Description

Simulates recurrent-event group sequential trials with information-based interim analyses and optional sample size re-estimation (SSR). Interim timing follows the blinded-information targeting used in the SSR study vignette: `get_cut_date()` searches for the earliest cut date where the blinded information reaches the planned information fraction, with an unblinded fallback if the blinded fit is unavailable.

## Usage

```
sim_ssr_nbinom(
  n_sims,
  enroll_rate,
  fail_rate,
  dropout_rate = NULL,
  max_followup,
  design,
  n_max = NULL,
  strategies = c("No adaptation", "Blinded SSR", "Unblinded SSR"),
  adapt_analysis = NULL,
  min_if_futility = 0,
  max_enrollment_frac_for_adapt = 1,
  min_months_to_close_for_adapt = 0,
  analysis_lag_months = 0,
  event_gap = NULL,
  bound_info = c("unblinded_ml", "blinded_ml", "unblinded_mom", "blinded_mom"),
  first_min_time = 1,
  min_analysis_gap = 0.5,
  ignore_futility = FALSE,
  metadata = NULL,
  test_type = c("wald", "score"),
  seed = TRUE
)
```

## Arguments

| | |
|---|---|
| `n_sims` | Number of simulated trials. |
| `enroll_rate` | Enrollment-rate data frame passed to `nb_sim()`. |
| `fail_rate` | Failure-rate data frame passed to `nb_sim()`. This defines the data-generating truth. |
| `dropout_rate` | Optional dropout-rate data frame passed to `nb_sim()`. |
| `max_followup` | Maximum follow-up per patient in the simulated trial. |
| `design` | A planning object of class `gsNB`, typically returned by `gsNBCalendar()` (optionally after `toInteger()`). |
| `n_max` | Maximum total enrollment allowed after SSR. Defaults to 150% of the planned final enrollment, rounded up and constrained to be at least the planned sample size. |

`strategies` Character vector of strategies to simulate. Must be chosen from `"No adaptation"`, `"Blinded SSR"`, and `"Unblinded SSR"`.

`adapt_analysis` Interim analysis index where SSR may be applied. Defaults to the last interim analysis (`design$k - 1`).

`min_if_futility` Minimum observed information fraction required before allowing a futility stop. Default is `0`.

`max_enrollment_frac_for_adapt` Maximum fraction of the planned enrollment already accrued at the adaptation cut for SSR to be allowed. Default is `1`.

`min_months_to_close_for_adapt` Minimum predicted months remaining to planned enrollment close required to allow SSR. Default is `0`.

`analysis_lag_months` Additional months of enrollment counted after a futility stop to approximate operational lag. Default is `0`.

`event_gap` Optional event-gap duration. If `NULL`, inherits `design$nb_design$inputs$event_gap` when available; otherwise defaults to `0`.

`bound_info` Information measure used for efficacy/futility bounds. Choices are `"unblinded_ml"` (default), `"blinded_ml"`, `"unblinded_mom"`, and `"blinded_mom"`.

`first_min_time` Minimum calendar time allowed for the first information-based interim search. Default is `1`.

`min_analysis_gap` Minimum gap between successive information-based interim searches. Default is `0.5`.

`ignore_futility` Logical. If `TRUE`, lower-bound crossings do not stop the trial.

`metadata` Optional named list or one-row data frame of scenario labels to repeat across the returned rows.

`test_type` Type of test statistic passed to `mutze_test()`: `"wald"` (default) or `"score"`. See `mutze_test()` for details.

`seed` Random-seed control passed to `future.apply::future_lapply()`. Follows the same conventions as `sim_gs_nbinom()`.

## Details

The function compares one or more strategies:

**"No adaptation"** The trial keeps the planned sample size.

**"Blinded SSR"** A blinded nuisance-parameter update is applied at `adapt_analysis` using `blinded_ssr()`.

**"Unblinded SSR"** An unblinded nuisance-parameter update is applied at `adapt_analysis` using `unblinded_ssr()`.

The returned `trial_results` include stage-specific columns such as `z_ia1`, `if_ia1`, `ia1_time`, `participants_with_events_ia1`, `events_observed_ia1`, and analogous columns for later analyses. For the actual stopping stage, `participants_with_events_stop` and `events_observed_stop` summarize the observed burden carried into the final decision for each simulated trial.

## Value

An object of class `sim_ssr_nbinom` with components:

**trial_results** One row per simulation and strategy, containing trial-level outcomes and stage-specific summaries in wide format.

**analysis_results** One row per simulation, strategy, and analysis in long format.

**settings** A list of key design and simulation settings.

## Examples

```
set.seed(123)
enroll_rate <- data.frame(rate = 12, duration = 6)
fail_rate <- data.frame(
  treatment = c("Control", "Experimental"),
  rate = c(0.6, 0.42),
  dispersion = 0.4
)
dropout_rate <- data.frame(
  treatment = c("Control", "Experimental"),
  rate = c(0.05, 0.05),
  duration = c(12, 12)
)
fixed_design <- sample_size_nbinom(
  lambda1 = 0.6,
  lambda2 = 0.42,
  dispersion = 0.4,
  power = 0.8,
  alpha = 0.025,
  accrual_rate = 12,
  accrual_duration = 6,
  trial_duration = 12,
  max_followup = 6
)
gs_design <- gsNBCalendar(
  fixed_design,
  k = 3,
  test.type = 4,
  alpha = 0.025,
  sfu = sfHSD,
  sfupar = -2,
  sfl = sfHSD,
  sflpar = 1,
  analysis_times = c(4, 8, 12)
)
sim_res <- sim_ssr_nbinom(
  n_sims = 2,
  enroll_rate = enroll_rate,
  fail_rate = fail_rate,
  dropout_rate = dropout_rate,
  max_followup = 6,
  design = gs_design,
```

```
  seed = 123
)
names(sim_res)
head(sim_res$trial_results)
```

---

`summarize_gs_sim` *Summarize group sequential simulation results*

---

## Description

Provides a summary of the operating characteristics of the group sequential design based on simulation results.

## Usage

```
summarize_gs_sim(x, info_trim = 0.01)
```

## Arguments

| | |
|---|---|
| `x` | A data frame returned by `check_gs_bound()` (or `sim_gs_nbinom()` if bounds are manually checked). Must contain columns `cross_upper`, `cross_lower`. |
| `info_trim` | Proportion of observations trimmed from each tail when summarizing information estimates. Defaults to `0.01` to reduce sensitivity to occasional unstable fitted information estimates in small interim data sets. |

## Value

A list containing:

**n_sim** Number of simulations

**power** Overall power (probability of crossing upper bound)

**futility** Overall futility rate (probability of crossing lower bound and not upper)

**analysis_summary** Data frame with per-analysis statistics (sample size, events, information, crossings, and optional exposure columns when present in `x`).

## Examples

```
design <- gsDesign::gsDesign(k = 2, n.fix = 80, test.type = 2, timing = c(0.5, 1))
sim_df <- data.frame(
  sim = c(1, 1, 2, 2),
  analysis = c(1, 2, 1, 2),
  z_stat = c(2.4, NA, -0.5, 1.9),
  blinded_info = c(40, 80, 40, 80),
  unblinded_info = c(40, 80, 40, 80),
  n_enrolled = c(30, 60, 30, 60),
  events_total = c(12, 25, 10, 22)
)
bounds_checked <- check_gs_bound(sim_df, design)
summarize_gs_sim(bounds_checked)
```

summarize_ssr_sim *Summarize adaptive SSR simulation results*

## Description

Produces trial-level and analysis-level summaries from the output of `sim_ssr_nbinom()`. The summary includes expected sample size, stopping probabilities, expected participants with events, and expected events observed.

## Usage

```
summarize_ssr_sim(x, by = "strategy")
```

## Arguments

| | |
|---|---|
| x | A `sim_ssr_nbinom` object or a data frame containing the `trial_results` component returned by `sim_ssr_nbinom()`. |
| by | Character vector of grouping columns for the trial-level summary. Defaults to `"strategy"`. |

## Value

A list with:

**trial_summary** Trial-level grouped summary.

**analysis_summary** Analysis-level grouped summary.

## Examples

```
set.seed(123)
enroll_rate <- data.frame(rate = 10, duration = 4)
fail_rate <- data.frame(
  treatment = c("Control", "Experimental"),
  rate = c(0.5, 0.35),
  dispersion = 0.3
)
fixed_design <- sample_size_nbinom(
  lambda1 = 0.5,
  lambda2 = 0.35,
  dispersion = 0.3,
  power = 0.8,
  alpha = 0.025,
  accrual_rate = 10,
  accrual_duration = 4,
  trial_duration = 8,
  max_followup = 4
)
gs_design <- gsNBCalendar(
  fixed_design,
```

```
  k = 3,
  test.type = 4,
  alpha = 0.025,
  analysis_times = c(3, 5, 8)
)
sim_res <- sim_ssr_nbinom(
  n_sims = 2,
  enroll_rate = enroll_rate,
  fail_rate = fail_rate,
  max_followup = 4,
  design = gs_design,
  strategies = "No adaptation",
  seed = 321
)
summarize_ssr_sim(sim_res)$trial_summary
```

---

`summary.gsNB` *Summary for gsNB objects*

---

## Description

Provides a textual summary of a group sequential design for negative binomial outcomes, similar to the summary provided by `gsDesign::gsDesign()`. For tabular output, use `gsDesign::gsBoundSummary()` directly on the gsNB object.

## Usage

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

## Arguments

| | |
|---|---|
| `object` | An object of class gsNB. |
| `...` | Additional arguments (currently ignored). |

## Value

A character string summarizing the design (invisibly). The summary is also printed to the console.

## Examples

```
nb_ss <- sample_size_nbinom(
  lambda1 = 0.5, lambda2 = 0.3, dispersion = 0.1, power = 0.9,
  accrual_rate = 10, accrual_duration = 20, trial_duration = 24
)
gs_design <- gsNBCalendar(nb_ss, k = 3, analysis_times = c(12, 18, 24))
summary(gs_design)

# For tabular bounds summary, use gsBoundSummary() directly:
gsBoundSummary(gs_design)
```

summary.sample_size_nbinom_result *Summary for sample_size_nbinom_result objects*

## Description

Provides a textual summary of the sample size calculation for negative binomial outcomes, similar to the summary for gsNB objects.

## Usage

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

## Arguments

| | |
|---|---|
| object | An object of class sample_size_nbinom_result. |
| ... | Additional arguments (currently ignored). |

## Value

A character string summarizing the design (invisibly). The summary is also printed to the console.

## Examples

```
x <- sample_size_nbinom(
  lambda1 = 0.5, lambda2 = 0.3, dispersion = 0.1, power = 0.8,
  accrual_rate = 10, accrual_duration = 20, trial_duration = 24
)
class(x)
summary(x)
```

toInteger *Convert group sequential design to integer sample sizes*

## Description

Generic function to round sample sizes in a group sequential design to integers. This extends the gsDesign::toInteger() function from the gsDesign package to work with gsNB objects.

## Usage

```
toInteger(x, ...)

## S3 method for class 'gsDesign'
toInteger(x, ratio = x$ratio, roundUpFinal = TRUE, ...)

## S3 method for class 'gsNB'
toInteger(x, ratio = x$nb_design$inputs$ratio, roundUpFinal = TRUE, ...)
```

## Arguments

| | |
|---|---|
| x | An object of class gsNB or gsDesign. |
| ... | Additional arguments passed to methods. |
| ratio | Randomization ratio (n2/n1). If an integer is provided, rounding is done to a multiple of ratio + 1. If ratio < 1 and 1/ratio is an integer (e.g., 1:2 allocation, ratio = 0.5), rounding is done to a multiple of 1/ratio + 1. Default uses the ratio from the original design. |
| roundUpFinal | If TRUE (default), the final sample size is rounded up to ensure the target is met. If FALSE, rounding is to the nearest integer. |

## Details

This function rounds the final sample size while maintaining the randomization ratio. When calendar analysis times are available, interim sample sizes remain expected enrollment counts at those calendar times after rescaling the accrual rate to the rounded final sample size.

When analysis_times were provided to gsNBCalendar(), expected events, exposure, and statistical information (n.I) are recomputed at each analysis time based on the new sample size and expected exposures.

## Value

An object of the same class as input with integer sample sizes.

## Methods (by class)

- toInteger(gsDesign): Method for gsDesign objects (calls gsDesign::toInteger()).
- toInteger(gsNB): Method for gsNB objects.
  Rounds sample sizes in a group sequential negative binomial design to integers, respecting the randomization ratio.

## Examples

```
nb_ss <- sample_size_nbinom(
  lambda1 = 0.5, lambda2 = 0.3, dispersion = 0.1, power = 0.9,
  accrual_rate = 10, accrual_duration = 20, trial_duration = 24
)
gs_design <- gsNBCalendar(nb_ss, k = 3, analysis_times = c(12, 18, 24))
gs_integer <- toInteger(gs_design)
```

unblinded_ssr *Unblinded sample size re-estimation for recurrent events*

## Description

Estimates the event rates and dispersion from unblinded interim data and calculates the required sample size to maintain power, assuming the planned treatment effect holds (or using the observed control rate).

## Usage

```
unblinded_ssr(
  data,
  ratio = 1,
  lambda1_planning,
  lambda2_planning,
  rr0 = 1,
  power = 0.8,
  alpha = 0.025,
  accrual_rate,
  accrual_duration,
  trial_duration,
  dropout_rate = 0,
  max_followup = NULL,
  event_gap = NULL
)
```

## Arguments

| | |
|---|---|
| data | A data frame containing the unblinded interim data. Must include columns events (number of events), tte (total exposure/follow-up time), and treatment (treatment group identifier, e.g., 1 for control, 2 for experimental). This is typically the output of cut_data_by_date(). |
| ratio | Planned allocation ratio (experimental / control). Default is 1. |
| lambda1_planning | Planned event rate for the control group used in original calculation. |
| lambda2_planning | Planned event rate for the experimental group used in original calculation. |
| rr0 | Rate ratio under the null hypothesis (lambda2/lambda1). Default is 1. |
| power | Target power (1 - beta). Default is 0.8. |
| alpha | One-sided significance level. Default is 0.025. |
| accrual_rate | Vector of accrual rates (patients per unit time). |
| accrual_duration | Vector of durations for each accrual rate. Must be same length as accrual_rate. |
| trial_duration | Total planned duration of the trial. |

| | |
|---|---|
| dropout_rate | Dropout rate (hazard rate). Default is 0. |
| max_followup | Maximum follow-up time for any patient. Default is NULL (infinite). |
| event_gap | Gap duration after each event during which no new events are counted. Default is NULL (no gap). |

## Details

If the maximum likelihood negative binomial fit fails to converge, the function falls back to method-of-moments estimation via `estimate_nb_mom()` rather than erroring out. The observed Fisher information is then computed analytically from the MoM-estimated rates and dispersion using the same subject-level weight formula as `calculate_blinded_info()`. This keeps SSR updates well-defined under extreme overdispersion or sparse interim data.

## Value

A list containing:

**n_total_unblinded** Re-estimated total sample size using unblinded estimates.

**dispersion_unblinded** Estimated dispersion parameter (k) from unblinded data.

**lambda1_unblinded** Estimated control event rate from unblinded data.

**lambda2_unblinded** Estimated experimental event rate from unblinded data.

**info_fraction** Estimated information fraction at interim (unblinded information / target information).

**unblinded_info** Estimated statistical information from the unblinded interim data.

**target_info** Target statistical information required for the planned power.

**fallback** Character label for which estimator was used (`"ml"` or `"mom"`).

## Examples

```
interim <- data.frame(
  events = c(1, 2, 1, 3),
  tte = c(0.8, 1.0, 1.2, 0.9),
  treatment = c("Control", "Control", "Experimental", "Experimental")
)
unblinded_ssr(
  interim,
  ratio = 1,
  lambda1_planning = 0.5,
  lambda2_planning = 0.3,
  power = 0.8,
  alpha = 0.025,
  accrual_rate = 10,
  accrual_duration = 12,
  trial_duration = 18
)
```

`update_gsNB` *Update group sequential bounds with observed information*

## Description

Given a planned `gsNB` (or `gsDesign`) object and observed statistical information at one or more analyses, recompute the group sequential boundaries and return an updated design object together with a `gsDesign::gsBoundSummary()`-style table.

## Usage

```
update_gsNB(design, observed_info, spending_time = NULL)
```

## Arguments

| | |
|---|---|
| `design` | A `gsNB` or `gsDesign` object produced by `gsNBCalendar()` (or `gsDesign::gsDesign()`). |
| `observed_info` | Numeric vector of observed statistical information at each analysis conducted so far. Its length must be between 1 and `design$k`. If shorter than `design$k`, information at future analyses is projected from the planned design. |
| `spending_time` | Optional numeric vector the same length as `observed_info` giving the spending time (between 0 and 1) for each analysis. When `NULL`, spending time equals the information fraction. If shorter than `design$k`, future spending times are taken from the planned design. |

## Details

The observed information determines the covariance structure of the test statistics (via the information fraction `timing`), while `spending_time` controls how much of the error-spending budget has been used. When `spending_time` is `NULL` (the default), spending is driven by the observed information fraction. Supplying an explicit `spending_time` is useful when the monitoring charter specifies calendar-driven spending that differs from the observed information fraction.

## Value

A list with components:

**design** The updated `gsDesign` object with recalculated boundaries.

**bounds** A data frame from `gsDesign::gsBoundSummary()` showing Z-boundaries, nominal p-values, approximate treatment effects at the boundary, and cumulative crossing probabilities at each analysis.

**info** A data frame with one row per analysis containing the information fraction (`IF`), spending time (`spending_time`), upper and lower Z-boundaries, and cumulative upper and lower spending.

## Examples

```
library(gsDesign)
nb_ss <- sample_size_nbinom(
  lambda1 = 0.5, lambda2 = 0.3, dispersion = 0.1, power = 0.9,
  accrual_rate = 10, accrual_duration = 20, trial_duration = 24
)
gs <- gsNBCalendar(nb_ss, k = 3, analysis_times = c(12, 18, 24))

# After observing information at the first interim
upd <- update_gsNB(gs, observed_info = gs$n.I[1] * 0.95)
upd$bounds
upd$info
```

# Index


blinded_ssr, 3
blinded_ssr(), *6*, *37*, *43*

calculate_blinded_info, 5
calculate_blinded_info(), *51*
check_gs_bound, 6
check_gs_bound(), *45*
compute_info_at_time, 7
compute_info_at_time(), *37*
cut_completers, 9
cut_data_by_date, 10
cut_data_by_date(), *3*, *9*, *26*, *39*, *50*
cut_date_for_completers, 11
cut_date_for_completers(), *9*

estimate_nb_mom, 12
estimate_nb_mom(), *5*, *51*

fit_nb_glmm, 13
fit_nb_glmm(), *19, 20*, *23*, *25*
future.apply::future_lapply(), *39*

get_analysis_date, 14
get_cut_date, 15
get_cut_date(), *39*, *42*
gsDesign::gsBoundSummary(), *18*, *47*, *52*
gsDesign::gsDesign(), *17, 18*, *47*, *52*
gsDesign::toInteger(), *48, 49*
gsNBCalendar, 16
gsNBCalendar(), *33*, *37*, *42*, *49*, *52*

impute_nb, 19
impute_nb(), *13*
impute_nb_composite, 22
impute_nb_composite(), *20*
impute_nb_mar, 23
impute_nb_mar(), *25*
impute_nb_mnar_ref, 24
impute_nb_mnar_ref(), *20*

lapply(), *39*

MASS::glm.nb(), *26*, *29*
mutze_test, 26
mutze_test(), *39*, *43*

nb_sim, 28
nb_sim(), *9–11*, *14, 15*, *42*
nb_sim_seasonal, 30
nb_sim_seasonal(), *11*

print.gsNBsummary, 31
print.mutze_test *(*mutze_test*)*, 26
print.sample_size_nbinom_result, 32
print.sample_size_nbinom_summary, 32

run_ssr_shiny, 33

sample_size_nbinom, 34
sample_size_nbinom(), *6*, *8*, *16, 17*, *33*
set.seed(), *39*
shiny::runApp(), *33*
sim_gs_nbinom, 38
sim_gs_nbinom(), *7*, *43*, *45*
sim_ssr_nbinom, 41
sim_ssr_nbinom(), *33*
stats::rnbinom(), *35*
summarize_gs_sim, 45
summarize_ssr_sim, 46
summarize_ssr_sim(), *33*
summary.gsNB, 47
summary.sample_size_nbinom_result, 48

toInteger, 48
toInteger(), *42*

unblinded_ssr, 50
unblinded_ssr(), *43*
update_gsNB, 52