Package 'nof1kit'

August 8, 2026

Title Design, Monitor, and Analyze Single-Case (N-of-1) Intensive Longitudinal Studies

Version 0.1.0

Description Tools for the stages of a single-case (N-of-1) experimental study that come before analysis: generating randomization schedules that can be preregistered and reproduced exactly, under run-length constraints, exporting them for mobile data collection, validating incoming ecological momentary assessment (EMA) records, and monitoring compliance. Designed around the workflow of a 70-day randomized N-of-1 study collected with stock 'iOS' tools at 92.9% compliance; the package ships with that study's complete dataset.

License MIT + file LICENSE

URL https://github.com/haomeng797-ship-it/nof1kit, https://haomeng797-ship-it.github.io/nof1kit/

BugReports https://github.com/haomeng797-ship-it/nof1kit/issues

Encoding UTF-8

Imports jsonlite, stats, tools, utils

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Miura Meng [aut, cre] (ORCID: <https://orcid.org/0009-0004-1522-1997>)

Maintainer Miura Meng <haomeng797@gmail.com>

Repository CRAN

Date/Publication 2026-08-08 11:50:07 UTC

Contents

`check_schedule` *Diagnose a randomization schedule*

Description

Reports the design properties a reviewer (or a preregistration reader) would want to see: condition counts, run-length distribution, number of alternations, and the lag-1 autocorrelation of the condition sequence.

Usage

```
check_schedule(schedule)
```

Arguments

`schedule`	A `nof1_schedule` from `design_schedule()`, or any data frame with `day` and `condition` columns.

Value

A list with elements `counts`, `max_run`, `run_table`, `alternations`, and `lag1_autocor`.

Examples

```
sched <- design_schedule(n_days = 70, seed = 20260218)
check_schedule(sched)
```

`compliance` *Compliance against scheduled prompts*

Description

Computes EMA compliance as the proportion of *scheduled prompts* that received a response inside their response window.

Usage

```
compliance(data, start_date, n_days, times, window = 3, through = NULL)
```

Arguments

`data`	A data frame from `read_ema()`, containing `timestamp`.
`start_date`	Date the study began (day 1).
`n_days`	Study length in days.
`times`	Character vector of prompt times in `"HH:MM"`, e.g. `c("09:00", "15:00", "21:00")`.
`window`	Response window in hours. A record counts toward a prompt if it arrives between the prompt time and `window` hours later.
`through`	Compute compliance as of this moment. Defaults to the last timestamp in the data, so a study still running is not penalised for prompts that have not happened yet.

Details

The definition matters more than the arithmetic. Compliance is often reported as the number of records divided by the number of prompts, which lets a participant who answers one prompt three times appear more compliant than one who answers three prompts once. Late entries, likewise, are data but not evidence that a prompt was answered when it was asked.

This function therefore counts each scheduled prompt at most once, and only when a record falls within `window` hours of it. Records outside every window are returned separately as `n_off_window`: they are not discarded, they are simply not evidence of compliance.

Value

An object of class `nof1_compliance`: a list with `rate`, `n_answered`, `n_expected`, `n_off_window`, and a data frame `prompts` with one row per scheduled prompt and a logical `answered`.

Examples

```
d <- data.frame(timestamp = as.POSIXct(
  c("2026-02-18 09:30:00", "2026-02-18 15:10:00", "2026-02-19 09:05:00"),
  tz = "UTC"))
compliance(d, start_date = "2026-02-18", n_days = 2,
           times = c("09:00", "15:00", "21:00"))
```

`design_schedule` *Generate a randomized N-of-1 intervention schedule*

Description

Produces a day-by-day condition sequence for a single-case experimental study, under the restricted randomization commonly needed in N-of-1 designs: balanced condition counts and a cap on how many consecutive days the same condition may repeat.

Usage

```
design_schedule(n_days, conditions = c(0L, 1L), max_run = 2L, seed)
```

Arguments

`n_days`	Integer. Length of the study in days.
`conditions`	Vector of condition labels. Defaults to `c(0L, 1L)` (control / treatment), matching the schedule format used by the companion iOS Shortcuts logger.
`max_run`	Integer. Maximum allowed run of identical consecutive conditions. Defaults to 2.
`seed`	Integer. Required. The seed makes the schedule reproducible and is intended to be reported in the preregistration.

Details

Sampling is exact, not by rejection. The number of valid completions from every intermediate state (remaining counts, current run) is counted by dynamic programming, and the sequence is then drawn day by day with probability proportional to those counts. The result is a uniform draw over the set of *all* schedules that satisfy the constraints, so no valid schedule is more probable than any other.

Naive rejection sampling is not workable here: for a balanced 70-day two-condition design with `max_run = 2`, fewer than one permutation in a million satisfies the run constraint.

Value

An object of class `nof1_schedule`: a data frame with columns `day` and `condition`, with the generating parameters stored as attributes.

Examples

```
sched <- design_schedule(n_days = 70, max_run = 2, seed = 20260218)
head(sched)
check_schedule(sched)
```

`read_ema` *Read EMA records into a tidy data frame*

Description

Reads the CSV or JSON produced by a mobile collection tool and returns a data frame with parsed timestamps and a study-day index, ready for `validate_ema()` and `compliance()`.

Usage

```
read_ema(path, start_date = NULL, tz = "UTC", timestamp_col = "timestamp")
```

Arguments

`path`	Path to a `.csv` or `.json` file.
`start_date`	Date (or something `as.Date()` accepts) on which the study began. Used to compute `study_day` when the file does not already have it.
`tz`	Time zone to interpret timestamps in. Defaults to `"UTC"`, which matches the ISO 8601 output of the companion app.
`timestamp_col`	Name of the column holding the timestamp. Real files call it many things (`datetime`, `time`, `recorded_at`); give the name here and it is renamed to `timestamp` on the way in.

Details

The expected columns are `timestamp` plus whatever was measured. Only `timestamp` is required. If `start_date` is supplied (or the file already carries a `study_day` column) a `study_day` index is attached, counting from 1 on the first day of the study.

Value

A data frame with `timestamp` as `POSIXct`, `study_day` as integer when derivable, and all other columns unchanged.

Examples

```
f <- system.file("extdata", "example_ema.csv", package = "nof1kit")
if (nzchar(f)) head(read_ema(f))
```

`sim_power` *Simulation-based power for an N-of-1 design*

Description

Estimates the power to detect a treatment effect in a single-case design by simulating studies of the given length, analyzing each one, and recording how often the effect is detected.

Usage

```
sim_power(
  n_days = 70,
  effect = 0.5,
  sd = 1,
  phi = 0,
  max_run = 2L,
  alpha = 0.05,
  n_sims = 500L,
  schedule = NULL,
  seed = NULL
)
```

Arguments

`n_days`	Study length in days.
`effect`	True treatment effect, in the same units as `sd`. Use 0 to estimate the false positive rate.
`sd`	Residual standard deviation of the outcome (its marginal SD, not the innovation SD).
`phi`	Lag-1 autocorrelation of the errors, in `[0, 1)`. Daily mood and affect measures commonly fall between 0.2 and 0.5.
`max_run`	Maximum run of identical consecutive conditions in the generated schedules.
`alpha`	Significance level.
`n_sims`	Number of simulated studies.
`schedule`	Optional fixed condition vector (0/1) of length `n_days`. When given, every simulation uses it instead of drawing a new one.
`seed`	Optional seed for reproducibility.

Details

Closed-form power formulas assume independent observations. Daily measurements from one person are not independent: today's mood is correlated with yesterday's. Positive autocorrelation means each new day carries less information than a genuinely new observation would, so the effective sample size is smaller than the number of days.

This function therefore analyses each simulated study two ways, and reports both. `power` comes from a model with AR(1) errors, which accounts for the dependence. `power_ols` comes from ordinary least squares, which ignores it.

The direction in which OLS goes wrong depends on the schedule, which is why design and analysis cannot be chosen separately. Under the rapidly alternating schedules that `design_schedule()` produces, a slowly drifting AR(1) error is nearly orthogonal to the condition sequence: the drift cancels across adjacent days, the true variance of the effect estimate is smaller than independence implies, and OLS is therefore conservative. It loses power but does not produce false positives.

Under a schedule that changes slowly, the opposite happens. A design that runs control for the first half and treatment for the second is nearly collinear with any drift, so autocorrelation is readily mistaken for an effect. In simulations at `phi = 0.7`, such a design rejects a true null about 40 percent of the time at a nominal 5 percent level. Pass `schedule` to see this for a sequence you are considering.

Set `effect = 0` to get the false positive rate instead of power.

Each simulated study gets a freshly drawn randomization schedule under the same run constraint, so the estimate reflects the design rather than one particular sequence. Supply `schedule` to hold the sequence fixed instead.

Value

An object of class `nof1_power`: a list with `power`, `power_ols`, `n_days`, `effect`, `phi`, `alpha`, `n_sims`, and `n_failed` (simulations where the AR(1) fit did not converge and were dropped).

Examples

```
# Power to detect a half-SD effect over 70 days, with moderate autocorrelation
# (n_sims is kept small here; use the default 500 for a real planning run)
sim_power(n_days = 70, effect = 0.5, phi = 0.3, n_sims = 40, seed = 1)

# With no effect, `power` is the false positive rate
sim_power(n_days = 70, effect = 0, phi = 0.5, n_sims = 40, seed = 1)
```

`validate_ema` *Check EMA records for the problems that actually occur*

Description

Runs the integrity checks a single-case dataset needs before analysis: values outside their declared range, duplicated timestamps, missing values in required columns, records falling outside the study period, and timestamps that failed to parse.

Usage

```
validate_ema(data, ranges = NULL, required = "timestamp", n_days = NULL)
```

Arguments

data	A data frame from read_ema().
ranges	Named list of c(min, max) giving the valid range of each measured variable, e.g. list(mood = c(0, 100)). Columns not listed are not range-checked.
required	Character vector of columns that must not be missing.
n_days	Optional study length. When given, records with a study_day outside 1:n_days are flagged.

Details

The result is returned as data, not printed. Reports are one way to consume it; a preregistered pipeline that halts on failure is another.

Value

An object of class nof1_validation: a list with n_records, n_issues, and a data frame issues with columns check, row, column, and detail. n_issues == 0 means every check passed.

Examples

```
d <- data.frame(
  timestamp = as.POSIXct(c("2026-02-18 09:00:00", "2026-02-18 09:00:00"), tz = "UTC"),
  mood = c(72, 140),
  study_day = c(1L, 1L)
)
v <- validate_ema(d, ranges = list(mood = c(0, 100)))
v
```

---

write_schedule *Export a schedule for the mobile data-collection layer*

Description

Writes a schedule as JSON in the exact {"1": 0, "2": 1, ...} format consumed by the companion iOS Shortcuts EMA logger (https://github.com/haomeng797-ship-it/melatonin-ema-logger), so a generated design can be dropped straight into an existing collection pipeline.

Usage

```
write_schedule(schedule, path)
```

Arguments

schedule	A nof1_schedule from design_schedule().
path	File path for the JSON output, e.g. "schedule.json".

Value

The path, invisibly.

Examples

```
sched <- design_schedule(n_days = 70, seed = 20260218)
tmp <- tempfile(fileext = ".json")
write_schedule(sched, tmp)
```

Index