

Package: gsDesignTune (via r-universe)

May 13, 2026

Title Dependency-Aware Scenario Exploration for Group Sequential Designs

Version 0.1.0

Description Provides systematic, dependency-aware exploration of group sequential designs created with 'gsDesign'. Supports reproducible grid and random search over user-defined candidate sets, parallel evaluation via the 'future' framework, standardized metric extraction, and auditable reporting for design space evaluation and trade-off analysis. Methods for group sequential design are described in Anderson (2025) <doi:10.32614/CRAN.package.gsDesign>. The 'future' framework for parallel processing is described in Bengtsson (2021) <doi:10.32614/RJ-2021-048>.

License MIT + file LICENSE

URL <https://nanx.me/gsDesignTune/>,
<https://github.com/nanxstats/gsDesignTune>

BugReports <https://github.com/nanxstats/gsDesignTune/issues>

Encoding UTF-8

Depends R (>= 4.1.0)

Imports digest, future.apply, ggplot2, gsDesign, progressr, R6, rlang, tinytable

Suggests future, knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libxml2-dev libssl-dev libnode-dev

Repository <https://nanxstats.r-universe.dev>

Date/Publication 2026-04-13 20:54:13 UTC

RemoteUrl <https://github.com/nanxstats/gsdowntune>

RemoteRef HEAD

RemoteSha 481a075fb0a0e8bc725b210845cbe1e06ef79472

Contents

gsDesignTune	2
GSDTuneJob	3
gsSurvCalendarTune	7
gsSurvTune	8
spending_specs	8
SpendingFamily	9
SpendingSpec	10
toString.function	11
tune_choice	12
tune_dep	12
tune_fixed	13
tune_int	13
tune_seq	14
tune_specs	14
tune_values	15

Index **16**

gsDesignTune	<i>Create a tune job for gsDesign::gsDesign()</i>
--------------	---

Description

gsDesignTune() is a drop-in replacement for [gsDesign::gsDesign\(\)](#) that returns a tune job object instead of immediately running a single design.

Usage

```
gsDesignTune(..., upper = NULL, lower = NULL)
```

Arguments

...	Arguments to gsDesign::gsDesign() . Any argument can be replaced by a tune_*() specification.
upper, lower	Optional spending specifications provided as SpendingSpec or SpendingFamily. When supplied, these are translated to the underlying (sfu, sfupar) / (sfl, sflpar) arguments.

Details

Any argument can be replaced by a tuning specification created by tune_*(). Use SpendingSpec / SpendingFamily via upper= and lower= for dependency-aware spending function tuning.

Value

A GSDTuneJob R6 object.

Examples

```
job <- gsDesignTune(
  k = 3,
  test.type = 4,
  alpha = tune_values(list(0.025, 0.03))
)

job$run(strategy = "grid", parallel = FALSE, seed = 1)
utils::head(job$results())
```

GSDTuneJob

GSDTuneJob

Description

GSDTuneJob

GSDTuneJob

Details

R6 class representing a dependency-aware tuning job for group sequential designs created by [gsDesign::gsDesign\(\)](#) or [gsDesign::gsSurv\(\)](#).

Value

An R6 class generator. Use `$new()` to create a GSDTuneJob object.

Public fields

`target` Target design function name ("gsDesign" or "gsSurv").

`base_args` Named list of fixed arguments passed to the target function.

`tune_specs` Named list of tuning specifications for explored arguments.

`param_space` Internal parameter space used for configuration generation.

`spec` Audit record including base/tuned args and `sessionInfo()`.

Methods

Public methods:

- `GSDTuneJob$new()`
- `GSDTuneJob$run()`
- `GSDTuneJob$results()`
- `GSDTuneJob$summarize()`
- `GSDTuneJob$table()`
- `GSDTuneJob$design()`
- `GSDTuneJob$call_args()`
- `GSDTuneJob$best()`
- `GSDTuneJob$pareto()`
- `GSDTuneJob$plot()`
- `GSDTuneJob$report()`
- `GSDTuneJob$clone()`

Method `new()`: Create a new tune job.

Usage:

```
GSDTuneJob$new(target = c("gsDesign", "gsSurv", "gsSurvCalendar"), args)
```

Arguments:

`target` Target function name ("gsDesign", "gsSurv", or "gsSurvCalendar").

`args` Named list of evaluated arguments.

Method `run()`: Evaluate configurations under a search strategy.

Usage:

```
GSDTuneJob$run(  
  strategy = c("grid", "random"),  
  n = NULL,  
  parallel = TRUE,  
  seed = NULL,  
  cache_dir = NULL,  
  metrics_fun = NULL  
)
```

Arguments:

`strategy` Search strategy ("grid" or "random").

`n` Number of configurations for random search.

`parallel` Whether to evaluate configurations in parallel.

`seed` Optional seed for reproducibility.

`cache_dir` Optional directory to cache design objects as RDS.

`metrics_fun` Optional metric hook.

Method `results()`: Return the results data.frame.

Usage:

```
GSDTuneJob$results()
```

Method `summarize()`: Summarize the run (counts + numeric metric summaries).

Usage:

```
GSDTuneJob$summarize()
```

Method `table()`: Render a compact results table with `tinytable`.

Returns a `tinytable` object which knits to HTML/LaTeX automatically in R Markdown and can also be printed explicitly with `print(x, output = "html")` or `print(x, output = "latex")`.

Usage:

```
GSDTuneJob$table(  
  data = NULL,  
  columns = NULL,  
  n = NULL,  
  caption = NULL,  
  notes = NULL,  
  digits = 3,  
  theme = "striped"  
)
```

Arguments:

`data` Optional results-like `data.frame` to render. Defaults to `$results()`.

`columns` Optional character vector of columns to display, in order. When `NULL`, a compact default set is chosen automatically. Explicit column selections are preserved, including `audit/internal` columns.

`n` Optional number of rows to display.

`caption` Optional table caption.

`notes` Optional table notes passed to `tinytable::tt()`.

`digits` Number of digits for numeric columns.

`theme` `Tinytable` theme passed to `tinytable::tt()`. Defaults to `"striped"`.

Method `design()`: Retrieve a design object for configuration `i`.

Usage:

```
GSDTuneJob$design(i)
```

Arguments:

`i` Row index of the configuration.

Method `call_args()`: Return the underlying argument list for configuration `i`.

Usage:

```
GSDTuneJob$call_args(i)
```

Arguments:

`i` Row index of the configuration.

Method `best()`: Rank configurations by a metric (with optional constraints).

Usage:

```
GSDTuneJob$best(metric, direction = c("min", "max"), constraints = NULL)
```

Arguments:

`metric` Metric column name.
`direction` Ranking direction ("min" or "max").
`constraints` Optional constraints (function or expression).

Method `pareto()`: Compute a Pareto (non-dominated) set for multiple metrics.

Usage:

```
GSDTuneJob$pareto(metrics, directions)
```

Arguments:

`metrics` Metric column names.
`directions` Directions for each metric ("min"/"max").

Method `plot()`: Create a quick exploration plot.

Usage:

```
GSDTuneJob$plot(metric, x, color = NULL, facet = NULL)
```

Arguments:

`metric` Y-axis metric column name.
`x` X-axis column name.
`color` Optional color column name.
`facet` Optional faceting column name.

Method `report()`: Render an HTML report.

Usage:

```
GSDTuneJob$report(path)
```

Arguments:

`path` Output HTML file path.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
GSDTuneJob$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
job <- GSDTuneJob$new(target = "gsDesign", args = list(k = 3, alpha = 0.025))  
job$spec$target
```

gsSurvCalendarTune *Create a tune job for* `gsDesign::gsSurvCalendar()`

Description

`gsSurvCalendarTune()` is a drop-in replacement for `gsDesign::gsSurvCalendar()` that returns a tune job object instead of immediately running a single design.

Usage

```
gsSurvCalendarTune(..., upper = NULL, lower = NULL)
```

Arguments

... Arguments to `gsDesign::gsSurvCalendar()`. Any argument can be replaced by a `tune_*`() specification.

upper, lower Optional spending specifications provided as `SpendingSpec` or `SpendingFamily`. When supplied, these are translated to the underlying `(sfu, sfupar) / (sf1, sf1par)` arguments.

Details

Any argument can be replaced by a tuning specification created by `tune_*`(). Use `SpendingSpec` / `SpendingFamily` via `upper=` and `lower=` for dependency-aware spending function tuning.

Value

A `GSDTuneJob` R6 object.

Examples

```
job <- gsSurvCalendarTune(
  calendarTime = tune_values(list(c(12, 24, 36), c(12, 24, 48))),
  spending = c("information", "calendar")
)

job$run(strategy = "grid", parallel = FALSE, seed = 1)
utils::head(job$results())
```

gsSurvTune	<i>Create a tune job for gsDesign::gsSurv()</i>
------------	---

Description

gsSurvTune() is a drop-in replacement for `gsDesign::gsSurv()` that returns a tune job object instead of immediately running a single design.

Usage

```
gsSurvTune(..., upper = NULL, lower = NULL)
```

Arguments

...	Arguments to <code>gsDesign::gsSurv()</code> . Any argument can be replaced by a <code>tune_*</code> () specification.
upper, lower	Optional spending specifications provided as <code>SpendingSpec</code> or <code>SpendingFamily</code> . When supplied, these are translated to the underlying <code>(sfu, sfupar) / (sfl, sflpar)</code> arguments.

Details

Any argument can be replaced by a tuning specification created by `tune_*`(). Use `SpendingSpec` / `SpendingFamily` via `upper=` and `lower=` for dependency-aware spending function tuning.

Value

A `GSDTuneJob` R6 object.

Examples

```
job <- gsSurvTune(
  k = 3,
  test.type = 4,
  hr = tune_values(list(0.6, 0.7))
)

job$run(strategy = "grid", parallel = FALSE, seed = 1)
utils::head(job$results())
```

spending_specs	<i>Spending function specifications</i>
----------------	---

Description

`SpendingSpec` and `SpendingFamily` provide a dependency-aware and user-friendly way to tune spending functions and their parameters.

SpendingFamily	<i>SpendingFamily</i>
----------------	-----------------------

Description

SpendingFamily

SpendingFamily

Details

An R6 class representing a set of spending function specifications. Each family member is a SpendingSpec.

Value

An R6 class generator. Use `$new()` to create a SpendingFamily object.

Public fields

`members` List of SpendingSpec objects.

Methods

Public methods:

- [SpendingFamily\\$new\(\)](#)
- [SpendingFamily\\$expand\(\)](#)
- [SpendingFamily\\$clone\(\)](#)

Method `new()`: Create a new spending family from one or more SpendingSpec.

Usage:

```
SpendingFamily$new(...)
```

Arguments:

... SpendingSpec objects.

Method `expand()`: Expand all members to spending settings.

Usage:

```
SpendingFamily$expand()
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
SpendingFamily$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
fam <- SpendingFamily$new(
  SpendingSpec$new(gsDesign::sfHSD, par = tune_fixed(-4)),
  SpendingSpec$new(gsDesign::sfLD0F, par = tune_fixed(0))
)
fam$expand()
```

SpendingSpec

SpendingSpec

Description

SpendingSpec

SpendingSpec

Details

An R6 class representing a single spending function (fun) and a tuning specification for its parameter (par).

Value

An R6 class generator. Use `$new()` to create a SpendingSpec object.

Public fields

fun Spending function (callable with signature (alpha, t, param)).

fun_label Label captured from the constructor call (used for plotting).

par Tuning specification for the spending parameter.

Methods**Public methods:**

- [SpendingSpec\\$new\(\)](#)
- [SpendingSpec\\$expand\(\)](#)
- [SpendingSpec\\$clone\(\)](#)

Method `new()`: Create a new spending specification.

Usage:

```
SpendingSpec$new(fun, par = tune_fixed(NULL))
```

Arguments:

fun Spending function.

par Spending parameter specification.

Method `expand()`: Expand to a list of spending settings (fun + concrete parameter values).

Usage:

```
SpendingSpec$expand()
```

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
SpendingSpec$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
spec <- SpendingSpec$new(gsDesign::sfHSD, par = tune_seq(-4, -2, length_out = 2))
spec$expand()
```

toString.function	<i>Convert a function to a short label string</i>
-------------------	---

Description

`gsDesignTune()` uses function-valued columns (for example, spending functions) in results tables. This method provides a stable, readable label for such functions to keep printing and plotting robust.

Usage

```
## S3 method for class ``function``
toString(x, ...)
```

Arguments

x	A function.
...	Unused (included for S3 method compatibility).

Value

A character scalar.

Examples

```
toString(stats::rnorm)
```

tune_choice	<i>Categorical choices</i>
-------------	----------------------------

Description

tune_choice() defines a finite set of categorical choices. Each argument in ... is treated as one choice (including functions and other objects).

Usage

```
tune_choice(...)
```

Arguments

... Candidate values.

Value

A gstune_spec object.

Examples

```
tune_choice("A", "B")
```

tune_dep	<i>Dependent tuning specification</i>
----------	---------------------------------------

Description

tune_dep() defines candidates for one argument as a function of other arguments.

Usage

```
tune_dep(depends_on, map)
```

Arguments

depends_on Character vector of argument names this specification depends on.
 map A function returning either a tune_*() specification or a fixed value. The function should have arguments matching depends_on (or use ...).

Value

A gstune_spec object.

Examples

```
# sfupar depends on sfu
tune_dep(
  depends_on = "sfu",
  map = function(sfu) {
    if (identical(sfu, gsDesign::sfLDOF)) tune_fixed(0) else tune_seq(-4, 4, 9)
  }
)
```

tune_fixed	<i>Fixed (non-tuned) value</i>
------------	--------------------------------

Description

Use `tune_fixed()` to explicitly mark a value as fixed. This is mainly useful inside dependent specifications such as `tune_dep()`.

Usage

```
tune_fixed(x)
```

Arguments

x Any R object.

Value

A `gstune_spec` object.

Examples

```
tune_fixed(0.025)
```

tune_int	<i>Integer sequence candidates</i>
----------	------------------------------------

Description

Integer sequence candidates

Usage

```
tune_int(from, to, by = 1)
```

Arguments

from, to Integer scalars.
by Integer scalar step size.

Value

A `gstune_spec` object.

Examples

```
tune_int(2, 5)
```

tune_seq	<i>Numeric sequence candidates</i>
----------	------------------------------------

Description

Numeric sequence candidates

Usage

```
tune_seq(from, to, length_out)
```

Arguments

from, to Numeric scalars.
length_out Integer scalar, the number of candidates.

Value

A `gstune_spec` object.

Examples

```
tune_seq(0.55, 0.75, length_out = 5)
```

tune_specs	<i>Tune specifications</i>
------------	----------------------------

Description

`gsDesignTune()` and `gsSurvTune()` treat most arguments as fixed values. Wrap an argument in a `tune_*` specification to explore candidate values.

tune_values	<i>Explicit candidate values</i>
-------------	----------------------------------

Description

tune_values() defines a finite set of candidate values. Values are provided as a list so vector-valued candidates (for example, timing) are treated as atomic.

Usage

```
tune_values(values)
```

Arguments

values A list of candidate values.

Value

A gstune_spec object.

Examples

```
tune_values(list(0.55, 0.65, 0.75))  
tune_values(list(c(0.33, 0.67, 1), c(0.5, 0.75, 1)))
```

Index

`gsDesign::gsDesign()`, 2, 3
`gsDesign::gsSurv()`, 3, 8
`gsDesign::gsSurvCalendar()`, 7
`gsDesignTune`, 2
`gsDesignTune()`, 11
`GSDTuneJob`, 3
`gsSurvCalendarTune`, 7
`gsSurvTune`, 8

`spending_specs`, 8
`SpendingFamily`, 9
`SpendingSpec`, 10

`tinytable::tt()`, 5
`toString.function`, 11
`tune_choice`, 12
`tune_dep`, 12
`tune_fixed`, 13
`tune_int`, 13
`tune_seq`, 14
`tune_specs`, 14
`tune_values`, 15