Package 'cvxoptdes'

November 19, 2026

Type Package

Title Convex Optimal Experiment Design

Version 1.0.0

Date 2026-11-01

Description Convex optimal experiment design across discrete candidate grids using open-source optimization solvers. Specifies and optimizes designs using an object-oriented interface (via 'R6') for linear models, generalized linear/additive models, nonlinear models, and random intercept linear mixed models. Optimization and augmentation of weighted, penalized and convex constrained approximate designs is supported using the Splitting Cone Solver (SCS). Optimal rounding to integer-valued designs is supported using point-exchange methods or mixed-integer programming with the 'HiGHS' solver. For details, see Chau et al. (2026) <doi:10.48550/arXiv.2610.14288>.

BugReports https://github.com/openanalytics/cvxoptdes/issues

URL https://repos.openanalytics.eu/docs/cvxoptdes/, https://github.com/openanalytics/cvxoptdes

Depends R (>= 4.1.0)

Imports methods, Matrix, stats, R6, reformulas, progress, mgcv, mirai

Suggests highs, knitr, rmarkdown

Encoding UTF-8

Language en-US

License Apache License (>= 2.0)

SystemRequirements GNU make, CMake

Biarch true

LazyData true

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Joris Chau [aut, cre, cph]

Maintainer Joris Chau <joris.chau@openanalytics.eu>

Repository CRAN

Date/Publication 2026-11-19 11:10:02 UTC

Contents

ethanol 2
fullerenes 3
gam_design 4
glm_design 15
imidazoles 27
lme_design 28
lm_design 40
nlm_design 56
scs_control_dflt 66

Index **69**

`ethanol` *Ethanol dataset*

Description

This dataset reports 88 measurements from an experiment in which ethanol was burned in a single cylinder automobile test engine. The data originates from the SemiPar package, which is now archived.

Usage

```
ethanol
```

Format

A data.frame containing 88 measurements with columns:

NOx numeric, concentration of nitric oxide (NO) and nitrogen dioxide (NO2) in engine exhaust, normalized by the work done by the engine

C numeric, the compression ratio of the engine

E numeric, the equivalence ratio at which the engine was run – a measure of the richness of the air/ethanol mix

Value

data.frame

Source

Brinkman, N.D. (1981). Ethanol fuel – a single-cylinder engine study of efficiency and exhaust emissions. SAE transactions Vol. 90, No 810345, 1410–1424.

References

Ruppert, D., Wand, M.P. and Carroll, R.J. (2003). *Semiparametric Regression*. Cambridge University Press.

Examples

```
ethanol
```

`fullerenes` *Fullerenes dataset*

Description

This dataset reports the production of o-xylenyl adducts of Buckminster fullerenes. Three process conditions (temperature, reaction time and ratio of sultine to C60) are varied to maximize the mole fraction of the desired ' product.

Usage

```
fullerenes
```

Format

A data.frame containing 120 experiments with columns:

reaction_time numeric, reaction time in flow reactor (min)

sultine_amount numeric, relative concentration of sultine to C60

temperature numeric, temperature of the reaction (Celsius)

product_amount numeric, mole fraction of the desired product

Value

data.frame

Source

Olympus: Fullerenes dataset

Examples

```
head(fullerenes)
```

gam_design *Penalized generalized additive model optimal design*

Description

An R6 class for managing penalized locally optimal experiment designs for generalized additive models. This class encapsulates the generalized additive model at a set of nominal parameter values, the discrete design grid, optimization of an approximate continuous design, augmentation of an existing design, and optimal rounding to exact integer designs.

Value

R6-object of class gam_design if called with gam_design$new().

Optimal design problem

The following penalized optimization problem is solved by $optimize():

$$w^* \;=\; \arg\max_{w} \Phi(M(w|X, v, \theta, F, g) + \lambda I) - \alpha \cdot c^T w$$

where,

- $w = \{w_1, \ldots, w_n\}$ is the vector of design weights.
- $c = \{c_1, \ldots, c_n\}$ is the cost assigned to each design weight with scaling/penalty parameter α.
- $v = \{v_1, \ldots, v_n\}$ is a vector of fixed user weights.
- $M(w|X, v, \theta, F, g)$ is the information matrix depending only on the weights w and v, and the design matrix X by default. If a non-trivial distribution family F is chosen, the information matrix also depends on the nominal parameter values θ, the chosen (exponential) distribution family F and the link function g.
- Φ is the chosen optimality criterion, generally a function of the information matrix.

By default, the error distribution family F is Gaussian with identity link function g. This implies that the information matrix (conditional on the error variance) is independent of the model parameters and given by:

$$M(w|X, v) \;\propto\; \sum_{i=1}^{n} w_i v_i x_i x_i^T$$

If F is not Gaussian or Gaussian with a non-canonical link function g, the information matrix is given by:

$$M(w|X, v, \theta, F, g) \;=\; \sum_{i=1}^{n} w_i v_i u(x_i, \theta) x_i x_i^T$$

with weights $u(x_i, \theta) = \frac{1}{V(\mu_i)[g'(\mu_i)]^2}$ depending on the parameter values θ. Here, $\mu = E[Y] = b'(\zeta)$ and $V(\mu) = b''(\zeta)$, with $b(\zeta)$ is the cumulant function of the exponential family F. See glm_design for the exact details.

Augmented optimal design

Given a vector of initial design weights w_0 and associated contribution weight γ, the following augmented penalized optimization problem is solved by `$augment()`:

$$w^* \;=\; \arg\max_w \Phi(M(w|\gamma, w_0, X, v, \theta, F, g) + \lambda I) - \alpha \cdot c^T w$$

using the augmented information matrix,

$$M(w|\gamma, w_0, X, v, \theta, F, g) \;= (1-\gamma)M(w_0|X, v, \theta, F, g) + \gamma M(w|X, v, \theta, F, g)$$

Stratified optimal design

The `$strata()` method stratifies the design by requiring that individual strata $X_1, \ldots, X_S$ are represented proportionally with given weights $p_1, \ldots, p_S$, ($\sum_i p_i = 1$). If stratification is applied, the following optimization problem is solved by `$optimize()`:

$$\begin{aligned} w^* \;=\; & \arg\max_w \Phi(M(w|X, v, \theta, F, g) + \lambda I) - \alpha \cdot c^T w, \\ & \text{subject to } \textstyle\sum_{i \in X_s} w_i = p_s, \qquad \text{for } s = 1, \ldots, S \end{aligned}$$

using the same notation as in the 'Optimal design problem' section above. This means that the total weight assigned to the design points in stratum X_i is equal to p_i for each $i = 1, \ldots, S$. Similarly, if stratification is applied, `$augment()` solves the augmented optimization problem *with* stratification constraints.

Sparse optimal design

The `$sparsify()` method solves the penalized optimization problem with an extra sparseness penalty $P(w)$ in the objective to promote sparsity in the approximate design w^*. See lm_design for the available penalty functions.

Optimality criteria

The optimality criteria $\Phi(\cdot)$ supported by `$optimize()`, `$augment()` and `$sparsify()` are: *D*-, *A*-, *I*-, *G*-, and *alias*-optimality, see lm_design for the definitions.

Super classes

`lm_design` -> `glm_design` -> `gam_design`

Public fields

`formula` a GAM model defined as a one-sided formula.

`theta` a named vector or list of nominal parameter values at which the information matrix $M(w|X, \theta, F, g)$ is evaluated. Only used if `$family` is not `gaussian(link = "identity")` (the default).

`family` a description of the model distribution family and link function. See family for details of family functions.

`data` a data frame or matrix containing all design points (rows) at which `formula` can be evaluated.

`weights` an optional numeric vector containing fixed user weights associated to each design point (row) in `data`.

`cost` an optional numeric vector containing costs associated to each design point (row) in `data`.

`design_weights` a numeric vector with optimal design weights populated after optimization with `$optimize()`, `$augment()`, or `$sparsify()`. The attribute `criterion` lists the used design criterion and the achieved optimal criterion value.

`strata` a list with elements `strata` and `proportions` imposing a partition of the design points into individual (disjoint) strata with given weight proportions.

`alpha` the cost scale/penalty parameter in the penalized optimization problem, only used if `$cost` is available.

`lambda` the ridge penalty added to the diagonal of the information matrix in the optimization problem. The ridge penalty is not used

`upper` the design weight upper limit. The default is 1, corresponding to no upper limit constraint on the design weights.

Active bindings

`X` the design matrix obtained via `mgcv::gam(formula, data = data, fit = FALSE, ...)` given the design points in `data` and the model `formula`.

Methods

Public methods:

- `gam_design$new()`
- `gam_design$update()`
- `gam_design$stratify()`
- `gam_design$augment()`
- `gam_design$optimize()`
- `gam_design$sparsify()`
- `gam_design$round()`
- `gam_design$crit()`
- `gam_design$vcov()`
- `gam_design$sep()`
- `gam_design$corr()`
- `gam_design$clone()`

`gam_design$new()`: Constructor. Initializes the experimental design to optimize with a linear model formula, discrete design grid, and optionally a distribution family with link function (and nominal set of parameter values).

Usage:

```
gam_design$new(
  formula,
  data,
  theta = NULL,
  family = NULL,
  weights = NULL,
  cost = NULL,
```

```
  contrasts = NULL,
  alpha = 1,
  lambda = 0,
  upper = 1,
  ...
)
```

Arguments:

`formula` a GAM model defined as a one-sided formula, see `gam`.

`data` a data frame or matrix containing all design points (rows) at which `formula` can be evaluated.

`theta` a named or unnamed vector or list of nominal parameter values at which the information matrix $M(w|X, \theta, F, g)$ is evaluated. Only needed if `$family` is not `NULL`.

`family` a family function describing the error distribution and link function in the model. Defaults to the gaussian family with `"identity"` link function.

`weights` an optional numeric vector containing fixed user weights associated to each design point (row) in `data`. Also accepts a one-sided formula made up of column names present in `data`. If `NULL`, the unweighted optimal design problem is solved.

`cost` an optional numeric vector containing costs associated to each design point (row) in `data`. Also accepts a one-sided formula made up of column names present in `data`. If `NULL`, the standard (non-penalized) optimal design problem is solved.

`contrasts` optional contrasts used when evaluating the linear model formula.

`alpha` numeric value, cost scale/penalty parameter in the penalized optimization problem, only used if `$cost` is available. The default is 1.

`lambda` numeric value, ridge penalty added to the diagonal of the information matrix in the penalized optimization problem. The default is 0.

`upper` numeric value in (0, 1], design weight upper limit. The default is 1, corresponding to no constraint on the design weights.

`...` any additional arguments passed to `mgcv::gam(formula, data = data, fit = FALSE, ...)` to evaluate the design matrix.

Returns: `gam_design` R6-object

`gam_design$update()`: Updates the fields of an existing object. Inputs should be named `formula`, `data`, `theta`, `family`, `weights`, `cost`, `alpha`, `lambda` and/or `upper` respectively.

Usage:

```
gam_design$update(...)
```

Arguments:

`...` named arguments `formula`, `data`, `theta`, `family`, `weights`, `cost`, `alpha`, `lambda` and/or `upper`.

Returns: none

`gam_design$stratify()`: Stratify the design by requiring that individual strata are represented proportionally. Call with `strata = NULL` to unset previously defined `strata`.

Usage:

```
gam_design$stratify(strata, proportions = NULL)
```

Arguments:

`strata` a factor variable of the same length as the number of rows in `data`. The factor levels determine the partition of the design points into individual (disjoint) strata. Also accepts a one-sided formula made up of column names present in `data`, in which case the strata are defined by the interaction between terms in the model.frame obtained from the formula.

`proportions` an optional numeric vector containing the weight proportions of the individual strata. Must be of the same length as the number of factor levels in `strata`. If `NULL`, each stratum is assigned the same weight proportion.

Returns: none

`gam_design$augment()`**:** Optimizes the augmented penalized approximate design given a vector of initial design weights

Usage:

```
gam_design$augment(
  design_weights,
  gamma = 0.5,
  criterion = c("D", "A", "I", "G", "alias"),
  scs_control = scs_control_dflt(),
  verbose = FALSE,
  return_value = TRUE
)
```

Arguments:

`design_weights` a numeric vector with initial design weights, must be the same length as the number of design points in `data`. Also accepts a one-sided formula made up of parameters in `theta` and column names in `data`.

`gamma` numeric contribution weight of the initial design in the augmented design problem, see also 'Augmented optimal design' above. Defaults to 0.5.

`criterion` character string specifying the optimality criterion to use, see also 'Details'. The following choices are supported:

`"D"` D-optimality (default).
`"A"` A-optimality.
`"I"` I-optimality.
`"G"` G-optimality.
`"alias"` alias-optimality.

`scs_control` an optional list of control parameters to tune the SCS optimization. See `scs_control_dflt` for the available control parameters and their default values.

`verbose` display verbose messages printed by the scs solver. Defaults to `FALSE`.

`return_value` return the optimal design weights. If set to `FALSE`, the design weights should be retrieved from the `design_weights` field.

Returns: numeric vector if `return_value = TRUE`, otherwise none

`gam_design$optimize()`**:** Optimizes the penalized approximate design based on the selected criterion.

Usage:

```
gam_design$optimize(
  criterion = c("D", "A", "I", "G", "alias"),
  scs_control = scs_control_dflt(),
  verbose = FALSE,
  return_value = TRUE
)
```

Arguments:

`criterion` character string specifying the optimality criterion to use, see also 'Details'. The following choices are supported:

"D" D-optimality (default).
"A" A-optimality.
"I" I-optimality.
"G" G-optimality.
"alias" alias-optimality.

`scs_control` an optional list of control parameters to tune the SCS optimization. See `scs_control_dflt` for the available control parameters and their default values.

`verbose` display verbose messages printed by the scs solver. Defaults to `FALSE`.

`return_value` return the optimal design weights. If set to `FALSE`, the design weights should be retrieved from the `design_weights` field.

Returns: numeric vector if `return_value = TRUE`, otherwise none

`gam_design$sparsify()`**:** Sparsify the penalized approximate design by iterative reweighting of the optimization problem.

Usage:

```
gam_design$sparsify(
  max_iter,
  criterion = c("D", "A", "I", "G", "alias"),
  reweight_fun = c("entropy", "log-sum", "power"),
  eps = .Machine$double.eps,
  tol = .Machine$double.eps^(1/4),
  p = 0.5,
  design_weights = NULL,
  scs_control = scs_control_dflt(),
  return_value = TRUE,
  show_progress = TRUE
)
```

Arguments:

`max_iter` positive integer, maximum number of reweighted optimizations/iterations to run.

`criterion` character string specifying the optimality criterion to use, see also 'Details'. The following choices are supported:

"D" D-optimality (default).
"A" A-optimality.
"I" I-optimality.

"G" G-optimality.
"alias" alias-optimality.

reweight_fun character reweighting function, one of "entropy", "log-sum", "power", see lm_design for the definitions.

eps positive numeric value to ensure finite weights in reweighting functions.

tol numeric tolerance used as a threshold for mean absolute relative difference in design weights between iterations to stop early.

p positive numeric power with $0 < p < 1$. Only used if reweight_fun = "power".

design_weights (optional) vector of design weights used as initial approximate design.

scs_control an optional list of control parameters to tune the SCS optimization. See scs_control_dflt for the available control parameters and their default values.

return_value return the optimal design weights. If set to FALSE, the design weights should be retrieved from the design_weights field.

show_progress logical, display a progress bar during the iterative reweighting procedure. Defaults to TRUE.

Details: The sparseness cost penalties are scaled by the same alpha tuning parameter as for a user-specific cost vector, see scs_control_dflt. If a cost vector c_1,...,c_n associated to the design weights is already specified. The sparseness penalty costs are multiplied by the existing cost vector.

Returns: numeric vector if return_value = TRUE, otherwise none

gam_design$round(): Round penalized approximate design to an exact (integer) design. By default, the approximate design is initialized from the design_weights field, which is the optimal (approximate) design calculated by $optimize(). For method "optimal", ridge and cost penalties and upper limit constraints present in $lambda, $cost and $upper (if specified) are included in the criterion optimized by the point-exchange algorithm. To override the initial approximate (optimal) design, a vector of design weights can be passed using the design_weights argument. The vector must be the same length as the number of design points in data. If design_weights is set to FALSE, the initial design is generated by random sampling.

Usage:

```
gam_design$round(
  m,
  method = c("optimal", "efficient", "dist-sum", "dist-min"),
  max_iter = 100,
  n_repeats = 10,
  replicates = TRUE,
  tol = .Machine$double.eps^(1/3),
  seed = NULL,
  design_weights = TRUE,
  augment_design = NULL,
  show_progress = TRUE,
  ...
)
```

Arguments:

m integer number of design points included in the exact design.

`method` the method used to calculate the exact design. The following choices are supported:

`"optimal"` optimality criterion point-exchange algorithm (default).
`"efficient"` efficient rounding method of Pukelsheim & Reider (1992).
`"dist-sum"` maximize total pairwise (Euclidean) distance between design points.
`"dist-min"` maximize minimum pairwise (Euclidean) distance between design points.

`max_iter` integer number of maximum iterations in point-exchange algorithm. Only used when `method = "optimal"`, `method = "dist-sum"` or `method = "dist-min"`.

`n_repeats` integer number of (random) repeats of the rounding/exchange algorithm. The exact design with the highest criterion value is returned. Defaults to 10.

`replicates` logical, allow replicate design points in the exact design, defaults to `TRUE`. Can also be an integer number, limiting the maximum number of allowed replicates per design point. This argument overrides the `$upper` value constraint if existing.

`tol` numeric tolerance used as lower threshold to continue iterations in point-exchange algorithm.

`seed` (optional) random seed used to generate jitter to break ties in rounding the initial exact design.

`design_weights` (optional) vector of design weights used as initial approximate design, with length equal to the number of available design points. If `TRUE` (default), the approximate design is initialized from the `design_weights` field. If `FALSE`, the initial approximate design is generated by random sampling.

`augment_design` (optional) integer design to augment, with length equal to the number of available design points.

`show_progress` logical, display a progress bar during the rounding procedure (only if `n_repeats > 1`). Defaults to `TRUE`.

`...` additional arguments to tune algorithm settings:

- `criterion`, character string specifying the optimality criterion for the rounding procedure in case `design_weights` has no `"criterion"` attribute, or to override the criterion specified in the `"criterion"` attribute. If specified, must be one of `"D"`, `"A"`, `"I"`, `"G"` or `"alias"`.

Returns: numeric vector

`gam_design$crit()`: Evaluate optimality criterion of a penalized approximate or rounded (exact) design. The approximate design is initialized from the `design_weights` field, which is the optimal (approximate) design calculated by `$optimize()`. To override the default approximate (optimal) design, a vector of design weights can be passed using the `design_weights` argument.

Usage:

```
gam_design$crit(
  design_weights = NULL,
  criterion = c("D", "A", "I", "G", "alias"),
  lambda = TRUE,
  alpha = TRUE
)
```

Arguments:

`design_weights` (optional) vector of design weights for which to evaluate optimality criterion.

`criterion` character string specifying the optimality criterion to use, see also 'Details'. The following choices are supported:

`"D"` D-optimality (default).
`"A"` A-optimality.
`"I"` I-optimality.
`"G"` G-optimality.
`"alias"` alias-optimality.

`lambda` logical, whether to include the ridge penalty in the calculated optimality criterion (see 'Optimality criteria'). Only relevant if `$lambda` is positive.

`alpha` logical, whether to include cost penalties in the calculated optimality criterion (see 'Optimality criteria'). Only relevant if cost penalties are present in the `$cost` field.

Returns: numeric criterion value

`gam_design$vcov()`: Calculates the variance-covariance matrix of the model parameters based on an approximate or rounded (exact) design $(w_1, \ldots, w_n)$ according to:

$$\Sigma \;=\; (X^T U^{1/2} \Omega U^{1/2} X)^{-1}$$

where Ω is $\mathrm{diag}(\omega_1, \ldots, \omega_n)$ with $\omega_i = w_i v_i$, and $U^{1/2}$ is $\mathrm{diag}(\sqrt{u(x_1, \theta)}, \ldots, \sqrt{u(x_n, \theta)})$, with GLM weights $u(x_i, \theta) = \frac{1}{V(\mu_i)[g'(\mu_i)]^2}$ evaluated at the nominal parameter values. The design weights are initialized from the `design_weights` field, which is the optimal (approximate) design calculated by `$optimize()`. Use the `m` argument to evaluate the covariance matrix for a specific number of design points, in which case the sum of the design weights is scaled to `m`. To override the default approximate (optimal) design weights, a vector of design weights can be passed using the `design_weights` argument. The vector must be the same length as the number of design points in `data`. If the vector of design weights is already an integer design, leave `m` unspecified, in which case the design is not rescaled.

Usage:

```
gam_design$vcov(m, design_weights = NULL, sigma2 = 1)
```

Arguments:

`m` (optional) integer number of design points at which to evaluate the estimated covariance matrix. If missing, the design weights remain unscaled, which for the default approximate (optimal) design is equivalent to `m = 1`.

`design_weights` (optional) vector of design weights for which to evaluate the estimated covariance matrix.

`sigma2` numeric error variance used to scale the covariance matrix. Defaults to 1, returning the unscaled covariance matrix.

Returns: numeric matrix

`gam_design$sep()`: Calculates the standard error of prediction or the square root prediction variance across the design grid based on an approximate or rounded (exact) design $(w_1, \ldots, w_n)$ according to:

$$\mathrm{SEP}(x_i) \;=\; \sigma \sqrt{x_i^T \Sigma x_i}, \quad \text{for} i = 1, \ldots, n$$

where Σ is the covariance matrix as returned by `$vcov()`. If `type = "response"`, the standard errors of prediction are transformed to the response scale by an application of the Delta method:

$$\mathrm{SEP}_y(x_i) \;=\; \mathrm{SEP}(x_i)\left|\frac{\partial \mu}{\partial \eta}\right|$$

Use the `newdata` argument to evaluate the standard error of prediction at design points not in the candidate design grid defined by `$data`. The design weights are initialized from the `design_weights` field, which is the optimal (approximate) design calculated by `$optimize()`. To override the default approximate (optimal) design weights, a vector of design weights can be passed using the `design_weights` argument. The vector must be the same length as the number of design points in `data`.

Usage:

```
gam_design$sep(
  m,
  newdata,
  design_weights = NULL,
  type = c("link", "response"),
  sigma2 = 1
)
```

Arguments:

`m` (optional) integer number of design points at which to evaluate the estimated covariance matrix. If missing, the design weights remain unscaled, which for the default approximate (optimal) design is equivalent to `m = 1`.

`newdata` (optional) data frame or matrix with design points (rows) at which to return the standard error of prediction and which can be evaluated by `$formula`.

`design_weights` (optional) vector of design weights for which to evaluate the estimated covariance matrix.

`type` the type of prediction required. The default choice `"link"` returns the standard error of prediction on the scale of the linear predictors, whereas `"response"` returns the standard error of prediction on the scale of the response variable.

`sigma2` numeric error variance used to scale the standard errors. Defaults to 1, returning the standardized standard error of prediction.

Returns: numeric vector

`gam_design$corr()`**:** Calculates the correlation matrix of the columns of the design matrix based on an approximate or rounded (exact) design $(w_1, \ldots, w_n)$ according to:

$$R \;=\; D^{-1/2}(X_c^T U^{1/2} \Omega U^{1/2} X_c) D^{-1/2}$$

where Ω is $\mathrm{diag}(\omega_1, \ldots, \omega_n)$ with $\omega_i = w_i v_i$, and X_c is the design matrix X centered by the (weighted) column means:

$$\frac{\sum_{i=1}^n \omega_i X_{ij}}{\sum_{i=1}^n \omega_i}, \quad j = 1, \ldots, p$$

$D^{-1/2}$ is $\mathrm{diag}(1/\sqrt{D_1}, \ldots, 1/\sqrt{D_p})$, with $D_1, \ldots, D_p$ the diagonal elements of $X_c^T U^{1/2} \Omega U^{1/2} X_c$. If `center = FALSE`, the uncentered correlation matrix is returned, which corresponds to the scale-invariant (unit diagonal) version of the information matrix $M(w|X, v, \theta, F, g)$. The design weights

are initialized from the `design_weights` field, which is the optimal (approximate) design calculated by `$optimize()`. To override the default approximate (optimal) design weights, a vector of design weights can be passed using the `design_weights` argument. The vector must be the same length as the number of design points in `data`.

Usage:

```
gam_design$corr(design_weights = NULL, center = TRUE)
```

Arguments:

`design_weights` (optional) vector of design weights for which to evaluate the information matrix.

`center` logical, whether to center the design columns when evaluating the correlation matrix, defaults to `TRUE`.

Returns: numeric matrix

`gam_design$clone()`**:** The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

`family`

`gam`

Examples

```
## Ethanol dataset

## initialize model
design <- gam_design$new(
  formula = ~s(E) + C,
  data = ethanol
)
## approximate I-optimal design
design$optimize(criterion = "I")
## integer optimal design w/ 11 design points
(exact_design <- design$round(m = 11, method = "optimal", criterion = "I", seed = 1))
design$subset(replicates = exact_design)

## GAM fitted to integer design
gam_fit <- mgcv::gam(NOx ~ s(E) + C, data = ethanol[exact_design > 0, ])
summary(gam_fit)

## Volcano height dataset

## pre-process data
data(volcano)
volcano <- volcano[c(TRUE, FALSE), c(TRUE, FALSE)]
```

```
data <- expand.grid(
  x1 = seq_len(dim(volcano)[1]),
  x2 = seq_len(dim(volcano)[2])
)
data[["y"]] <- c(volcano)

## initialize model
design <- gam_design$new(
  formula = ~s(x1, x2),
  data = data
)
## integer I-optimal design w/ 30 design points
exact_design <- design$round(
  m = 30, method = "optimal", criterion = "I", design_weights = FALSE
)
design$crit(exact_design, criterion = "I")

## GAM fitted to integer design
gam_fit <- mgcv::gam(y ~ s(x1, x2), data = data[exact_design > 0, ])
summary(gam_fit)
```

`glm_design` *Penalized generalized linear model locally optimal design*

Description

An R6 class for managing penalized locally optimal experiment designs for generalized linear models. This class encapsulates the generalized linear model at a set of nominal parameter values, the discrete design grid, optimization of an approximate continuous design, augmentation of an existing design, and optimal rounding to exact integer designs.

Value

R6-object of class `glm_design` if called with `glm_design$new()`.

Optimal design problem

The following penalized optimization problem is solved by `$optimize()`:

$$w^* \;=\; \arg\max_{w} \Phi(M(w|X, v, \theta, F, g) + \lambda I) - \alpha \cdot c^T w$$

where,

- $w = \{w_1, \ldots, w_n\}$ is the vector of design weights.
- $c = \{c_1, \ldots, c_n\}$ is the cost assigned to each design weight with scaling/penalty parameter α.
- $v = \{v_1, \ldots, v_n\}$ is a vector of fixed user weights.

- $M(w|X, v, \theta, F, g)$ is the information matrix depending on the weights w and v, the design matrix X, the nominal parameter values θ, the chosen (exponential) distribution family F and the link function g.
- Φ is the chosen optimality criterion, generally a function of the information matrix.

In standard canonical form, the probability density (or mass) function of the exponential family F is:

$$f(y \mid \zeta, \phi) = \exp\left(\frac{y\zeta - b(\zeta)}{a(\phi)} + c(y, \phi)\right)$$

The cumulant function $b(\zeta)$ admits $\mu = E[Y] = b'(\zeta)$ and $V(\mu) = b''(\zeta)$, the variance function expressed as a function of the mean. Given the link function g, the systematic component is $g(\mu) = \eta = x^T\theta$. The information matrix is given by:

$$M(w|X, v, \theta, F, g) \;=\; \sum_{i=1}^{n} w_i v_i u(x_i, \theta) x_i x_i^T$$

with weights $u(x_i, \theta) = \frac{1}{V(\mu_i)[g'(\mu_i)]^2}$ causing the information matrix to depend on the parameter values θ, unlike the information matrix $M(w|X, v)$ for a standard linear model.

Augmented optimal design

Given a vector of initial design weights w_0 and associated contribution weight γ, the following augmented penalized optimization problem is solved by `$augment()`:

$$w^* \;=\; \arg\max_w \Phi(M(w|\gamma, w_0, X, v, \theta, F, g) + \lambda I) - \alpha \cdot c^T w$$

using the augmented information matrix,

$$M(w|\gamma, w_0, X, v, \theta, F, g) \;= (1-\gamma)M(w_0|X, v, \theta, F, g) + \gamma M(w|X, v, \theta, F, g)$$

Stratified optimal design

The `$strata()` method stratifies the design by requiring that individual strata $X_1, \ldots, X_S$ are represented proportionally with given weights $p_1, \ldots, p_S$, ($\sum_i p_i = 1$). If stratification is applied, the following optimization problem is solved by `$optimize()`:

$$\begin{aligned} w^* \;=\; & \arg\max_w \Phi(M(w|X, v, \theta, F, g) + \lambda I) - \alpha \cdot c^T w, \\ & \text{subject to } \textstyle\sum_{i \in X_s} w_i = p_s, \qquad \text{for } s = 1, \ldots, S \end{aligned}$$

using the same notation as in the 'Optimal design problem' section above. This means that the total weight assigned to the design points in stratum X_i is equal to p_i for each $i = 1, \ldots, S$. Similarly, if stratification is applied, `$augment()` solves the augmented optimization problem *with* stratification constraints.

Sparse optimal design

The `$sparsify()` method solves the penalized optimization problem with an extra sparseness penalty $P(w)$ in the objective to promote sparsity in the approximate design w^*. See lm_design for the available penalty functions.

Optimality criteria

The optimality criteria $\Phi(\cdot)$ supported by `$optimize()`, `$augment()` and `$sparsify()` are: *D*-, *A*-, *I*-, *G*-, and *alias*-optimality, see lm_design for the definitions.

Super class

`lm_design` -> `glm_design`

Public fields

`formula` a linear model defined as a one-sided formula.

`theta` a named vector or list of nominal parameter values at which the information matrix $M(w|X, \theta, F, g)$ is evaluated.

`family` a description of the model distribution family and link function. See family for details of family functions.

`data` a data frame or matrix containing all design points (rows) at which `formula` can be evaluated.

`weights` an optional numeric vector containing fixed user weights associated to each design point (row) in `data`.

`cost` an optional numeric vector containing costs associated to each design point (row) in `data`.

`design_weights` a numeric vector with optimal design weights populated after optimization with `$optimize()`, `$augment()`, or `$sparsify()`. The attribute `criterion` lists the used design criterion and the achieved optimal criterion value.

`strata` a list with elements `strata` and `proportions` imposing a partition of the design points into individual (disjoint) strata with given weight proportions.

`alpha` the cost scale/penalty parameter in the penalized optimization problem, only used if `$cost` is available.

`lambda` the ridge penalty added to the diagonal of the information matrix in the optimization problem. The ridge penalty is not used

`upper` the design weight upper limit. The default is 1, corresponding to no upper limit constraint on the design weights.

Active bindings

`X` the design matrix obtained from all design points in `data` and the model `formula`.

Methods

Public methods:

- `glm_design$new()`
- `glm_design$update()`
- `glm_design$stratify()`
- `glm_design$augment()`
- `glm_design$optimize()`
- `glm_design$sparsify()`

- glm_design$round()
- glm_design$crit()
- glm_design$vcov()
- glm_design$sep()
- glm_design$corr()
- glm_design$clone()

`glm_design$new()`: Constructor. Initializes the experimental design to optimize with a linear model formula, discrete design grid, nominal parameter values and distribution family with link function.

Usage:

```
glm_design$new(
  formula,
  data,
  theta,
  family = gaussian(),
  weights = NULL,
  cost = NULL,
  contrasts = NULL,
  alpha = 1,
  lambda = 0,
  upper = 1,
  ...
)
```

Arguments:

`formula` a linear model defined as a one-sided formula.

`data` a data frame or matrix containing all design points (rows) at which `formula` can be evaluated.

`theta` a named or unnamed vector or list of nominal parameter values at which the information matrix $M(w|X, \theta, F, g)$ is evaluated.

`family` a family function describing the error distribution and link function in the model. Defaults to the gaussian family with `"identity"` link function.

`weights` an optional numeric vector containing fixed user weights associated to each design point (row) in `data`. Also accepts a one-sided formula made up of column names present in `data`. If `NULL`, the unweighted optimal design problem is solved.

`cost` an optional numeric vector containing costs associated to each design point (row) in `data`. Also accepts a one-sided formula made up of column names present in `data`. If `NULL`, the standard (non-penalized) optimal design problem is solved.

`contrasts` optional contrasts used when evaluating the linear model formula.

`alpha` numeric value, cost scale/penalty parameter in the penalized optimization problem, only used if `$cost` is available. The default is 1.

`lambda` numeric value, ridge penalty added to the diagonal of the information matrix in the penalized optimization problem. The default is 0.

`upper` numeric value in (0, 1], design weight upper limit. The default is 1, corresponding to no constraint on the design weights.

... any additional arguments passed to model.matrix.

Returns: glm_design R6-object

`glm_design$update()`: Updates the fields of an existing object. Inputs should be named `formula`, `data`, `theta`, `family`, `weights`, `cost`, `alpha`, `lambda` and/or `upper`.

Usage:

```
glm_design$update(...)
```

Arguments:

... named arguments `formula`, `data`, `theta`, `family`, `weights`, `cost`, `alpha`, `lambda` and/or `upper`.

Returns: none

`glm_design$stratify()`: Stratify the design by requiring that individual strata are represented proportionally. Call with `strata = NULL` to unset previously defined `strata`.

Usage:

```
glm_design$stratify(strata, proportions = NULL)
```

Arguments:

`strata` a factor variable of the same length as the number of rows in `data`. The factor levels determine the partition of the design points into individual (disjoint) strata. Also accepts a one-sided formula made up of column names present in `data`, in which case the strata are defined by the interaction between terms in the model.frame obtained from the formula.

`proportions` an optional numeric vector containing the weight proportions of the individual strata. Must be of the same length as the number of factor levels in `strata`. If `NULL`, each stratum is assigned the same weight proportion.

Returns: none

`glm_design$augment()`: Optimizes the augmented penalized approximate design given a vector of initial design weights

Usage:

```
glm_design$augment(
  design_weights,
  gamma = 0.5,
  criterion = c("D", "A", "I", "G", "alias"),
  scs_control = scs_control_dflt(),
  verbose = FALSE,
  return_value = TRUE
)
```

Arguments:

`design_weights` a numeric vector with initial design weights, must be the same length as the number of design points in `data`. Also accepts a one-sided formula made up of parameters in `theta` and column names in `data`.

`gamma` numeric contribution weight of the initial design in the augmented design problem, see also 'Augmented optimal design' above. Defaults to 0.5.

`criterion` character string specifying the optimality criterion to use, see also 'Details'. The following choices are supported:

"D" D-optimality (default).
"A" A-optimality.
"I" I-optimality.
"G" G-optimality.
"alias" alias-optimality.

`scs_control` an optional list of control parameters to tune the SCS optimization. See `scs_control_dflt` for the available control parameters and their default values.

`verbose` display verbose messages printed by the scs solver. Defaults to `FALSE`.

`return_value` return the optimal design weights. If set to `FALSE`, the design weights should be retrieved from the `design_weights` field.

Returns: numeric vector if `return_value = TRUE`, otherwise none

`glm_design$optimize()`: Optimizes the penalized approximate design based on the selected criterion.

Usage:

```
glm_design$optimize(
  criterion = c("D", "A", "I", "G", "alias"),
  scs_control = scs_control_dflt(),
  verbose = FALSE,
  return_value = TRUE
)
```

Arguments:

`criterion` character string specifying the optimality criterion to use, see also 'Details'. The following choices are supported:

"D" D-optimality (default).
"A" A-optimality.
"I" I-optimality.
"G" G-optimality.
"alias" alias-optimality.

`scs_control` an optional list of control parameters to tune the SCS optimization. See `scs_control_dflt` for the available control parameters and their default values.

`verbose` display verbose messages printed by the scs solver. Defaults to `FALSE`.

`return_value` return the optimal design weights. If set to `FALSE`, the design weights should be retrieved from the `design_weights` field.

Returns: numeric vector if `return_value = TRUE`, otherwise none

`glm_design$sparsify()`: Sparsify the penalized approximate design by iterative reweighting of the optimization problem.

Usage:

```
glm_design$sparsify(
  max_iter,
  criterion = c("D", "A", "I", "G", "alias"),
  reweight_fun = c("entropy", "log-sum", "power"),
  eps = .Machine$double.eps,
  tol = .Machine$double.eps^(1/4),
  p = 0.5,
  design_weights = NULL,
  scs_control = scs_control_dflt(),
  return_value = TRUE,
  show_progress = TRUE
)
```

Arguments:

`max_iter` positive integer, maximum number of reweighted optimizations/iterations to run.

`criterion` character string specifying the optimality criterion to use, see also 'Details'. The following choices are supported:

"D" D-optimality (default).
"A" A-optimality.
"I" I-optimality.
"G" G-optimality.
"alias" alias-optimality.

`reweight_fun` character reweighting function, one of "entropy", "log-sum", "power", see lm_design for the definitions.

`eps` positive numeric value to ensure finite weights in reweighting functions.

`tol` numeric tolerance used as a threshold for mean absolute relative difference in design weights between iterations to stop early.

`p` positive numeric power with $0 < p < 1$. Only used if `reweight_fun = "power"`.

`design_weights` (optional) vector of design weights used as initial approximate design.

`scs_control` an optional list of control parameters to tune the SCS optimization. See `scs_control_dflt` for the available control parameters and their default values.

`return_value` return the optimal design weights. If set to `FALSE`, the design weights should be retrieved from the `design_weights` field.

`show_progress` logical, display a progress bar during the iterative reweighting procedure. Defaults to `TRUE`.

Details: The sparseness cost penalties are scaled by the same `alpha` tuning parameter as for a user-specific cost vector, see `scs_control_dflt`. If a cost vector `c_1,...,c_n` associated to the design weights is already specified. The sparseness penalty costs are multiplied by the existing cost vector.

Returns: numeric vector if `return_value = TRUE`, otherwise none

`glm_design$round()`: Round penalized approximate design to an exact (integer) design. By default, the approximate design is initialized from the `design_weights` field, which is the optimal (approximate) design calculated by `$optimize()`. For method "optimal", ridge and cost penalties and upper limit constraints present in `$lambda`, `$cost` and `$upper` (if specified) are included

in the criterion optimized by the point-exchange algorithm. To override the initial approximate (optimal) design, a vector of design weights can be passed using the `design_weights` argument. The vector must be the same length as the number of design points in `data`. If `design_weights` is set to `FALSE`, the initial design is generated by random sampling.

Usage:

```
glm_design$round(
  m,
  method = c("optimal", "efficient", "dist-sum", "dist-min"),
  max_iter = 100,
  n_repeats = 10,
  replicates = TRUE,
  tol = .Machine$double.eps^(1/3),
  seed = NULL,
  design_weights = TRUE,
  augment_design = NULL,
  show_progress = TRUE,
  ...
)
```

Arguments:

`m` integer number of design points included in the exact design.

`method` the method used to calculate the exact design. The following choices are supported:

`"optimal"` optimality criterion point-exchange algorithm (default).
`"efficient"` efficient rounding method of Pukelsheim & Reider (1992).
`"dist-sum"` maximize total pairwise (Euclidean) distance between design points.
`"dist-min"` maximize minimum pairwise (Euclidean) distance between design points.

`max_iter` integer number of maximum iterations in point-exchange algorithm. Only used when `method = "optimal"`, `method = "dist-sum"` or `method = "dist-min"`.

`n_repeats` integer number of (random) repeats of the rounding/exchange algorithm. The exact design with the highest criterion value is returned. Defaults to 10.

`replicates` logical, allow replicate design points in the exact design, defaults to `TRUE`. Can also be an integer number, limiting the maximum number of allowed replicates per design point. This argument overrides the `$upper` value constraint if existing.

`tol` numeric tolerance used as lower threshold to continue iterations in point-exchange algorithm.

`seed` (optional) random seed used to generate jitter to break ties in rounding the initial exact design.

`design_weights` (optional) vector of design weights used as initial approximate design, with length equal to the number of available design points. If `TRUE` (default), the approximate design is initialized from the `design_weights` field. If `FALSE`, the initial approximate design is generated by random sampling.

`augment_design` (optional) integer design to augment, with length equal to the number of available design points.

`show_progress` logical, display a progress bar during the rounding procedure (only if `n_repeats > 1`). Defaults to `TRUE`.

... additional arguments to tune algorithm settings:
- – criterion, character string specifying the optimality criterion for the rounding procedure in case design_weights has no "criterion" attribute, or to override the criterion specified in the "criterion" attribute. If specified, must be one of "D", "A", "I", "G" or "alias".

Returns: numeric vector

glm_design$crit(): Evaluate optimality criterion of a penalized approximate or rounded (exact) design. The approximate design is initialized from the design_weights field, which is the optimal (approximate) design calculated by $optimize(). To override the default approximate (optimal) design, a vector of design weights can be passed using the design_weights argument.

Usage:

```
glm_design$crit(
  design_weights = NULL,
  criterion = c("D", "A", "I", "G", "alias"),
  lambda = TRUE,
  alpha = TRUE
)
```

Arguments:

design_weights (optional) vector of design weights for which to evaluate optimality criterion.

criterion character string specifying the optimality criterion to use, see also 'Details'. The following choices are supported:

"D" D-optimality (default).
"A" A-optimality.
"I" I-optimality.
"G" G-optimality.
"alias" alias-optimality.

lambda logical, whether to include the ridge penalty in the calculated optimality criterion (see 'Optimality criteria'). Only relevant if $lambda is positive.

alpha logical, whether to include cost penalties in the calculated optimality criterion (see 'Optimality criteria'). Only relevant if cost penalties are present in the $cost field.

Returns: numeric criterion value

glm_design$vcov(): Calculates the variance-covariance matrix of the model parameters based on an approximate or rounded (exact) design $(w_1, \ldots, w_n)$ according to:

$$\Sigma \;=\; (X^T U^{1/2} \Omega U^{1/2} X)^{-1}$$

where Ω is $\mathrm{diag}(\omega_1, \ldots, \omega_n)$ with $\omega_i = w_i v_i$, and $U^{1/2}$ is $\mathrm{diag}(\sqrt{u(x_1, \theta)}, \ldots, \sqrt{u(x_n, \theta)})$, with GLM weights $u(x_i, \theta) = \frac{1}{V(\mu_i)[g'(\mu_i)]^2}$ evaluated at the nominal parameter values. The design weights are initialized from the design_weights field, which is the optimal (approximate) design calculated by $optimize(). Use the m argument to evaluate the covariance matrix for a specific number of design points, in which case the sum of the design weights is scaled to m. To override the default approximate (optimal) design weights, a vector of design weights can be passed using the design_weights argument. The vector must be the same length as the number of design points in data. If the vector of design weights is already an integer design, leave m unspecified, in which case the design is not rescaled.

Usage:

```
glm_design$vcov(m, design_weights = NULL, sigma2 = 1)
```

Arguments:

`m` (optional) integer number of design points at which to evaluate the estimated covariance matrix. If missing, the design weights remain unscaled, which for the default approximate (optimal) design is equivalent to `m = 1`.

`design_weights` (optional) vector of design weights for which to evaluate the estimated covariance matrix.

`sigma2` numeric error variance used to scale the covariance matrix. Defaults to 1, returning the unscaled covariance matrix.

Returns: numeric matrix

`glm_design$sep()`: Calculates the standard error of prediction or the square root prediction variance across the design grid based on an approximate or rounded (exact) design $(w_1, \ldots, w_n)$ according to:

$$\mathrm{SEP}(x_i) \;=\; \sigma\sqrt{x_i^T \Sigma x_i}, \quad \text{for} i = 1, \ldots, n$$

where Σ is the covariance matrix as returned by `$vcov()`. If `type = "response"`, the standard errors of prediction are transformed to the response scale by an application of the Delta method:

$$\mathrm{SEP}_y(x_i) \;=\; \mathrm{SEP}(x_i)\left|\frac{\partial \mu}{\partial \eta}\right|$$

Use the `newdata` argument to evaluate the standard error of prediction at design points not in the candidate design grid defined by `$data`. The design weights are initialized from the `design_weights` field, which is the optimal (approximate) design calculated by `$optimize()`. To override the default approximate (optimal) design weights, a vector of design weights can be passed using the `design_weights` argument. The vector must be the same length as the number of design points in `data`.

Usage:

```
glm_design$sep(
  m,
  newdata,
  design_weights = NULL,
  type = c("link", "response"),
  sigma2 = 1
)
```

Arguments:

`m` (optional) integer number of design points at which to evaluate the estimated covariance matrix. If missing, the design weights remain unscaled, which for the default approximate (optimal) design is equivalent to `m = 1`.

`newdata` (optional) data frame or matrix with design points (rows) at which to return the standard error of prediction and which can be evaluated by `$formula`.

`design_weights` (optional) vector of design weights for which to evaluate the estimated covariance matrix.

`type` the type of prediction required. The default choice `"link"` returns the standard error of prediction on the scale of the linear predictors, whereas `"response"` returns the standard error of prediction on the scale of the response variable.

`sigma2` numeric error variance used to scale the standard errors. Defaults to 1, returning the standardized standard error of prediction.

Returns: numeric vector

`glm_design$corr()`: Calculates the correlation matrix of the columns of the design matrix based on an approximate or rounded (exact) design $(w_1, \ldots, w_n)$ according to:

$$R = D^{-1/2}(X_c^T U^{1/2} \Omega U^{1/2} X_c) D^{-1/2}$$

where Ω is $\mathrm{diag}(\omega_1, \ldots, \omega_n)$ with $\omega_i = w_i v_i$, and X_c is the design matrix X centered by the (weighted) column means:

$$\frac{\sum_{i=1}^n \omega_i X_{ij}}{\sum_{i=1}^n \omega_i}, \quad j = 1, \ldots, p$$

$D^{-1/2}$ is $\mathrm{diag}(1/\sqrt{D_1}, \ldots, 1/\sqrt{D_p})$, with $D_1, \ldots, D_p$ the diagonal elements of $X_c^T U^{1/2} \Omega U^{1/2} X_c$. If `center = FALSE`, the uncentered correlation matrix is returned, which corresponds to the scale-invariant (unit diagonal) version of the information matrix $M(w|X, v, \theta, F, g)$. The design weights are initialized from the `design_weights` field, which is the optimal (approximate) design calculated by `$optimize()`. To override the default approximate (optimal) design weights, a vector of design weights can be passed using the `design_weights` argument. The vector must be the same length as the number of design points in `data`.

Usage:

```
glm_design$corr(design_weights = NULL, center = TRUE)
```

Arguments:

`design_weights` (optional) vector of design weights for which to evaluate the information matrix.

`center` logical, whether to center the design columns when evaluating the correlation matrix, defaults to `TRUE`.

Returns: numeric matrix

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Lewis, S. L., Montgomery D. C., and Myers R. H. 2001. *Examples of Designed Experiments with Nonnormal Responses*. Journal of Quality Technology 3 (33): 265–78.

See Also

`family`

Examples

```
## Non-conforming tiles experiment
## Example 3 from [Lewis et al. (2001)]

## design grid
data <- expand.grid(
  x1 = c(-1, 1),
  x2 = c(-1, 1),
  x3 = c(-1, 1),
  x4 = c(-1, 1),
  x5 = c(-1, 1),
  x6 = c(-1, 1),
  x7 = c(-1, 1)
)
## initialize model, design grid, family and nominal parameters
design <- glm_design$new(
  formula = ~x1 + x2 + x3 + x4 + x5 + x6 + x7,
  data = data,
  family = binomial(link = "logit"),
  theta = c(-1.419, -0.578, 0.109, 0.266, -0.262, 0.434, -0.633, 0.425)
)
## approximate D-optimal design
design$optimize(criterion = "D")
## integer design w/ 8 design points
(exact_design <- design$round(m = 8, method = "optimal", n_repeats = 25, seed = 1))
design$subset(replicates = exact_design)

## Defect semiconductor wafers
## Example 5 from [Lewis et al. (2001)]

## design grid
data <- rbind(
  expand.grid(x1 = -1:1, x2 = -1:1, x3 = -1:1),
  data.frame(
    x1 = c(-1.732, 1.732, 0, 0, 0, 0),
    x2 = c(0, 0, -1.732, 1.732, 0, 0),
    x3 = c(0, 0, 0, 0, -1.732, 1.732)
  )
)
## initialize model, design grid, family and nominal parameters
design <- glm_design$new(
  formula = ~(x1 + x2 + x3)^2 + I(x1^2) + I(x2^2) + I(x3^2),
  data = data,
  family = poisson(link = "log"),
  theta = c(1.9293, 0.6103, 0.7028, -0.3531, 0, 0.2471, 0, -0.5209, 0, 0)
)
## approximate D-optimal design
design$optimize(criterion = "D")
## integer design w/ 18 design points
(exact_design <- design$round(m = 18, method = "optimal", seed = 1))
design$subset(replicates = exact_design)
```

`imidazoles` *Imidazoles dataset*

Description

This dataset reports the yield of the desired product and the reaction cost in a direct arylation reaction of imidazoles. Two continuous process conditions (temperature, initial concentration) and three reaction molecules (base, ligand, solvent) are varied to maximize the yield of the desired product while minimizing the cost of the reaction.

Usage

```
imidazoles
```

Format

A data.frame containing 1728 experiments with columns:

yield numeric, experimental yield of the reaction (%)

cost numeric, cost of the reaction scaled to the unit interval

concentration numeric, initial substrate concentration (M)

temperature numeric, temperature of the reaction (Celsius)

base character, common name of the reaction base molecule

base_smiles character, SMILES string of the reaction base molecule

solvent character, common name of the reaction solvent molecule

solvent_smiles character, SMILES string of the reaction solvent molecule

ligand character, common name of the reaction ligand molecule

ligand_smiles character, SMILES string of the reaction ligand molecule

Value

data.frame

Source

doi:10.1038/s4158602103213y

doi:10.1021/jacs.2c08592

References

Shields B.J. et al. (2021) *"Bayesian reaction optimization as a tool for chemical synthesis"*, Nature, Vol. 590 (7844).

Torres J.A.G. et al. (2022) *"A Multi-Objective Active Learning Platform and Web App for Reaction Optimization"*, J. Am. Chem. Soc., Vol. 144 (43).

Examples

```
head(imidazoles)
```

`lme_design` *Penalized linear mixed model optimal design*

Description

An R6 class for managing penalized optimal experiment designs for random intercept linear mixed models. This class encapsulates a random intercept linear mixed model, the discrete design grid, optimization of an approximate continuous design, augmentation of an existing design, and optimal rounding to exact integer designs including split-plot designs.

Value

R6-object of class `lme_design` if called with `lme_design$new()`.

Optimal design problem

The following penalized optimization problem is solved by `$optimize()`:

$$w^* \;=\; \arg\max_{w} \Phi(M(w|X,Z,\eta,v) + \lambda I) - \alpha \cdot c^T w$$

where,

- $w = \{w_1, \ldots, w_n\}$ is the vector of design weights.
- $c = \{c_1, \ldots, c_n\}$ is the cost assigned to each design weight with scaling/penalty parameter α.
- $v = \{v_1, \ldots, v_n\}$ is a vector of fixed user weights.
- $M(w|X,Z,\eta,v)$ is the information matrix depending on the weights w, the fixed-effects design matrix X, the random-effects design matrix Z, the variance component ratio $\eta = \sigma_u^2/\sigma_e^2$, and in the case of weighted regression the heterogeneous weights v.
- Φ is the chosen optimality criterion, generally a function of the information matrix.

Given a random intercept linear mixed model ($X\beta + Zu + \epsilon$) defined by `formula`, the information matrix of the fixed-effects model parameters, conditional on the random effects covariance matrix and the error variance, is given by:

$$M(w|X,Z,\eta,v) \;=\; X^T \Omega^{1/2} V^{-1} \Omega^{1/2} X$$

where $\Omega^{1/2}$ is the diagonal matrix with $(\sqrt{w_1 v_1}, \ldots, \sqrt{w_n v_n})$ on the diagonal, and V^{-1} is the inverse of the block-diagonal covariance matrix V. Partitioned into random effects levels $1, \ldots, S$, where group s has n_s observations, V consists of the level-specific blocks:

$$V_s = \sigma_e^2 \left(I_{n_s} + \eta \mathbf{1}_{n_s} \mathbf{1}_{n_s}^T \right)$$

Augmented optimal design

Given a vector of initial design weights w_0 and associated contribution weight γ, the following augmented penalized optimization problem is solved by `$augment()`:

$$w^* \;=\; \arg\max_w \Phi(M(w|\gamma, w_0, X, Z, \eta, v) + \lambda I) - \alpha \cdot c^T w$$

using the augmented information matrix,

$$M(w|\gamma, w_0, X, Z, \eta, v) \;= (1-\gamma)M(w_0|X, Z, \eta, v) + \gamma M(w|X, Z, \eta, v)$$

Stratified optimal design

The `$strata()` method stratifies the design by requiring that individual strata $X_1, \ldots, X_S$ are represented proportionally with given weights $p_1, \ldots, p_S$, ($\sum_i p_i = 1$). If stratification is applied, the following optimization problem is solved by `$optimize()`:

$$\begin{array}{rl} w^* \;=\; & \arg\max_w \Phi(M(w|X, Z, \eta, v) + \lambda I) - \alpha \cdot c^T w, \\ & \text{subject to } \sum_{i \in X_s} w_i = p_s, \qquad \text{for } s = 1, \ldots, S \end{array}$$

This means that the total weight assigned to the design points in stratum X_i is equal to p_i for each $i = 1, \ldots, S$. Similarly, if stratification is applied, `$augment()` solves the augmented optimization problem *with* stratification constraints.

Exact split-plot design

The `$round_split_plot()` method rounds an approximate design to an exact (integer) split-plot design. The rounding procedure enforces fixed levels of hard-to-change variables within each individual *whole plot*, where the whole plots are the blocks in the random effects factors (i.e. whole plot variables). Mathematically, for an exact design $m^* = \{m_1, \ldots, m_n\}$, the split-plot rounding procedure solves:

$$\begin{array}{rl} m^* \;=\; & \arg\max_m \Phi(M(m|X, Z, \eta, v) + \lambda I) - \alpha \cdot c^T (m / \sum m_i), \\ & \text{subject to } H(m) = Z(m) H_b \;\text{ and }\; \mathbf{1}_n^T Z(m) = M \mathbf{1}_b^T \end{array}$$

$H(m)$ and $Z(m)$ are the design matrices of the hard-to-change and random effects factors subject to m. H_b is a matrix with b rows selected from the levels of the hard-to-change variables, where b is the number of whole plots. The second constraint ensures that the number of design points at each whole plot level is exactly M, the whole plot size. The split-plot designs are only available in the form of *exact* designs, and not approximate designs (as returned by e.g. `$optimize()`).

Optimality criteria

The optimality criteria $\Phi(\cdot)$ supported by `$optimize()` and `$augment()` are: *D*-, *A*-, *I*-, *G*-, and *alias*-optimality, see lm_design for the definitions.

Super class

`lm_design` -> `lme_design`

Public fields

`formula` a two-part formula of the form `~ fix.eff ~ ran.eff` representing a random intercept linear mixed model, where `fix.eff` is a linear model formula encoding the fixed effects design matrix and `ran.eff` is an expression encoding the levels of the random effects factor.

`data` a data frame or matrix containing all design points (rows) at which `formula` can be evaluated.

`eta` numeric variance component ratio σ_u^2/σ_e^2, where σ_u^2 is the (single) variance component of the random effects and σ_e^2 is the error variance.

`cost` an optional numeric vector containing costs associated to each design point (row) in `data`.

`design_weights` a numeric vector with optimal design weights populated after optimization with `$optimize()` or `$augment()`. The attribute `criterion` lists the used design criterion and the achieved optimal criterion value.

`weights` an optional numeric vector containing fixed user weights associated to each design point (row) in `data`.

`strata` a list with elements `strata` and `proportions` imposing a partition of the design points into individual (disjoint) strata with given weight proportions.

`alpha` the cost scale/penalty parameter in the penalized optimization problem, only used if `$cost` is available.

`lambda` the ridge penalty added to the diagonal of the information matrix in the optimization problem. The ridge penalty is not used for the `alias` optimality criterion.

`upper` the design weight upper limit. The default is 1, corresponding to no upper limit constraint on the design weights.

Active bindings

`X` the fixed effects design matrix obtained from all design points in `data` and the model `formula`.

`Z` the random effects design matrix obtained from all design points in `data` and the model `formula`.

Methods

Public methods:

- `lme_design$new()`
- `lme_design$update()`
- `lme_design$stratify()`
- `lme_design$optimize()`
- `lme_design$augment()`
- `lme_design$round()`
- `lme_design$round_split_plot()`
- `lme_design$crit()`
- `lme_design$vcov()`
- `lme_design$sep()`
- `lme_design$corr()`
- `lme_design$clone()`

`lme_design$new()`: Constructor. Initializes the experimental design to optimize with a linear mixed model, discrete design grid, and fixed variance component ratio.

Usage:

```
lme_design$new(
  formula,
  data,
  eta,
  weights = NULL,
  cost = NULL,
  contrasts = NULL,
  alpha = 1,
  lambda = 0,
  upper = 1,
  ...
)
```

Arguments:

`formula` a two-part formula of the form `~ fix.eff ~ ran.eff`, where `fix.eff` is a linear model formula encoding the fixed effects design matrix and `ran.eff` is an expression encoding the random effects levels. `ran.eff` is evaluated as a factor, with the factor levels determining the random effects groups.

`data` a data frame or matrix containing all design points (rows) at which `formula` can be evaluated.

`eta` numeric variance component ratio σ_u^2/σ_e^2, where σ_u^2 is the (single) variance component of the random effects and σ_e^2 is the error variance.

`weights` an optional numeric vector containing fixed user weights associated to each design point (row) in `data`. Also accepts a one-sided formula using the column names present in `data`. If `NULL`, the unweighted optimal design problem is solved.

`cost` an optional numeric vector containing costs associated to each design point (row) in `data`. Also accepts a one-sided formula made up of column names present in `data`. If `NULL`, the standard (non-penalized) optimal design problem is solved.

`contrasts` optional contrasts used when evaluating the fixed effects formula `fix.eff`.

`alpha` numeric value, cost scale/penalty parameter in the penalized optimization problem, only used if `$cost` is available. The default is 1.

`lambda` numeric value, ridge penalty added to the diagonal of the information matrix in the penalized optimization problem. The default is 0.

`upper` numeric value in (0, 1], design weight upper limit. The default is 1, corresponding to no constraint on the design weights.

`...` any additional arguments passed to model.matrix when evaluating the fixed effects formula `fix.eff`.

Returns: `lme_design` R6-object

`lme_design$update()`: Updates the fields of an existing object. Inputs should be named `formula`, `data`, `eta`, `weights`, `cost`, `alpha`, `lambda` and/or `upper` respectively.

Usage:

```
lme_design$update(...)
```

Arguments:

`...` named arguments `formula`, `data`, `eta`, `weights`, `cost`, `alpha`, `lambda` and/or `upper`.

Returns: none

`lme_design$stratify()`: Stratify the design by requiring that individual strata are represented proportionally. Call with `strata = NULL` to unset previously defined `strata`.

Usage:

```
lme_design$stratify(strata, proportions = NULL)
```

Arguments:

`strata` a factor variable of the same length as the number of rows in `data`. The factor levels determine the partition of the design points into individual (disjoint) strata. Also accepts a one-sided formula made up of column names present in `data`, in which case the strata are defined by the interaction between terms in the model.frame obtained from the formula.

`proportions` an optional numeric vector containing the weight proportions of the individual strata. Must be of the same length as the number of factor levels in `strata`. If `NULL`, each stratum is assigned the same weight proportion.

Returns: none

`lme_design$optimize()`: Optimizes the penalized approximate design based on the selected criterion by iterative reweighting of a convex approximation of the optimization problem. `"alias"`-optimality is included for completeness only, as the optimality criterion does not vary with the variance component ratio η.

Usage:

```
lme_design$optimize(
  criterion = c("D", "A", "I", "G", "alias"),
  max_iter = 1,
  tol = .Machine$double.eps^(1/4),
  scs_control = scs_control_dflt(),
  verbose = FALSE,
  return_value = TRUE,
  show_progress = TRUE
)
```

Arguments:

`criterion` character string specifying the optimality criterion to use, see also 'Details'. The following choices are supported:

`"D"` D-optimality (default).
`"A"` A-optimality.
`"I"` I-optimality.
`"G"` G-optimality.
`"alias"` alias-optimality.

`max_iter` integer number of iterations sequential convex approximation. The default is 1, which returns the approximate design corresponding to a variance component ratio $\eta = 0$.

`tol` numeric tolerance used to scale the minimum improvement threshold in the backtracking line search to determine a new set of approximate design weights.

`scs_control` an optional list of control parameters to tune the SCS optimization. See `scs_control_dflt` for the available control parameters and their default values.

`verbose` display verbose messages printed by the scs solver. Defaults to `FALSE`.

`return_value` return the optimal design weights. If set to `FALSE`, the design weights should be retrieved from the `design_weights` field.

`show_progress` logical, display a progress bar during the rounding procedure (only if `max_iter` > 1). Defaults to `TRUE`.

Returns: numeric vector if `return_value = TRUE`, otherwise none

`lme_design$augment()`**:** Optimizes the augmented penalized approximate design given a vector of initial design weights by iterative reweighting of a convex approximation of the optimization problem. `"alias"`-optimality is included for completeness only, as the optimality criterion does not vary with the variance component ratio eta.

Usage:

```
lme_design$augment(
  design_weights,
  max_iter = 1,
  gamma = 0.5,
  criterion = c("D", "A", "I", "G", "alias"),
  tol = .Machine$double.eps^(1/4),
  scs_control = scs_control_dflt(),
  verbose = FALSE,
  return_value = TRUE,
  show_progress = TRUE
)
```

Arguments:

`design_weights` a numeric vector with initial design weights, must be the same length as the number of design points in `data`. Also accepts a one-sided formula made up of column names present in `data`.

`max_iter` integer number of iterations sequential convex approximation. The default is 1, which returns the approximate design corresponding to a variance component ratio $\eta = 0$.

`gamma` numeric contribution weight of the initial design in the augmented design problem, see also 'Augmented optimal design' above. Defaults to 0.5.

`criterion` character string specifying the optimality criterion to use, see also 'Details'. The following choices are supported:

`"D"` D-optimality (default).
`"A"` A-optimality.
`"I"` I-optimality.
`"G"` G-optimality.
`"alias"` alias-optimality.

`tol` numeric tolerance used to scale the minimum improvement threshold in the backtracking line search to determine a new set of approximate design weights.

`scs_control` an optional list of control parameters to tune the SCS optimization. See `scs_control_dflt` for the available control parameters and their default values.

`verbose` display verbose messages printed by the scs solver. Defaults to `FALSE`.

`return_value` return the optimal design weights. If set to `FALSE`, the design weights should be retrieved from the `design_weights` field.

`show_progress` logical, display a progress bar during the rounding procedure (only if `max_iter` > 1). Defaults to `TRUE`.

Returns: numeric vector if `return_value = TRUE`, otherwise none

`lme_design$round()`**:** Round penalized approximate design to an exact (integer) design. The approximate design is initialized from the `design_weights` field, which is the optimal (approximate) design calculated by `$optimize()`. For method `"optimal"`, cost penalties and upper limit constraints present in `$cost` and `$upper` (if specified) are included in the criterion optimized by the point-exchange procedure. To override the initial approximate (optimal) design, a vector of design weights can be passed using the `design_weights` argument. The vector must be the same length as the number of design points in `data`. If `design_weights` is set to `FALSE`, the initial design is generated by random sampling.

Usage:

```
lme_design$round(
  m,
  method = c("optimal", "efficient"),
  max_iter = 100,
  n_repeats = 10,
  replicates = TRUE,
  tol = .Machine$double.eps^(1/3),
  seed = NULL,
  design_weights = TRUE,
  augment_design = NULL,
  show_progress = TRUE,
  ...
)
```

Arguments:

`m` integer number of design points included in the exact design.

`method` the method used to calculate the exact design. The following choices are supported:

`"optimal"` optimality criterion point-exchange algorithm (default).
`"efficient"` efficient rounding method of Pukelsheim & Reider (1992).

`max_iter` integer number of maximum iterations in point-exchange algorithms. Only used when `method = "optimal"`.

`n_repeats` integer number of (random) repeats of the rounding/exchange algorithm. The exact design with the highest criterion value is returned. Defaults to 10.

`replicates` logical, allow replicate design points in the exact design, defaults to `TRUE`. Can also be an integer number, limiting the maximum number of allowed replicates per design point. This argument overrides the `$upper` value constraint if existing.

`tol` numeric tolerance used as lower threshold to continue iterations in point-exchange algorithms.

`seed` (optional) random seed used to generate jitter to break ties in rounding the initial exact design. Not used in the case of (`mirai`) asynchronous evaluation, which requires the random seed to be set via the `seed` argument in `mirai::daemons`.

`design_weights` (optional) vector of design weights used as initial approximate design, with length equal to the number of available design points. If `TRUE` (default), the approximate design is initialized from the `design_weights` field. If `FALSE`, the initial approximate design is generated by random sampling.

`augment_design` (optional) integer design to augment, with length equal to the number of available design points.

`show_progress` logical, display a progress bar during the rounding procedure (only if `n_repeats > 1`). Defaults to `TRUE`.

`...` additional arguments to tune algorithm/solver settings:

- `criterion`, character string specifying the optimality criterion for the rounding procedure in case `design_weights` has no `"criterion"` attribute, or to override the criterion specified in the `"criterion"` attribute. If specified, must be one of `"D"`, `"A"`, `"I"`, `"G"` or `"alias"`.

Details:

Parallel evaluation: The repeated rounding/exchange algorithms can be run in parallel using the `mirai` asynchronous evaluation framework. The parallel background R processes are configured by the user outside of `$round()` with `mirai::daemons`. If daemon connections are available, `$round()` automatically runs the repeated exchange algorithms in parallel using the configured daemons. If the underlying BLAS/LAPACK libraries are already multi-threaded, this may inhibit the use of additional threads by the `mirai` asynchronous evaluation. In this case, it is recommended to restrict BLAS/LAPACK to a single thread with the `OPENBLAS_NUM_THREADS`, `MKL_NUM_THREADS` or `OMP_NUM_THREADS` environment variables, depending on the BLAS library that is used. In addition, note that there is a small overhead in populating the environments of the background R processes, which may result in increased runtime compared to sequential evaluation in the main R process, especially when `n_repeats` is small or the rounding/exchange algorithm does not take much time to complete.

Returns: numeric vector

`lme_design$round_split_plot()`**:** Round penalized approximate design to an exact (integer) split-plot design. This rounding procedure enforces fixed levels of the defined `hard_to_change` factors within each *whole plot*, where the whole plots are determined by the random effects factors in the model `$formula`. Stratification constraints imposed with `$stratify()` are not taken into account as these are overridden by the split-plot design constraints. The approximate design is initialized from the `design_weights` field, which is the optimal (approximate) design calculated by `$optimize()`. To override the initial approximate (optimal) design, a vector of design weights can be passed using the `design_weights` argument. The vector must be the same length as the number of design points in `data`.

Usage:

```
lme_design$round_split_plot(
  m,
  hard_to_change,
  max_iter = 100,
  n_repeats = 10,
  replicates = TRUE,
```

```
  tol = .Machine$double.eps^(1/3),
  seed = NULL,
  design_weights = NULL,
  augment_design = NULL,
  show_progress = TRUE,
  ...
)
```

Arguments:

`m` integer whole plot size(s). If a single integer, the number of design points per level of the whole plot factor(s) is set equal to `m`, i.e. the total number of design points in the split-plot design is `m` times the number of whole plot levels. If a vector, the length of `m` must be the same as the number of whole plot levels, with the elements of `m` specifying the sizes of the individual whole plots.

`hard_to_change` hard-to-change factor variable of the same length as the number of rows in `data`. Also accepts a one-sided formula made up of column names present in `data`, in which case the factor levels are defined by the interaction between terms in the model.frame obtained from the formula.

`max_iter` integer number of maximum iterations in point-exchange algorithms.

`n_repeats` integer number of (random) repeats of the rounding/exchange algorithm. The exact design with the highest criterion value is returned. Defaults to 10.

`replicates` logical, allow replicate design points in the exact design, defaults to `TRUE`. Can also be an integer number, limiting the maximum number of allowed replicates per design point. This argument overrides the `$upper` value constraint if existing.

`tol` numeric tolerance used a lower threshold to continue iterations in point-exchange algorithms.

`seed` (optional) random seed used to generate jitter to break ties in rounding the initial exact design. Not used in the case of (`mirai`) asynchronous evaluation, which requires the random seed to be set via the `seed` argument in `mirai::daemons`.

`design_weights` (optional) vector of design weights used as initial approximate design.

`augment_design` (optional) integer design to augment, with length equal to the number of available design points.

`show_progress` logical, display a progress bar monitoring the repeats of the rounding/exchange procedure. Defaults to `TRUE`.

`...` additional arguments to tune algorithm settings:

- – `criterion`, character string specifying the optimality criterion for the rounding procedure in case `design_weights` has no `"criterion"` attribute, or to override the criterion specified in the `"criterion"` attribute. If specified, must be one of `"D"`, `"A"`, `"I"`, or `"G"`.

Details:

Parallel evaluation: The repeated rounding/exchange algorithms can be run in parallel using the `mirai` asynchronous evaluation framework. The parallel background R processes are configured by the user outside of `$round()` with `mirai::daemons`. If daemon connections are available, `$round()` automatically runs the repeated exchange algorithms in parallel using the configured daemons. If the underlying BLAS/LAPACK libraries are already multi-threaded, this may inhibit the use of additional threads by the `mirai` asynchronous evaluation. In this case, it is recommended to restrict BLAS/LAPACK to a single thread with

the OPENBLAS_NUM_THREADS, MKL_NUM_THREADS or OMP_NUM_THREADS environment variables, depending on the BLAS library that is used. In addition, note that there is a small overhead in populating the environments of the background R processes, which may result in increased runtime compared to sequential evaluation in the main R process, especially when n_repeats is small or the rounding/exchange algorithm does not take much time to complete.

Returns: numeric vector

`lme_design$crit()`: Evaluate optimality criterion of a penalized approximate or rounded (exact) design. If cost penalties are present in the `$cost` field, the penalized optimality criterion as described in the 'Optimality criteria' section of lm_design are returned. The approximate design is initialized from the `design_weights` field, which is the optimal (approximate) design calculated by `$optimize()`. To override the default approximate (optimal) design, a vector of design weights can be passed using the `design_weights` argument.

Usage:

```
lme_design$crit(
  design_weights = NULL,
  criterion = c("D", "A", "I", "G", "alias"),
  lambda = TRUE,
  alpha = TRUE
)
```

Arguments:

`design_weights` (optional) vector of design weights for which to evaluate optimality criterion.

`criterion` character string specifying the optimality criterion to use, see also 'Details'. The following choices are supported:

`"D"` D-optimality (default).
`"A"` A-optimality.
`"I"` I-optimality.
`"G"` G-optimality.
`"alias"` alias-optimality.

`lambda` logical, whether to include the ridge penalty in the calculated optimality criterion (see 'Optimality criteria'). Only relevant if `$lambda` is positive.

`alpha` logical, whether to include cost penalties in the calculated optimality criterion (see 'Optimality criteria'). Only relevant if cost penalties are present in the `$cost` field.

Returns: numeric criterion value

`lme_design$vcov()`: Calculates the variance-covariance matrix of the model parameters based on an approximate or rounded (exact) design $(w_1, \ldots, w_n)$ according to:

$$\Sigma \;=\; (X^T \Omega^{1/2} V^{-1} \Omega^{1/2} X)^{-1}$$

Here $\Omega^{1/2} = \mathrm{diag}(\sqrt{w_1 v_1}, \ldots, \sqrt{w_n v_n})$ and V^{-1} is the inverse of the block-diagonal covariance matrix $V = \mathrm{diag}[V_1, \ldots, V_S]$, consisting of the level-specific blocks $V_s = \sigma_e^2 \left(I_{n_s} + \eta \mathbf{1}_{n_s} \mathbf{1}_{n_s}^T\right)$, with n_s is the number of observations for random effects level s. The design weights are initialized from the `design_weights` field, which is the optimal (approximate) design calculated by `$optimize()`. Use the `m` argument to evaluate the covariance matrix for a specific number of

design points, in which case the sum of the design weights is scaled to `m`. To override the default approximate (optimal) design weights, a vector of design weights can be passed using the `design_weights` argument. The vector must be the same length as the number of design points in `data`. If the vector of design weights is already an integer design, leave `m` unspecified, in which case the design is not rescaled.

Usage:

```
lme_design$vcov(m, design_weights = NULL, sigma2 = 1)
```

Arguments:

`m` (optional) integer number of design points at which to evaluate the estimated covariance matrix. If missing, the design weights remain unscaled, which for the default approximate (optimal) design is equivalent to `m = 1`.

`design_weights` (optional) vector of design weights for which to evaluate the estimated covariance matrix.

`sigma2` numeric error variance used to scale the covariance matrix. Defaults to 1, returning the unscaled covariance matrix.

Returns: numeric matrix

`lme_design$sep()`: Calculates the standard error of prediction or the square root prediction variance across the design grid based on an approximate or rounded (exact) design $(w_1, \ldots, w_n)$ according to:

$$\mathrm{SEP}(x_i) \;=\; \sigma\sqrt{x_i^T \Sigma x_i}, \quad \text{for} i = 1, \ldots, n$$

where Σ is the covariance matrix as returned by `$vcov()`. Use the `newdata` argument to evaluate the standard error of prediction at design points not in the candidate design grid defined by `$data`. The design weights are initialized from the `design_weights` field, which is the optimal (approximate) design calculated by `$optimize()`. To override the default approximate (optimal) design weights, a vector of design weights can be passed using the `design_weights` argument. The vector must be the same length as the number of design points in `data`.

Usage:

```
lme_design$sep(m, newdata, design_weights = NULL, sigma2 = 1)
```

Arguments:

`m` (optional) integer number of design points at which to evaluate the estimated covariance matrix. If missing, the design weights remain unscaled, which for the default approximate (optimal) design is equivalent to `m = 1`.

`newdata` (optional) data frame or matrix with design points (rows) for which to return the standard error of prediction and which can be evaluated by `$formula`.

`design_weights` (optional) vector of design weights for which to evaluate the estimated covariance matrix.

`sigma2` numeric error variance used to scale the standard errors. Defaults to 1, returning the standardized standard error of prediction.

Returns: numeric vector

`lme_design$corr()`: Calculates the correlation matrix of the design matrix columns based on an approximate or rounded (exact) design $(w_1, \ldots, w_n)$ according to:

$$R \;=\; D^{-1/2}(X_c^T \Omega^{1/2} V^{-1} \Omega^{1/2} X_c) D^{-1/2}$$

where $\Omega = \mathrm{diag}(\sqrt{w_1 v_1}, \ldots, \sqrt{w_n v_n})$ and V^{-1} is the inverse of the block-diagonal covariance matrix $V = \mathrm{diag}[V_1, \ldots, V_S]$. X_c is the design matrix X centered by the (weighted) column means:

$$\frac{\sum_{i=1}^{n} \omega_i X_{ij}}{\sum_{i=1}^{n} \omega_i}, \quad j = 1, \ldots, p$$

$D^{-1/2}$ is $\mathrm{diag}(1/\sqrt{D_1}, \ldots, 1/\sqrt{D_p})$, with $D_1, \ldots, D_p$ the diagonal elements of $X_c^T \Omega^{1/2} V^{-1} \Omega^{1/2} X_c$. If `center = FALSE`, the uncentered correlation matrix is returned, which corresponds to the scale-invariant (unit diagonal) version of the information matrix $M(w|X, v)$. The design weights are initialized from the `design_weights` field, which is the optimal (approximate) design calculated by `$optimize()`. To override the default approximate (optimal) design weights, a vector of design weights can be passed using the `design_weights` argument. The vector must be the same length as the number of design points in `data`.

Usage:

```
lme_design$corr(design_weights = NULL, center = TRUE, unscaled = FALSE)
```

Arguments:

`design_weights` (optional) vector of design weights for which to evaluate the information matrix.

`center` logical, whether to center the design columns when evaluating the correlation matrix, defaults to `TRUE`.

`unscaled` logical value, whether to include V^{-1} in the calculation of R. If `TRUE`, the ordinary correlation matrix (without random effects) $R = D^{-1/2}(X_c^T \Omega X_c) D^{-1/2}$ is returned.

Returns: numeric matrix

`lme_design$clone()`**:** The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Getting started

## unbalanced design grid
data <- data.frame(
  Treatment = c(-1, -1, 0, 0, 1, 1, -1, 0, 0, 1, -1, 1),
  Block = factor(rep(c("B1", "B2", "B3", "B4", "B5", "B6"), each = 2))
)
## fixed effects quadratic model in Treatment ~ random intercepts by Block
design <- lme_design$new(
  formula = ~ Treatment + I(Treatment^2) ~ Block,
  eta = 10,       ## variance component ratio
  data = data
)

## approximate D-optimal design
design$optimize(criterion = "D", max_iter = 10)
```

```
## integer design w/ 18 design points
(exact_design <- design$round(m = 6, method = "optimal", seed = 1))
design$subset(replicates = exact_design)

## Split-plot design

data <- expand.grid(
  A = -1:1,
  B = -1:1,
  C = -1:1,
  week = factor(1:7, levels = 1:7)
)

## two-factor interaction model w/ random week effect
design <- lme_design$new(
  formula = ~ (A + B + C)^2 ~ week,
  eta = 1,
  data = data
)
## initial D-optimal approximate design
design$optimize(criterion = "D", max_iter = 5, return_value = FALSE)

## D-optimal split-plot design w/ 3 points per whole plot
## and hard-to-change factor A
exact_design <- design$round_split_plot(
  m = 3,
  hard_to_change = ~A
)
design$subset(replicates = exact_design)

## Penalized optimal design

design$update(
  cost = ~A,         ## increase cost of high levels
  alpha = 1          ## penalty scaling term
)
## initial D-optimal approximate design
design$optimize(criterion = "D", max_iter = 5, return_value = FALSE)

exact_design <- design$round_split_plot(
  m = 3,
  hard_to_change = ~A
)
design$subset(replicates = exact_design)
```

`lm_design` *Penalized linear optimal design*

Description

An R6 class for managing penalized optimal experiment designs for linear models. This class encapsulates the linear model formula, the discrete design grid, optimization of an approximate continuous design, augmentation of an existing design, and optimal rounding to exact integer designs.

Value

R6-object of class `lm_design` if called with `lm_design$new()`.

Optimal design problem

The following penalized optimization problem is solved by `$optimize()`:

$$w^* \;=\; \arg\max_w \Phi(M(w|X,v) + \lambda I) - \alpha \cdot c^T w$$

where,

- $w = \{w_1, \ldots, w_n\}$ is the vector of design weights.
- $c = \{c_1, \ldots, c_n\}$ is the cost assigned to each design weight with scaling/penalty parameter α.
- $v = \{v_1, \ldots, v_n\}$ is a vector of fixed user weights.
- $M(w|X,v)$ is the information matrix depending on the weights w, the design matrix X created by model.matrix, and in the case of weighted regression the heterogeneous weights v.
- λI is a diagonal matrix to regularize the information matrix with ridge penalty parameter λ.
- Φ is the chosen optimality criterion, generally a function of the information matrix.

Given the linear model `formula`, the information matrix (conditional on the error variance) is:

$$M(w|X) \;\propto\; \sum_{i=1}^{n} w_i v_i x_i x_i^T$$

Augmented optimal design

Given a vector of initial design weights w_0 and associated contribution weight $\gamma \in [0,1]$, the following augmented penalized optimization problem is solved by `$augment()`:

$$w^* \;=\; \arg\max_w \Phi(M(w|\gamma, w_0, X, v) + \lambda I) - \alpha \cdot c^T w$$

using the augmented information matrix,

$$M(w|\gamma, w_0, X, v) \;= (1-\gamma) M(w_0|X,v) + \gamma M(w|X,v)$$

Stratified optimal design

The `$strata()` method stratifies the design by requiring that individual strata $X_1, \ldots, X_S$ are represented proportionally with given weights $p_1, \ldots, p_S$, ($\sum_i p_i = 1$). If stratification is applied, the following optimization problem is solved by `$optimize()`:

$$\begin{aligned} w^* \;=\; & \arg\max_w \Phi(M(w|X,v) + \lambda I) - \alpha \cdot c^T w, \\ & \text{subject to } \textstyle\sum_{i \in X_s} w_i = p_s, \qquad \text{for } s = 1, \ldots, S \end{aligned}$$

using the same notation as in the 'Optimal design problem' section above. This means that the total weight assigned to the design points in stratum X_i is equal to p_i for each $i = 1, \ldots, S$. Similarly, if stratification is applied, `$augment()` solves the augmented optimization problem with stratification constraints.

Sparse optimal design

The `$sparsify()` method solves the penalized optimization problem using a custom penalty function $P(w)$ to promote sparsity in the approximate design w^*. Because the sparse optimization problem is not convex, a sequential convex approximation is used substituting the (concave) sparseness penalty by linearized (convex) costs on the design weights w similar to the cost penalties above. The following options are supported:

`"entropy"`**-penalty:** Shannon entropy penalty, $P(w) = -\sum_i^n w_i \log(w_i)$, with linearized costs (iteration k):

$$c_i^{(k)} = -(1 + \log(w_i^{(k)} + \epsilon))$$

`"log-sum"`**-penalty:** Log-sum penalty, $P(w) = \sum_i^n \log(w_i + \epsilon)$, with linearized costs (iteration k):

$$c_i^{(k)} = \frac{1}{w_i^{(k)} + \epsilon}$$

`"power"`**-penalty:** Power (bridge) penalty, $P(w) = \sum_i^n (w_i + \epsilon)^p$, with linearized costs (iteration k):

$$c_i^{(k)} = p(w_i^{(k)} + \epsilon)^{p-1}, \quad 0 < p < 1$$

Optimality criteria

The following optimality criteria $\Phi(\cdot)$ are supported in `$optimize()`, `$augment()` and `$sparsify()`:

`"D"`**-optimality:** Maximize the volume of the information matrix M, or equivalently minimize the volume of the confidence ellipsoid of the parameter estimates:

$$\Phi(M) = \det(M)^{1/p}$$

where p is the number of columns in the design matrix X. With non-zero ridge/cost penalties, the (penalized) optimality criterion is calculated as:

$$\Phi_c(w, M) = \exp\left(\log(\Phi(M + \lambda I)) - \alpha \cdot c^T w\right)$$

`"A"`**-optimality:** Maximize the inverse trace of the inverse information M^{-1}, or equivalently minimize the average variance of the parameter estimates:

$$\Phi(M) = \left(\mathrm{Tr}(M^{-1})\right)^{-1}$$

With non-zero ridge/cost penalties, the (penalized) optimality criterion is calculated as:

$$\Phi_c(w, M) = \Phi(M + \lambda I) - \alpha \cdot c^T w$$

`"I"`**-optimality:** Maximize the inverse of the average prediction variance across the design space, (equivalently minimize the average prediction variance):

$$\Phi(M) = \left(\sum_{i=1}^{n} x_i^T M^{-1} x_i \right)^{-1}$$

With non-zero ridge/cost penalties, the (penalized) optimality criterion is calculated as:

$$\Phi_c(w, M) = \Phi(M + \lambda I) - \alpha \cdot c^T w$$

`"G"`**-optimality:** Maximize the inverse of the maximum prediction variance across the design space, (equivalently minimize the maximum prediction variance):

$$\Phi(M) = \left(\max_{x_i} x_i^T M^{-1} x_i \right)^{-1}$$

With non-zero ridge/cost penalties, the (penalized) optimality criterion is calculated as:

$$\Phi_c(w, M) = \Phi(M + \lambda I) - \alpha \cdot c^T w$$

`"alias"`**-optimality:** Maximize one minus the average absolute cosine similarity (also uncentered correlation or normalized dot product) between columns of the design matrix X:

$$\Phi(w, v, X) = 1 - \frac{2}{p(p-1)} \sum_{i<j} \left| \frac{\langle \omega x_i, x_j \rangle}{\|\sqrt{\omega} x_i\| \|\sqrt{\omega} x_j\|} \right|$$

where ωx_i is the elementwise-product of $\omega = (w_1 v_1, \ldots, w_n v_n)$ and the i-th design column $x_i = (x_{1,i}, \ldots, x_{n,i})$. If all columns in the design matrix X are centered, this is equivalent to minimizing the average correlation between columns in X. With non-zero cost penalties, the (penalized) optimality criterion is calculated as:

$$\Phi_c(w, v, X) = \Phi(w, v, X) - \alpha \cdot c^T w$$

Public fields

`formula` a linear model defined as a one-sided formula.

`data` a data frame or matrix containing all design points (rows) at which `formula` can be evaluated.

`cost` an optional numeric vector containing costs associated to each design point (row) in `data`.

`design_weights` a numeric vector with optimal design weights populated after optimization with `$optimize()`, `$augment()`, or `$sparsify()`. The attribute `criterion` lists the used design criterion and the achieved optimal criterion value.

`weights` an optional numeric vector containing fixed user weights associated to each design point (row) in `data`.

`strata` a list with elements `strata` and `proportions` imposing a partition of the design points into individual (disjoint) strata with given weight proportions.

`alpha` the cost scale/penalty parameter in the penalized optimization problem, only used if `$cost` is available.

`lambda` the ridge penalty added to the diagonal of the information matrix in the optimization problem. The ridge penalty is not used for the `alias` optimality criterion.

`upper` the design weight upper limit. The default is 1, corresponding to no upper limit constraint on the design weights.

Active bindings

`X` the design matrix obtained from all design points in `data` and the model `formula`.

`orthogonal` the dot product (orthogonality) constraint matrix. This matrix contains all dot product (upper limit) constraints between design columns parsed from the `orthogonal` formula(s) in `$new(orthogonal = ...)` or `$update(orthogonal = ...)`. For design columns with no dot product constraint, the matrix entry is `Inf`.

`zeros` the proportion of zeros vector. This vector contains the required proportion of zeros for each column in the design matrix parsed from the `zeros` formula(s) in `$new(zeros = ...)` or `$update(zeros = ...)`. For design columns with no constraint, the value is `NA`.

`col_means` the imposed column mean vector. This vector contains the imposed column mean constraints for each column in the design matrix parsed from the `col_means` formula in `$new(col_means = ...)` or `$update(col_means = ...)`. For design columns with no mean constraint, the value is `NA`.

Methods

Public methods:

- `lm_design$new()`
- `lm_design$update()`
- `lm_design$stratify()`
- `lm_design$augment()`
- `lm_design$optimize()`
- `lm_design$sparsify()`
- `lm_design$round()`
- `lm_design$crit()`
- `lm_design$vcov()`
- `lm_design$sep()`
- `lm_design$corr()`
- `lm_design$subset()`
- `lm_design$clone()`

`lm_design$new()`**:** Constructor. Initializes the experimental design to optimize with a linear model formula and discrete design grid.

Usage:

```
lm_design$new(
  formula,
  data,
  orthogonal = NULL,
  weights = NULL,
  cost = NULL,
  zeros = NULL,
  col_means = NULL,
  contrasts = NULL,
  alpha = 1,
  lambda = 0,
```

```
  upper = 1,
  ...
)
```

Arguments:

`formula` a linear model defined as a one-sided formula.

`data` a data frame or matrix containing all design points (rows) at which `formula` can be evaluated.

`orthogonal` an optional two- or three-part formula or list of formulas imposing dot product (orthogonality) constraints between the columns of the design matrix. In the three-part formula `x1 + x2 ~ y1 + y2 ~ 0`, dot products are constrained between all combinations of columns in the first part of the formula (`x1`, `x2`) and the second part of the formula (`y1`, `y2`). The third part (`0`) specifies the upper limit on the dot product(s). If the formula only has two parts, e.g. `x1 + x2 ~ y1 + y2`, zero dot product constraints are imposed. Also accepts a list of formulas, in which case constraints are added in the order in which they appear in the list.

`weights` an optional numeric vector containing fixed user weights associated to each design point (row) in `data`. Also accepts a one-sided formula using the column names present in `data`. If `NULL`, the unweighted optimal design problem is solved.

`cost` an optional numeric vector containing costs associated to each design point (row) in `data`. Also accepts a one-sided formula using the column names present in `data`. If `NULL`, the standard (non-penalized) optimal design problem is solved.

`zeros` an optional two-part formula or list of formulas imposing constraints on the proportion of zero values in the columns of the design matrix. In the two-part formula `x1 + x2 ~ 1/3`, the proportion of zeros in the columns specified by the first part of the formula (`x1`, `x2`) is constrained to be equal to the (nonnegative) numeric value in the second part of the formula (1/3). Note that imposing constraints on the proportion of zeros in the optimized design is only relevant if the design grid actually contain zero entries. For a list of formulas, zero constraints are added in the order in which they appear in the list, analogous to the `orthogonal` argument.

`col_means` an optional two-part formula imposing constraints on the column means of the design matrix. In the two-part formula `x1 + x2 ~ 0`, the means of the columns specified by the first part of the formula (`x1`, `x2`) are constrained to be equal to the numeric value in the second part of the formula (0).

`contrasts` optional contrasts used when evaluating the linear model formula.

`alpha` numeric value, cost scale parameter in the penalized optimization problem, only used if `$cost` is available. The default is 1.

`lambda` numeric value, ridge penalty added to the diagonal of the information matrix in the penalized optimization problem. The default is 0.

`upper` numeric value in (0, 1], design weight upper limit. The default is 1, corresponding to no constraint on the design weights.

`...` any additional arguments passed to model.matrix.

Returns: `lm_design` `R6`-object

`lm_design$update()`: Updates the fields of an existing object. Inputs should be named `formula`, `data`, `weights`, `orthogonal`, `cost`, `zeros`, `col_means`, `alpha`, `lambda` and/or `upper`.

Usage:

`lm_design$update(...)`

Arguments:

`...` named arguments `formula`, `data`, `weigts`, `orthogonal`, `cost`, `zeros`, `col_means`, `alpha`, `lambda` and/or `upper`.

Returns: none

`lm_design$stratify()`: Stratify the design by requiring that individual strata are represented proportionally. Call with `strata = NULL` to unset previously defined `strata`.

Usage:

```
lm_design$stratify(strata, proportions = NULL)
```

Arguments:

`strata` a factor variable of the same length as the number of rows in `data`. The factor levels determine the partition of the design points into individual (disjoint) strata. Also accepts a one-sided formula using column names present in `data`, in which case the strata are defined by the interaction between terms in the model.frame obtained from the formula.

`proportions` an optional numeric vector containing the weight proportions of the individual strata. Must be of the same length as the number of factor levels in `strata`. If `NULL`, each stratum is assigned the same weight proportion.

Returns: none

`lm_design$augment()`: Optimizes the augmented penalized approximate design given a vector of initial design weights

Usage:

```
lm_design$augment(
  design_weights,
  gamma = 0.5,
  criterion = c("D", "A", "I", "G", "alias"),
  scs_control = scs_control_dflt(),
  verbose = FALSE,
  return_value = TRUE
)
```

Arguments:

`design_weights` a numeric vector with initial design weights, must be the same length as the number of design points in `data`. Also accepts a one-sided formula made up of column names present in `data`.

`gamma` numeric contribution weight of the initial design in the augmented design problem, see also 'Augmented optimal design' above. Defaults to 0.5.

`criterion` character string specifying the optimality criterion to use, see also 'Details'. The following choices are supported:

`"D"` D-optimality (default).
`"A"` A-optimality.
`"I"` I-optimality.
`"G"` G-optimality.

"alias" alias-optimality.

scs_control an optional list of control parameters to tune the SCS optimization. See scs_control_dflt for the available control parameters and their default values.

verbose display verbose messages printed by the scs solver. Defaults to FALSE.

return_value return the optimal design weights. If set to FALSE, the design weights can be obtained from the design_weights field.

Returns: numeric vector if return_value = TRUE, otherwise none

lm_design$optimize(): Optimizes the penalized approximate design based on the selected criterion.

Usage:

```
lm_design$optimize(
  criterion = c("D", "A", "I", "G", "alias"),
  scs_control = scs_control_dflt(),
  verbose = FALSE,
  return_value = TRUE
)
```

Arguments:

criterion character string specifying the optimality criterion to use, see also 'Details'. The following choices are supported:

"D" D-optimality (default).
"A" A-optimality.
"I" I-optimality.
"G" G-optimality.
"alias" alias-optimality.

scs_control an optional list of control parameters to tune the SCS optimization. See scs_control_dflt for the available control parameters and their default values.

verbose display verbose messages printed by the scs solver. Defaults to FALSE.

return_value return the optimal design weights. If set to FALSE, the design weights can be obtained from the design_weights field.

Returns: numeric vector if return_value = TRUE, otherwise none

lm_design$sparsify(): Sparsify the penalized approximate design by iterative reweighting of the optimization problem.

Usage:

```
lm_design$sparsify(
  max_iter,
  criterion = c("D", "A", "I", "G", "alias"),
  reweight_fun = c("entropy", "log-sum", "power"),
  eps = .Machine$double.eps,
  tol = .Machine$double.eps^(1/4),
  p = 0.5,
  design_weights = NULL,
```

```
  recenter = FALSE,
  scs_control = scs_control_dflt(),
  return_value = TRUE,
  show_progress = TRUE
)
```

Arguments:

`max_iter` positive integer, maximum number of reweighted optimizations/iterations to run.

`criterion` character string specifying the optimality criterion to use, see also 'Details'. The following choices are supported:

"D" D-optimality (default).
"A" A-optimality.
"I" I-optimality.
"G" G-optimality.
"alias" alias-optimality.

`reweight_fun` character reweighting function, one of "entropy", "log-sum", "power", see 'Details' for the definitions.

`eps` positive numeric value to ensure finite weights in reweighting functions, see 'Details'.

`tol` numeric tolerance used as a threshold for mean absolute relative difference in design weights between iterations to stop early.

`p` positive numeric power with $0 < p < 1$. Only used if `reweight_fun = "power"`.

`design_weights` (optional) vector of design weights used as initial approximate design.

`recenter` logical, recenter design columns based on the current design weights after each iteration. Defaults to `FALSE`.

`scs_control` an optional list of control parameters to tune the SCS optimization. See `scs_control_dflt` for the available control parameters and their default values.

`return_value` return the optimal design weights. If set to `FALSE`, the design weights can be obtained from the `design_weights` field.

`show_progress` logical, display a progress bar during the iterative reweighting procedure. Defaults to `TRUE`.

Details: The sparseness cost penalties are scaled by the same `alpha` tuning parameter as for a user-specific cost vector, see `scs_control_dflt`. If a cost vector `c_1,...,c_n` associated to the design weights is already specified. The sparseness penalty costs are multiplied by the existing cost vector.

Returns: numeric vector if `return_value = TRUE`, otherwise none

`lm_design$round()`: Round penalized approximate design to an exact (integer) design. By default, the approximate design is initialized from the `design_weights` field, which is the optimal (approximate) design calculated by `$optimize()`. For method "optimal", ridge and cost penalties and upper limit constraints present in `$lambda`, `$cost` and `$upper` (if specified) are included in the criterion optimized by the point-exchange algorithm. To override the initial approximate (optimal) design, a vector of design weights can be passed using the `design_weights` argument. The vector must be the same length as the number of design points in `data`. If `design_weights` is set to `FALSE`, the initial design is generated by random sampling.

Usage:

```
lm_design$round(
  m,
  method = c("optimal", "efficient", "dist-sum", "dist-min", "orthogonal"),
  max_iter = 100,
  n_repeats = 10,
  replicates = TRUE,
  tol = .Machine$double.eps^(1/3),
  seed = NULL,
  design_weights = TRUE,
  augment_design = NULL,
  show_progress = TRUE,
  ...
)
```

Arguments:

`m` integer number of design points included in the exact design.

`method` the method used to calculate the exact design. The following choices are supported:

`"optimal"` optimality criterion point-exchange algorithm (default).
`"efficient"` efficient rounding method of Pukelsheim & Reider (1992).
`"dist-sum"` maximize total pairwise (Euclidean) distance between design points.
`"dist-min"` maximize minimum pairwise (Euclidean) distance between design points.
`"orthogonal"` minimizes aliasing subject to orthogonality constraints, see 'Details'.

`max_iter` integer number of maximum iterations in point-exchange algorithm. Only used when `method = "optimal"`, `method = "dist-sum"` or `method = "dist-min"`.

`n_repeats` integer number of (random) repeats of the rounding/exchange algorithm. The exact design with the highest criterion value is returned. Not used for `method = "orthogonal"`. Defaults to 10.

`replicates` logical, allow replicate design points in the exact design, defaults to `TRUE`. Can also be an integer number, limiting the maximum number of allowed replicates per design point. This argument overrides the `$upper` value constraint if existing.

`tol` numeric tolerance used as lower threshold to continue iterations in point-exchange algorithm. Also used as threshold to ensure positive diagonal entries of $M(w|X)$ when `method = "orthogonal"`.

`seed` (optional) random seed used to generate jitter to break ties in rounding the initial exact design. Not used in the case of (`mirai`) asynchronous evaluation, which requires the random seed to be set via the `seed` argument in `mirai::daemons`.

`design_weights` (optional) vector of design weights used as initial approximate design, with length equal to the number of available design points. If `TRUE` (default), the approximate design is initialized from the `design_weights` field. If `FALSE`, the initial approximate design is generated by random sampling.

`augment_design` (optional) integer design to augment, with length equal to the number of available design points.

`show_progress` logical, display a progress bar monitoring the repeats of the rounding/exchange algorithm. Defaults to `TRUE`.

`...` additional arguments to tune algorithm/solver settings:

- – criterion, character string specifying the optimality criterion for the rounding procedure in case design_weights has no "criterion" attribute, or to override the criterion specified in the "criterion" attribute. If specified, must be one of "D", "A", "I", "G" or "alias".
- – highs_control, a named list with control arguments supported by the HiGHS solver, only used if method = "orthogonal". Defaults to highs_control = list(time_limit = 180, log_to_console = TRUE).
- – highs_write_mps, .mps file path. If provided, the HiGHS MILP is exported in MPS (Mathematical Programming System) format to the provided file path. Can also be TRUE, in which case a new .mps file is created in the current working directory. Only used if method = "orthogonal".

Details:

Parallel evaluation: The repeated rounding/exchange algorithms can be run in parallel using the mirai asynchronous evaluation framework. The parallel background R processes are configured by the user outside of $round() with mirai::daemons. If daemon connections are available, $round() automatically runs the repeated exchange algorithms in parallel using the configured daemons. If the underlying BLAS/LAPACK libraries are already multithreaded, this may inhibit the use of additional threads by the mirai asynchronous evaluation. In this case, it is recommended to restrict BLAS/LAPACK to a single thread with the OPENBLAS_NUM_THREADS, MKL_NUM_THREADS or OMP_NUM_THREADS environment variables, depending on the BLAS library that is used. In addition, note that there is a small overhead in populating the environments of the background R processes, which may result in increased runtime compared to sequential evaluation in the main R process, especially when n_repeats is small or the rounding/exchange algorithm does not take much time to complete.

HiGHS solver: The "orthogonal" rounding procedure uses HiGHS to find an exact design subject to orthogonality constraints. This requires either a local installation of the HiGHS solver or the highs package to be installed. To point to a local HiGHS installation, the environment variable HIGHS_HOME must be set to the highs root directory when installing cvxoptdes. If installation fails, verify that highs_c_api.h exists, typically present in $HIGHS_HOME/include/highs/interfaces, which is required to compile the source code.

Returns: numeric vector

lm_design$crit(): Evaluate optimality criterion of a penalized approximate or rounded (exact) design. The approximate design is initialized from the design_weights field, which is the optimal (approximate) design calculated by $optimize(). To override the default approximate (optimal) design, a vector of design weights can be passed using the design_weights argument.

Usage:

```
lm_design$crit(
  design_weights = NULL,
  criterion = c("D", "A", "I", "G", "alias"),
  lambda = TRUE,
  alpha = TRUE
)
```

Arguments:

design_weights (optional) vector of design weights for which to evaluate optimality criterion.

`criterion` character string specifying the optimality criterion to use, see also 'Details'. The following choices are supported:

"D" D-optimality (default).
"A" A-optimality.
"I" I-optimality.
"G" G-optimality.
"alias" alias-optimality.

`lambda` logical, whether to include the ridge penalty in the calculated optimality criterion (see 'Optimality criteria'). Only relevant if `$lambda` is positive.

`alpha` logical, whether to include cost penalties in the calculated optimality criterion (see 'Optimality criteria'). Only relevant if cost penalties are present in the `$cost` field.

Returns: numeric criterion value

`lm_design$vcov()`: Calculates the variance-covariance matrix of the model coefficients based on an approximate or rounded (exact) design $(w_1, \ldots, w_n)$ according to:

$$\Sigma \;=\; (X^T \Omega X)^{-1}$$

where Ω is $\mathrm{diag}(\omega_1, \ldots, \omega_n)$ with $\omega_i \;=\; w_i v_i$. The design weights are initialized from the `design_weights` field, which is the optimal (approximate) design calculated by `$optimize()`. Use the `m` argument to evaluate the covariance matrix for a specific number of design points, in which case the sum of the design weights is scaled to `m`. To override the default approximate (optimal) design weights, a vector of design weights can be passed using the `design_weights` argument. The vector must be the same length as the number of design points in `data`. If the vector of design weights is already an integer design, leave `m` unspecified.

Usage:

```
lm_design$vcov(m, design_weights = NULL, sigma2 = 1)
```

Arguments:

`m` (optional) integer number of design points at which to evaluate the estimated covariance matrix. If missing, the design weights remain unscaled, which for the default approximate (optimal) design is equivalent to `m = 1`.

`design_weights` (optional) vector of design weights for which to evaluate the estimated covariance matrix.

`sigma2` numeric error variance used to scale the covariance matrix. Defaults to 1, returning the unscaled covariance matrix.

Returns: numeric matrix

`lm_design$sep()`: Calculates the standard error of prediction or the square root prediction variance across the design grid based on an approximate or rounded (exact) design $(w_1, \ldots, w_n)$ according to:

$$\mathrm{SEP}(x_i) \;=\; \sigma \sqrt{x_i^T \Sigma x_i}, \quad \text{for} i = 1, \ldots, n$$

where Σ is the covariance matrix as returned by `$vcov()`. Use the `newdata` argument to evaluate the standard error of prediction at design points not in the candidate design grid defined by `$data`. The design weights are initialized from the `design_weights` field, which is the optimal

(approximate) design calculated by `$optimize()`. To override the default approximate (optimal) design weights, a vector of design weights can be passed using the `design_weights` argument. The vector must be the same length as the number of design points in `data`.

Usage:

```
lm_design$sep(m, newdata, design_weights = NULL, sigma2 = 1)
```

Arguments:

`m` (optional) integer number of design points at which to evaluate the estimated covariance matrix. If missing, the design weights remain unscaled, which for the default approximate (optimal) design is equivalent to `m = 1`.

`newdata` (optional) data frame or matrix with design points (rows) at which to return the standard error of prediction and which can be evaluated by `$formula`.

`design_weights` (optional) vector of design weights for which to evaluate the estimated covariance matrix.

`sigma2` numeric error variance used to scale the standard errors. Defaults to 1, returning the standardized standard error of prediction.

Returns: numeric vector

`lm_design$corr()`**:** Calculates the correlation matrix of the columns of the design matrix based on an approximate or rounded (exact) design $(w_1, \ldots, w_n)$ according to:

$$R \;=\; D^{-1/2}(X_c^T \Omega X_c) D^{-1/2}$$

where Ω is $\mathrm{diag}(\omega_1, \ldots, \omega_n)$ with $\omega_i = w_i v_i$, and X_c is the design matrix X centered by the (weighted) column means:

$$\frac{\sum_{i=1}^n \omega_i X_{ij}}{\sum_{i=1}^n \omega_i}, \quad j = 1, \ldots, p$$

$D^{-1/2}$ is $\mathrm{diag}(1/\sqrt{D_1}, \ldots, 1/\sqrt{D_p})$, with $D_1, \ldots, D_p$ the diagonal elements of $X_c^T \Omega X_c$. If `center = FALSE`, the uncentered correlation matrix is returned, which corresponds to the scale-invariant (unit diagonal) version of the information matrix $M(w|X, v)$. The design weights are initialized from the `design_weights` field, which is the optimal (approximate) design calculated by `$optimize()`. To override the default approximate (optimal) design weights, a vector of design weights can be passed using the `design_weights` argument. The vector must be the same length as the number of design points in `data`.

Usage:

```
lm_design$corr(design_weights = NULL, center = TRUE)
```

Arguments:

`design_weights` (optional) vector of design weights for which to evaluate the information matrix.

`center` logical, whether to center the design columns when evaluating the correlation matrix, defaults to `TRUE`.

Returns: numeric matrix

`lm_design$subset()`**:** Subsets the design points associated to an exact design vector with integer replicates as returned by `$round()`. The vector must be the same length as the number of design points in `data`.

Usage:

```
lm_design$subset(replicates)
```

Arguments:

`replicates` integer vector of design point replicates of the same length as the candidate grid in `data`

Returns: data.frame

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Pukelsheim, F., and Rieder S. (1992) *"Efficient rounding of approximate designs"*, Biometrika.

See Also

https://www.cvxgrp.org/scs/

https://ergo-code.github.io/HiGHS/dev/installation/

https://mirai.r-lib.org/

Examples

```
## Getting started

## 3 factors w/ 5, 5 and 3 levels
data <- expand.grid(
    x1 = -2:2,
    x2 = -2:2,
    x3 = -1:1
)
design <- lm_design$new(
    formula = ~(x1 + x2 + x3)^2 + I(x1^2) + I(x2^2) + I(x3^2),   ## full quadratic model
    data = data    ## design points
)
## approximate D-optimal design
design$optimize(criterion = "D")
## integer D-optimal design w/ 13 design points
(exact_design <- design$round(m = 13, method = "optimal", seed = 1))
design$subset(replicates = exact_design)
## integer design covariance and correlation matrices
design$vcov(design_weights = exact_design, sigma2 = 1)
design$corr(design_weights = exact_design)

## Penalized optimal design
```

```
## assign cost c = 1 if x3 = 1, and c = 0 otherwise
design$update(
  cost = ~pmax(x3, 0),
  alpha = 1
)
## penalized approximate D-optimal design
design$optimize(criterion = "D")

## assign v = 1 if x3 = 1, and v = 0.1 otherwise
design$update(
 weights = ~pmax(x3, 0.1),
 cost = NULL
)
## penalized approximate D-optimal design
design$optimize(criterion = "D")
## exact design w/ 15 design points
design$round(m = 15, method = "optimal", seed = 1)

## drop weights term
design$update(weights = NULL)

## Stratified optimal design

## add stratification constraints
design$stratify(
  strata = ~x3,
  proportions = c(1/4, 1/2, 1/4)
)
## approximate D-optimal design w/ stratification
design$optimize(criterion = "D")
## verify stratification
tapply(design$design_weights, design$data$x3, sum)
## exact design w/ stratification
design$round(m = 12, method = "optimal", seed = 1)
## drop stratification
design$stratify(strata = NULL)

## Orthogonality constraints

## orthogonal main effects wrt all other columns
design$update(
  orthogonal = x1 + x2 + x3 ~ . ~ 0,
  alpha = 0.075  ## sparseness penalty
)
design$orthogonal
## approximate D-optimal design w/ orthogonality constraints
design$optimize(criterion = "D")
## sparsify support by iterative entropy reweighting
design$sparsify(max_iter = 5, criterion = "D", reweight_fun = "entropy")

if(requireNamespace("highs", quietly = TRUE)) {
 ## integer design w/ orthogonality constraints
 (ortho_design <- design$round(m = 8, method = "orthogonal", seed = 1))
```

```
 ## verify correlations
 design$corr(ortho_design)
}

## Optimal design augmentation

## Example from [Goos & Jones (2011), Chapter 3]
## 6 factors w/ 2 levels each
data <- expand.grid(
  x1 = c(-1, 1),
  x2 = c(-1, 1),
  x3 = c(-1, 1),
  x4 = c(-1, 1),
  x5 = c(-1, 1),
  x6 = c(-1, 1)
)

## linear model w/ main effects and selected interactions
design <- lm_design$new(
  formula = ~1 + x1 + x2 + x3 + x4 + x5 + x6 + x2:x3 + x1:x6 + x2:x6 + x3:x6 + x4:x6 + x1:x4,
  data = data
)
## initial design of size n=12
init_design <- c(0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
 0, 1, 0, 0, 0, 0, 1)
## approximate optimal augmented design
design$augment(design_weights = init_design, gamma = 0.4, criterion = "D")
## exact augmented design
(augment_design <- design$round(m = 8, method = "optimal", seed = 1, augment_design = init_design))
## proposed design points
design$subset(replicates = augment_design - init_design)

## augmented design w/ orthogonality constraints
design_ortho <- design$clone()
design_ortho$update(
  orthogonal = list(
   x1 + x2 + x3 + x4 + x5 + x6 ~ x1 + x2 + x3 + x4 + x5 + x6 ~ 0  ## uncorrelated main effects
  )
)
## approximate augmented design
design_ortho$augment(
  design_weights = init_design, gamma = 0.4, criterion = "D", return_value = FALSE
)

if(requireNamespace("highs", quietly = TRUE)) {
  ## integer design w/ orthogonality constraints
  ortho_augment_design <- design_ortho$round(
    m = 8, method = "orthogonal", augment_design = init_design
  )
  ## verify correlations
  design_ortho$corr(ortho_augment_design)
}
```

`nlm_design` *Penalized nonlinear locally optimal design*

Description

An R6 class for managing penalized locally optimal experiment designs for nonlinear models. This class encapsulates the nonlinear model at a set of nominal parameter values, the discrete design grid, optimization of an approximate continuous design, augmentation of an existing design, and optimal rounding to exact integer designs.

Value

R6-object of class `nlm_design` if called with `nlm_design$new()`.

Optimal design problem

The following penalized optimization problem is solved by `$optimize()`:

$$w^* \;=\; \arg\max_w \Phi(M(w|X,v,\theta) + \lambda I) - \alpha \cdot c^T w$$

where,

- $w = \{w_1, \ldots, w_n\}$ is the vector of design weights.
- $c = \{c_1, \ldots, c_n\}$ is the cost assigned to each design weight with scaling/penalty parameter α.
- $v = \{v_1, \ldots, v_n\}$ is a vector of fixed user weights.
- $M(w|X,v,\theta)$ is the information matrix depending on the weights w and v, the design matrix X and the nominal parameter values θ.
- λI is a diagonal matrix to regularize the information matrix with ridge penalty parameter λ.
- Φ is the chosen optimality criterion, generally a function of the information matrix.

Given the nonlinear model `f` and nominal parameter values `theta`, the information matrix (conditional on the error variance) is given by:

$$M(w|X,v,\theta) \;\propto\; \sum_{i=1}^{n} w_i v_i F(x_i,\theta) F(x_i,\theta)^T$$

where $F(x_i,\theta)$ is the gradient vector at $f(x_i,\theta)$, i.e. the i-th row of the Jacobian matrix.

Augmented optimal design

Given a vector of initial design weights w_0 and associated contribution weight γ, the following augmented penalized optimization problem is solved by `$augment()`:

$$w^* \;=\; \arg\max_w \Phi(M(w|\gamma,w_0,X,v,\theta) + \lambda I) - \alpha \cdot c^T w$$

using the augmented information matrix,

$$M(w|\gamma,w_0,X,v,\theta) \;= (1-\gamma) M(w_0|X,v,\theta) + \gamma M(w|X,v,\theta)$$

Stratified optimal design

The `$strata()` method stratifies the design by requiring that individual strata $X_1, \ldots, X_S$ are represented proportionally with given weights $p_1, \ldots, p_S$, ($\sum_i p_i = 1$). If stratification is applied, the following optimization problem is solved by `$optimize()`:

$$\begin{aligned} w^* \;=\; & \arg\max_w \Phi(M(w|X, v, \theta) + \lambda I) - \alpha \cdot c^T w, \\ & \text{subject to } \textstyle\sum_{i \in X_s} w_i = p_s, \qquad \text{for } s = 1, \ldots, S \end{aligned}$$

using the same notation as in the 'Optimal design problem' section above. This means that the total weight assigned to the design points in stratum X_i is equal to p_i for each $i = 1, \ldots, S$. Similarly, if stratification is applied, `$augment()` solves the augmented optimization problem *with* stratification constraints.

Sparse optimal design

The `$sparsify()` method solves the penalized optimization problem with an extra sparseness penalty $P(w)$ in the objective to promote sparsity in the approximate design w^*. See lm_design for the available penalty functions.

Optimality criteria

The optimality criteria $\Phi(\cdot)$ supported by `$optimize()`, `$augment()` and `$sparsify()` are: *D*-, *A*-, *I*-, *G*-, and *alias*-optimality, see lm_design for the definitions.

Super class

`lm_design` -> `nlm_design`

Public fields

`formula` a nonlinear model defined as a one-sided formula.

`theta` a named vector or list of nominal parameter values at which the information matrix $M(w, \theta)$ is evaluated.

`data` a data frame or matrix containing all design points (rows) at which `formula` can be evaluated.

`cost` an optional numeric vector containing costs associated to each design point (row) in `data`.

`weights` an optional numeric vector containing fixed user weights associated to each design point (row) in `data`.

`design_weights` a numeric vector with optimal design weights populated after optimization with `$optimize()`, `$augment()`, or `$sparsify()`. The attribute `criterion` lists the used design criterion and the achieved optimal criterion value.

`strata` a list with elements `strata` and `proportions` imposing a partition of the design points into individual (disjoint) strata with given weight proportions.

`alpha` the cost scale/penalty parameter in the penalized optimization problem, only used if `$cost` is available.

`lambda` the ridge penalty added to the diagonal of the information matrix in the optimization problem. The ridge penalty is not used for the `alias` optimality criterion.

`upper` the design weight upper limit. The default is 1, corresponding to no upper limit constraint on the design weights.

Active bindings

`X` the Jacobian matrix obtained from all design points in `data` and the model `formula`.

Methods

Public methods:

- `nlm_design$new()`
- `nlm_design$update()`
- `nlm_design$stratify()`
- `nlm_design$augment()`
- `nlm_design$optimize()`
- `nlm_design$sparsify()`
- `nlm_design$round()`
- `nlm_design$crit()`
- `nlm_design$vcov()`
- `nlm_design$sep()`
- `nlm_design$corr()`
- `nlm_design$clone()`

`nlm_design$new()`: Constructor. Initializes the experimental design to optimize with a nonlinear model formula and discrete design grid.

Usage:

```
nlm_design$new(
  formula,
  data,
  theta,
  weights = NULL,
  cost = NULL,
  alpha = 1,
  lambda = 0,
  upper = 1
)
```

Arguments:

`formula` a nonlinear model defined as a one-sided formula.

`data` a data frame or matrix containing all design points (rows) at which `formula` can be evaluated.

`theta` a named vector or list of nominal parameter values at which the information matrix $M(w, \theta)$ is evaluated.

`weights` an optional numeric vector containing fixed user weights associated to each design point (row) in `data`. Also accepts a one-sided formula made up of column names present in `data`. If `NULL`, the unweighted optimal design problem is solved.

`cost` an optional numeric vector containing costs associated to each design point (row) in `data`. Also accepts a one-sided formula made up of column names present in `data`. If `NULL`, the standard (non-penalized) optimal design problem is solved.

`alpha` numeric value, cost scale/penalty parameter in the penalized optimization problem, only used if `$cost` is available. The default is 1.

`lambda` numeric value, ridge penalty added to the diagonal of the information matrix in the penalized optimization problem. The default is 0.

`upper` numeric value in (0, 1], design weight upper limit. The default is 1, corresponding to no constraint on the design weights.

Returns: `nlm_design` R6-object

`nlm_design$update()`: Updates the fields of an existing object. Inputs should be named `formula`, `data`, `theta`, `weights`, `cost`, `alpha`, `lambda` and/or `upper`.

Usage:

```
nlm_design$update(...)
```

Arguments:

`...` named arguments `formula`, `data`, `theta`, `weights`, `cost`, `alpha`, `lambda` and/or `upper`.

Returns: none

`nlm_design$stratify()`: Stratify the design by requiring that individual strata are represented proportionally. Call with `strata = NULL` to unset previously defined `strata`.

Usage:

```
nlm_design$stratify(strata, proportions = NULL)
```

Arguments:

`strata` a factor variable of the same length as the number of rows in `data`. The factor levels determine the partition of the design points into individual (disjoint) strata. Also accepts a one-sided formula made up of column names present in `data`, in which case the strata are defined by the interaction between terms in the model.frame obtained from the formula.

`proportions` an optional numeric vector containing the weight proportions of the individual strata. Must be of the same length as the number of factor levels in `strata`. If `NULL`, each stratum is assigned the same weight proportion.

Returns: none

`nlm_design$augment()`: Optimizes the augmented penalized approximate design given a vector of initial design weights

Usage:

```
nlm_design$augment(
  design_weights,
  gamma = 0.5,
  criterion = c("D", "A", "I", "G", "alias"),
  scs_control = scs_control_dflt(),
  verbose = FALSE,
  return_value = TRUE
)
```

Arguments:

`design_weights` a numeric vector with initial design weights, must be the same length as the number of design points in `data`. Also accepts a one-sided formula made up of parameters in `theta` and column names in `data`.

`gamma` numeric contribution weight of the initial design in the augmented design problem, see also 'Augmented optimal design' above. Defaults to 0.5.

`criterion` character string specifying the optimality criterion to use, see also 'Details'. The following choices are supported:

"D" D-optimality (default).
"A" A-optimality.
"I" I-optimality.
"G" G-optimality.
"alias" alias-optimality.

`scs_control` an optional list of control parameters to tune the SCS optimization. See `scs_control_dflt` for the available control parameters and their default values.

`verbose` display verbose messages printed by the scs solver. Defaults to `FALSE`.

`return_value` return the optimal design weights. If set to `FALSE`, the design weights should be retrieved from the `design_weights` field.

Returns: numeric vector if `return_value = TRUE`, otherwise none

`nlm_design$optimize()`: Optimizes the penalized approximate design based on the selected criterion.

Usage:

```
nlm_design$optimize(
  criterion = c("D", "A", "I", "G", "alias"),
  scs_control = scs_control_dflt(),
  verbose = FALSE,
  return_value = TRUE
)
```

Arguments:

`criterion` character string specifying the optimality criterion to use, see also 'Details'. The following choices are supported:

"D" D-optimality (default).
"A" A-optimality.
"I" I-optimality.
"G" G-optimality.
"alias" alias-optimality.

`scs_control` an optional list of control parameters to tune the SCS optimization. See `scs_control_dflt` for the available control parameters and their default values.

`verbose` display verbose messages printed by the scs solver. Defaults to `FALSE`.

`return_value` return the optimal design weights. If set to `FALSE`, the design weights should be retrieved from the `design_weights` field.

Returns: numeric vector if `return_value = TRUE`, otherwise none

`nlm_design$sparsify()`: Sparsify the penalized approximate design by iterative reweighting of the optimization problem.

Usage:

```
nlm_design$sparsify(
  max_iter,
  criterion = c("D", "A", "I", "G", "alias"),
  reweight_fun = c("entropy", "log-sum", "power"),
  eps = .Machine$double.eps,
  tol = .Machine$double.eps^(1/4),
  p = 0.5,
  design_weights = NULL,
  scs_control = scs_control_dflt(),
  return_value = TRUE,
  show_progress = TRUE
)
```

Arguments:

`max_iter` positive integer, maximum number of reweighted optimizations/iterations to run.

`criterion` character string specifying the optimality criterion to use, see also 'Details'. The following choices are supported:

"D" D-optimality (default).
"A" A-optimality.
"I" I-optimality.
"G" G-optimality.
"alias" alias-optimality.

`reweight_fun` character reweighting function, one of "entropy", "log-sum", "power", see lm_design for the definitions.

`eps` positive numeric value to ensure finite weights in reweighting functions.

`tol` numeric tolerance used as a threshold for mean absolute relative difference in design weights between iterations to stop early.

`p` positive numeric power with $0 < p < 1$. Only used if `reweight_fun = "power"`.

`design_weights` (optional) vector of design weights used as initial approximate design.

`scs_control` an optional list of control parameters to tune the SCS optimization. See `scs_control_dflt` for the available control parameters and their default values.

`return_value` return the optimal design weights. If set to `FALSE`, the design weights should be retrieved from the `design_weights` field.

`show_progress` logical, display a progress bar during the iterative reweighting procedure. Defaults to `TRUE`.

Details: The sparseness cost penalties are scaled by the same `alpha` tuning parameter as for a user-specific cost vector, see `scs_control_dflt`. If a cost vector `c_1,...,c_n` associated to the design weights is already specified. The sparseness penalty costs are multiplied by the existing cost vector.

Returns: numeric vector if `return_value = TRUE`, otherwise none

`nlm_design$round()`: Round penalized approximate design to an exact (integer) design. By default, the approximate design is initialized from the `design_weights` field, which is the optimal (approximate) design calculated by `$optimize()`. For method "optimal", ridge and cost penalties and upper limit constraints present in `$lambda`, `$cost` and `$upper` (if specified) are included

in the criterion optimized by the point-exchange algorithm. To override the initial approximate (optimal) design, a vector of design weights can be passed using the `design_weights` argument. The vector must be the same length as the number of design points in `data`. If `design_weights` is set to `FALSE`, the initial design is generated by random sampling.

Usage:

```
nlm_design$round(
  m,
  method = c("optimal", "efficient", "dist-sum", "dist-min"),
  max_iter = 100,
  n_repeats = 10,
  replicates = TRUE,
  tol = .Machine$double.eps^(1/3),
  seed = NULL,
  design_weights = TRUE,
  augment_design = NULL,
  show_progress = TRUE,
  ...
)
```

Arguments:

`m` integer number of design points included in the exact design.

`method` the method used to calculate the exact design. The following choices are supported:

`"optimal"` optimality criterion point-exchange algorithm (default).
`"efficient"` efficient rounding method of Pukelsheim & Reider (1992).
`"dist-sum"` maximize total pairwise (Euclidean) distance between design points.
`"dist-min"` maximize minimum pairwise (Euclidean) distance between design points.

`max_iter` integer number of maximum iterations in point-exchange algorithm. Only used when `method = "optimal"`, `method = "dist-sum"` or `method = "dist-min"`.

`n_repeats` integer number of (random) repeats of the rounding/exchange algorithm. The exact design with the highest criterion value is returned. Defaults to 10.

`replicates` logical, allow replicate design points in the exact design, defaults to `TRUE`. Can also be an integer number, limiting the maximum number of allowed replicates per design point. This argument overrides the `$upper` value constraint if existing.

`tol` numeric tolerance used as lower threshold to continue iterations in point-exchange algorithm.

`seed` (optional) random seed used to generate jitter to break ties in rounding the initial exact design.

`design_weights` (optional) vector of design weights used as initial approximate design, with length equal to the number of available design points. If `TRUE` (default), the approximate design is initialized from the `design_weights` field. If `FALSE`, the initial approximate design is generated by random sampling.

`augment_design` (optional) integer design to augment, with length equal to the number of available design points.

`show_progress` logical, display a progress bar during the rounding procedure (only if `n_repeats > 1`). Defaults to `TRUE`.

... additional arguments to tune algorithm settings:

- – criterion, character string specifying the optimality criterion for the rounding procedure in case design_weights has no "criterion" attribute, or to override the criterion specified in the "criterion" attribute. If specified, must be one of "A", "I", "D", "G" or "alias".

Returns: numeric vector

nlm_design$crit(): Evaluate optimality criterion of a penalized approximate or rounded (exact) design. The approximate design is initialized from the design_weights field, which is the optimal (approximate) design calculated by $optimize(). To override the default approximate (optimal) design, a vector of design weights can be passed using the design_weights argument.

Usage:

```
nlm_design$crit(
  design_weights = NULL,
  criterion = c("D", "A", "I", "G", "alias"),
  lambda = TRUE,
  alpha = TRUE
)
```

Arguments:

design_weights (optional) vector of design weights for which to evaluate optimality criterion.

criterion character string specifying the optimality criterion to use, see also 'Details'. The following choices are supported:

"D" D-optimality (default).
"A" A-optimality.
"I" I-optimality.
"G" G-optimality.
"alias" alias-optimality.

lambda logical, whether to include the ridge penalty in the calculated optimality criterion (see 'Optimality criteria'). Only relevant if $lambda is positive.

alpha logical, whether to include cost penalties in the calculated optimality criterion (see 'Optimality criteria'). Only relevant if cost penalties are present in the $cost field.

Returns: numeric criterion value

nlm_design$vcov(): Calculates the variance-covariance matrix of the model parameters based on an approximate or rounded (exact) design $(w_1, \ldots, w_n)$ according to:

$$\Sigma \;=\; (F(X,\theta)^T \Omega F(X,\theta))^{-1}$$

where Ω is $\mathrm{diag}(\omega_1, \ldots, \omega_n)$ with $\omega_i = w_i v_i$, and $F(X,\theta)$ is the Jacobian matrix at the nominal parameter values. The design weights are initialized from the design_weights field, which is the optimal (approximate) design calculated by $optimize(). Use the m argument to evaluate the covariance matrix for a specific number of design points, in which case the sum of the design weights is scaled to m. To override the default approximate (optimal) design weights, a vector of design weights can be passed using the design_weights argument. The vector must be the same length as the number of design points in data. If the vector of design weights is already an integer design, leave m unspecified, in which case the design is not rescaled.

Usage:

```
nlm_design$vcov(m, design_weights = NULL, sigma2 = 1)
```

Arguments:

`m` (optional) integer number of design points at which to evaluate the estimated covariance matrix. If missing, the design weights remain unscaled, which for the default approximate (optimal) design is equivalent to `m = 1`.

`design_weights` (optional) vector of design weights for which to evaluate the estimated covariance matrix.

`sigma2` numeric error variance used to scale the covariance matrix. Defaults to 1, returning the unscaled covariance matrix.

Returns: numeric matrix

`nlm_design$sep()`**:** Calculates the standard error of prediction or the square root prediction variance across the design grid based on an approximate or rounded (exact) design $(w_1, \ldots, w_n)$ according to:

$$\mathrm{SEP}(x_i) \;=\; \sigma\sqrt{F(x_i,\theta)^T \Sigma F(x_i,\theta)}, \quad \text{for} i = 1, \ldots, n$$

where Σ is the covariance matrix as returned by `$vcov()`. Use the `newdata` argument to evaluate the standard error of prediction at design points not in the candidate design grid defined by `$data`. The design weights are initialized from the `design_weights` field, which is the optimal (approximate) design calculated by `$optimize()`. To override the default approximate (optimal) design weights, a vector of design weights can be passed using the `design_weights` argument. The vector must be the same length as the number of design points in `data`.

Usage:

```
nlm_design$sep(m, newdata, design_weights = NULL, sigma2 = 1)
```

Arguments:

`m` (optional) integer number of design points at which to evaluate the estimated covariance matrix. If missing, the design weights remain unscaled, which for the default approximate (optimal) design is equivalent to `m = 1`.

`newdata` (optional) data frame or matrix with design points (rows) at which to return the standard error of prediction and which can be evaluated by `$formula`.

`design_weights` (optional) vector of design weights for which to evaluate the estimated covariance matrix.

`sigma2` numeric error variance used to scale the standard errors. Defaults to 1, returning the standardized standard error of prediction.

Returns: numeric vector

`nlm_design$corr()`**:** Calculates the correlation matrix of the columns of the Jacobian matrix based on an approximate or rounded (exact) design $(w_1, \ldots, w_n)$ according to:

$$R \;=\; D^{-1/2}(J_c^T \Omega J_c) D^{-1/2}$$

where Ω is $\mathrm{diag}(\omega_1, \ldots, \omega_n)$ with $\omega_i = w_i v_i$, and J_c is the Jacobian matrix $J = F(X, \theta)$ centered by the (weighted) column means:

$$\frac{\sum_{i=1}^n \omega_i J_{ij}}{\sum_{i=1}^n \omega_i}, \quad j = 1, \ldots, p$$

Furthermore, $D^{-1/2}$ is $\mathrm{diag}(1/\sqrt{D_1}, \ldots, 1/\sqrt{D_p})$, with $D_1, \ldots, D_p$ the diagonal elements of $J_c^T \Omega J_c$. If `center = FALSE`, the uncentered correlation matrix is returned, which corresponds to the scale-invariant (unit diagonal) version of the information matrix $M(w|X, v, \theta)$. The design weights are initialized from the `design_weights` field, which is the optimal (approximate) design calculated by `$optimize()`. To override the default approximate (optimal) design weights, a vector of design weights can be passed using the `design_weights` argument. The vector must be the same length as the number of design points in `data`.

Usage:

```
nlm_design$corr(design_weights = NULL, center = TRUE)
```

Arguments:

`design_weights` (optional) vector of design weights for which to evaluate the information matrix.

`center` logical, whether to center the Jacobian columns when evaluating the correlation matrix, defaults to `TRUE`.

Returns: numeric matrix

`nlm_design$clone()`**:** The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Lubricant example from [Bates & Watts (1998), Appendix 1, A1.8]

## design grid
data <- expand.grid(
  x1 = seq(from = 0, to = 100, by = 10), ## temp. (C)
  x2 = seq(from = 0, to = 7, by = 0.5)   ## pressure (atm)
)
## initialize model, nominal parameters and design grid
design <- nlm_design$new(
  formula = ~b1/(b2 + x1) + b3 * x2 + b4 * x2^2 + b5 * x2^3 +
              (b6 * x2 + b7 * x2^3) * exp(-x1/(b8 + b9 * x2^2)),
  theta = c(b1 = 1054.541, b2 = 206.546, b3 = 1.46, b4 = -0.26,
            b5 = 0.023, b6 = 0.401, b7 = 0.035, b8 = 57.405, b9 = -0.477),
  data = data
)
## approximate I-optimal design
design$optimize(criterion = "I")
## integer design w/ 11 design points
(exact_design <- design$round(m = 11, method = "optimal", seed = 1))
design$subset(replicates = exact_design)

## Penalized optimal design

design$update(
```

```
  cost = ~abs(x1 - 25) / 100 + abs(x2 - 1) / 7,                  ## condition weights
  alpha = 0.1,
  upper = 0.1                                                    ## max weight per point
)
## re-optimize
design$optimize(criterion = "I")
## optimal rounding w/ penalization
design$round(m = 14, method = "optimal", seed = 1)

## Stratified optimal design

design$update(
  cost = NULL,   ## drop cost penalties
  data = subset(data, x1 %in% c(0, 20, 40, 100))
)
## partition design space by temperature conditions
## by default, equal weight proportions are assigned to each condition
design$stratify(
  strata = ~x1
)
## stratified optimal design
design$optimize(criterion = "I")
## verify stratification
tapply(design$design_weights, design$data$x1, sum)
## optimal rounding w/ stratification
design$round(m = 12, method = "optimal", seed = 1)
```

`scs_control_dflt` *Tunable SCS solver parameters*

Description

Allow the user to tune the parameters of the SCS solver procedure.

Usage

```
scs_control_dflt(
  maxiter = 1e+05,
  eps_rel = 1e-05,
  eps_abs = 1e-05,
  eps_infeas = 1e-07,
  alpha_relax = 1.5,
  rho_x = 1e-06,
  scale = 0.1,
  normalize = TRUE,
  acceleration_lookback = 10,
  acceleration_interval = 5,
  adaptive_scale = TRUE,
```

```
  adaptive_diag_scale = TRUE,
  time_limit_secs = 60
)
```

Arguments

`maxiter`	positive integer, maximum number of iterations to run.
`eps_rel`	positive numeric, relative feasibility tolerance.
`eps_abs`	positive numeric, absolute feasibility tolerance.
`eps_infeas`	positive numeric, infeasibility tolerance (primal and dual).
`alpha_relax`	numeric value in (0, 2), douglas-Rachford relaxation parameter.
`rho_x`	positive numeric, primal scale factor.
`scale`	positive numeric, initial dual scale factor, updated if `adaptive_scale` is `TRUE`.
`normalize`	logical, whether to perform heuristic data rescaling.
`acceleration_lookback`	positive integer, relative amount of memory used for Anderson acceleration. More memory requires more time to compute but can give more reliable steps. If `acceleration_lookback` < 1, Anderson acceleration is disabled.
`acceleration_interval`	positive integer, run Anderson acceleration every `acceleration_interval` iteration(s).
`adaptive_scale`	logical, whether to heuristically adapt dual scale through the solve.
`adaptive_diag_scale`	logical, whether to refine the metric R per row, using the primal residual profile. Requires `adaptive_scale` (silently disabled without it).
`time_limit_secs`	positive numeric, time limit for solve run in seconds (can be fractional). The value 0 is interpreted as no limit. The default is 60.

Value

A `list` with components:

- maxiter
- eps_rel
- eps_abs
- eps_infeas
- alpha_relax
- rho_x
- scale
- normalize
- acceleration_lookback
- acceleration_interval

- adaptive_scale
- adaptive_diag_scale
- time_limit_secs

with meanings as explained under 'Arguments'.

See Also

https://www.cvxgrp.org/scs/api/settings

Examples

```
## default tuning parameters
scs_control_dflt()
```

Index

∗ **datasets**
ethanol, 2
fullerenes, 3
imidazoles, 27

ethanol, 2

factor, *8*, *19*, *32*, *36*, *46*, *59*
family, *5*, *7*, *14*, *17, 18*, *25*
formula, *5*, *7, 8*, *17–19*, *30–33*, *36*, *43*, *45, 46*, *57–59*
fullerenes, 3

gam, *7*, *14*
gam_design, 4
gaussian, *7*, *18*
glm_design, *4, 5*, 15

imidazoles, 27
interaction, *8*, *19*, *32*, *36*, *46*, *59*

lm_design, *5*, *10*, *16, 17*, *21*, *29*, *37*, 40, *57*, *61*
lme_design, 28

model.frame, *8*, *19*, *32*, *36*, *46*, *59*
model.matrix, *19*, *31*, *41*, *45*

nlm_design, 56

scs_control_dflt, *8–10*, *20, 21*, *33*, *47, 48*, *60, 61*, 66