Package 'GreedyExperimentalDesign'

May 7, 2026

Type Package

Title Greedy Experimental Design Construction

Version 1.6.1

Date 2026-04-30

Author Adam Kapelner [aut, cre] (ORCID: 0000-0001-5985-6792),
David Azriel [aut],
Abba Krieger [aut]

Maintainer Adam Kapelner <kapelner@qc.cuny.edu>

Description Computes experimental designs for two-arm experiments with covariates using multiple methods, including: (0) complete randomization and randomization with forced-balance; (1) greedy optimization of a balance objective function via pairwise switching; (2) numerical optimization via 'gurobi'; (3) rerandomization; (4) Karp's method for one covariate; (5) exhaustive enumeration for small sample sizes; (6) binary pair matching using 'nbpMatching'; (7) binary pair matching plus method (1) to further optimize balance; (8) binary pair matching plus method (3) to further optimize balance; (9) Hadamard designs; and (10) simultaneous multiple kernels. For the greedy, rerandomization, and related methods, three objective functions are supported: Mahalanobis distance, standardized sums of absolute differences, and kernel distances via the 'kernlab' library. This package is the result of a stream of research that can be found in Krieger, A. M., Azriel, D. A., and Kapelner, A. (2019). ``Nearly Random Designs with Greatly Improved Balance." Biometrika 106(3), 695-701 <doi:10.1093/biomet/asz026>. Krieger, A. M., Azriel, D. A., and Kapelner, A. (2023). ``Better experimental design by hybridizing binary matching with imbalance optimization." Canadian Journal of Statistics, 51(1), 275-292 <doi:10.1002/cjs.11685>.

License GPL-3

Encoding UTF-8

Depends R (>= 4.1.0), rJava (>= 0.9-6)

SystemRequirements Java (>= 22.0); wgpu-native >= 0.19.0 (optional, for GPU acceleration, see https://github.com/gfx-rs/wgpu-native)

LinkingTo Rcpp

Imports Rcpp, checkmate, nbpMatching, rlist, stringr, stringi, kernlab, ggplot2, graphics, grDevices, stats

Suggests testthat (>= 3.0.0), pkgload, R6

Config/testthat/edition 3

URL https://github.com/kapelner/GreedyExperimentalDesign

RoxygenNote 7.3.3

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-04-30 07:40:09 UTC

Contents

all_elements_same_cpp_wrap

Tests if a vector has all elements the same

Description

Tests if a vector has all elements the same

Usage

```
all_elements_same_cpp_wrap(w)
```

Arguments

w	The vector to be queried

Value

A boolean if it has all same elements

Author(s)

Adam Kapelner

Examples

```
## Not run:
all_elements_same_cpp_wrap(c(1, 1, 1))
all_elements_same_cpp_wrap(c(1, 2, 1))

## End(Not run)
```

complete_randomization

Implements complete randomization (without forced balance)

Description

For debugging, you can use `set.seed` to be assured of deterministic output.

Usage

```
complete_randomization(n, r, form = "one_zero")
```

Arguments

n	number of observations
r	number of randomized designs you would like
form	Which form should it be in? The default is one_zero for 1/0's or pos_one_min_one for +1/-1's.

Value

a matrix where each column is one of the r designs

Author(s)

Adam Kapelner

Examples

```
## Not run:
complete_randomization(n = 6, r = 2)

## End(Not run)
```

---

complete_randomization_with_forced_balanced

Implements forced balanced randomization

Description

For debugging, you can use set.seed to be assured of deterministic output.

Usage

```
complete_randomization_with_forced_balanced(
  n,
  r,
  form = "one_zero",
  seed = NULL
)
```

Arguments

n	number of observations
r	number of randomized designs you would like
form	Which form should it be in? The default is one_zero for 1/0's or pos_one_min_one for +1/-1's.
seed	An integer which is the seed to be set within C++. Default is NULL which means the seed is set from the system clock.

Value

a matrix where each column is one of the r designs

Author(s)

Adam Kapelner

Examples

```
## Not run:
complete_randomization_with_forced_balanced(n = 6, r = 2, seed = 1)

## End(Not run)
```

`computeBinaryMatchStructure` *Compute Binary Matching Strcuture*

Description

This method creates an object of type binary_match_structure and will compute pairs. You can then use the functions `initBinaryMatchExperimentalDesignSearchObject` and `resultsBinaryMatchSearch` to create randomized allocation vectors. For one column in X, we just sort to find the pairs trivially.

Usage

```
computeBinaryMatchStructure(
  X,
  mahal_match = FALSE,
  compute_dist_matrix = NULL,
  D = NULL,
  symmetry_tol = 1e-12,
  use_safe_inverse = FALSE
)
```

Arguments

X	The design matrix with n rows (one for each subject) and p columns (one for each measurement on the subject). This is the design matrix you wish to search for a more optimal design.
mahal_match	Match using Mahalanobis distance. Default is FALSE.
compute_dist_matrix	The function that computes the distance matrix between every two observations in X, its only argument. The default is NULL signifying euclidean squared distance optimized in C++.
D	A distance matrix precomputed. The default is NULL indicating the distance matrix should be computed.

`symmetry_tol` Tolerance for symmetry check on `D`. Default is `1e-12`.

`use_safe_inverse`
Should a regularized inverse be used for the Mahalanobis objective? Default is `FALSE`.

Value

An object of type `binary_experimental_design` which can be further operated upon.

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(16), nrow = 8)
bms = computeBinaryMatchStructure(X)
bms$indicies_pairs

## End(Not run)
```

---

compute_distance_matrix_cpp_wrap

Computes a Euclidean-squared distance matrix rapidly

Description

Computes a Euclidean-squared distance matrix rapidly

Usage

```
compute_distance_matrix_cpp_wrap(X)
```

Arguments

`X` A numeric matrix with n rows representing each subject and p columns which are measurements on each subject

Value

The n x n Euclidean distances squared

Author(s)

Adam Kapelner

Examples

```
## Not run:
X = matrix(c(0, 1, 2, 3), nrow = 2)
compute_distance_matrix_cpp_wrap(X)

## End(Not run)
```

`compute_distance_matrix_gpu` *Compute a squared Euclidean distance matrix through the optional native backend*

Description

Compute a squared Euclidean distance matrix through the optional native backend

Usage

```
compute_distance_matrix_gpu(X, backend = "auto", device = 0)
```

Arguments

`X`	A numeric matrix.
`backend`	One of `"auto"`, `"cpu"`, or `"gpu"`.
`device`	Zero-based device id. -1 for native C++.

Value

An `n` x `n` squared Euclidean distance matrix.

`compute_gram_matrix` *Gram Matrix Computation*

Description

Computes the Gram Matrix for a user-specified kernel using the library `kernlab`. Note that this function automatically standardizes the columns of the data entered.

Usage

```
compute_gram_matrix(X, kernel_type, params = c())
```

Arguments

X	The design matrix with n rows (one for each subject) and p columns (one for each measurement on the subject). This is the design matrix you wish to search for a more optimal design.
kernel_type	One of the following: "vanilla", "rbf", "poly", "tanh", "bessel", "laplace", "anova" or "spline".
params	A vector of numeric parameters. Each kernel_type has different numbers of parameters required. For more information see documentation for the kernlab library.

Value

The n x n gram matrix for the given kernel on the given data.

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(12), nrow = 6)
K = compute_gram_matrix(X, kernel_type = "rbf", params = 0.5)
dim(K)

## End(Not run)
```

---

compute_kernel_matrix_gpu

Compute a kernel matrix through the optional native backend

Description

Compute a kernel matrix through the optional native backend

Usage

```
compute_kernel_matrix_gpu(X, kernel = "gaussian", ...)
```

Arguments

X	A numeric matrix.
kernel	One of "gaussian", "laplacian", "inv_mult_quad", "exponential", or "poly".
...	Optional kernel parameters. Supported names are gamma and poly_s.

Value

An n x n kernel matrix.

compute_multiple_kernel_objective_vals_gpu *Compute multiple kernel objective values using GPU*

Description

Compute multiple kernel objective values using GPU

Usage

```
compute_multiple_kernel_objective_vals_gpu(
  W,
  Kgrams,
  weights,
  initial_objs,
  running_sums,
  max_reds,
  maximum_gain_scaling,
  device = 0
)
```

Arguments

W	The design matrix (r x n)
Kgrams	A list of Gram matrices
weights	A vector of weights for the kernels
initial_objs	A vector of initial objective values
running_sums	A vector of current kernel sums
max_reds	A vector of max reduction log objective values
maximum_gain_scaling	Scaling factor
device	The device ID

compute_objective_val *Computes Objective Value From Allocation Vector*

Description

Returns the objective value given a design vector as well an an objective function. This is sometimes duplicated in Java. However, within Java, tricks are played to make optimization go faster so Java's objective values may not always be the same as the true objective function (e.g. logs or constants dropped).

Usage

```
compute_objective_val(
  X,
  indic_T,
  objective = "abs_sum_diff",
  inv_cov_X = NULL,
  use_safe_inverse = FALSE
)
```

Arguments

X	The n x p design matrix
indic_T	The n-length binary allocation vector
objective	The objective function to use. Default is abs_sum_diff and the other option is mahal_dist.
inv_cov_X	Optional: the inverse sample variance covariance matrix. Use this argument if you will be doing many calculations since passing this in will cache this data.
use_safe_inverse	Should a regularized inverse be used for the Mahalanobis objective? Default is FALSE.

Author(s)

Adam Kapelner

Examples

```
## Not run:
X = matrix(rnorm(12), nrow = 6)
indic_T = c(1, 0, 1, 0, 1, 0)
compute_objective_val(X, indic_T, objective = "abs_sum_diff")

## End(Not run)
```

---

compute_objective_vals_gpu *Compute kernel objective values through the optional native backend*

Description

Compute kernel objective values through the optional native backend

Usage

```
compute_objective_vals_gpu(W, Kgram, device = 0)
```

Arguments

W	An r x n numeric matrix of design vectors.
Kgram	An n x n kernel Gram matrix.
device	Zero-based device id. -1 for native C++.

Value

A numeric vector with one quadratic form per row of W.

---

compute_randomization_metrics

Computes Randomization Metrics (explained in paper) about a design algorithm

---

Description

Computes Randomization Metrics (explained in paper) about a design algorithm

Usage

```
compute_randomization_metrics(designs)
```

Arguments

designs	A matrix where each column is one design.

Value

A list of resulting data: the probability estimates for each pair in the design of randomness where estmates close to ~0.5 represent random assignment, then the entropy metric the distance metric, the maximum eigenvalue of the allocation var-cov matrix (operator norm) and the squared Frobenius norm (the sum of the squared eigenvalues)

Author(s)

Adam Kapelner

Examples

```
## Not run:
designs = matrix(c(1, 0, 1, 0, 0, 1, 0, 1), nrow = 4, ncol = 2)
compute_randomization_metrics(designs)

## End(Not run)
```

compute_randomization_metrics_gpu

Compute randomization metrics through the optional native backend

Description

Compute randomization metrics through the optional native backend

Usage

```
compute_randomization_metrics_gpu(W, device = 0)
```

Arguments

W	An r x n integer matrix of design vectors (1/0).
device	Zero-based device id. -1 for native C++.

Value

An n x n matrix of same-group probabilities.

create_all_ys_cpp_wrap

Create all binary Y's convenience function using a randomized design

Description

Create all binary Y's convenience function using a randomized design

Usage

```
create_all_ys_cpp_wrap(pCs, pTs, W, two_n, nY)
```

Arguments

pCs	Control-group success probabilities (length two_n)
pTs	Treatment-group success probabilities (length two_n)
W	Assignment matrix with nY rows and two_n columns
two_n	Total number of units
nY	Number of Y vectors to generate

Value

A matrix of boolean Y's

Author(s)

Adam Kapelner

Examples

```
## Not run:
pCs = rep(0.2, 4)
pTs = rep(0.8, 4)
W = matrix(c(1, 0, 1, 0, 0, 1, 0, 1), nrow = 2, byrow = TRUE)
create_all_ys_cpp_wrap(pCs, pTs, W, two_n = 4, nY = 2)

## End(Not run)
```

---

full_greedy_search_gpu *Run a full greedy search on GPU (Upload Once, In-Place)*

Description

Run a full greedy search on GPU (Upload Once, In-Place)

Usage

```
full_greedy_search_gpu(X, Sinv, start_indicT, max_iters = 100, device = 0)
```

Arguments

X	The design matrix
Sinv	Inverse covariance matrix
start_indicT	Starting allocation
max_iters	Maximum iterations
device	Device ID

---

ged_gpu_available *Check for optional GPU support*

Description

Check for optional GPU support

Usage

```
ged_gpu_available()
```

Value

A logical scalar. Attribute `wgpu_compiled` is `TRUE` when the package was compiled with WebGPU backend support.

ged_gpu_devices *List optional GPU devices*

Description

List optional GPU devices

Usage

```
ged_gpu_devices()
```

Value

A data frame with columns `id`, `name`, `backend`, `type`, and `usable`. It is empty when GPU support was not compiled into the package.

generate_block_design_cpp_wrap *Generates homogeneous block design allocations rapidly*

Description

Generates homogeneous block design allocations rapidly

Usage

```
generate_block_design_cpp_wrap(B, nR, dummy_block)
```

Arguments

`B`	The number of blocks in the design
`nR`	The number of allocation vectors
`dummy_block`	The subvector of allocations in each block that will be permuted

Value

A matrix with rows being the nR random block allocation of sample size B x length(dummy_block).

Author(s)

Adam Kapelner

Examples

```
## Not run:
generate_block_design_cpp_wrap(B = 2, nR = 3, dummy_block = c(1, 0))

## End(Not run)
```

`generate_stdzied_design_matrix` *Generates a design matrix with standardized predictors.*

Description

This function is useful for debugging.

Usage

```
generate_stdzied_design_matrix(n = 50, p = 1, covariate_gen = rnorm, ...)
```

Arguments

`n`	Number of rows in the design matrix
`p`	Number of columns in the design matrix
`covariate_gen`	The function to use to draw the covariate realizations (assumed to be iid). This defaults to `rnorm` for $N(0,1)$ draws.
`...`	Optional arguments to be passed to the `covariate_dist` function.

Value

THe design matrix

Author(s)

Adam Kapelner

Examples

```
## Not run:
X = generate_stdzied_design_matrix(n = 6, p = 2)
colMeans(X)

## End(Not run)
```

gen_pm_designs_cpp_wrap *Create PM designs*

Description

Create PM designs

Usage

```
gen_pm_designs_cpp_wrap(indicies_pairs, n, r)
```

Arguments

indicies_pairs	A matrix of n x 2 indicies where each row is a pair of subjects' indicies
n	Half the number of subjects i.e. the number of pairs
r	The number of assignments to generate

Value

A matrix of r x 2n PM designs of +1/-1 assignments

Author(s)

Adam Kapelner

Examples

```
## Not run:
indicies_pairs = matrix(c(1, 2, 3, 4), ncol = 2, byrow = TRUE)
gen_pm_designs_cpp_wrap(indicies_pairs, n = 2, r = 3)

## End(Not run)
```

gen_var_cov_matrix_block_designs *Computes varcov matrix for block designs*

Description

The varcov matrix for block designs consists of a block- diagonal matrix with B blocks (the number of blocks in the design) with off-diagonal entries = -1 / (n/B - 1) where n is the number of subjected in the study.

Usage

```
gen_var_cov_matrix_block_designs(n, prop_T, B, use_cache = TRUE)
```

Arguments

n	number of observations
prop_T	the proportion of treatments allocated
B	the number of blocks
use_cache	Cache results for repeated calls with identical inputs. Default is TRUE.

Value

varcov matrix for the specific block design

Author(s)

Adam Kapelner

Examples

```
## Not run:
gen_var_cov_matrix_block_designs(n = 12, prop_T = 0.5, B = 3)

## End(Not run)
```

GreedyExperimentalDesign

Greedy Experimental Design Search

Description

A tool to find many types of a priori experimental designs

Author(s)

Adam Kapelner <kapelner@qc.cuny.edu>

References

Kapelner, A

`greedy_orthogonalization_curation` *Curate More Orthogonal Vectors Greedily*

Description

This function takes a set of allocation vectors and pares them down one-by-one by eliminating the vector that can result in the largest reduction in Avg[|r_ij|]. It is recommended to begin with a set of unmirrored vectors for speed. Then add the mirrors later for whichever subset you wish.

Usage

```
greedy_orthogonalization_curation(W, Rmin = 2, verbose = FALSE)
```

Arguments

`W`	A matrix in in the set $-1, 1^{Rxn}$ which have R allocation vectors for an experiment of sample size n.
`Rmin`	The minimum number of vectors to consider in a design. The default is the true bottom, two.
`verbose`	Default is `FALSE` but if not, it will print out a message for each iteration.

Value

A list with two elements: (1) `avg_abs_rij_by_R` which is a data frame with R - Rmin + 1 rows and columns R and average absolute r_ij and (2) `Wsorted` which provides the collection of vectors in sorted by best average absolute r_ij in row order from best to worst.

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
W = matrix(sample(c(-1, 1), 6 * 8, replace = TRUE), nrow = 6)
res = greedy_orthogonalization_curation(W, Rmin = 3, verbose = FALSE)
res$avg_abs_rij_by_R

## End(Not run)
```

`greedy_orthogonalization_curation2` *Curate More Orthogonal Vectors Greedily*

Description

This function takes a set of allocation vectors and pares them down one-by-one by eliminating the vector that can result in the largest reduction in Avg[|r_ij|]. It is recommended to begin with a set of unmirrored vectors for speed. Then add the mirrors later for whichever subset you wish.

Usage

```
greedy_orthogonalization_curation2(W, R0 = 100, verbose = FALSE)
```

Arguments

`W`	A matrix in $-1, 1^{Rxn}$ which have R allocation vectors for an experiment of sample size n.
`R0`	The minimum number of vectors to consider in a design. The default is the true bottom, two.
`verbose`	Default is `FALSE` but if not, it will print out a message for each iteration.

Value

A list with two elements: (1) `avg_abs_rij_by_R` which is a data frame with R - Rmin + 1 rows and columns R and average absolute r_ij and (2) `Wsorted` which provides the collection of vectors in sorted by best average absolute r_ij in row order from best to worst.

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
W = matrix(sample(c(-1, 1), 6 * 8, replace = TRUE), nrow = 6)
W2 = greedy_orthogonalization_curation2(W, R0 = 4, verbose = FALSE)
dim(W2)

## End(Not run)
```

hadamardExperimentalDesign *Create a Hadamard Design*

Description

This method returns unique designs according to a Hadamard matrix. For debugging, you can use `set.seed` to be assured of deterministic output.

Usage

```
hadamardExperimentalDesign(X, strict = TRUE, form = "one_zero")
```

Arguments

`X`	The design matrix with n rows (one for each subject) and p columns (one for each measurement on the subject). The measurements aren't used to compute the Hadamard designs, only the number of rows.
`strict`	Hadamard matrices are not available for all n.
`form`	Which form should it be in? The default is `one_zero` for 1/0's or `pos_one_min_one` for +1/-1's.

Value

An matrix of dimension R x n where R is the number of Hadamard allocations.

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(16), nrow = 8)
W = hadamardExperimentalDesign(X, strict = TRUE, form = "one_zero")
dim(W)

## End(Not run)
```

`imbalanced_block_designs` *Implements unequally allocated block designs*

Description

For debugging, you can use `set.seed` to be assured of deterministic output. The following quantities in this design must be integer valued or an error will be thrown: n_B := n / B and n_B * prop_T

Usage

```
imbalanced_block_designs(n, prop_T, B, r, form = "one_zero", seed = NULL)
```

Arguments

`n`	number of observations
`prop_T`	the proportion of treatments allocated
`B`	the number of blocks
`r`	number of randomized designs you would like
`form`	Which form should it be in? The default is `one_zero` for 1/0's or `pos_one_min_one` for +1/-1's.
`seed`	An integer which is the seed to be set within C++. Default is `NULL` which means the seed is set from the system clock.

Value

a matrix where each column is one of the `r` designs

Author(s)

Adam Kapelner

Examples

```
## Not run:
imbalanced_block_designs(n = 12, prop_T = 0.5, B = 3, r = 2, seed = 1)

## End(Not run)
```

`imbalanced_complete_randomization` *Implements unequally allocated complete randomization*

Description

For debugging, you can use `set.seed` to be assured of deterministic output.

Usage

```
imbalanced_complete_randomization(n, prop_T, r, form = "one_zero", seed = NULL)
```

Arguments

`n`	number of observations
`prop_T`	the proportion of treatments needed
`r`	number of randomized designs you would like
`form`	Which form should it be in? The default is `one_zero` for 1/0's or `pos_one_min_one` for +1/-1's.
`seed`	An integer which is the seed to be set within C++. Default is `NULL` which means the seed is set from the system clock.

Value

a matrix where each column is one of the `r` designs

Author(s)

Adam Kapelner

Examples

```
## Not run:
imbalanced_complete_randomization(n = 10, prop_T = 0.3, r = 2, seed = 1)

## End(Not run)
```

`initBinaryMatchExperimentalDesignSearchObject` *Begin a Binary Match Search*

Description

This method creates an object of type pairwise_matching_experimental_design_search and will immediately initiate a search through allocation space for pairwise match designs based on the structure computed in the function `computeBinaryMatchStructure`. For debugging, you can use set the `seed` parameter and `num_cores = 1` to be assured of deterministic output.

Usage

```
initBinaryMatchExperimentalDesignSearchObject(
  binary_match_structure,
  max_designs = 1000,
  wait = FALSE,
  start = TRUE,
  num_cores = 1,
  seed = NULL,
  prop_flips = 1,
  verbose = TRUE
)
```

Arguments

`binary_match_structure`	The `binary_experimental_design` object where the pairs are computed.
`max_designs`	How many random allocation vectors you wish to return. The default is 1000.
`wait`	Should the `R` terminal hang until all `max_designs` vectors are found? The default is `FALSE`.
`start`	Should we start searching immediately (default is `TRUE`).
`num_cores`	The number of CPU cores you wish to use during the search. The default is `1`.
`seed`	The set to set for deterministic output. This should only be set if `num_cores = 1` otherwise the output will not be deterministic. Default is `NULL` for no seed set.
`prop_flips`	Proportion of flips. Default is all. Lower for more correlated assignments (useful for research only).
`verbose`	Should the algorithm emit progress output? Default is `TRUE`.

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(16), nrow = 8)
bms = computeBinaryMatchStructure(X)
bm = initBinaryMatchExperimentalDesignSearchObject(
  bms,
  max_designs = 4,
  num_cores = 1,
  start = TRUE,
  wait = TRUE,
  seed = 1,
  verbose = FALSE
)
bm

## End(Not run)
```

`initBinaryMatchFollowedByGreedyExperimentalDesignSearchObject` *Begin a Search for Binary Matching Followed by Greedy Switch Designs*

Description

This method creates an object of type binary_then_greedy_experimental_design and will find optimal matched pairs which are then greedily switched in order to further minimize a balance metric. You can then use the function `resultsBinaryMatchThenGreedySearch` to obtain the randomized allocation vectors. For one column in X, the matching just sorts the values to find the pairs trivially.

Usage

```
initBinaryMatchFollowedByGreedyExperimentalDesignSearchObject(
  X,
  diff_method = FALSE,
  compute_dist_matrix = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`X`	The design matrix with n rows (one for each subject) and p columns (one for each measurement on the subject). This is the design matrix you wish to search for a more optimal design.
`diff_method`	Once the subjects (i.e. row vectors) are paired, do we create a set of n/2 difference vectors and feed that into greedy? If `TRUE`, this technically breaks the objective function, but it is shown to have better performance. The default is thus `FALSE`.

`compute_dist_matrix`
: The function that computes the distance matrix between every two observations in `X`, its only argument. The default is `NULL` signifying euclidean squared distance optimized in C++.

`verbose`
: Should the algorithm emit progress output? Default is `TRUE`.

`...`
: Arguments passed to `initGreedyExperimentalDesignObject`. It is recommended to set `max_designs` otherwise it will default to 10,000.

Value

An object of type `binary_experimental_design` which can be further operated upon.

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(16), nrow = 8)
obj = initBinaryMatchFollowedByGreedyExperimentalDesignSearchObject(
  X,
  max_designs = 4,
  num_cores = 1,
  objective = "abs_sum_diff",
  start = TRUE,
  wait = TRUE,
  verbose = FALSE
)
obj

## End(Not run)
```

`initBinaryMatchFollowedByRerandomizationDesignSearchObject` *Begin a Search for Binary Matching Followed by Rerandomization*

Description

This method creates an object of type binary_then_rerandomization_experimental_design and will find optimal matched pairs which are then rerandomized in order to further minimize a balance metric. You can then use the function `resultsBinaryMatchThenRerandomizationSearch` to obtain the randomized allocation vectors. For one column in X, the matching just sorts the values to find the pairs trivially.

Usage

```
initBinaryMatchFollowedByRerandomizationDesignSearchObject(
  X,
  compute_dist_matrix = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

X	The design matrix with n rows (one for each subject) and p columns (one for each measurement on the subject). This is the design matrix you wish to search for a more optimal design.
compute_dist_matrix	The function that computes the distance matrix between every two observations in X, its only argument. The default is NULL signifying euclidean squared distance optimized in C++.
verbose	Should the algorithm emit progress output? Default is TRUE.
...	Arguments passed to initGreedyExperimentalDesignObject. It is recommended to set max_designs otherwise it will default to 10,000.

Value

An object of type binary_experimental_design which can be further operated upon.

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(16), nrow = 8)
obj = initBinaryMatchFollowedByRerandomizationDesignSearchObject(
  X,
  max_designs = 4,
  num_cores = 1,
  objective = "abs_sum_diff",
  obj_val_cutoff_to_include = Inf,
  start = TRUE,
  wait = TRUE,
  verbose = FALSE
)
obj

## End(Not run)
```

`initGreedyExperimentalDesignObject` *Begin A Greedy Pair Switching Search*

Description

This method creates an object of type greedy_experimental_design and will immediately initiate a search through allocation space for forced balance designs. For debugging, you can use set the `seed` parameter and `num_cores = 1` to be assured of deterministic output.

Usage

```
initGreedyExperimentalDesignObject(
  X = NULL,
  nT = NULL,
  max_designs = 10000,
  objective = "mahal_dist",
  indicies_pairs = NULL,
  Kgram = NULL,
  wait = FALSE,
  start = TRUE,
  max_iters = Inf,
  semigreedy = FALSE,
  diagnostics = FALSE,
  num_cores = 1,
  seed = NULL,
  verbose = TRUE,
  use_safe_inverse = FALSE
)
```

Arguments

`X`	The design matrix with n rows (one for each subject) and p columns (one for each measurement on the subject). This is the design matrix you wish to search for a more optimal design. This parameter must be specified unless you choose objective type `"kernel"` in which case, the `Kgram` parameter must be specified.
`nT`	The number of treatments to assign. Default is `NULL` which is for forced balance allocation i.e. nT = nC = n / 2 where n is the number of rows in X (or Kgram if X is unspecified).
`max_designs`	The maximum number of designs to be returned. Default is 10,000. Make this large so you can search however long you wish as the search can be stopped at any time by using the `stopSearch` method
`objective`	The objective function to use when searching design space. This is a string with valid values `"mahal_dist"` (the default), `"abs_sum_diff"` or `"kernel"`.

indicies_pairs A matrix of size $n/2$ times 2 whose rows are indicies pairs. The values of the entire matrix must enumerate all indicies $1, ..., n$. The default is NULL meaning to use all possible pairs.

Kgram If the objective = kernel, this argument is required to be an n x n matrix whose entries are the evaluation of the kernel function between subject i and subject j. Default is NULL.

wait Should the R terminal hang until all max_designs vectors are found? The deafult is FALSE.

start Should we start searching immediately (default is TRUE).

max_iters Should we impose a maximum number of greedy switches? The default is Inf which a flag for "no limit."

semigreedy Should we use a fully greedy approach or the quicker semi-greedy approach? The default is FALSE corresponding to the fully greedy approach.

diagnostics Returns diagnostic information about the iterations including (a) the initial starting vectors, (b) the switches at every iteration and (c) information about the objective function at every iteration (default is FALSE to decrease the algorithm's run time).

num_cores The number of CPU cores you wish to use during the search. The default is 1.

seed The set to set for deterministic output. This should only be set if num_cores = 1 otherwise the output will not be deterministic. Default is NULL for no seed set.

verbose Should the algorithm emit progress output? Default is TRUE.

use_safe_inverse Should a regularized inverse be used for the Mahalanobis objective? Default is FALSE.

Value

An object of type greedy_experimental_design_search which can be further operated upon

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(20), nrow = 10)
ged = initGreedyExperimentalDesignObject(
  X,
  max_designs = 5,
  num_cores = 1,
  objective = "abs_sum_diff",
  start = TRUE,
  wait = TRUE,
  verbose = FALSE
)
```

```
ged

## End(Not run)
```

`initGreedyMultipleKernelExperimentalDesignObject` *Begin A Greedy Multiple Kernel Design Search*

Description

This method creates an object of type greedy_multiple_kernel_experimental_design and will immediately initiate a search through allocation space for forced balance designs. For debugging, you can use set the `seed` parameter and `num_cores = 1` to be assured of deterministic output.

Usage

```
initGreedyMultipleKernelExperimentalDesignObject(
  X = NULL,
  kernel_names = NULL,
  Kgrams = NULL,
  kernel_weights = NULL,
  objective = "added_pct_reduction",
  maximum_gain_scaling = TRUE,
  nT = NULL,
  max_designs = 10000,
  kernel_pre_num_designs = 1000,
  wait = FALSE,
  start = TRUE,
  max_iters = Inf,
  semigreedy = FALSE,
  diagnostics = FALSE,
  num_cores = 1,
  seed = NULL,
  verbose = TRUE,
  use_safe_inverse = FALSE
)
```

Arguments

`X`	The design matrix with n rows (one for each subject) and p columns (one for each measurement on the subject). This parameter must be specified unless you choose objective type `"kernel"` in which case, the `Kgrams` parameter must be specified.
`kernel_names`	A vector of M >= 1 strings indicating the kernels to be used. Valid values are `"mahalanobis"`, `"gaussian"`, `"laplacian"`, `"exponential"`, `"inv_mult_quad"`, `"poly_2"`, `"poly_3"`. Or a special value `"manual"` to indicate the kernels are specified manually using the `Kgrams` parameter.

Kgrams A list of M >= 1 elements where each is a n x n matrix whose entries are the evaluation of the kernel function between subject i and subject j. Default is NULL.

kernel_weights A vector of M weights for each kernel which should sum to 1. Default is NULL which specifies equal weighting.

objective The method used to aggregate the kernel objective functions together. Default is "added_pct_reduction".

maximum_gain_scaling
Should we scale the kernels to have the same maximum gain? Default is TRUE.

nT The number of treatments to assign. Default is NULL which is for forced balance allocation i.e. nT = nC = n / 2 where n is the number of rows in X (or Kgram if X is unspecified).

max_designs The maximum number of designs to be returned. Default is 10,000. Make this large so you can search however long you wish as the search can be stopped at any time by using the stopSearch method.

kernel_pre_num_designs
The number of initial designs to search through before starting the greedy search for each kernel. Default is 1000.

wait Should the R terminal hang until all max_designs vectors are found? The deafult is FALSE.

start Should we start searching immediately (default is TRUE).

max_iters The maximum number of iterations of the greedy search algorithm to run. Default is Inf.

semigreedy Should we use a fully greedy approach or the quicker semi-greedy approach? The default is FALSE corresponding to the fully greedy approach.

diagnostics Should the objective function values at each iteration be saved? Default is FALSE.

num_cores The number of CPU cores to use for the search. Default is 1.

seed The set to set for deterministic output. This should only be set if num_cores = 1 otherwise the output will not be deterministic. Default is NULL for no seed set.

verbose Should the algorithm emit progress output? Default is TRUE.

use_safe_inverse
Should a regularized inverse be used for the Mahalanobis objective? Default is FALSE.

Value

An object of type greedy_multiple_kernel_experimental_design which can be further operated upon

Author(s)

Adam Kapelner

Examples

```
 ## Not run:
library(MASS)
data(Boston)
 #pretend the Boston data was an experiment setting
#first pull out the covariates
 X = Boston[, 1 : 13]
 #begin the greedy design search
mk = initGreedyMultipleKernelExperimentalDesignObject(X,
max_designs = 100, num_cores = 3, kernel_names = c("mahalanobis", "gaussian"))
#wait
res = resultsMultipleKernelGreedySearch(mk, max_vectors = 2)
design = res$ending_indicTs[1, ] #ordered already by best-->worst
 design
#how far have we come of the 100 we set out to do?
mk
#we can cut it here
stopSearch(mk)

## End(Not run)
```

---

initGurobiNumericalOptimizationExperimentalDesignObject

Begin Gurobi Optimized Search

Description

This method creates an object of type optimal_experimental_design and will immediately initiate a search through allocation space for forced balance designs. Make sure you setup Gurobi properly first. This means applying for a license, downloading, installing, registering it on your computer using the grbgetkey command with the license file in the default directory. Then, in R, install the package from the file in your gurobi directory.

Usage

```
initGurobiNumericalOptimizationExperimentalDesignObject(
  X = NULL,
  objective = "mahal_dist",
  Kgram = NULL,
  num_cores = 2,
  w_0 = NULL,
  initial_time_limit_sec = 5 * 60,
  restart_time_limit_sec = 60,
  max_number_of_restarts = 0,
  max_no_good_cuts = 0,
  verbose = TRUE,
  gurobi_params = list(),
  use_safe_inverse = FALSE,
```

```
  r,
  pool_solutions = NULL,
  pool_gap = 0.2,
  pool_gap_abs = NULL,
  pool_search_mode = 2,
  mip_gap = 1e-04,
  mip_gap_abs = 1e-10,
  mip_focus = 1,
  heuristics = 0.2,
  cuts = 2,
  presolve = 2
)
```

Arguments

X	The design matrix with n rows (one for each subject) and p columns (one for each measurement on the subject). This is the design matrix you wish to search for a more optimal design.
objective	The objective function to use when searching design space. This is a string with valid values "mahal_dist" (the default) or "kernel".
Kgram	If the objective = kernel, this argument is required to be an n x n matrix whose entries are the evaluation of the kernel function between subject i and subject j. Default is NULL.
num_cores	Number of cores to use during search. Default is 2.
w_0	The initial starting location (optional).
initial_time_limit_sec	The maximum amount of time the optimizer can run for in seconds. The default is 5 * 60.
restart_time_limit_sec	The maximum amount of time each restart can run for in seconds. The default is 60.
max_number_of_restarts	The maximum number of restarts to attempt if too few unique solutions are returned. Default is 0.
max_no_good_cuts	The maximum number of no-good cuts to attempt. Default is 0 (disabled).
verbose	Should Gurobi log to console? Default is TRUE.
gurobi_params	A list of optional parameters to be passed to Gurobi (see their documentation online).
use_safe_inverse	Should a regularized inverse be used for the Mahalanobis objective? Default is FALSE.
r	Number of solution vectors to request from the Gurobi pool.
pool_solutions	Number of solutions to request from the Gurobi pool. Defaults to 10 * r.
pool_gap	Relative optimality gap for the pool. Default is 0.2. Use NULL to skip.

`pool_gap_abs` Absolute optimality gap for the pool. Default is `NULL` to skip.

`pool_search_mode`
Solution pool search mode. Default is `2` for diverse solutions.

`mip_gap` Relative MIP gap target (stops when `|best-bound - best-incumbent| / |best-incumbent| <= mip_gap`). Lower values force deeper search. Default is `1e-4`.

`mip_gap_abs` Absolute MIP gap target (stops when `|best-bound - best-incumbent| <= mip_gap_abs`). Lower values force deeper search. Default is `1e-10`.

`mip_focus` Search focus: `0` (balance), `1` (find feasible solutions), `2` (prove optimality), `3` (bound improvement). Default is `1`.

`heuristics` Heuristics effort in `[0,1]` where higher values spend more time on heuristics. Default is `0.2`.

`cuts` Cut aggressiveness: `-1` (automatic), `0` (off), `1` (conservative), `2` (aggressive), `3` (very aggressive). Default is `2`.

`presolve` Presolve aggressiveness: `-1` (automatic), `0` (off), `1` (conservative), `2` (aggressive). Default is `2`.

Details

Currently, this method does not return multiple vectors. This will be improved in a later version. If you want this functionality now, use the hacked-up method `gurobi_multiple_designs`.

Value

A list object which houses the results from Gurobi. Depending on the `gurobi_parms`, the data within will be different. The most relevant tags are `x` for the best found solution and `objval` for the object

Author(s)

Adam Kapelner

Examples

```
## Not run:
if ("gurobi" %in% loadedNamespaces()) {
  set.seed(1)
  X = matrix(rnorm(12), nrow = 6)
  gobj = initGurobiNumericalOptimizationExperimentalDesignObject(
    X,
    r = 2,
    num_cores = 1,
    initial_time_limit_sec = 5,
    verbose = FALSE
  )
  gobj$n
}

## End(Not run)
```

initKarpExperimentalDesignObject
Begin Karp Search

Description

This method creates an object of type karp_experimental_design and will immediately initiate a search through allocation space. Note that the Karp search only works for one covariate (i.e. $p=1$) and the objective "abs_sum_diff".

Usage

```
initKarpExperimentalDesignObject(
  X,
  wait = FALSE,
  balanced = TRUE,
  start = TRUE,
  verbose = TRUE
)
```

Arguments

X	The design matrix with n rows (one for each subject) and p columns (one for each measurement on the subject). This is the design matrix you wish to search for a more karp design.
wait	Should the R terminal hang until all max_designs vectors are found? The deafult is FALSE.
balanced	Should the final vector be balanced? Default and recommended is TRUE.
start	Should we start searching immediately (default is TRUE).
verbose	Should the algorithm emit progress output? Default is TRUE.

Value

An object of type karp_experimental_design_search which can be further operated upon

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(10), nrow = 10)
kobj = initKarpExperimentalDesignObject(
  X,
  start = TRUE,
```

```
  wait = TRUE,
  balanced = TRUE,
  verbose = FALSE
)
kobj

## End(Not run)
```

`initOptimalExperimentalDesignObject` *Begin a Search for the Optimal Solution*

Description

This method creates an object of type optimal_experimental_design and will immediately initiate a search through allocation space. Since this search takes exponential time, for most machines, this method is futile beyond 28 samples. You've been warned! For debugging, you can use set `num_cores = 1` to be assured of deterministic output.

Usage

```
initOptimalExperimentalDesignObject(
  X = NULL,
  objective = "mahal_dist",
  Kgram = NULL,
  wait = FALSE,
  start = TRUE,
  num_cores = 1,
  verbose = TRUE,
  use_safe_inverse = FALSE
)
```

Arguments

`X`	The design matrix with n rows (one for each subject) and p columns (one for each measurement on the subject). This is the design matrix you wish to search for a more optimal design.
`objective`	The objective function to use when searching design space. This is a string with valid values `"mahal_dist"` (the default), `"abs_sum_diff"` or `"kernel"`.
`Kgram`	If the `objective = kernel`, this argument is required to be an `n x n` matrix whose entries are the evaluation of the kernel function between subject i and subject j. Default is `NULL`.
`wait`	Should the `R` terminal hang until all `max_designs` vectors are found? The deafult is `FALSE`.
`start`	Should we start searching immediately (default is `TRUE`).
`num_cores`	The number of CPU cores you wish to use during the search. The default is `1`.

verbose Should the algorithm emit progress output? Default is TRUE.

use_safe_inverse
Should a regularized inverse be used for the Mahalanobis objective? Default is FALSE.

Value

An object of type optimal_experimental_design_search which can be further operated upon

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(12), nrow = 6)
obj = initOptimalExperimentalDesignObject(
  X,
  objective = "abs_sum_diff",
  num_cores = 1,
  start = TRUE,
  wait = TRUE,
  verbose = FALSE
)
obj

## End(Not run)
```

---

initRerandomizationExperimentalDesignObject

Begin a Rerandomization Search

Description

This method creates an object of type rerandomization_experimental_design and will immediately initiate a search through allocation space for forced-balance designs. For debugging, you can use set the seed parameter and num_cores = 1 to be assured of deterministic output.

Usage

```
initRerandomizationExperimentalDesignObject(
  X = NULL,
  obj_val_cutoff_to_include,
  max_designs = 1000,
  objective = "mahal_dist",
  Kgram = NULL,
```

```
  wait = FALSE,
  start = TRUE,
  num_cores = 1,
  seed = NULL,
  verbose = TRUE,
  use_safe_inverse = FALSE
)
```

Arguments

X	The design matrix with n rows (one for each subject) and p columns (one for each measurement on the subject). This is the design matrix you wish to search for a more optimal design.
obj_val_cutoff_to_include	Only allocation vectors with objective values lower than this threshold will be returned. If the cutoff is infinity, you are doing BCRD and you should use the complete_randomization_with_forced_balanced function instead.
max_designs	The maximum number of designs to be returned. Default is 10,000. Make this large so you can search however long you wish as the search can be stopped at any time by using the stopSearch method
objective	The objective function to use when searching design space. This is a string with valid values "mahal_dist" (the default), "abs_sum_diff" or "kernel".
Kgram	If the objective = kernel, this argument is required to be an n x n matrix whose entries are the evaluation of the kernel function between subject i and subject j. Default is NULL.
wait	Should the R terminal hang until all max_designs vectors are found? The default is FALSE.
start	Should we start searching immediately (default is TRUE).
num_cores	The number of CPU cores you wish to use during the search. The default is 1.
seed	The set to set for deterministic output. This should only be set if num_cores = 1 otherwise the output will not be deterministic. Default is NULL for no seed set.
verbose	Should the algorithm emit progress output? Default is TRUE.
use_safe_inverse	Should a regularized inverse be used for the Mahalanobis objective? Default is FALSE.

Value

An object of type rerandomization_experimental_design_search which can be further operated upon.

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(20), nrow = 10)
obj = initRerandomizationExperimentalDesignObject(
  X,
  max_designs = 5,
  num_cores = 1,
  objective = "abs_sum_diff",
  obj_val_cutoff_to_include = Inf,
  start = TRUE,
  wait = TRUE,
  verbose = FALSE
)
obj

## End(Not run)
```

---

optimize_asymmetric_treatment_assignment

Compute Optimal Number of Treatments/Controls

Description

Given a total budget and asymmetric treatment and control costs, calculate the number of treatments and controls that optimize the variance of the estimator. The number of treatments is rounded up by default.

Usage

```
optimize_asymmetric_treatment_assignment(
  c_treatment = NULL,
  c_control = NULL,
  c_total_max = NULL,
  n = NULL
)
```

Arguments

c_treatment	The cost of a treatment assignment. Default is NULL for symmetric costs.
c_control	The cost of a control assignment. Default is NULL for symmetric costs.
c_total_max	The total cost constraint of any allocation. Either this or n must be specified. Default is NULL.
n	The total cost constraint as specified by the total number of subjects. Either this or c_total must be specified. Default is NULL.

Value

A list with three keys: n, nT, nC plus specified arguments

Author(s)

Adam Kapelner

Examples

```
## Not run:
optimize_asymmetric_treatment_assignment(n = 100)
optimize_asymmetric_treatment_assignment(n = 100, c_treatment = 2, c_control = 1)
optimize_asymmetric_treatment_assignment(c_total_max = 50, c_treatment = 2, c_control = 1)

## End(Not run)
```

---

plot.greedy_experimental_design_search

Plots a summary of a greedy search object object

Description

Plots a summary of a greedy search object object

Usage

```
## S3 method for class 'greedy_experimental_design_search'
plot(x, ...)
```

Arguments

x	The greedy search object object to be summarized in the plot
...	Other parameters to pass to the default plot function

Value

An array of order statistics from plot_obj_val_order_statistic as a list element

Author(s)

Adam Kapelner

plot.greedy_multiple_kernel_experimental_design *Plots a summary of a* greedy_multiple_kernel_experimental_design *object*

Description

Plots a summary of a greedy_multiple_kernel_experimental_design object

Usage

```
## S3 method for class 'greedy_multiple_kernel_experimental_design'
plot(x, ...)
```

Arguments

x	The greedy_multiple_kernel_experimental_design object to be plotted
...	Optional arguments for the plot function

Author(s)

Adam Kapelner

plot_obj_val_by_iter *Plots the objective value by iteration*

Description

Plots the objective value by iteration

Usage

```
plot_obj_val_by_iter(res, runs = NULL)
```

Arguments

res	Results from a greedy search object
runs	A vector of run indices you would like to see plotted (default is to plot the first up to 9)

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(20), nrow = 10)
ged = initGreedyExperimentalDesignObject(
  X,
  max_designs = 5,
  num_cores = 1,
  diagnostics = TRUE,
  objective = "abs_sum_diff",
  start = TRUE,
  wait = TRUE,
  verbose = FALSE
)
res = resultsGreedySearch(ged, max_vectors = 2)
plot_obj_val_by_iter(res)

## End(Not run)
```

---

plot_obj_val_order_statistic

Plots an order statistic of the object value as a function of number of searches

---

Description

Plots an order statistic of the object value as a function of number of searches

Usage

```
plot_obj_val_order_statistic(
  obj,
  order_stat = 1,
  skip_every = 5,
  type = "o",
  ...
)
```

Arguments

obj	The greedy search object object whose search history is to be visualized
order_stat	The order statistic that you wish to plot. The default is 1 for the minimum.
skip_every	Plot every nth point. This makes the plot generate much more quickly. The default is 5.
type	The type parameter for plot.
...	Other arguments to be passed to the plot function.

Value

An array of order statistics as a list element

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(20), nrow = 10)
ged = initGreedyExperimentalDesignObject(
  X,
  max_designs = 5,
  num_cores = 1,
  objective = "abs_sum_diff",
  start = TRUE,
  wait = TRUE,
  verbose = FALSE
)
plot_obj_val_order_statistic(ged, order_stat = 1, skip_every = 1)

## End(Not run)
```

---

print.binary_match_structure

Prints a summary of a `binary_match_structure` *object*

Description

Prints a summary of a `binary_match_structure` object

Usage

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

Arguments

`x`	The `binary_match_structure` object to be summarized in the console
`...`	Other parameters to pass to the default print function

Author(s)

Adam Kapelner

print.binary_then_greedy_experimental_design

Prints a summary of a `binary_then_greedy_experimental_design` *object*

Description

Prints a summary of a `binary_then_greedy_experimental_design` object

Usage

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

Arguments

`x`	The `binary_then_greedy_experimental_design` object to be summarized in the console
`...`	Other parameters to pass to the default print function

Author(s)

Adam Kapelner

print.binary_then_rerandomization_experimental_design

Prints a summary of a `binary_then_rerandomization_experimental_design` *object*

Description

Prints a summary of a `binary_then_rerandomization_experimental_design` object

Usage

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

Arguments

`x`	The `binary_then_rerandomization_experimental_design` object to be summarized in the console
`...`	Other parameters to pass to the default print function

Author(s)

Adam Kapelner

print.greedy_experimental_design_search

Prints a summary of a `greedy_experimental_design_search` *object*

Description

Prints a summary of a `greedy_experimental_design_search` object

Usage

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

Arguments

`x`	The `greedy_experimental_design_search` object to be summarized in the console
`...`	Other parameters to pass to the default print function

Author(s)

Adam Kapelner

print.greedy_multiple_kernel_experimental_design

Prints a summary of a `greedy_multiple_kernel_experimental_design` *object*

Description

Prints a summary of a `greedy_multiple_kernel_experimental_design` object

Usage

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

Arguments

`x`	The `greedy_multiple_kernel_experimental_design` object to be printed in the console
`...`	Optional arguments for the `print` function

Author(s)

Adam Kapelner

print.karp_experimental_design_search *Prints a summary of a* karp_experimental_design_search *object*

Description

Prints a summary of a karp_experimental_design_search object

Usage

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

Arguments

x	The karp_experimental_design_search object to be summarized in the console
...	Other parameters to pass to the default print function

Author(s)

Adam Kapelner

print.optimal_experimental_design_search *Prints a summary of a* optimal_experimental_design_search *object*

Description

Prints a summary of a optimal_experimental_design_search object

Usage

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

Arguments

x	The optimal_experimental_design_search object to be summarized in the console
...	Other parameters to pass to the default print function

Author(s)

Adam Kapelner

print.pairwise_matching_experimental_design_search

Prints a summary of a `pairwise_matching_experimental_design_search` *object*

Description

Prints a summary of a `pairwise_matching_experimental_design_search` object

Usage

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

Arguments

`x`	The `pairwise_matching_experimental_design_search` object to be summarized in the console
`...`	Other parameters to pass to the default print function

Author(s)

Adam Kapelner

print.rerandomization_experimental_design_search

Prints a summary of a `rerandomization_experimental_design_search` *object*

Description

Prints a summary of a `rerandomization_experimental_design_search` object

Usage

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

Arguments

`x`	The `rerandomization_experimental_design_search` object to be summarized in the console
`...`	Other parameters to pass to the default print function

Author(s)

Adam Kapelner

`resultsBinaryMatchSearch` *Binary Pair Match Search*

Description

Returns the results (thus far) of the binary pair match design search

Usage

```
resultsBinaryMatchSearch(obj, form = "one_zero")
```

Arguments

`obj`	The `pairwise_matching_experimental_design_search` object that is currently running the search
`form`	Which form should the assignments be in? The default is `one_zero` for 1/0's or `pos_one_min_one` for +1/-1's.

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(16), nrow = 8)
bms = computeBinaryMatchStructure(X)
bm = initBinaryMatchExperimentalDesignSearchObject(
  bms,
  max_designs = 4,
  num_cores = 1,
  start = TRUE,
  wait = TRUE,
  seed = 1,
  verbose = FALSE
)
res = resultsBinaryMatchSearch(bm, form = "one_zero")
dim(res)

## End(Not run)
```

`resultsBinaryMatchThenGreedySearch` *Returns unique allocation vectors that are binary matched*

Description

Returns unique allocation vectors that are binary matched

Usage

```
resultsBinaryMatchThenGreedySearch(
  obj,
  num_vectors = NULL,
  compute_obj_vals = FALSE,
  form = "one_zero",
  use_safe_inverse = FALSE
)
```

Arguments

`obj`	The `binary_then_greedy_experimental_design` object where the pairs are computed.
`num_vectors`	How many random allocation vectors you wish to return. The default is `NULL` indicating you want all of them.
`compute_obj_vals`	Should we compute all the objective values for each allocation? Default is `FALSE`.
`form`	Which form should it be in? The default is `one_zero` for 1/0's or `pos_one_min_one` for +1/-1's.
`use_safe_inverse`	Should a regularized inverse be used for the Mahalanobis objective? Default is `FALSE`.

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(16), nrow = 8)
obj = initBinaryMatchFollowedByGreedyExperimentalDesignSearchObject(
  X,
  max_designs = 4,
  num_cores = 1,
  objective = "abs_sum_diff",
```

```
  start = TRUE,
  wait = TRUE,
  verbose = FALSE
)
res = resultsBinaryMatchThenGreedySearch(obj, num_vectors = 3, form = "one_zero")
dim(res$indicTs)

## End(Not run)
```

`resultsBinaryMatchThenRerandomizationSearch` *Returns unique allocation vectors that are binary matched*

Description

Returns unique allocation vectors that are binary matched

Usage

```
resultsBinaryMatchThenRerandomizationSearch(
  obj,
  num_vectors = NULL,
  compute_obj_vals = FALSE,
  form = "one_zero",
  use_safe_inverse = FALSE
)
```

Arguments

`obj`	The `binary_then_greedy_experimental_design` object where the pairs are computed.
`num_vectors`	How many random allocation vectors you wish to return. The default is `NULL` indicating you want all of them.
`compute_obj_vals`	Should we compute all the objective values for each allocation? Default is `FALSE`.
`form`	Which form should it be in? The default is `one_zero` for 1/0's or `pos_one_min_one` for +1/-1's.
`use_safe_inverse`	Should a regularized inverse be used for the Mahalanobis objective? Default is `FALSE`.

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(16), nrow = 8)
obj = initBinaryMatchFollowedByRerandomizationDesignSearchObject(
  X,
  max_designs = 4,
  num_cores = 1,
  objective = "abs_sum_diff",
  obj_val_cutoff_to_include = Inf,
  start = TRUE,
  wait = TRUE,
  verbose = FALSE
)
res = resultsBinaryMatchThenRerandomizationSearch(obj, num_vectors = 3, form = "one_zero")
dim(res$indicTs)

## End(Not run)
```

`resultsGreedySearch` *Returns the results (thus far) of the greedy design search*

Description

Returns the results (thus far) of the greedy design search

Usage

```
resultsGreedySearch(obj, max_vectors = 9, form = "one_zero")
```

Arguments

`obj`	The `greedy_experimental_design` object that is currently running the search
`max_vectors`	The number of design vectors you wish to return. `NULL` returns all of them. This is not recommended as returning over 1,000 vectors is time-intensive. The default is 9.
`form`	Which form should it be in? The default is `one_zero` for 1/0's or `pos_one_min_one` for +1/-1's.

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(20), nrow = 10)
ged = initGreedyExperimentalDesignObject(
  X,
  max_designs = 5,
  num_cores = 1,
  objective = "abs_sum_diff",
  start = TRUE,
  wait = TRUE,
  verbose = FALSE
)
res = resultsGreedySearch(ged, max_vectors = 2)
res$obj_vals

## End(Not run)
```

---

resultsGurobiNumericalOptimizeSearch

Query the Gurobi Results

Description

Returns the results (thus far) of the Gurobi numerical optimization design search

Usage

```
resultsGurobiNumericalOptimizeSearch(obj)
```

Arguments

obj	The gurobi_numerical_optimization_experimental_design_search object that is currently running the search

Author(s)

Adam Kapelner

Examples

```
## Not run:
if ("gurobi" %in% loadedNamespaces()) {
  set.seed(1)
  X = matrix(rnorm(12), nrow = 6)
  gobj = initGurobiNumericalOptimizationExperimentalDesignObject(
    X,
    r = 2,
    num_cores = 1,
```

```
    initial_time_limit_sec = 5,
    verbose = FALSE
  )
  res = resultsGurobiNumericalOptimizeSearch(gobj)
  res$obj_vals
}

## End(Not run)
```

`resultsKarpSearch` *Returns the results (thus far) of the karp design search*

Description

Returns the results (thus far) of the karp design search

Usage

```
resultsKarpSearch(obj)
```

Arguments

`obj`	The `karp_experimental_design` object that is currently running the search

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(10), nrow = 10)
kobj = initKarpExperimentalDesignObject(
  X,
  start = TRUE,
  wait = TRUE,
  balanced = TRUE,
  verbose = FALSE
)
res = resultsKarpSearch(kobj)
res$obj_val

## End(Not run)
```

`resultsMultipleKernelGreedySearch` *Returns the results of a greedy multiple kernel search*

Description

Returns the results of a greedy multiple kernel search

Usage

```
resultsMultipleKernelGreedySearch(obj, max_vectors = NULL, form = "one_zero")
```

Arguments

`obj`	The `greedy_multiple_kernel_experimental_design` object where the search was run.
`max_vectors`	How many random allocation vectors you wish to return. The default is `NULL` indicating you want all of them.
`form`	Which form should it be in? The default is `one_zero` for 1/0's or `pos_one_min_one` for +1/-1's.

Author(s)

Adam Kapelner

Examples

```
 ## Not run:
library(MASS)
data(Boston)
 #pretend the Boston data was an experiment setting
#first pull out the covariates
 X = Boston[, 1 : 13]
 #begin the greedy design search
mk = initGreedyMultipleKernelExperimentalDesignObject(X,
max_designs = 100, num_cores = 3, kernel_names = c("mahalanobis", "gaussian"))
#wait
res = resultsMultipleKernelGreedySearch(mk, max_vectors = 2, form = "one_zero")
res$obj_vals

## End(Not run)
```

`resultsOptimalSearch` *Returns the results (thus far) of the optimal design search*

Description

Returns the results (thus far) of the optimal design search

Usage

```
resultsOptimalSearch(obj, num_vectors = 2, form = "one_zero")
```

Arguments

`obj`	The `optimal_experimental_design` object that is currently running the search
`num_vectors`	How many allocation vectors you wish to return. The default is 1 meaning the best vector. If `Inf`, it means all vectors.
`form`	Which form should it be in? The default is `one_zero` for 1/0's or `pos_one_min_one` for +1/-1's.

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(12), nrow = 6)
obj = initOptimalExperimentalDesignObject(
  X,
  objective = "abs_sum_diff",
  num_cores = 1,
  start = TRUE,
  wait = TRUE,
  verbose = FALSE
)
res = resultsOptimalSearch(obj, num_vectors = 2, form = "one_zero")
res$opt_obj_val

## End(Not run)
```

`resultsRerandomizationSearch` *Returns the results (thus far) of the rerandomization design search*

Description

Returns the results (thus far) of the rerandomization design search

Usage

```
resultsRerandomizationSearch(
  obj,
  include_assignments = FALSE,
  form = "one_zero"
)
```

Arguments

`obj`	The `rerandomization_experimental_design` object that is currently running the search
`include_assignments`	Do we include the assignments (takes time) and default is `FALSE`.
`form`	Which form should the assignments be in? The default is `one_zero` for 1/0's or `pos_one_min_one` for +1/-1's.

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(20), nrow = 10)
obj = initRerandomizationExperimentalDesignObject(
  X,
  max_designs = 5,
  num_cores = 1,
  objective = "abs_sum_diff",
  obj_val_cutoff_to_include = Inf,
  start = TRUE,
  wait = TRUE,
  verbose = FALSE
)
res = resultsRerandomizationSearch(obj, include_assignments = TRUE, form = "one_zero")
dim(res$ending_indicTs)

## End(Not run)
```

safe_cov_inverse *Computes a numerically stable inverse of a covariance matrix*

Description

Computes a numerically stable inverse of a covariance matrix

Usage

```
safe_cov_inverse(X, ridge = 1e-08, max_ridge_steps = 6)
```

Arguments

X	The n x p design matrix
ridge	Initial ridge penalty added to the diagonal
max_ridge_steps	Maximum number of ridge escalation attempts

Value

The inverse covariance matrix

Author(s)

Adam Kapelner

Examples

```
## Not run:
X = matrix(rnorm(20), nrow = 10)
Sinv = safe_cov_inverse(X)
dim(Sinv)

## End(Not run)
```

searchTimeElapsed *Returns the amount of time elapsed*

Description

Returns the amount of time elapsed

Usage

```
searchTimeElapsed(obj)
```

Arguments

obj	The experimental_design object that is currently running the search

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(20), nrow = 10)
ged = initGreedyExperimentalDesignObject(
  X,
  max_designs = 1,
  num_cores = 1,
  start = TRUE,
  wait = TRUE,
  objective = "abs_sum_diff",
  verbose = FALSE
)
searchTimeElapsed(ged)

## End(Not run)
```

---

shuffle_cpp_wrap *Shuffles a vector rapidly*

Description

Shuffles a vector rapidly

Usage

```
shuffle_cpp_wrap(w, seed = NA_integer_)
```

Arguments

w	The vector to be shuffled
seed	Optional integer seed; use NA to draw from the system clock

Value

The vector with elements shuffled

Author(s)

Adam Kapelner

Examples

```
## Not run:
shuffle_cpp_wrap(1:5, seed = 1)

## End(Not run)
```

`standardize_data_matrix` *Standardizes the columns of a data matrix.*

Description

Standardizes the columns of a data matrix.

Usage

```
standardize_data_matrix(X)
```

Arguments

`X`	The n x p design matrix

Value

The n x p design matrix with columns standardized

Author(s)

Adam Kapelner

Examples

```
## Not run:
X = matrix(rnorm(12), nrow = 6)
Xstd = standardize_data_matrix(X)
colMeans(Xstd)

## End(Not run)
```

startSearch *Starts the parallelized greedy design search.*

Description

Once begun, this function cannot be run again.

Usage

```
startSearch(obj)
```

Arguments

obj	The experimental_design object that will be running the search

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(20), nrow = 10)
ged = initGreedyExperimentalDesignObject(
  X,
  max_designs = 3,
  num_cores = 1,
  start = FALSE,
  wait = FALSE,
  objective = "abs_sum_diff",
  verbose = FALSE
)
startSearch(ged)
stopSearch(ged)

## End(Not run)
```

stopSearch *Stops the parallelized greedy design search.*

Description

Once stopped, it cannot be restarted.

Usage

```
stopSearch(obj)
```

Arguments

obj	The experimental_design object that is currently running the search

Author(s)

Adam Kapelner

Examples

```
## Not run:
set.seed(1)
X = matrix(rnorm(20), nrow = 10)
ged = initGreedyExperimentalDesignObject(
  X,
  max_designs = 3,
  num_cores = 1,
  start = TRUE,
  wait = FALSE,
  objective = "abs_sum_diff",
  verbose = FALSE
)
stopSearch(ged)

## End(Not run)
```

---

summary.binary_match_structure

Prints a summary of a binary_match_structure *object*

Description

Prints a summary of a binary_match_structure object

Usage

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

Arguments

object	The binary_match_structure object to be summarized in the console
...	Other parameters to pass to the default summary function

Author(s)

Adam Kapelner

`summary.binary_then_greedy_experimental_design` *Prints a summary of a* `binary_then_greedy_experimental_design` *object*

Description

Prints a summary of a `binary_then_greedy_experimental_design` object

Usage

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

Arguments

`object`	The `binary_then_greedy_experimental_design` object to be summarized in the console
`...`	Other parameters to pass to the default summary function

Author(s)

Adam Kapelner

`summary.binary_then_rerandomization_experimental_design` *Prints a summary of a* `binary_then_rerandomization_experimental_design` *object*

Description

Prints a summary of a `binary_then_rerandomization_experimental_design` object

Usage

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

Arguments

`object`	The `binary_then_rerandomization_experimental_design` object to be summarized in the console
`...`	Other parameters to pass to the default summary function

Author(s)

Adam Kapelner

summary.greedy_experimental_design_search *Prints a summary of a* greedy_experimental_design_search *object*

Description

Prints a summary of a greedy_experimental_design_search object

Usage

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

Arguments

object	The greedy_experimental_design_search object to be summarized in the console
...	Other parameters to pass to the default summary function

Author(s)

Adam Kapelner

summary.greedy_multiple_kernel_experimental_design *Prints a summary of a* greedy_multiple_kernel_experimental_design *object*

Description

Prints a summary of a greedy_multiple_kernel_experimental_design object

Usage

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

Arguments

object	The greedy_multiple_kernel_experimental_design object to be summarized in the console
...	Optional arguments for the summary function

Author(s)

Adam Kapelner

summary.karp_experimental_design_search
Prints a summary of a karp_experimental_design_search *object*

Description

Prints a summary of a karp_experimental_design_search object

Usage

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

Arguments

object	The karp_experimental_design_search object to be summarized in the console
...	Other parameters to pass to the default summary function

Author(s)

Adam Kapelner

summary.optimal_experimental_design_search
Prints a summary of a optimal_experimental_design_search *object*

Description

Prints a summary of a optimal_experimental_design_search object

Usage

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

Arguments

object	The optimal_experimental_design_search object to be summarized in the console
...	Other parameters to pass to the default summary function

Author(s)

Adam Kapelner

summary.pairwise_matching_experimental_design_search

Prints a summary of a `pairwise_matching_experimental_design_search` *object*

Description

Prints a summary of a `pairwise_matching_experimental_design_search` object

Usage

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

Arguments

`object`	The `pairwise_matching_experimental_design_search` object to be summarized in the console
`...`	Other parameters to pass to the default summary function

Author(s)

Adam Kapelner

summary.rerandomization_experimental_design_search

Prints a summary of a `rerandomization_experimental_design_search` *object*

Description

Prints a summary of a `rerandomization_experimental_design_search` object

Usage

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

Arguments

`object`	The `rerandomization_experimental_design_search` object to be summarized in the console
`...`	Other parameters to pass to the default summary function

Author(s)

Adam Kapelner

Index