Package 'fastrerandomize'

December 22, 2025

Title Hardware-Accelerated Rerandomization for Improved Balance

Version 0.3

Description Provides hardware-accelerated tools for performing rerandomization and randomization testing in experimental research. Using a 'JAX' backend, the package enables exact rerandomization inference even for large experiments with hundreds of billions of possible randomizations. Key functionalities include generating pools of acceptable rerandomizations based on covariate balance, conducting exact randomization tests, and performing pre-analysis evaluations to determine optimal rerandomization acceptance thresholds. The package supports various hardware acceleration frameworks including 'CPU', 'CUDA', and 'METAL', making it versatile across accelerated computing environments. This allows researchers to efficiently implement stringent rerandomization designs and conduct valid inference even with large sample sizes. The package is partly based on Jerzak and Goldstein (2023) <doi:10.48550/arXiv.2310.00861>.

URL `https://github.com/cjerzak/fastrerandomize-software`

BugReports `https://github.com/cjerzak/fastrerandomize-software/issues`

Depends R (>= 3.5.0)

License GPL-3

Encoding UTF-8

LazyData false

Imports reticulate, assertthat, utils, stats, graphics

Suggests knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 7.3.3

NeedsCompilation no

Author Fucheng Warren Zhu [aut] (ORCID:
<`https://orcid.org/0009-0001-5692-7572`>),
Aniket Sachin Kamat [aut] (ORCID:
<`https://orcid.org/0009-0003-6411-1084`>),
Connor Jerzak [aut, cre] (ORCID:
<`https://orcid.org/0000-0003-1914-8905`>),

Rebecca Goldstein [aut] (ORCID:
<https://orcid.org/0000-0002-9944-8440>)

Maintainer Connor Jerzak <`connor.jerzak@gmail.com`>

Repository CRAN

Date/Publication 2025-12-22 22:30:02 UTC

Contents

`build_backend` *A function to build the environment for fastrerandomize. Builds a conda environment in which 'JAX' and 'np' are installed. Users can also create a conda environment where 'JAX' and 'np' are installed themselves.*

Description

A function to build the environment for fastrerandomize. Builds a conda environment in which 'JAX' and 'np' are installed. Users can also create a conda environment where 'JAX' and 'np' are installed themselves.

Usage

```
build_backend(conda_env = "fastrerandomize_env", conda = "auto")
```

Arguments

`conda_env`	(default = `"fastrerandomize_env"`) Name of the conda environment in which to place the backends.
`conda`	(default = `auto`) The path to a conda executable. Using `"auto"` allows reticulate to attempt to automatically find an appropriate conda binary.

Value

Invisibly returns NULL; this function is used for its side effects of creating and configuring a conda environment for `fastrerandomize`. This function requires an Internet connection. You can find out a list of conda Python paths via: `Sys.which("python")`

Examples

```
## Not run:
# Create a conda environment named "fastrerandomize_env"
# and install the required Python packages (jax, numpy, etc.)
build_backend(conda_env = "fastrerandomize_env", conda = "auto")

# If you want to specify a particular conda path:
# build_backend(conda_env = "fastrerandomize_env", conda = "/usr/local/bin/conda")

## End(Not run)
```

`check_jax_availability` *Check if 'Python' and 'JAX' are available*

Description

This function checks if 'Python' and 'JAX' can be accessed via ‘reticulate‘. If not, it returns ‘NULL‘ and prints a message suggesting to run ‘build_backend()‘.

Usage

```
check_jax_availability(conda_env = "fastrerandomize_env", conda = "auto")
```

Arguments

`conda_env`	A character string specifying the name of the conda environment. Default is ‘"fastrerandomize_env"‘.
`conda`	The path to a conda executable, or ‘"auto"‘. Default is ‘"auto"‘.

Value

Returns ‘TRUE‘ (invisibly) if both ’Python’ and ’JAX’ are available; otherwise returns ‘NULL‘.

Examples

```
## Not run:
  check_jax_availability()

## End(Not run)
```

`compute_diff_at_tau_for_oneW_R`

Compute potential outcome difference in means for a single assignment under a hypothesized tau in base R

Description

Compute potential outcome difference in means for a single assignment under a hypothesized tau in base R

Usage

```
compute_diff_at_tau_for_oneW_R(Wprime, obsY, obsW, tau)
```

Arguments

`Wprime`	A 0/1 assignment vector for which to compute the diff in means.
`obsY`	Observed outcome vector.
`obsW`	Observed assignment vector.
`tau`	The hypothesized true effect for the shift in outcomes under treatment.

Value

Scalar difference in means for the assignment Wprime.

`diagnose_rerandomization` *Diagnostic map from observed (or targeted) balance to precision and stringency*

Description

Implements the calculations in Theorem 1 and Appendix D of the paper involving: (1) Realized RMSE from an observed Mahalanobis distance M (or SMDs); (2) Ex-ante RMSE when accepting assignments with M < a (equivalently, with acceptance probability q under complete randomization); (3) largest acceptance probability q that attains a user-specified precision goal, provided via an RMSE target or via a power target (alpha, 1-beta, |tau|).

Usage

```
diagnose_rerandomization(
  smd = NULL,
  M = NULL,
  d = NULL,
  n_T,
  n_C,
  sigma = NULL,
  R2 = NULL,
  rmse_goal = NULL,
  tau = NULL,
  alpha = 0.05,
  power = 0.8,
  two_sided = TRUE,
  q_min = 1e-09,
  q_tol = 1e-10
)
```

Arguments

`smd`	Optional numeric vector of standardized mean differences; if supplied, M is computed as sum(smd^2), and d = length(smd).
`M`	Optional scalar Mahalanobis distance M; if provided without ‘smd‘, you must also supply ‘d‘ (the number of covariates used in M).
`d`	Optional integer number of covariates (needed if supplying only ‘M‘).
`n_T`	Integer, number of treated units.
`n_C`	Integer, number of control units.
`sigma`	Optional outcome noise SD (sigma). If ‘NULL‘, absolute RMSEs cannot be formed; dimensionless "per-sigma" factors are still returned.
`R2`	Optional model R^2 for Y ~ X under the linear potential-outcomes model. Must lie in [0,1). If ‘NULL‘, RMSEs that require R^2 are returned as NA, but the "per-sigma" formulas that do not need R^2 are still shown when possible.

`rmse_goal`	Optional numeric target for RMSE (same units as Y). If supplied (with sigma and R2), the largest q achieving this ex-ante goal is returned.
`tau`	Optional effect size |tau| (same units as Y) to back out an RMSE goal via a normal approximation to power.
`alpha`	Size of a two-sided test (default 0.05).
`power`	Desired power 1 - beta (default 0.80). Used only if ‘tau‘ is given.
`two_sided`	Logical; if FALSE, uses a one-sided z-threshold for power inversion.
`q_min`	Lower bound for numerical search over q (default 1e-9).
`q_tol`	Absolute tolerance for q root-finding (default 1e-10).

Details

Realized (conditional) RMSE: with standardized/whitened X and typical orientation,

$$\mathrm{RMSE}_{\text{realized}} \approx \sqrt{\sigma^2\left(\frac{1}{n_T}+\frac{1}{n_C}\right)+\frac{\sigma^2_{\text{Prog}}}{d}\,M} \;=\; \sigma\sqrt{\left(\frac{1}{n_T}+\frac{1}{n_C}\right)+\frac{R^2}{1-R^2}\,\frac{M}{d}}\,,$$

and the conservative upper bound replaces σ^2_{Prog}/d by σ^2_{Prog}.

Ex-ante (design-stage) RMSE under thresholding: with acceptance rule M <= a (acceptance probability q),

$$\mathbb{E}[\mathrm{MSE} \mid M \le a] = \left(\frac{1}{n_T}+\frac{1}{n_C}\right)\left(\sigma^2 + v_a(d)\,\sigma^2_{\text{Prog}}\right),$$

where $v_a(d) = \Pr(\chi^2_{d+2} \le c)/\Pr(\chi^2_d \le c)$ and $c = a/(\frac{1}{n_T}+\frac{1}{n_C})$. Since $q = \Pr(\chi^2_d \le c)$, we can parameterize by q: $v(q;d) = \Pr(\chi^2_{d+2} \le \chi^2_{d;q})/q$, with $\chi^2_{d;q}$ the q-th quantile.

Power inversion (Appendix D): for two-sided size α and power $1-\beta$, a normal approximation suggests the RMSE goal $|\tau|/(z_{1-\alpha/2}+z_{1-\beta})$.

Value

A list of class ‘"fastrerandomize_diagnostic"‘ with elements:

- `inputs`: Echo of parsed inputs and derived quantities (M, d, S=1/n_T+1/n_C).
- `realized`: `rmse_factor` (dimensionless, per sigma), `rmse`, and conservative `rmse_upper_factor`, `rmse_upper`.
- `power_check`: If `tau`, `alpha`, `power`, `sigma`, `R2` are given, includes `z_needed`, `z_realized`, and `already_sufficient`.
- `recommendation`: If a target is supplied (via `rmse_goal` or `tau`), returns `q_star`, `a_star`, `v_star`, `rmse_exante`, `expected_M_accepted`, and `expected_draws_per_accept = 1/q_star`.

Examples

```
# Example 1: observed SMDs, realized precision only (dimensionless factors)
smd <- c(0.10, -0.05, 0.08, 0.02)  # standardized mean differences
out1 <- diagnose_rerandomization(smd = smd, n_T = 100, n_C = 100)
print(out1)
```

```
# Example 2: same, but supply sigma and R^2 for absolute RMSE
out2 <- diagnose_rerandomization(smd = smd, n_T = 100, n_C = 100, sigma = 1.2, R2 = 0.4)

# Example 3: choose q to hit a power target (two-sided alpha=.05, 80% power, |tau|=0.2)
out3 <- diagnose_rerandomization(smd = smd, n_T = 100, n_C = 100, sigma = 1.2, R2 = 0.4,
                tau = 0.2, alpha = 0.05, power = 0.80)

# Analyze rerandomization recommendation given contextual factors
out3$recommendation

# Example 4: choose q to hit an absolute RMSE goal directly
out4 <- diagnose_rerandomization(M = sum(smd^2), d = length(smd), n_T = 100, n_C = 100,
                sigma = 1.2, R2 = 0.4, rmse_goal = 0.25)
```

`diff_in_means_R` *Simple difference in means in base R*

Description

Simple difference in means in base R

Usage

```
diff_in_means_R(Y, W)
```

Arguments

`Y`	Numeric outcome vector.
`W`	0/1 treatment assignment vector.

Value

Scalar difference in means: mean(Y|W=1) - mean(Y|W=0).

`fastrerandomize_class` *Constructor for fastrerandomize randomizations*

Description

Create an S3 object of class `fastrerandomize_randomizations` that stores the randomizations (and optionally balance statistics) generated by functions such as `generate_randomizations`.

Usage

```
fastrerandomize_class(
  randomizations,
  balance = NULL,
  fastrr_env = NULL,
  call = NULL
)
```

Arguments

`randomizations`	A matrix or array where each row (or slice) represents one randomization.
`balance`	A numeric vector or similar object holding balance statistics for each randomization, or `NULL` if not applicable.
`fastrr_env`	Associated `fastrr_env` environment.
`call`	The function call, if you wish to store it for reference (optional).

Value

An object of class `fastrerandomize_randomizations`.

`fastrerandomize_test` *Constructor for fastrerandomize randomization test objects*

Description

Constructor for fastrerandomize randomization test objects

Usage

```
fastrerandomize_test(p_value, FI, tau_obs, fastrr_env = NULL, call = NULL, ...)
```

Arguments

`p_value`	A numeric value representing the p-value of the test.
`FI`	A numeric vector (length 2) representing the fiducial interval, or `NULL` if not requested.
`tau_obs`	A numeric value (or vector) representing the estimated treatment effect.
`fastrr_env`	Associated ‘fastrr_env‘ environment.
`call`	An optional function call, stored for reference.
`...`	Other slots you may want to store (e.g. additional diagnostics).

Value

An object of class `fastrerandomize_test`.

`fast_distance` *JAX-accelerated distance calculations*

Description

Compute pairwise distances between the rows of one matrix `A` or two matrices `A` and `B`, using JAX-backed, JIT-compiled kernels. Supports common metrics: Euclidean, squared Euclidean, Manhattan, Chebyshev, Cosine, Minkowski (with optional feature weights), and Mahalanobis (with full or diagonal inverse covariance).

The function automatically batches computations to avoid excessive device memory use.

Usage

```
fast_distance(
  A,
  B = NULL,
 metric = c("euclidean", "sqeuclidean", "manhattan", "chebyshev", "cosine", "minkowski",
    "mahalanobis"),
  p = 2,
  weights = NULL,
  cov_inv = NULL,
  approximate_inv = TRUE,
  squared = FALSE,
  row_batch_size = NULL,
  as_dist = FALSE,
  return_type = "R",
  verbose = FALSE,
  conda_env = "fastrerandomize_env",
  conda_env_required = TRUE
)
```

Arguments

`A`	A numeric matrix with rows as observations and columns as features.
`B`	Optional numeric matrix with the same number of columns as `A`. If `NULL`, distances are computed within `A` (i.e., $n \times n$).
`metric`	Character; one of `"euclidean"`, `"sqeuclidean"`, `"manhattan"`, `"chebyshev"`, `"cosine"`, `"minkowski"`, `"mahalanobis"`. Default is `"euclidean"`.
`p`	Numeric order for Minkowski distance (must be > 0). Default is `2`.
`weights`	Optional numeric vector of length `ncol(A)` with nonnegative feature weights. Used for `"minkowski"` and `"manhattan"` (the latter is equivalent to Minkowski with `p = 1`).
`cov_inv`	Optional inverse covariance matrix (p x p) for Mahalanobis (ignored if `approximate_inv = TRUE`). If not supplied and `approximate_inv = FALSE`, it is estimated from `rbind(A, B)` and inverted in JAX.

`approximate_inv`	Logical; if `TRUE` and `metric = "mahalanobis"`, uses a diagonal inverse (reciprocal variances) for speed and robustness. Default `TRUE`.
`squared`	Logical; if `TRUE`, return squared distances when supported (`"euclidean"` and `"mahalanobis"`). Ignored for other metrics. Default `FALSE`.
`row_batch_size`	Optional integer; number of rows of `A` to process per batch. If `NULL`, a safe size is chosen automatically.
`as_dist`	Logical; if `TRUE` and `B` is `NULL`, return a base `dist` object (for symmetric metrics). Default `FALSE`.
`return_type`	Either `"R"` (convert to base R matrix/`dist`) or `"jax"` (return a JAX array). Default `"R"`.
`verbose`	Logical; print batching progress. Default `FALSE`.
`conda_env`	Character; conda environment name used by `reticulate`. Default `"fastrerandomize_env"`.
`conda_env_required`	Logical; whether the specified conda environment must be used. Default `TRUE`.

Details

- **Mahalanobis**: with `approximate_inv = TRUE`, the diagonal of the pooled covariance is used (variance stabilizer); otherwise a full inverse covariance is used. - **Weighted distances**: supply `weights` (length `p`) for `"minkowski"` and `"manhattan"` (the latter uses `p = 1`). - Computations run in float32 and are JIT-compiled with JAX; where applicable, GPU/Metal/CPU device selection follows your existing backend.

Value

An $n \times m$ distance matrix in the format specified by `return_type`. If `as_dist = TRUE` and `B = NULL` (symmetric case), returns a `dist` object.

See Also

`dist`

Examples

```
## Not run:
# Simple Euclidean within-matrix distances (returns an n x n matrix)
X <- matrix(rnorm(50 * 8), 50, 8)
D <- fast_distance(X, metric = "euclidean")

# Cosine distance between two sets
A <- matrix(rnorm(100 * 16), 100, 16)
B <- matrix(rnorm(120 * 16), 120, 16)
Dcos <- fast_distance(A, B, metric = "cosine")

# Minkowski with p = 3 and feature weights
w <- runif(ncol(A))
Dm3 <- fast_distance(A, B, metric = "minkowski", p = 3, weights = w)
```

```
# Mahalanobis (diagonal approx, fast & robust)
Dmah_diag <- fast_distance(X, metric = "mahalanobis", approximate_inv = TRUE)

# Mahalanobis with full inverse (computed internally)
Dmah_full <- fast_distance(X, metric = "mahalanobis", approximate_inv = FALSE)

# Return a base R 'dist' object
D_dist <- fast_distance(X, metric = "euclidean", as_dist = TRUE)

## End(Not run)
```

`find_fiducial_interval_R`

Fiducial interval logic in base R, for randomization test

Description

Fiducial interval logic in base R, for randomization test

Usage

```
find_fiducial_interval_R(
  obsW,
  obsY,
  allW,
  tau_obs,
  alpha = 0.05,
  c_initial = 2,
  n_search_attempts = 500
)
```

Arguments

`obsW`	Observed assignment (0/1).
`obsY`	Observed outcome.
`allW`	Matrix of candidate random assignments (rows = assignments).
`tau_obs`	Observed difference in means with obsW, obsY.
`alpha`	Significance level (default 0.05).
`c_initial`	A numeric step scale (default 2).
`n_search_attempts`	Number of bracket search attempts (default 500).

Value

2-element numeric vector [lower, upper] or [NA, NA] if none accepted.

`generate_randomizations`

Generate randomizations for a rerandomization-based experimental design

---

Description

This function generates randomizations for experimental design using either exact enumeration or Monte Carlo sampling methods. It provides a unified interface to both approaches while handling memory and computational constraints appropriately.

Usage

```
generate_randomizations(
  n_units,
  n_treated,
  X = NULL,
  randomization_accept_prob,
  threshold_func = NULL,
  max_draws = 10^6,
  batch_size = 1000,
  randomization_type = "monte_carlo",
  approximate_inv = TRUE,
  file = NULL,
  return_type = "R",
  verbose = TRUE,
  conda_env = "fastrerandomize_env",
  conda_env_required = TRUE
)
```

Arguments

`n_units`	An integer specifying the total number of experimental units.
`n_treated`	An integer specifying the number of units to be assigned to treatment.
`X`	A numeric matrix of covariates used for balance checking. Cannot be `NULL`.
`randomization_accept_prob`	A numeric value between 0 and 1 specifying the probability threshold for accepting randomizations based on balance.
`threshold_func`	A 'JAX' function that computes a balance measure for each randomization. Only used for Monte Carlo sampling.
`max_draws`	An integer specifying the maximum number of randomizations to draw in Monte Carlo sampling.
`batch_size`	An integer specifying batch size for Monte Carlo processing.
`randomization_type`	A string specifying the type of randomization: either `"exact"` or `"monte_carlo"`.

`approximate_inv`	A logical value indicating whether to use an approximate inverse (diagonal of the covariance matrix) instead of the full matrix inverse when computing balance metrics. This can speed up computations for high-dimensional covariates. Default is `TRUE`.
`file`	A string specifying where to save candidate randomizations (if saving, not returning).
`return_type`	A string specifying the format of the returned randomizations and balance measures. Allowed values are `"R"` for base R objects (e.g., `matrix`, `numeric`) or `"jax"` for 'JAX' arrays. Default is `"R"`.
`verbose`	A logical value indicating whether to print progress information. Default is `TRUE`.
`conda_env`	A character string specifying the name of the conda environment to use via `reticulate`. Default is `"fastrerandomize_env"`.
`conda_env_required`	A logical indicating whether the specified conda environment must be strictly used. If `TRUE`, an error is thrown if the environment is not found. Default is `TRUE`.

Details

The function supports two methods of generating randomizations:

1. Exact enumeration: Generates all possible randomizations (memory intensive but exact).
2. Monte Carlo sampling: Generates randomizations through sampling (more memory efficient).

For large problems (e.g., X with >20 rows), Monte Carlo sampling is recommended.

Value

Returns an S3 object with slots:

- `assignments` An array where each row represents one possible treatment assignment vector containing the accepted randomizations.
- `balance_measures` A numeric vector containing the balance measure for each corresponding randomization.
- `fastrr_env` The fastrerandomize environment.
- `file_output` If file is specified, results are saved to the given file path instead of being returned.

See Also

`generate_randomizations_exact` for the exact enumeration method. `generate_randomizations_mc` for the Monte Carlo sampling method.

Examples

```
## Not run:
# Generate synthetic data
X <- matrix(rnorm(20*5), 20, 5)

# Generate randomizations using exact enumeration
RandomizationSet_Exact <- generate_randomizations(
               n_units = nrow(X),
               n_treated = round(nrow(X)/2),
               X = X,
               randomization_accept_prob=0.1,
               randomization_type="exact")

# Generate randomizations using Monte Carlo sampling
RandomizationSet_MC <- generate_randomizations(
               n_units = nrow(X),
               n_treated = round(nrow(X)/2),
               X = X,
               randomization_accept_prob = 0.1,
               randomization_type = "monte_carlo",
               max_draws = 100000,
               batch_size = 1000)

## End(Not run)
```

`generate_randomizations_exact`

Generate Complete Randomizations with Optional Balance Constraints

Description

Generates all possible treatment assignments for a completely randomized experiment, optionally filtering them based on covariate balance criteria. The function can generate either all possible randomizations or a subset that meets specified balance thresholds using Hotelling's T-squared statistic.

Usage

```
generate_randomizations_exact(
  n_units,
  n_treated,
  X = NULL,
  randomization_accept_prob = 1,
  approximate_inv = TRUE,
  threshold_func = NULL,
  verbose = TRUE,
  conda_env = "fastrerandomize_env",
  conda_env_required = TRUE
)
```

Arguments

`n_units`	An integer specifying the total number of experimental units
`n_treated`	An integer specifying the number of units to be assigned to treatment
`X`	A numeric matrix of covariates where rows represent units and columns represent different covariates. Default is `NULL`, in which case all possible randomizations are returned without balance filtering.
`randomization_accept_prob`	A numeric value between 0 and 1 specifying the quantile threshold for accepting randomizations based on balance statistics. Default is 1 (accept all randomizations).
`approximate_inv`	A logical value indicating whether to use an approximate inverse (diagonal of the covariance matrix) instead of the full matrix inverse when computing balance metrics. This can speed up computations for high-dimensional covariates. Default is `TRUE`.
`threshold_func`	A function that calculates balance statistics for candidate randomizations. Default is `VectorizedFastHotel2T2` which computes Hotelling's T-squared statistic.
`verbose`	A logical value indicating whether to print progress information. Default is `TRUE`.
`conda_env`	A character string specifying the name of the conda environment to use via `reticulate`. Default is "fastrerandomize_env".
`conda_env_required`	A logical indicating whether the specified conda environment must be strictly used. If `TRUE`, an error is thrown if the environment is not found. Default is TRUE.

Details

The function works in two main steps: 1. Generates all possible combinations of treatment assignments given n_units and n_treated 2. If covariates (X) are provided, filters these combinations based on balance criteria using the specified threshold function

The balance filtering process uses Hotelling's T-squared statistic by default to measure multivariate balance between treatment and control groups. Randomizations are accepted if their balance measure is below the specified quantile threshold.

Value

The function returns a *list* with two elements: `candidate_randomizations`: an array of randomization vectors `M_candidate_randomizations`: an array of their balance measures.

Note

This function requires 'JAX' and 'NumPy' to be installed and accessible through the reticulate package.

References

Hotelling, H. (1931). The generalization of Student's ratio. The Annals of Mathematical Statistics, 2(3), 360-378.

See Also

`generate_randomizations` for full randomization generation function. `generate_randomizations_mc` for the Monte Carlo version.

Examples

```
## Not run:
# Generate synthetic data
X <- matrix(rnorm(60), nrow = 10)  # 10 units, 6 covariates

# Generate balanced randomizations with covariates
BalancedRandomizations <- generate_randomizations_exact(
  n_units = 10,
  n_treated = 5,
  X = X,
  randomization_accept_prob = 0.25  # Keep top 25% most balanced
)

## End(Not run)
```

---

generate_randomizations_mc

Draws a random sample of acceptable randomizations from all possible complete randomizations using Monte Carlo sampling

Description

This function performs sampling with replacement to generate randomizations in a memory-efficient way. It processes randomizations in batches to avoid memory issues and filters them based on covariate balance. The function uses JAX for fast computation and memory management.

Usage

```
generate_randomizations_mc(
  n_units,
  n_treated,
  X,
  randomization_accept_prob = 1,
  threshold_func = NULL,
  max_draws = 1e+05,
  batch_size = 1000,
  approximate_inv = TRUE,
```

```
    verbose = TRUE,
    conda_env = "fastrerandomize_env",
    conda_env_required = TRUE
  )
```

Arguments

`n_units`	An integer specifying the total number of experimental units.
`n_treated`	An integer specifying the number of units to be assigned to treatment.
`X`	A numeric matrix of covariates used for balance checking. Cannot be NULL.
`randomization_accept_prob`	A numeric value between 0 and 1 specifying the probability threshold for accepting randomizations based on balance. Default is 1
`threshold_func`	A JAX function that computes a balance measure for each randomization. Must be vectorized using `jax$vmap` with `in_axes = list(NULL, 0L, NULL, NULL)`, and inputs covariates (matrix of X), treatment_assignment (vector of 0s and 1s), n0 (scalar), n1 (scalar). Default is `VectorizedFastHotel2T2` which uses Hotelling's T-squared statistic.
`max_draws`	An integer specifying the maximum number of randomizations to draw.
`batch_size`	An integer specifying how many randomizations to process at once. Lower values use less memory but may be slower.
`approximate_inv`	A logical value indicating whether to use an approximate inverse (diagonal of the covariance matrix) instead of the full matrix inverse when computing balance metrics. This can speed up computations for high-dimensional covariates. Default is `TRUE`.
`verbose`	A logical value indicating whether to print detailed information about batch processing progress, and GPU memory usage. Default is `FALSE`.
`conda_env`	A character string specifying the name of the conda environment to use via `reticulate`. Default is `"fastrerandomize_env"`.
`conda_env_required`	A logical indicating whether the specified conda environment must be strictly used. If `TRUE`, an error is thrown if the environment is not found. Default is `TRUE`.

Details

The function works by:

1. Generating batches of random permutations.
2. Computing balance measures for each permutation using the provided threshold function.
3. Keeping only the top permutations that meet the acceptance probability threshold.
4. Managing memory by clearing unused objects and caches between batches.

The function uses smaller data types (int8, float16) where possible to reduce memory usage. It also includes assertions to verify array shapes and dimensions throughout.

Value

The function returns a *list* with two elements: `candidate_randomizations`: an array of randomization vectors `M_candidate_randomizations`: an array of their balance measures.

See Also

`generate_randomizations` for full randomization generation function. `generate_randomizations_exact` for the exact version.

Examples

```
## Not run:
# Generate synthetic data
X <- matrix(rnorm(100*5), 100, 5) # 5 covariates

# Generate 1000 randomizations for 100 units with 50 treated
rand_less_strict <- generate_randomizations_mc(
               n_units = 100,
               n_treated = 50,
               X = X,
               randomization_accept_prob=0.01,
               max_draws = 100000,
               batch_size = 1000)

# Use a stricter balance criterion
rand_more_strict <- generate_randomizations_mc(
               n_units = 100,
               n_treated = 50,
               X = X,
               randomization_accept_prob=0.001,
               max_draws = 1000000,
               batch_size = 1000)

## End(Not run)
```

`generate_randomizations_R`

Generate randomizations in base R, filtering by Hotelling's T^2 acceptance

---

Description

Base R function to either do exact enumeration or Monte Carlo random permutations, then keep the fraction whose T^2 is below the acceptance cutoff.

Usage

```
generate_randomizations_R(
  n_units,
  n_treated,
  X,
  accept_prob,
  random_type,
  max_draws,
  batch_size
)
```

Arguments

`n_units`	Integer, total number of units.
`n_treated`	Integer, number of units to be assigned to treatment.
`X`	Covariate matrix (n_units x p).
`accept_prob`	Numeric in [0, 1]: keep the fraction of randomizations that have the lowest T^2 up to this quantile.
`random_type`	Either "exact" or "monte_carlo".
`max_draws`	If ‘random_type="monte_carlo"‘, how many permutations to sample.
`batch_size`	If ‘random_type="monte_carlo"‘, how many permutations to handle per chunk.

Value

A list with:

- `randomizations`: a matrix (rows = accepted assignments).
- `balance`: numeric vector of T^2 values for each accepted assignment.

`hotellingT2_R` *Compute Hotelling’s T-squared statistic in base R*

Description

This function provides a base R implementation of Hotelling’s T-squared balance measure, renamed with ‘_R‘ for clarity that it is the R-based analog to the JAX version in fastrerandomize.

Usage

```
hotellingT2_R(X, W)
```

Arguments

`X`	A numeric matrix of covariates (observations in rows).
`W`	A 0/1 treatment assignment vector of the same length as nrow(X).

Value

A numeric scalar: Hotelling's T-squared statistic for that assignment.

`plot.fastrerandomize_randomizations`

Plot method for fastrerandomize_randomizations objects

Description

Generates a histogram of the balance measures for accepted randomizations.

Usage

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

Arguments

`x`	An object of class `fastrerandomize_randomizations`.
`...`	Further graphical parameters passed to `plot`.

Value

No return value. This function is called for the side effect of generating a histogram of the accepted balance measures of object with class `fastrerandomize_randomizations`.

`plot.fastrerandomize_test`

Plot method for fastrerandomize_test objects

Description

Plots a simple visualization of the observed effect and the fiducial interval (if present) on a horizontal axis.

Usage

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

Arguments

`x`	An object of class `fastrerandomize_test`.
`...`	Further graphical parameters passed to `plot`.

Value

No output returned. Performs side effect of plotting `fastrerandomize_test` class objects.

`print.fastrerandomize_randomizations`

Print method for fastrerandomize_randomizations objects

Description

Print method for fastrerandomize_randomizations objects

Usage

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

Arguments

`x`	An object of class `fastrerandomize_randomizations`.
`...`	Further arguments passed to or from other methods.

Value

Prints an object of class `fastrerandomize_randomizations`.

`print.fastrerandomize_test`

Print method for fastrerandomize_test objects

Description

Print method for fastrerandomize_test objects

Usage

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

Arguments

`x`	An object of class `fastrerandomize_test`.
`...`	Further arguments passed to or from other methods.

Value

No return value, prints object of class `fastrerandomize_test`.

`print2` *Print timestamped messages with optional quieting*

Description

This function prints messages prefixed with the current timestamp in a standardized format. Messages can be suppressed using the quiet parameter.

Usage

```
print2(text, quiet = FALSE)
```

Arguments

`text`	A character string containing the message to be printed.
`quiet`	A logical value indicating whether to suppress output. Default is `FALSE`.

Details

The function prepends the current timestamp in "YYYY-MM-DD HH:MM:SS" format to the provided message.

Value

No return value, called for side effect of printing with timestamp.

See Also

`Sys.time` for the underlying timestamp functionality.

Examples

```
# Print a basic message with timestamp
print2("Processing started")

# Suppress output
print2("This won't show", quiet = TRUE)

# Use in a loop
for(i in 1:3) {
  print2(sprintf("Processing item %d", i))
}
```

`QJEData` *QJEData: Agricultural Treatment Experiment Data*

Description

Data from a field experiment studying moral hazard in tenancy contracts in agriculture.

After subsetting, this dataset includes observations on 968 experimental units with the following variables of interest: household composition, treatment assignment, and agricultural outcomes.

Usage

```
data(QJEData)
```

Format

A data frame with 968 rows and 7 columns:

children Numeric (integer). Number of children in the household. Larger numbers may reflect increased household labor needs and different investment or effort incentives.

married Numeric/binary. Whether the household head is currently married (1) or not (0). Marital status may influence decision-making and risk preferences in farming.

hh_size Numeric (integer). Household size. Differences in family labor availability or consumption needs can influence effort levels and thus relate to moral hazard in production decisions.

hh_sexrat Numeric. The ratio of adult men to adult women in the household. Imbalances in the male–female ratio can affect labor division and investment decisions.

treat1 Numeric/binary. Primary treatment indicator (e.g., whether a farmer is offered a specific tenancy contract or cost-sharing arrangement).

R_yield_ELA_sqm Numeric. Crop yield per square meter (e.g., kilograms of output per square meter). This is a principal outcome measure for evaluating productivity and treatment impact on farm performance.

ELA_Fertil_D Numeric/binary. Indicator for whether fertilizer was used (1) or not (0). This measures input investment—a key mechanism in moral hazard models (farmers may alter input use under different contracts).

Source

Burchardi, K.B., et al. (2019). Moral hazard: Experimental evidence from tenancy contracts. *The Quarterly Journal of Economics*, 134(1), 281-347.

`randomization_test` *Fast randomization test*

Description

Fast randomization test

Usage

```
randomization_test(
  obsW = NULL,
  obsY = NULL,
  alpha = 0.05,
  candidate_randomizations = NULL,
  candidate_randomizations_array = NULL,
  n0_array = NULL,
  n1_array = NULL,
  findFI = FALSE,
  c_initial = 2,
  conda_env = "fastrerandomize_env",
  conda_env_required = TRUE
)
```

Arguments

`obsW`	A numeric vector where `0`'s correspond to control units and `1`'s to treated units.
`obsY`	An optional numeric vector of observed outcomes. If not provided, the function assumes a NULL value.
`alpha`	The significance level for the test. Default is `0.05`.
`candidate_randomizations`	A numeric matrix of candidate randomizations.
`candidate_randomizations_array`	An optional 'JAX' array of candidate randomizations. If not provided, the function coerces `candidate_randomizations` into a 'JAX' array.
`n0_array`	An optional array specifying the number of control units.
`n1_array`	An optional array specifying the number of treated units.
`findFI`	A logical value indicating whether to find the fiducial interval. Default is FALSE.
`c_initial`	A numeric value representing the initial criterion for the fiducial interval search. Default is `2`.
`conda_env`	A character string specifying the name of the conda environment to use via `reticulate`. Default is `"fastrerandomize_env"`.
`conda_env_required`	A logical indicating whether the specified conda environment must be strictly used. If `TRUE`, an error is thrown if the environment is not found. Default is `TRUE`.

Value

Returns an S3 object with slots:

- `p_value` A numeric value or vector representing the p-value of the test (or the expected p-value under the prior structure specified in the function inputs).
- `FI` A numeric vector representing the fiducial interval if `findFI=TRUE`.
- `tau_obs` A numeric value or vector representing the estimated treatment effect(s).
- `fastrr_env` The fastrerandomize environment.

References

- Zhang, Y. and Zhao, Q., 2023. What is a randomization test?. Journal of the American Statistical Association, 118(544), pp.2928-2942.

See Also

`generate_randomizations` for randomization generation function.

Examples

```
## Not run:
# A small synthetic demonstration with 6 units, 3 treated and 3 controls:

# Generate pre-treatment covariates
X <- matrix(rnorm(24*2), ncol = 2)

# Generate candidate randomizations
RandomizationSet_MC <- generate_randomizations(
  n_units = nrow(X),
  n_treated = round(nrow(X)/2),
  X = X,
  randomization_accept_prob = 0.1,
  randomization_type = "monte_carlo",
  max_draws = 100000,
  batch_size = 1000
)

# Generate outcome
W <- RandomizationSet_MC$randomizations[1,]
obsY <- rnorm(nrow(X), mean = 2 * W)

# Perform randomization test
results_base <- randomization_test(
  obsW = W,
  obsY = obsY,
  candidate_randomizations = RandomizationSet_MC$randomizations
)
print(results_base)

# Perform randomization test with fiducial interval
result_fi <- randomization_test(
```

```
  obsW = W,
  obsY = obsY,
  candidate_randomizations = RandomizationSet_MC$randomizations,
  findFI = TRUE
)
print(result_fi)

## End(Not run)
```

`randomization_test_R` *Base R randomization test: difference in means + optional fiducial interval*

Description

Base R randomization test: difference in means + optional fiducial interval

Usage

```
randomization_test_R(obsW, obsY, allW, findFI = FALSE, alpha = 0.05)
```

Arguments

`obsW`	Observed assignment (0/1).
`obsY`	Observed outcome vector.
`allW`	Matrix of candidate random assignments (rows = assignments).
`findFI`	Logical, whether to compute fiducial interval as well.
`alpha`	Significance level (default 0.05).

Value

A list with p_value, tau_obs, and (optionally) FI if ‘findFI=TRUE‘.

`summary.fastrerandomize_randomizations` *Summary method for fastrerandomize_randomizations objects*

Description

Summary method for fastrerandomize_randomizations objects

Usage

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

Arguments

`object`	An object of class `fastrerandomize_randomizations`.
`...`	Further arguments passed to or from other methods.

Value

A list with summary statistics, printed by default.

`summary.fastrerandomize_test` *Summary method for fastrerandomize_test objects*

Description

Summary method for fastrerandomize_test objects

Usage

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

Arguments

`object`	An object of class `fastrerandomize_test`.
`...`	Further arguments passed to or from other methods.

Value

Returns an (invisible) list with a summary of `fastrerandomize_test` class objects.

`YOPData` *YOPData*

Description

Data from a re-analysis of the Youth Opportunities Program anti-poverty RCT in Uganda, with satellite imagery neural representations linked to RCT units.

Usage

```
data(YOPData)
```

Format

A list containing two data frames:

RCTData	Treatment, outcome, and geolocation information
ImageEmbeddings	CLIP-RSICD neural embeddings of satellite imagery

Source

- Blattman, C., Fiala, N. and Martinez, S. (2020). The Long-term Impacts of Grants on Poverty: Nine-year Evidence from Uganda's Youth Opportunities Program. American Economic Review: Insights, 2(3), 287-304.
- Jerzak, C.T., Johansson, F.D. and Daoud, A. (2023). Image-based Treatment Effect Heterogeneity. Conference on Causal Learning and Reasoning, 531-552. PMLR.

Index