Package 'FDOTT'

October 1, 2025

Type Package

Title Optimal Transport Based Testing in Factorial Design

Version 0.1.0

Date 2025-09-24

Description Perform optimal transport based tests in factorial designs as introduced in Groppe et al. (2025) <doi:10.48550/arXiv.2509.13970> via the FDOTT() function. These tests are inspired by ANOVA and its nonparametric counterparts. They allow for testing linear relationships in factorial designs between finitely supported probability measures on a metric space. Such relationships include equality of all measures (no treatment effect), interaction effects between a number of factors, as well as main and simple factor effects.

License GPL (>= 3)

Imports Rcpp (>= 1.0.12), ROI, future.apply, progressr, transport, slam, rrapply, stats, methods

Depends R (>= 4.1)

Suggests ROI.plugin.glpk, future

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.3.3

Encoding UTF-8

NeedsCompilation yes

Author Michel Groppe [aut, cre], Linus Niemöller [aut]

Maintainer Michel Groppe <michel.groppe@uni-goettingen.de>

Repository CRAN

Date/Publication 2025-10-01 07:00:20 UTC

Contents

`cost_matrix_lp` *Cost Matrix of ℓ^p-Form.*

Description

Compute cost matrices of ℓ^p-form.

Usage

```
cost_matrix_lp(x, y = NULL, p = 2, q = 1)
```

Arguments

`x`	matrix of size $n \times d$ containing vectors $x_1, \ldots, x_n \in \mathbb{R}^d$ (row-wise).
`y`	matrix of size $m \times d$ containing vectors $y_1, \ldots, y_m \in \mathbb{R}^d$ (row-wise); `y = NULL` means that $y_i = x_i$.
`p`	number $p \in (0, \infty]$.
`q`	number $q \in (0, \infty)$.

Value

A $n \times m$ matrix with entry at i, j being equal to

$$\|x_i - y_j\|_p^q = \left[\sum_{k=1}^{d} |x_{i,k} - y_{j,k}|^p\right]^{q/p}$$

For `p = Inf`, this is to be understood as the maximum norm to the power of q.

Examples

```
n <- 3
m <- 4
d <- 5
x <- runif(n * d) |> matrix(n, d)
y <- runif(m * d) |> matrix(m, d)
costm <- cost_matrix_lp(x, y)
print(costm)
```

FDOTT *Test linear relationships between probability vectors in factorial designs*

Description

Perform FDOTT, an optimal transport (OT) based test in factorial designs, to test linear relationships between probability vectors, based on samples from them.

Usage

```
FDOTT(
  samples,
  costm,
  H0 = "*",
  fac.names = NULL,
  method = c("plug-in", "bootstrap-deriv", "bootstrap-m", "permutation"),
  num.sim = 1000,
  null.mu = NULL,
  m.p = 0.5,
  is.metric = is_metric_cost_mat(costm, tol.ti = Inf),
  verbose = FALSE
)
```

Arguments

samples	nested list of depth D (representing a D-way layout) containing count vectors. A count vector is a vector of length N that contains the number of times a sample was observed at the respective points. Can also be given as a matrix (row-wise), which is viewed as a one-way layout.
costm	semi-metric cost matrix $c \in \mathbb{R}^{N \times N}$.
H0	null hypothesis, see details.
fac.names	names of the D factors. Used for printing. Default NULL corresponds to "F1" for factor 1, and so on.
method	the method to use to simulate from the null distribution, see details.
num.sim	number of samples to draw from the limiting null distribution.
null.mu	probability vectors μ underlying the null distribution used only for method = "plug-in". Must be of the same structure as samples.
m.p	exponent $p \in (0, 1)$ used only for method = "bootstrap-m".
is.metric	value indicating whether c is a metric cost matrix, see is_metric_cost_mat.
verbose	logical value indicating whether additional information should be printed.

Details

Denote with μ the matrix (row-wise) of the probability vectors (in lexicographical order of the factor combinations) that underlie `samples`. FDOTT deals with null hypotheses of the form

$$H_0^L : \; L\mu = 0 \,,$$

where L is a suitable matrix with row sums all equal to 0. The FDOTT statistic is defined as

$$T^L(\hat{\mu}_n) := \frac{\sqrt{\rho_n}}{s} \sum_{m=1}^{M} \mathrm{OT}_c^{\pm}([L\hat{\mu}_n]_m, 0) \,,$$

where ρ_n and s are scaling factors, $[L\mu]_m$ is the m-th row-vector of $L\mu$ and $\mathrm{OT}_c^{\pm}$ the extended OT functional, see `ot_cost_sgn`. The test is based on the asymptotic distribution of $T^L(\hat{\mu}_n)$ under under the null, for more details see Groppe et al. (2025).

The form of H_0^L allows for testing hypotheses like interaction effects in classical ANOVA, obtained by formally substituting means by measures. The following values are allowed for `H0`:

- `H0 = "*"` (the default). Test all interaction (including main effects) of the factors. A specific interaction or main effect can be tested by including the corresponding indices of the factors in a list, e.g., `H0 = list("*", c(1, 3))` corresponds to the interaction effect between factor 1 and 3. Note that in a one-way layout, `H0 = "*"` reduces to `H0 = "="`.
- `H0 = "|"`. Test all simple factor effects. A specific simple factor effect can be tested by by including the corresponding indices of the factors in a list, e.g., `H0 = list("|", c(1, 3))` corresponds to the simple factor effect of factor 1 and 3 within the other remaining factors.
- `H0 = "="`. Test for treatment effect, i.e., whether all underlying probability vectors are the same. Note that each pairwise comparison can be tested simultaneously via `FDOTT_HSD`.
- `H0 = L`. Test H_0^L for the directly supplied L matrix. The name of the tested effect (useful for printing) and the scaling s (by default `nrow(L)`) can be supplied by setting the `"effect"` and `"scaling"` attribute of `L`, respectively.
- `H0 = list(...)`. Test a combined null hypothesis. Each element of the list represents a null hypothesis and can be given by one of the options above. This is useful in combination with `FDOTT_HSD`, which allows to test all the given null hypotheses simultaneously.

To simulate from the limiting null distribution, there are four different methods:

- `"plug-in"`: uses the limiting distribution where μ is substituted by its empirical version (or `null.mu`, when specified).
- `"bootstrap-deriv"`: uses the so-called derivative bootstrap.
- `"bootstrap-m"`: uses m-out-of-n bootstrap with $m = \lfloor n^p \rfloor$.
- `"permutation"`: uses a permutation approach, only works for `H0 = "="`.

These simulations can be done in parallel via `future::plan` and the progress can be shown with `progressr::with_progress`.

Value

A FDOTT object containing:

fac.lvls	vector of levels of the factors
mu	matrix, empirical version $\hat{\mu}_n$ of μ that is based on samples
n	vector of sample sizes n
L	matrix L for the null hypothesis H_0^L
p.value	the p-value
statistic	the value of the test statistic $T^L(\hat{\mu}_n)$
null.samples	samples drawn from the null distribution

References

M. Groppe, L. Niemöller, S. Hundrieser, D. Ventzke, A. Blob, S. Köster and A. Munk (2025). Optimal Transport Based Testing in Factorial Design. arXiv preprint. doi:10.48550/arXiv.2509.13970.

See Also

FDOTT_HSD

Examples

```
# enable txt progressbar
progressr::handlers("txtprogressbar")
# enable parallel computation
if (requireNamespace("future")) {
    future::plan(future::multisession)
}

# use higher number to better approximate null distribution and get more accurate p-value
num.sim <- 10

### one-way layout

N <- 2
costm <- cost_matrix_lp(1:N)

K <- 3
n <- c(300, 360, 200)

# underlying probability vectors, all measures are equal
mu <- matrix(1 / N, K, N, TRUE)

set.seed(123)
samples <- tab_sample(n, mu)
# show progress
progressr::with_progress({
    # default in one-way layout is H0 = "="
    res <- FDOTT(samples, costm, num.sim = num.sim)
})
```

```
print(res)

# measures are not equal
mu[2, ] <- c(0.1, 0.9)

set.seed(123)
samples <- tab_sample(n, mu)
res2 <- FDOTT(samples, costm, num.sim = num.sim)
print(res2)
# find out which measures are not equal via HSD
res3 <- FDOTT_HSD(res2)
print(res3)

### two-way layout

K1 <- K2 <- 2
N <- 3
costm <- cost_matrix_lp(1:N)

n <- list(list(300, 360), list(280, 200))

# underlying probability vectors (two-way layout)
# no interaction effect, only factor 2 has main effect
mu <- list(
    list(c(0, 0.5, 0.5), c(0.25, 0.25, 0.5)),
    list(c(0, 0.5, 0.5), c(0.25, 0.25, 0.5))
)

# test interaction effect and main effects, equivalent to H0 <- "*"
H0 <- list(list("*", 1:2), list("*", 1), list("*", 2))

set.seed(123)
samples <- tab_sample(n, mu)
res <- FDOTT(samples, costm, H0 = H0, num.sim = num.sim)
print(res)

# find out exactly which effect gets rejected via HSD
res1 <- FDOTT_HSD(res)
print(res1)

# now with interaction effect
mu[[1]][[1]] <- c(0.3, 0.3, 0.4)

# only test for interaction effect
H0 <- list("*", 1:2)

set.seed(123)
samples <- tab_sample(n, mu)
res2 <- FDOTT(samples, costm, H0 = H0, num.sim = num.sim, method = "bootstrap-deriv")
print(res2)

### custom effect
```

```
K <- 2
N <- 2
costm <- cost_matrix_lp(1:N)
num.sim <- 100

# null hypothesis H0: mu^1 - 0.5 * mu^2 - 0.5 * mu^3 = 0
L <- matrix(c(1, -0.5, -0.5), 1, 3)
# give custom name
attr(L, "effect") <- "mu^1 = 0.5 * (mu^2 + mu^3)"

# underlying probability vectors
mu <- matrix(c(0.4, 0.6, 0.6, 0.4, 0.2, 0.8), 3, 2, TRUE)
print(L %*% mu)

n <- c(250, 280, 230)

# test L, as well as mu^1 = mu^2 = mu^3
H0 <- list(L, "=")

set.seed(123)
samples <- tab_sample(n, mu)
res <- FDOTT(samples, costm, H0 = H0, num.sim = num.sim)
print(res)
# find out which effect is responsible for rejection
res2 <- FDOTT_HSD(res)
print(res2)

# L %*% mu = 0 not satisfied anymore
mu[2, ] <- c(1, 0)
print(L %*% mu)

# only test for L %*% mu = 0
H0 <- L

set.seed(123)
samples <- tab_sample(n, mu)
res3 <- FDOTT(samples, costm, H0 = H0, num.sim = num.sim)
print(res3)
```

FDOTT_HSD *Test multiple linear relationships between probability vectors in factorial designs*

Description

Perform an optimal transport based HSD test to deal with multiple comparisons simultaneously.

Usage

```
FDOTT_HSD(test, weights = NULL, group.sizes = TRUE)
```

Arguments

`test`	a `FDOTT` object, i.e., output of `FDOTT`.
`weights`	weight vector of length K. `weights = NULL` means that no weights are used. For `weights = TRUE` the standard weighting is used.
`group.sizes`	integer vector summing to the number of comparisons M. Used to split the null hypothesis into sub-hypotheses of the specified sizes. The default `group.sizes = TRUE` extracts these sizes from `test`. For `group.sizes = NULL`, each equation is its own group.

Details

Let $H_0^L : L\mu = 0$ be the null hypothesis of `test`. In the case of rejection, it is of interest to find out exactly which row-equations are not satisfied with statistical significance. To this end, $L\mu = 0$ can be split into a number of sub-hypotheses which are tested simultaneously via an approach inspired by Tukey's HSD test, see Groppe et al. (2025) for more details.

Value

A `FDOTT_HSD` object containing:

`p.value`	the p-values
`statistic`	the values of the test statistics
`null.samples`	samples drawn from the null distribution

References

M. Groppe, L. Niemöller, S. Hundrieser, D. Ventzke, A. Blob, S. Köster and A. Munk (2025). Optimal Transport Based Testing in Factorial Design. arXiv preprint. doi:10.48550/arXiv.2509.13970.

See Also

`FDOTT`

Examples

```
# see FDOTT for more examples

# enable parallel computation
if (requireNamespace("future")) {
    future::plan(future::multisession)
}

K <- 3
N <- 2
costm <- cost_matrix_lp(1:N)

# use higher number to better approximate null distribution and get more accurate p-value
num.sim <- 10
```

```
# underlying probability vectors (one-way layout)
# only mu^1 and mu^3 are equal
mu <- matrix(0.5, K, N, TRUE)
mu[2, ] <- c(0.2, 0.8)

n <- c(300, 360, 200)

set.seed(123)
samples <- tab_sample(n, mu)
res <- FDOTT(samples, costm, num.sim = num.sim) |> FDOTT_HSD()
# significant differences for mu^1 = mu^2 and mu^2 = mu^3
print(res)
```

`is_metric_cost_mat` *Check metric properties of cost matrices*

Description

Check if a cost matrix satisfies symmetry, positive definiteness and the triangle inequality.

Usage

```
is_metric_cost_mat(x, tol.sym = 1e-08, tol.pd = 0, tol.ti = 1e-08)
```

Arguments

`x`	numeric square matrix.
`tol.sym`	tolerance used to check symmetry.
`tol.pd`	tolerance used to check positive definiteness.
`tol.ti`	tolerance used to check the triangle inequality.

Details

The following three properties of a square matrix $x \in \mathbb{R}^{N \times N}$ are checked:

- symmetry; if $x_{ij} = x_{ji}$,
- positive definiteness; if $x_{ii} = 0$ and $x_{ij} > 0$ for all $i \neq j$,
- triangle inequality; if $x_{ij} \leq x_{ik} + x_{kj}$.

If symmetry and positive definiteness are satisfied, then x is called a semi-metric cost matrix. If additionally also the triangle inequality holds, then x is a metric cost matrix.

Value

A list containing logical entries `metric`, `semi.metric`, `sym`, `pos.def` and `tri.ineq` that indicate whether the corresponding property is satisfied.

See Also

cost_matrix_lp

Examples

```
x <- cost_matrix_lp(1:5)
res <- is_metric_cost_mat(x)
res2 <- is_metric_cost_mat(x^2)
# x is a metric cost matrix
print(res$metric)
# x^2 is only a semi-metric cost matrix,
# because the triangle inequality is not satisfied
print(res2$semi.metric)
print(res2$tri.ineq)
```

---

ot_barycenter *Compute optimal transport barycenters*

Description

Compute the optimal transport (OT) barycenter of multiple probability vectors via linear programming.

Usage

```
ot_barycenter(
  mu,
  costm,
  w = NULL,
  solver = ot_test_lp_solver(),
  constr_mat = NULL
)
```

Arguments

mu	matrix (row-wise) or list containing K probability vectors of length N.
costm	cost matrix $c \in \mathbb{R}^{N \times N}$.
w	weight vector $w \in \mathbb{R}_+^K$. The default is $w = (1/K, \ldots, 1/K)$.
solver	the LP solver to use, see ot_test_lp_solver.
constr_mat	the constraint matrix for the underlying LP.

Details

The OT barycenter is defined as the minimizer of the cost functional,

$$B_c^w(\mu^1, \ldots, \mu^K) := \min_\nu \sum_{k=1}^{k} w_k \, \mathrm{OT}_c(\mu^k, \nu)\,,$$

where the minimum is taken over all probability vectors ν. The OT barycenter is solved via linear programming (LP) and the underlying solver can be controlled via the parameter `solver`.

Value

A list containing the entries `cost` and `barycenter`.

Examples

```
K <- 3
N <- 2
costm <- cost_matrix_lp(1:N)
w <- rep(1 / K, K)

# all measures are equal
mu <- matrix(1 / N, K, N, TRUE)

# to run this, a LP solver must be available for ROI (ROI.plugin.glpk by default)
if (requireNamespace("ROI.plugin.glpk")) {
    solver <- ot_test_lp_solver("glpk")
    print(ot_barycenter(mu, costm, w = w, solver = solver))
}

# not all measures are equal
mu[2, ] <- 1:N / sum(1:N)
if (requireNamespace("ROI.plugin.glpk")) {
    solver <- ot_test_lp_solver("glpk")
    print(ot_barycenter(mu, costm, w = w, solver = solver))
}
```

`ot_barycenter_test` *Test equality of probability vectors*

Description

Perform optimal transport (OT) barycenter based tests for equality of probability vectors in a one-way layout.

Usage

```
ot_barycenter_test(
  samples,
  costm,
  null.mu = NULL,
  w = NULL,
  num.sim = 1000,
  solver = ot_test_lp_solver(),
  is.metric = is_metric_cost_mat(costm, tol.ti = Inf),
  verbose = FALSE
)
```

Arguments

samples	matrix (row-wise) or nested list containing K count vectors. A count vector is a vector of length N that contains the number of times a sample was observed at the respective points.
costm	semi-metric cost matrix $c \in \mathbb{R}^{N \times N}$.
null.mu	probability measures μ underlying the null distribution. Must be of the same structure as samples.
w	weight vector $w \in \mathbb{R}_+^K$.
num.sim	number of samples to draw from the limiting null distribution.
solver	the LP solver to use, see ot_test_lp_solver.
is.metric	value indicating whether c is a metric cost matrix, see is_metric_cost_mat.
verbose	logical value indicating whether additional information should be printed.

Details

Denote with $\mu^1, \ldots, \mu^K$ the probability measures that underlie the samples contained in samples. To test for the one-way null hypothesis $H_0 : \mu^1 = \ldots = \mu^K$, this test employs the OT barycenter statistic which is defined as $T^B(\mu) := \sqrt{\rho_n} B_c^w(\mu^1, \ldots, \mu^K)$, where ρ_n is a scaling factor and B_c^w is the OT barycenter functional, see ot_barycenter.

The test is based on the asymptotic distribution of T^B under under the null, for more details see the reference.

These simulations can be done in parallel via future::plan and the progress can be shown with progressr::with_progress.

Especially for large N and K, simulating a sufficient number of samples from the limiting null distribution might take a while. Consider using FDOTT instead.

Value

An object of class "ot_barycenter_test" containing:

mu	empirical version of μ that is based on samples
n	the sample sizes

`p.value`	the p-value
`statistic`	the value of the test statistic
`null.samples`	samples drawn from the null distribution

References

TODO

See Also

`FDOTT` with `H0 = "="`.

Examples

```
# enable txt progressbar
progressr::handlers("txtprogressbar")
# enable parallel computation
if (requireNamespace("future")) {
    future::plan(future::multisession)
}

K <- 3
N <- 2
costm <- cost_matrix_lp(1:N)

# use higher number to better approximate null distribution and get more accurate p-value
num.sim <- 10

n <- c(300, 360, 200)

# underlying probability vectors
mu <- matrix(1 / N, K, N, TRUE)

# to run this, a LP solver must be available for ROI (ROI.plugin.glpk by default)
if (requireNamespace("ROI.plugin.glpk")) {
    solver <- ot_test_lp_solver("glpk")
    set.seed(123)
    samples <- tab_sample(n, mu)
    progressr::with_progress({
        res <- ot_barycenter_test(samples, costm, num.sim = num.sim, solver = solver)
    })
    print(res)
}

# measures are not equal anymore
mu[2, ] <- 1:N / sum(1:N)

if (requireNamespace("ROI.plugin.glpk")) {
    solver <- ot_test_lp_solver("glpk")
    set.seed(123)
    samples <- tab_sample(n, mu)
```

```
    progressr::with_progress({
        res2 <- ot_barycenter_test(samples, costm, num.sim = num.sim, solver = solver)
    })
    print(res2)
}
```

`ot_cost_sgn` *Compute optimal transport costs for signed measures*

Description

Compute the optimal transport (OT) cost between signed measures that have the same total mass.

Usage

```
ot_cost_sgn(mu, nu, costm, mode = c("all", "diag"))
```

Arguments

`mu`	matrix (row-wise) or list containing K_1 vectors of length N.
`nu`	matrix (row-wise) or list containing K_2 vectors of length N or `NULL`.
`costm`	cost matrix $c \in \mathbb{R}^{N \times N}$.
`mode`	controls which of the pairwise OT costs are computed.

Details

The extended OT functional for vectors $\mu,\ \nu \in \mathbb{R}^N$ with $\sum_{i=1}^N \mu_i = \sum_{i=1}^N \nu_i$ is defined as

$$\mathrm{OT}_c^{\pm}(\mu, \nu) := \mathrm{OT}_c(\mu^+ + \nu^-,\ \nu^+ + \mu^-)\,,$$

where $\mu^+ = \max(0, \mu)$ and $\mu^- = -\min(0, \mu)$ denote the positive and negative part of μ, and OT_c is the standard OT functional. To compute the standard OT, the function `transport::transport` is used. The values may be computed in parallel via `future::plan`.

Value

The OT cost between the vectors in `mu` and `nu`.

For `mode = "all"` the whole matrix of size $K_1 \times K_2$ is returned. If `mu` or `nu` is a vector, then this matrix is also returned as a vector. `nu = NULL` means that `nu = mu` and only the lower triangular part is actually computed and then reflected.

If `mode = "diag"`, then only the diagonal is returned (requiring $K_1 = K_2$).

See Also

`transport::transport`

Examples

```
# enable parallel computation
if (requireNamespace("future")) {
    future::plan(future::multisession)
}

# generate random signed measures with total mass 0 (row-wise)
rsum0 <- \(K, N) {
    x <- runif(K * N) |> matrix(K, N)
    x <- sweep(x, 1, rowSums(x) / N, "-")
    x[, 1] <- x[, 1] - rowSums(x)
    x
}

K1 <- 3
K2 <- 2
N <- 4
costm <- cost_matrix_lp(1:N)

set.seed(123)
mu <- rsum0(K1, N)
nu <- rsum0(K2, N)

print(ot_cost_sgn(mu[2, ], nu[2, ], costm))

# mode = "diag" requires K1 = K2
print(ot_cost_sgn(mu[1:2, ], nu, costm, mode = "diag"))

print(ot_cost_sgn(mu, nu, costm))

# only works properly if costm is semi-metric
print(ot_cost_sgn(mu, NULL, costm))
# but it requires less computations than
print(ot_cost_sgn(mu, mu, costm))
```

`ot_test_lp_solver` *Control* FDOTT *linear programming solver*

Description

Create an object that controls the linear programming (LP) solver to use.

Usage

```
ot_test_lp_solver(name = NULL, ...)
```

Arguments

`name`	name of the LP solver.
`...`	optional control arguments passed to the corresponding LP solver.

Details

name can be any LP solver that is compatible with the ROI package infrastructure. In particular, the corresponding plugin package ROI.plugin.name must be installed. The default value corresponding to name = NULL can be set via options(FDOTT.lp_solver = name) (the default is "glpk").

Value

A ot_test_lp_solver_control object containing:

name	the name of the LP solver
control	list of control arguments passed to the LP solver

See Also

ROI::ROI_available_solvers

Examples

```
## Not run:
# glpk is already the default
options(FDOTT.lp_solver = "glpk")
## End(Not run)
# plugin needs to be installed, else we get error
if (requireNamespace("ROI.plugin.glpk")) {
    # add control parameter (specific to glpk)
    sol <- ot_test_lp_solver("glpk", verbose = TRUE)
    print(sol)
} else {
    cat("'ROI.plugin.glpk' needs to be installed!\n")
}
```

---

simulate_finite_FDOTT *Simulations for* FDOTT

Description

Perform simulations for the test statistic used in FDOTT.

Usage

```
simulate_finite_FDOTT(mu, costm, n, H0 = "*", num.sim = 1000)

simulate_limit_FDOTT_null(
  mu,
  costm,
  n = NULL,
  delta = NULL,
```

```
  H0 = "*",
  num.sim = 1000,
  method = c("plug-in", "bootstrap-deriv", "bootstrap-m"),
  m.p = 0.5,
  mean = NULL
)

simulate_limit_FDOTT_alt(mu, costm, delta, H0 = "*", num.sim = 1000)
```

Arguments

mu	matrix (row-wise) or nested list containing K probability vectors.
costm	semi-metric cost matrix $c \in \mathbb{R}^{N \times N}$.
n	samples sizes. Must be of the same structure as mu.
H0	null hypothesis, see FDOTT for more information.
num.sim	number of samples to draw from the limiting null distribution.
delta	asymptotic sample size ratios. Must be of the same structure as mu.
method	the method to use to simulate from the null distribution.
m.p	exponent $p \in (0, 1)$ used for method = "bootstrap-m".
mean	mean of the Gaussians appearing in the limiting distribution. Must be of the same structure as mu.

Details

See FDOTT for the definition of the test statistic and more details.

simulate_finite_FDOTT simulates from the finite sample distribution.

simulate_limit_FDOTT_null and simulate_limit_FDOTT_alt simulate from the limiting distribution under the null or alternative, respectively.

All these simulations can be done in parallel via future::plan and the progress can be shown with progressr::with_progress.

Value

A vector containing the simulated samples.

Examples

```
# enable txt progressbar
progressr::handlers("txtprogressbar")
# enable parallel computation
if (requireNamespace("future")) {
    future::plan(future::multisession)
}

K <- 3
N <- 2
```

```
costm <- cost_matrix_lp(1:N)

# use higher values for better approximation
num.sim <- 10
n <- rep(300, K)

delta <- rep(1 / K, K)

# under one-way null
mu <- matrix(1 / N, K, N, TRUE)

set.seed(123)
lhs <- simulate_finite_FDOTT(mu, costm, n, num.sim = num.sim)
rhs <- simulate_limit_FDOTT_null(mu, costm, delta = delta, num.sim = num.sim)

h1 <- density(lhs)
h2 <- density(rhs)
plot(h1$x, h1$y, xlim = range(h1$x, h2$x), ylim = range(h1$y, h2$y), type = "l",
     col = "red", xlab = "x", ylab = "density", main = "KDEs")
lines(h2$x, h2$y, col = "blue")
legend("topright", c("Finite", "Limit"), col = c("red", "blue"), pch = 15)

# under one-way alternative
mu[2, ] <- 1:N / sum(1:N)

set.seed(123)
lhs <- simulate_finite_FDOTT(mu, costm, n, num.sim = num.sim)
rhs <- simulate_limit_FDOTT_alt(mu, costm, delta, num.sim = num.sim)

h1 <- density(lhs)
h2 <- density(rhs)
plot(h1$x, h1$y, xlim = range(h1$x, h2$x), ylim = range(h1$y, h2$y), type = "l",
     col = "red", xlab = "x", ylab = "density", main = "KDEs")
lines(h2$x, h2$y, col = "blue")
legend("topright", c("Finite", "Limit"), col = c("red", "blue"), pch = 15)
```

`simulate_finite_ot_barycenter_test` *Simulations for* `ot_barycenter_test`

Description

Perform simulations for the test statistic used in `ot_barycenter_test`.

Usage

```
simulate_finite_ot_barycenter_test(
  mu,
  costm,
```

```
  n,
  w = NULL,
  num.sim = 1000,
  solver = ot_test_lp_solver()
)

simulate_limit_ot_barycenter_test_null(
  mu,
  costm,
  n = NULL,
  delta = NULL,
  w = NULL,
  num.sim = 1000,
  solver = ot_test_lp_solver(),
  mean = NULL
)

simulate_limit_ot_barycenter_test_alt(
  mu,
  costm,
  delta,
  w = NULL,
  num.sim = 1000,
  solver = ot_test_lp_solver()
)
```

Arguments

mu	matrix (row-wise) or list containing K probability vectors.
costm	semi-metric cost matrix $c \in \mathbb{R}^{N \times N}$.
n	vector of samples sizes.
w	weight vector $w \in \mathbb{R}_+^K$.
num.sim	number of samples to draw from the limiting null distribution.
solver	the LP solver to use, see ot_test_lp_solver.
delta	vector of asymptotic sample size ratios.
mean	mean of the Gaussians appearing in the limiting distribution. Must be of the same structure as mu.

Details

See ot_barycenter_test for the definition of the test statistic and more details.

simulate_finite_ot_barycenter_test simulates from the finite sample distribution.

simulate_limit_ot_barycenter_test_null and simulate_limit_ot_barycenter_test_alt simulate from the limiting distribution under the null or alternative, respectively.

All these simulations can be done in parallel via future::plan and the progress can be shown with progressr::with_progress.

Value

A vector containing the simulated samples.

Examples

```
# enable txt progressbar
progressr::handlers("txtprogressbar")
# enable parallel computation
if (requireNamespace("future")) {
    future::plan(future::multisession)
}

K <- 3
N <- 2
costm <- cost_matrix_lp(1:N)

# use higher values for better approximation
num.sim <- 10
n <- rep(300, K)

delta <- rep(1 / K, K)

# under null
mu <- matrix(1 / N, K, N, TRUE)

# to run this, a LP solver must be available for ROI (ROI.plugin.glpk by default)
if (requireNamespace("ROI.plugin.glpk")) {
    solver <- ot_test_lp_solver("glpk")
    set.seed(123)
    # show progress bar
    progressr::with_progress({
     lhs <- simulate_finite_ot_barycenter_test(mu, costm, n, num.sim = num.sim, solver = solver)
     rhs <- simulate_limit_ot_barycenter_test_null(mu, costm, delta = delta, num.sim = num.sim,
                                                   solver = solver)
    })
    h1 <- density(lhs)
    h2 <- density(rhs)
    plot(h1$x, h1$y, xlim = range(h1$x, h2$x), ylim = range(h1$y, h2$y), type = "l",
         col = "red", xlab = "x", ylab = "density", main = "KDEs")
    lines(h2$x, h2$y, col = "blue")
    legend("topright", c("Finite", "Limit"), col = c("red", "blue"), pch = 15)
}

# under alternative
mu[2, ] <- 1:N / sum(1:N)

if (requireNamespace("ROI.plugin.glpk")) {
    solver <- ot_test_lp_solver("glpk")
    set.seed(123)
   lhs <- simulate_finite_ot_barycenter_test(mu, costm, n, num.sim = num.sim, solver = solver)
    rhs <- simulate_limit_ot_barycenter_test_alt(mu, costm, delta, num.sim = num.sim,
                                                 solver = solver)
```

```
    h1 <- density(lhs)
    h2 <- density(rhs)
    plot(h1$x, h1$y, xlim = range(h1$x, h2$x), ylim = range(h1$y, h2$y), type = "l",
         col = "red", xlab = "x", ylab = "density", main = "KDEs")
    lines(h2$x, h2$y, col = "blue")
    legend("topright", c("Finite", "Limit"), col = c("red", "blue"), pch = 15)
}
```

`tab_sample` *Generate tabulated samples from probability vectors*

Description

Generate count vectors instead of samples, i.e., vectors giving the number of times a sample was observed at the respective points.

Usage

```
tab_sample(n, mu, prob = FALSE)
```

Arguments

`n`	vector or nested list of sample sizes.
`mu`	matrix (row-wise) or nested list containing probability vectors to sample from. The structure of `n` and `mu` must be the same.
`prob`	logical value indicating whether probabilities (instead of counts) should be returned.

Value

The count vectors corresponding to the generated samples. Has the same structure as `mu`.

Examples

```
## matrix example

mu <- matrix(c(0.01, 0.99, 0.5, 0.5), 2, 2, TRUE)
n <- c(80, 20)

set.seed(123)
cv <- tab_sample(n, mu)
print(cv)
# sample sizes are rowsums
print(rowSums(cv))
# empirical probability vectors
print(sweep(cv, 1, n, "/"))
set.seed(123)
# same result
```

```
print(tab_sample(n, mu, prob = TRUE))

## list example

mu <- list(
   list(c(0.3, 0.7), c(0.25, 0.75)),
   list(c(0, 1), c(0.5, 0.5))
)
n <- list(list(100, 120), list(80, 90))

set.seed(123)
cv <- tab_sample(n, mu)
print(cv)
# empirical probability vectors
print(rapply(cv, \(x) x / sum(x), how = "replace"))
set.seed(123)
print(tab_sample(n, mu, prob = TRUE))
```

Index