

Package ‘rsamplr’

March 31, 2026

Title Sampling Algorithms and Spatially Balanced Sampling

Version 0.2.0

Description Fast tools for unequal probability sampling in multi-dimensional spaces, implemented in Rust for high performance. The package offers a wide range of methods, including Sampford (Sampford, 1967, <[doi:10.1093/biomet/54.3-4.499](https://doi.org/10.1093/biomet/54.3-4.499)>) and correlated Poisson sampling (Bondeson and Thorburn, 2008, <[doi:10.1111/j.1467-9469.2008.00596.x](https://doi.org/10.1111/j.1467-9469.2008.00596.x)>), pivotal sampling (Deville and Tillé, 1998, <[doi:10.1093/biomet/91.4.893](https://doi.org/10.1093/biomet/91.4.893)>), and balanced sampling such as the cube method (Deville and Tillé, 2004, <[doi:10.1093/biomet/91.4.893](https://doi.org/10.1093/biomet/91.4.893)>) to ensure auxiliary totals are respected. Spatially balanced approaches, including the local pivotal method (Grafström et al., 2012, <[doi:10.1111/j.1541-0420.2011.01699.x](https://doi.org/10.1111/j.1541-0420.2011.01699.x)>), spatially correlated Poisson sampling (Grafström, 2012, <[doi:10.1016/j.jspi.2011.07.003](https://doi.org/10.1016/j.jspi.2011.07.003)>), and locally correlated Poisson sampling (Prentius, 2024, <[doi:10.1002/env.2832](https://doi.org/10.1002/env.2832)>), provide efficient designs when the target variable is linked to auxiliary information.

URL <https://www.envisim.se/>, <https://github.com/envisim/rust-samplr/>

BugReports <https://github.com/envisim/rust-samplr/issues>

License AGPL-3

Encoding UTF-8

Language en-GB

Depends R (>= 4.5)

RoxygenNote 7.3.3

SystemRequirements Cargo (Rust's package manager), rustc >= 1.84.1

NeedsCompilation yes

Author Wilmer Prentius [aut, cre] (ORCID: <<https://orcid.org/0000-0002-3561-290X>>), Anton Grafström [ctb] (ORCID: <<https://orcid.org/0000-0002-4345-4024>>), Authors of the dependent Rust crates [aut] (see inst/AUTHORS file)

Maintainer Wilmer Prentius <wilmer.prentius@slu.se>

Repository CRAN

Date/Publication 2026-03-31 06:00:02 UTC

Contents

.dbd_defaults	2
.sampling_defaults	3
Balanced sampling	3
Distributionally balanced designs	5
Doubly balanced sampling	6
draw	8
Iterative evaluation of distributional designs	9
local_mean_variance	10
pips_from_vector	11
sample_to_indicator	12
Spatial balance measure	13
Spatially balanced sampling	15
Unequal probability sampling	18
Index	21

.dbd_defaults	<i>Distributionally balanced designs defaults</i>
---------------	---

Description

Distributionally balanced designs defaults

Usage

```
.dbd_defaults(
  annealing_temp = 0.1,
  annealing_cooling = 0.999,
  spatial_init = FALSE,
  ...
)
```

Arguments

annealing_temp The initial temperature to use in simulated annealing

annealing_cooling The annealing_cooling rate to use in simulated annealing

spatial_init If TRUE a spatial initialization strategy is used

... Arguments passed on to [.sampling_defaults](#)

eps A small value used when comparing floats.

max_iter The maximum number of iterations used in iterative algorithms.

Value

A validated list of arguments used internally in dbd functions.

`.sampling_defaults` *Sampling defaults*

Description

Sampling defaults

Usage

```
.sampling_defaults(eps = 1e-10, max_iter = 1000L, bucket_size = 50L)
```

Arguments

<code>eps</code>	A small value used when comparing floats.
<code>max_iter</code>	The maximum number of iterations used in iterative algorithms.
<code>bucket_size</code>	The maximum size of the k-d-tree nodes. A higher value gives a slower k-d-tree, but is faster to create and takes up less memory.

Value

A validated list of arguments used internally in sampling functions.

Balanced sampling *Balanced sampling*

Description

Selects balanced samples with prescribed inclusion probabilities from finite populations.

Usage

```
cube(probabilities, balance_mat, ...)  
cube_stratified(probabilities, balance_mat, strata, ...)
```

Arguments

<code>probabilities</code>	A vector of inclusion probabilities.
<code>balance_mat</code>	A matrix of balancing covariates.
<code>...</code>	Arguments passed on to .sampling_defaults
	<code>eps</code> A small value used when comparing floats.
<code>strata</code>	An integer vector with stratum numbers for each unit.

Details

For the cube method, a fixed sized sample is obtained if the first column of `balance_mat` is the inclusion probabilities. For `cube_stratified`, the inclusion probabilities are inserted automatically.

Value

A vector of sample indices.

Functions

- `cube()`: The cube method
- `cube_stratified()`: The stratified cube method

References

Deville, J. C. and Tillé, Y. (2004). Efficient balanced sampling: the cube method. *Biometrika*, 91(4), 893-912.

Chauvet, G. and Tillé, Y. (2006). A fast algorithm for balanced sampling. *Computational Statistics*, 21(1), 53-62.

Chauvet, G. (2009). Stratified balanced sampling. *Survey Methodology*, 35, 115-119.

Examples

```
set.seed(12345);
N = 1000;
n = 100;
prob = rep(n/N, N);
xb = matrix(c(prob, runif(N * 2)), ncol = 3);
strata = c(rep(1L, 100), rep(2L, 200), rep(3L, 300), rep(4L, 400));

s = cube(prob, xb);
plot(xb[, 2], xb[, 3], pch = ifelse(sample_to_indicator(s, N), 19, 1));

s = cube_stratified(prob, xb[, -1], strata);
plot(xb[, 2], xb[, 3], pch = ifelse(sample_to_indicator(s, N), 19, 1));

# Respects inclusion probabilities
set.seed(12345);
prob = c(0.2, 0.25, 0.35, 0.4, 0.5, 0.5, 0.55, 0.65, 0.7, 0.9);
N = length(prob);
xb = matrix(c(prob, runif(N * 2)), ncol = 3);

ep = rep(0L, N);
r = 10000L;

for (i in seq_len(r)) {
  s = cube(prob, xb);
  ep[s] = ep[s] + 1L;
}
```

```
print(ep / r - prob);
```

Distributionally balanced designs

Distributionally balanced designs

Description

Construct distributionally balanced sampling designs.

Usage

```
dbd_circular(sample_size, spread_mat, ...)
```

```
dbd_tc(sample_size, spread_mat, ...)
```

Arguments

<code>sample_size</code>	The size of the desired sample.
<code>spread_mat</code>	A matrix of spreading covariates.
<code>...</code>	Arguments passed on to <code>.dbd_defaults</code>
<code>annealing_temp</code>	The initial temperature to use in simulated annealing
<code>annealing_cooling</code>	The annealing_cooling rate to use in simulated annealing
<code>spatial_init</code>	If TRUE a spatial initialization strategy is used

Details

Use the `draw.dbd()` method to draw or select a single sample from the design.

Value

A `dbd` object containing all possible samples.

Functions

- `dbd_circular()`: Distributionally Balanced Sampling as circular sequence
- `dbd_tc()`: Distributionally Balanced Sampling using Tactical Configuration

References

Grafström, A., & Prentius, W. (2026). Distributionally balanced sampling designs. arXiv preprint arXiv:2603.11916.

Grafström, A., & Prentius, W. (2026). Distributionally balanced sampling designs via minimum tactical configurations. arXiv preprint arXiv:2603.24439.

Examples

```

set.seed(12345);
N = 1000L;
n = 100L;
prob = rep(n / N, N);
xs = matrix(runif(N * 2), ncol = 2);

# Construct circular dbd design
design = dbd_circular(
  n,
  xs,
  annealing_temp = 0.1,
  annealing_cooling = 0.999,
  max_iter = 1e5L
);
s = draw(design); # Draw sample from design
plot(xs[, 1], xs[, 2], pch = ifelse(sample_to_indicator(s, N), 19, 1));

# Construct dbd design using tactical configuration
design = dbd_tc(
  n,
  xs,
  annealing_temp = 0.1,
  annealing_cooling = 0.999,
  max_iter = 1e5L
);
s = draw(design); # Draw sample from design
plot(xs[, 1], xs[, 2], pch = ifelse(sample_to_indicator(s, N), 19, 1));

```

Doubly balanced sampling

Doubly balanced sampling

Description

Selects doubly balanced samples with prescribed inclusion probabilities from finite populations.

Usage

```
local_cube(probabilities, spread_mat, balance_mat, ...)
```

```
local_cube_stratified(probabilities, spread_mat, balance_mat, strata, ...)
```

Arguments

`probabilities` A vector of inclusion probabilities.
`spread_mat` A matrix of spreading covariates.

balance_mat	A matrix of balancing covariates.
...	Arguments passed on to <code>.sampling_defaults</code>
eps	A small value used when comparing floats.
bucket_size	The maximum size of the k-d-tree nodes. A higher value gives a slower k-d-tree, but is faster to create and takes up less memory.
strata	An integer vector with stratum numbers for each unit.

Details

For the `local_cube` method, a fixed sized sample is obtained if the first column of `balance_mat` is the inclusion probabilities. For `local_cube_stratified`, the inclusion probabilities are inserted automatically.

Value

A vector of sample indices.

Functions

- `local_cube()`: The local cube method
- `local_cube_stratified()`: The stratified local cube method

References

- Deville, J. C. and Tillé, Y. (2004). Efficient balanced sampling: the cube method. *Biometrika*, 91(4), 893-912.
- Chauvet, G. and Tillé, Y. (2006). A fast algorithm for balanced sampling. *Computational Statistics*, 21(1), 53-62.
- Chauvet, G. (2009). Stratified balanced sampling. *Survey Methodology*, 35, 115-119.
- Grafström, A. and Tillé, Y. (2013). Doubly balanced spatial sampling with spreading and restitution of auxiliary totals. *Environmetrics*, 24(2), 120-131

Examples

```
set.seed(12345);
N = 1000;
n = 100;
prob = rep(n/N, N);
xb = matrix(c(prob, runif(N * 2)), ncol = 3);
xs = matrix(runif(N * 2), ncol = 2);
strata = c(rep(1L, 100), rep(2L, 200), rep(3L, 300), rep(4L, 400));

s = local_cube(prob, xs, xb);
plot(xs[, 1], xs[, 2], pch = ifelse(sample_to_indicator(s, N), 19, 1));

s = local_cube_stratified(prob, xs, xb[, -1], strata);
plot(xs[, 1], xs[, 2], pch = ifelse(sample_to_indicator(s, N), 19, 1));
```

```

# Respects inclusion probabilities
set.seed(12345);
prob = c(0.2, 0.25, 0.35, 0.4, 0.5, 0.5, 0.55, 0.65, 0.7, 0.9);
N = length(prob);
xb = matrix(c(prob, runif(N * 2)), ncol = 3);
xs = matrix(runif(N * 2), ncol = 2);

ep = rep(0L, N);
r = 10000L;

for (i in seq_len(r)) {
  s = local_cube(prob, xs, xb);
  ep[s] = ep[s] + 1L;
}

print(ep / r - prob);

```

draw

Draw from a sampling design

Description

Draw from a sampling design

Draw a sample from a distributionally balanced design

Usage

```
draw(design, ...)
```

```
## S3 method for class 'dbd'
draw(design, ...)
```

Arguments

design	A dbd design object.
...	Additional arguments passed to methods. draw.dbd accepts sample_id (integer). If sample_id is 0L, returns a random sample (default).

Value

a vector of sample indices.

Functions

- draw(): draw generic

Examples

```
set.seed(12345);
N = 1000L;
n = 100L;
prob = rep(n / N, N);
xs = matrix(runif(N * 2), ncol = 2);

# Construct dbd design using tactical configuration
design = dbd_tc(
  n,
  xs,
  annealing_temp = 0.1,
  annealing_cooling = 0.999,
  max_iter = 1e5L
);
s = draw(design); # Draw sample from design
plot(xs[, 1], xs[, 2], pch = ifelse(sample_to_indicator(s, N), 19, 1));
```

Iterative evaluation of distributional designs

Iterative evaluation of distributional designs

Description

Evaluates the energy distance along a simulated annealing scheme.

Usage

```
dbd_circular_iter(sample_size, spread_mat, ...)
```

```
dbd_tc_iter(sample_size, spread_mat, ...)
```

Arguments

<code>sample_size</code>	The size of the desired sample.
<code>spread_mat</code>	A matrix of spreading covariates.
<code>...</code>	Additional arguments, see <code>.dbd_defaults()</code> . Note: <ul style="list-style-type: none">• <code>max_iter</code> determines the number of iterations to run;• <code>iter_by</code> determines the reporting interval.

Value

A matrix with the average energies and the standard deviations of the energies as columns, per the reporting intervals.

Functions

- `dbd_tc_iter()`: Iterative evolution of distributional design using Tactical Configuration

Examples

```

set.seed(12345);
N = 1000L;
n = 100L;
prob = rep(n / N, N);
xs = matrix(runif(N * 2), ncol = 2);

# Report mean and sd energy at every 1e4L iteration
dbd_circular_iter(
  n,
  xs,
  annealing_temp = 0.1,
  annealing_cooling = 0.999,
  max_iter = 1e5L,
  iter_by = 1e4L
);

# Report mean and sd energy at every 1e4L iteration
dbd_tc_iter(
  n,
  xs,
  annealing_temp = 0.1,
  annealing_cooling = 0.999,
  max_iter = 1e5L,
  iter_by = 1e4L
);

```

local_mean_variance *Variance estimator for spatially balanced samples*

Description

Variance estimator of HT estimator of population total.

Usage

```
local_mean_variance(values, probabilities, spread_mat, neighbours = 4L)
```

Arguments

values	A vector of values of the variable of interest.
probabilities	A vector of inclusion probabilities.
spread_mat	A matrix of spreading covariates.
neighbours	The number of neighbours to construct the means around.

Value

A vector of sample indices.

References

Grafström, A., & Schelin, L. (2014). How to select representative samples. *Scandinavian Journal of Statistics*, 41(2), 277-290.

Examples

```
set.seed(12345);
N = 1000;
n = 100;
prob = rep(n/N, N);
xs = matrix(runif(N * 2), ncol = 2);
y = runif(N);

s = lpm_2(prob, xs);
local_mean_variance(y[s], prob[s], xs[s, ], 4);

# Compare SRS, empirical
r = 1000L;
v = matrix(0.0, r, 3L);

for (i in seq_len(r)) {
  s = lpm_2(prob, xs);
  v[i, 1] = local_mean_variance(y[s], prob[s], xs[s, ], 4);
  v[i, 2] = N^2 * sd(y[s]) / n;
  v[i, 3] = sum(y[s] / prob[s]);
}

# Local mean variance, SRS variance, MSE
print(c(mean(v[, 1]), mean(v[, 2]), mean((v[, 3] - sum(y))^2)));
```

pips_from_vector

Inclusion probabilities proportional-to-size

Description

Computes the first-order inclusion probabilities from a vector of positive numbers, for an inclusion probabilities proportional-to-size design.

Usage

```
pips_from_vector(values, sample_size)
```

Arguments

values A vector of positive numbers
sample_size The wanted sample size

Value

A vector of inclusion probabilities proportional-to-size.

Examples

```
set.seed(12345);  
N = 1000;  
n = 100;  
x = matrix(runif(N * 2), ncol = 2);  
prob = pips_from_vector(x[, 1], n);  
s = lpm_2(prob, x);  
plot(x[, 1], x[, 2], pch = ifelse(sample_to_indicator(s, N), 19, 1));
```

sample_to_indicator *Transform a sample vector into an inclusion indicator vector*

Description

Transform a sample vector into an inclusion indicator vector

Usage

```
sample_to_indicator(sample, population_size)
```

Arguments

sample A vector of sample indices.
population_size The total size of the population.

Value

An inclusion indicator vector, i.e. a population_size-sized vector of 0/1.

Examples

```
s = c(1, 2, 10);  
si = sample_to_indicator(s, 10);
```

Spatial balance measure

Spatial balance measure

Description

Calculates the spatial balance of a sample.

Usage

```
spatial_balance_local(  
  sample,  
  probabilities,  
  spread_mat,  
  balance_probabilities = TRUE  
)
```

```
spatial_balance_local_equal(  
  sample,  
  spread_mat,  
  sample_size = length(sample),  
  balance_probabilities = TRUE  
)
```

```
spatial_balance_voronoi(sample, probabilities, spread_mat)
```

```
spatial_balance_voronoi_equal(sample, spread_mat, sample_size = length(sample))
```

```
spatial_balance_energy(sample, probabilities, spread_mat)
```

```
spatial_balance_energy_equal(sample, spread_mat, sample_size = length(sample))
```

```
spatial_balance_all(sample, probabilities, spread_mat)
```

```
spatial_balance_all_equal(sample, spread_mat, sample_size = length(sample))
```

```
balance_deviation(sample, probabilities, spread_mat)
```

Arguments

<code>sample</code>	A vector of sample indices.
<code>probabilities</code>	A vector of inclusion probabilities.
<code>spread_mat</code>	A matrix of spreading covariates.
<code>balance_probabilities</code>	If true (default), includes the vector of inclusion probabilities as a balancing variable.
<code>sample_size</code>	The sample size

Value

the measure, or in case of `balance_deviation`, the vector of deviations.

Functions

- `spatial_balance_local()`: Local spatial balance
- `spatial_balance_local_equal()`: Local spatial balance
- `spatial_balance_voronoi()`: Voronoi spatial balance
- `spatial_balance_voronoi_equal()`: Voronoi spatial balance
- `spatial_balance_energy()`: Energy spatial balance
- `spatial_balance_energy_equal()`: Energy spatial balance
- `spatial_balance_all()`: Energy spatial balance
- `spatial_balance_all_equal()`: Energy spatial balance
- `balance_deviation()`: Balance deviation

References

Stevens Jr, D. L., & Olsen, A. R. (2004). Spatially balanced sampling of natural resources. *Journal of the American statistical Association*, 99(465), 262-278.

Grafström, A., Lundström, N.L.P. & Schelin, L. (2012). Spatially balanced sampling through the Pivotal method. *Biometrics* 68(2), 514-520.

Prentius, W., & Grafström, A. (2024). How to find the best sampling design: A new measure of spatial balance. *Environmetrics*, 35(7), e2878.

Examples

```
set.seed(12345);
N = 500;
n = 70;
prob = rep(n / N, N);
xs = matrix(runif(N * 2), ncol = 2);

s = lpm_2(prob, xs);
spatial_balance_voronoi(s, prob, xs);
spatial_balance_local(s, prob, xs, TRUE);
spatial_balance_local(s, prob, xs, FALSE);
spatial_balance_energy(s, prob, xs);
balance_deviation(s, prob, xs);

# Compare SRS
r = 1000L;
sb_v = matrix(0.0, r, 2L);
sb_l = matrix(0.0, r, 4L);
sb_e = matrix(0.0, r, 2L);
bal = matrix(0.0, r, 2L * ncol(xs));
```

```

for (i in seq_len(r)) {
  s1 = lpm_2(prob, xs);
  s2 = sample(N, n);
  sb_v[i, ] = c(
    spatial_balance_voronoi(s1, prob, xs),
    spatial_balance_voronoi(s2, prob, xs)
  );
  sb_l[i, ] = c(
    spatial_balance_local(s1, prob, xs, TRUE),
    spatial_balance_local(s2, prob, xs, TRUE),
    spatial_balance_local(s1, prob, xs, FALSE),
    spatial_balance_local(s2, prob, xs, FALSE)
  );
  sb_e[i, ] = c(
    spatial_balance_energy(s1, prob, xs),
    spatial_balance_energy(s2, prob, xs)
  );
  bal[i, ] = c(
    balance_deviation(s1, prob, xs),
    balance_deviation(s2, prob, xs)
  );
}

# Spatial balance measure (voronoi), LPM vs SRS
print(colMeans(sb_v));
# Spatial balance measure (local), LPM vs SRS
print(colMeans(sb_l));
# Spatial balance measure (energy), LPM vs SRS
print(colMeans(sb_e));
# Abs. balance deviation, LPM vs SRS
print(colMeans(abs(bal)));

```

Spatially balanced sampling

Spatially balanced sampling

Description

Selects spatially balanced samples with prescribed inclusion probabilities from finite populations.

Usage

`lpm_1(probabilities, spread_mat, ...)`

`lpm_1s(probabilities, spread_mat, ...)`

`lpm_2(probabilities, spread_mat, ...)`

```
scps(probabilities, spread_mat, ...)
lcps(probabilities, spread_mat, ...)
lpm_2_hierarchical(probabilities, spread_mat, sizes, ...)
```

Arguments

`probabilities` A vector of inclusion probabilities.

`spread_mat` A matrix of spreading covariates.

`...` Arguments passed on to `.sampling_defaults`

`eps` A small value used when comparing floats.

`bucket_size` The maximum size of the k-d-tree nodes. A higher value gives a slower k-d-tree, but is faster to create and takes up less memory.

`sizes` A vector of integers containing the sizes of the subsamples.

Details

`lpm_2_hierarchical` selects an initial sample using the LPM2 algorithm, and then splits this sample into subsamples of given sizes, using successive, hierarchical selection with LPM2. When using `lpm_2_hierarchical`, the inclusion probabilities must sum to an integer, and the sizes vector (the subsamples) must sum to the same integer.

Value

A vector of sample indices, or in the case of hierarchical sampling, a matrix where the first column contains sample indices and the second column contains subsample indices (groups).

Functions

- `lpm_1()`: Local pivotal method 1
- `lpm_1s()`: Local pivotal method 1s
- `lpm_2()`: Local pivotal method 2
- `scps()`: Spatially correlated Poisson sampling
- `lcps()`: Locally correlated Poisson sampling
- `lpm_2_hierarchical()`: Hierarchical Local pivotal method 2

References

- Deville, J.-C., & Tillé, Y. (1998). Unequal probability sampling without replacement through a splitting method. *Biometrika* 85, 89-101.
- Grafström, A. (2012). Spatially correlated Poisson sampling. *Journal of Statistical Planning and Inference*, 142(1), 139-147.
- Grafström, A., Lundström, N.L.P. & Schelin, L. (2012). Spatially balanced sampling through the Pivotal method. *Biometrics* 68(2), 514-520.

Lisic, J. J., & Cruze, N. B. (2016, June). Local pivotal methods for large surveys. In Proceedings of the Fifth International Conference on Establishment Surveys.

Prentius, W. (2024). Locally correlated Poisson sampling. *Environmetrics*, 35(2), e2832.

Examples

```

set.seed(12345);
N = 1000;
n = 100;
prob = rep(n/N, N);
xs = matrix(runif(N * 2), ncol = 2);
sizes = c(10L, 20L, 30L, 40L);

s = lpm_1(prob, xs);
plot(xs[, 1], xs[, 2], pch = ifelse(sample_to_indicator(s, N), 19, 1));

s = lpm_1s(prob, xs);
plot(xs[, 1], xs[, 2], pch = ifelse(sample_to_indicator(s, N), 19, 1));

s = lpm_2(prob, xs);
plot(xs[, 1], xs[, 2], pch = ifelse(sample_to_indicator(s, N), 19, 1));

s = scps(prob, xs);
plot(xs[, 1], xs[, 2], pch = ifelse(sample_to_indicator(s, N), 19, 1));

s = lpm_2_hierarchical(prob, xs, sizes);
plot(xs[, 1], xs[, 2], pch = ifelse(sample_to_indicator(s, N), 19, 1));

s = lcps(prob, xs); # May have a long execution time
plot(xs[, 1], xs[, 2], pch = ifelse(sample_to_indicator(s, N), 19, 1));

# Respects inclusion probabilities
set.seed(12345);
prob = c(0.2, 0.25, 0.35, 0.4, 0.5, 0.5, 0.55, 0.65, 0.7, 0.9);
N = length(prob);
xs = matrix(c(prob, runif(N * 2)), ncol = 3);

ep = rep(0L, N);
r = 10000L;

for (i in seq_len(r)) {
  s = lpm_2(prob, xs);
  ep[s] = ep[s] + 1L;
}

print(ep / r - prob);

```

Unequal probability sampling
Unequal probability sampling

Description

Selects samples with prescribed inclusion probabilities from finite populations.

Usage

```
rpm(probabilities, ...)  
spm(probabilities, ...)  
cps(probabilities, ...)  
poisson(probabilities, ...)  
conditional_poisson(probabilities, sample_size, ...)  
systematic(probabilities, ...)  
systematic_random_order(probabilities, ...)  
brewer(probabilities, ...)  
pareto(probabilities, ...)  
sampford(probabilities, ...)
```

Arguments

<code>probabilities</code>	A vector of inclusion probabilities.
<code>...</code>	Arguments passed on to <code>.sampling_defaults</code>
<code>eps</code>	A small value used when comparing floats.
<code>max_iter</code>	The maximum number of iterations used in iterative algorithms.
<code>sample_size</code>	The wanted sample size

Details

`sampford` and `conditional_poisson` may return an error if a solution is not found within `max_iter`.

Value

A vector of sample indices.

Functions

- rpm(): Random pivotal method
- spm(): Sequential pivotal method
- cps(): Correlated Poisson sampling
- poisson(): Poisson sampling
- conditional_poisson(): Conditional Poisson sampling
- systematic(): Systematic sampling
- systematic_random_order(): Systematic sampling with random order
- brewer(): Brewer sampling
- pareto(): Pareto sampling
- sampford(): Sampford sampling

References

- Bondesson, L., & Thorburn, D. (2008). A list sequential sampling method suitable for real-time sampling. *Scandinavian Journal of Statistics*, 35(3), 466-483.
- Brewer, K. E. (1975). A Simple Procedure for Sampling pi-ps wor. *Australian Journal of Statistics*, 17(3), 166-172.
- Chauvet, G. (2012). On a characterization of ordered pivotal sampling. *Bernoulli*, 18(4), 1320-1340.
- Deville, J.-C., & Tillé, Y. (1998). Unequal probability sampling without replacement through a splitting method. *Biometrika* 85, 89-101.
- Grafström, A. (2009). Non-rejective implementations of the Sampford sampling design. *Journal of Statistical Planning and Inference*, 139(6), 2111-2114.
- Rosén, B. (1997). On sampling with probability proportional to size. *Journal of statistical planning and inference*, 62(2), 159-191.
- Sampford, M. R. (1967). On sampling without replacement with unequal probabilities of selection. *Biometrika*, 54(3-4), 499-513.

Examples

```
set.seed(12345);
N = 1000;
n = 100;
prob = rep(n/N, N);
xs = matrix(runif(N * 2), ncol = 2);

s = rpm(prob);
plot(xs[, 1], xs[, 2], pch = ifelse(sample_to_indicator(s, N), 19, 1));

s = spm(prob);
plot(xs[, 1], xs[, 2], pch = ifelse(sample_to_indicator(s, N), 19, 1));

s = cps(prob);
plot(xs[, 1], xs[, 2], pch = ifelse(sample_to_indicator(s, N), 19, 1));
```

```
s = poisson(prob);
plot(xs[, 1], xs[, 2], pch = ifelse(sample_to_indicator(s, N), 19, 1));

s = brewer(prob);
plot(xs[, 1], xs[, 2], pch = ifelse(sample_to_indicator(s, N), 19, 1));

s = pareto(prob);
plot(xs[, 1], xs[, 2], pch = ifelse(sample_to_indicator(s, N), 19, 1));

s = systematic(prob);
plot(xs[, 1], xs[, 2], pch = ifelse(sample_to_indicator(s, N), 19, 1));

s = systematic_random_order(prob);
plot(xs[, 1], xs[, 2], pch = ifelse(sample_to_indicator(s, N), 19, 1));

# Conditional poisson and sampford are not guaranteed to find a solution
prob2 = rep(0.5, 10L);
s = conditional_poisson(prob2, 5L, max_iter = 10000L);
plot(xs[1:10, 1], xs[1:10, 2], pch = ifelse(sample_to_indicator(s, 10L), 19, 1));

s = sampford(prob2, max_iter = 10000L);
plot(xs[1:10, 1], xs[1:10, 2], pch = ifelse(sample_to_indicator(s, 10L), 19, 1));

# Respects inclusion probabilities
set.seed(12345);
prob = c(0.2, 0.25, 0.35, 0.4, 0.5, 0.5, 0.55, 0.65, 0.7, 0.9);
N = length(prob);

ep = rep(0L, N);
r = 10000L;

for (i in seq_len(r)) {
  s = poisson(prob);
  ep[s] = ep[s] + 1L;
}

print(ep / r - prob);
```

Index

.dbd_defaults, 2, 5
.dbd_defaults(), 9
.sampling_defaults, 2, 3, 3, 7, 16, 18

balance_deviation (Spatial balance measure), 13
Balanced sampling, 3
brewer (Unequal probability sampling), 18

conditional_poisson (Unequal probability sampling), 18
cps (Unequal probability sampling), 18
cube (Balanced sampling), 3
cube_stratified (Balanced sampling), 3

dbd_circular (Distributionally balanced designs), 5
dbd_circular_iter (Iterative evaluation of distributional designs), 9
dbd_tc (Distributionally balanced designs), 5
dbd_tc_iter (Iterative evaluation of distributional designs), 9
Distributionally balanced designs, 5
Doubly balanced sampling, 6
draw, 8
draw.dbd(), 5

Iterative evaluation of distributional designs, 9

lcps (Spatially balanced sampling), 15
local_cube (Doubly balanced sampling), 6
local_cube_stratified (Doubly balanced sampling), 6
local_mean_variance, 10
lpm_1 (Spatially balanced sampling), 15
lpm_1s (Spatially balanced sampling), 15
lpm_2 (Spatially balanced sampling), 15
lpm_2_hierarchical (Spatially balanced sampling), 15

pareto (Unequal probability sampling), 18
pips_from_vector, 11
poisson (Unequal probability sampling), 18

rpm (Unequal probability sampling), 18

sampford (Unequal probability sampling), 18
sample_to_indicator, 12
scps (Spatially balanced sampling), 15
Spatial balance measure, 13
spatial_balance_all (Spatial balance measure), 13
spatial_balance_all_equal (Spatial balance measure), 13
spatial_balance_energy (Spatial balance measure), 13
spatial_balance_energy_equal (Spatial balance measure), 13
spatial_balance_local (Spatial balance measure), 13
spatial_balance_local_equal (Spatial balance measure), 13
spatial_balance_voronoi (Spatial balance measure), 13
spatial_balance_voronoi_equal (Spatial balance measure), 13
Spatially balanced sampling, 15
spm (Unequal probability sampling), 18
systematic (Unequal probability sampling), 18
systematic_random_order (Unequal probability sampling), 18

Unequal probability sampling, 18