Package: spbal (via r-universe)

October 30, 2026

Type Package

Title Spatially Balanced Sampling Algorithms

Version 1.1.0

Description Encapsulates a number of spatially balanced sampling algorithms, namely, Balanced Acceptance Sampling (equal, unequal, seed point, panels), Halton frames (for discretizing a continuous resource), Halton Iterative Partitioning (equal probability), Simple Random Sampling, Dynamic Assignment Sampling and Doubly balanced Dynamic Assignment Sampling. Robertson, B. L., Brown, J. A., McDonald, T. and Jaksons, P. (2013) <doi:10.1111/biom.12059>. Robertson, B. L., McDonald, T., Price, C. J. and Brown, J. A. (2017) <doi:10.1016/j.spl.2017.05.004>. Robertson, B. L., McDonald, T., Price, C. J. and Brown, J. A. (2018) <doi:10.1007/s10651-018-0406-6>. Robertson, B. L., van Dam-Bates, P. and Gansell, O. (2021a) <doi:10.1007/s10651-020-00481-1>. Robertson, B. L., Davies, P., Gansell, O., van Dam-Bates, P., McDonald, T. (2025) <doi:10.1111/anzs.12435>. Robertson, B.L., Price, C.J., Reale, M., Davies, P. (2026) <doi:10.1093/biomtc/ujag011>. Robertson, B.L., Price, C.J., Reale, M., Davies, P. (2026) <doi:10.1093/jssam/smag021>.

Depends R (>= 3.6.0)

License MIT + file LICENSE

Encoding UTF-8

Imports units, sf, Rcpp (>= 1.0.0)

SystemRequirements C++17

Suggests knitr, rmarkdown, testthat (>= 3.0.0), bookdown, ggplot2, gridExtra, reshape2

VignetteBuilder knitr

LinkingTo Rcpp, RcppArmadillo, RcppThread

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Phil Davies [aut, cre], Blair Robertson [aut], Paul van Dam-Bates [aut], Oliver Gansell [aut]

Maintainer Phil Davies <philip.davies@canterbury.ac.nz>

Repository https://cran.r-universe.dev

Date/Publication 2026-10-30 14:28:22 UTC

RemoteUrl https://github.com/cran/spbal

RemoteRef HEAD

RemoteSha 067f8a2edb5cf0098bc4b9a7efec0207a9ed0409

Contents

arma_dist_al . . . 2
BAS . . . 3
BoundingBox . . . 5
cppBASpts . . . 6
cppBASptsIndexed . . . 7
cppRSHalton_br . . . 8
DAS . . . 9
DoubleAssignmentProblem_cpp . . . 10
doubleDAS . . . 11
getPanel . . . 12
getSample . . . 14
HaltonFrame . . . 15
HIP . . . 17
LinearAssignmentProblem_cpp . . . 19
native_lapjv . . . 20
SRS . . . 20

Index **22**

`arma_dist_al` *Calculate a distance matrix.*

Description

Computes a distance matrix for the supplied matrix of n-dimensional points using Euclidean distance.

Usage

```
arma_dist_al(X)
```

Arguments

X	An n-dimensional matrix of points.

Value

The distance matrix for the point matrix X.

Author(s)

Phil Davies.

Examples

```
# Distance matrix for the two 3D points.
spbal::arma_dist_al(X = rbind(c(1, 2, 3), c(4, 3, 2)))
# Distance matrix for three 2D points.
spbal::arma_dist_al(X = matrix(c(2, 3, 0, 9, 4, 5), nrow = 3, byrow = TRUE))
```

BAS *Balanced Acceptance Sampling (BAS).*

Description

BAS draws spatially balanced samples from areal resources. To draw BAS samples, spbal requires a study region shapefile and the region's bounding box. An initial sample size is also needed, which can be easily increased or decreased within spbal for master sampling applications

Usage

```
BAS(
  shapefile = NULL,
  n = 100,
  boundingbox = NULL,
  minRadius = NULL,
  panels = NULL,
  panel_overlap = NULL,
  stratum = NULL,
  seeds = NULL,
  verbose = FALSE
)
```

Arguments

shapefile	Shape file as a polygon (sp or sf) to select sites for.
n	Number of sites to select. If using stratification it is a named vector containing sample sizes of each group.
boundingbox	Bounding box around the study area. If a bounding box is not supplied then spbal will generate a bounding box for the shapefile.
minRadius	If specified, the minimum distance, in meters, allowed between sample points. This is applied to the $sample points. Points that meet the minRadius criteria are retuned in the minRadius output variable.
panels	A list of integers that define the size of each panel in a non-overlapping panels design. The length of the list determines the number of panels required. The sum of the integers in the panels parameter will determine the total number of samples selected, n. The default value for panels is NULL, this indicates that a non-overlapping panel design is not wanted.
panel_overlap	A list of integers that define the overlap into the previous panel. Is only used when the panels parameter is not NULL. The default value for panel_overlap is NULL. The length of panel_overlap must be equal to the length of panels. The first value is always forced to zero as the first panel never overlaps any region.
stratum	The name of a column in the data.frame attached to shapefile that defines the strata of interest.
seeds	A vector of 2 seeds, u1 and u2. If not specified, the default is NULL and will be defined randomly using function `generateUVector`.
verbose	Boolean if you want to see any output printed to screen. Helpful if taking a long time. Default is FALSE i.e. no informational messages are displayed.

Value

A list containing three variables, `$sample` containing locations in the BAS sample, in BAS order, `$seeds`, the u1 and u2 seeds used to generate the sample and `$minRadius` containing points from $sample that meet the minRadius criteria. If the minRadius parameter is NULL then the $minRadius returned will also be NULL.

The sample points are returned in the form of a simple feature collection of POINT objects. They have the following attributes:

- `SiteID` A unique identifier for every sample point. This encodes the BAS order.
- `spbalSeqID` A unique identifier for every sample point. This encodes the BAS sample order.
- `geometry` The XY co-ordinates of the sample point in the CRS of the original shapefile.

Author(s)

This function was first written by Paul van Dam-Bates for the package BASMasterSample and later simplified by Phil Davies.

Examples

```
# Equal probability BAS sample ------------------------------------------------

# Use the North Carolina shapefile supplied in the sf R package.
shp_file <- sf::st_read(system.file("shape/nc.shp", package="sf"))
shp_gates <- shp_file[shp_file$NAME == "Gates",]

# Vertically aligned master sample bounding box.
bb <- spbal::BoundingBox(shapefile = shp_gates)

set.seed(511)
n_samples <- 20
# Equal probability BAS sample.
result <- spbal::BAS(shapefile = shp_gates,
                     n = n_samples,
                     boundingbox = bb)
BAS20 <- result$sample
# display first three sample points.
BAS20[1:3,]

# Increase the BAS sample size ------------------------------------------------
n_samples <- 50
result2 <- spbal::BAS(shapefile = shp_gates,
                      n = n_samples,
                      boundingbox = bb,
                      seeds = result$seed)
BAS50 <- result2$sample
BAS50[1:3,]

# Check, first n_samples points in both samples must be the same.
all.equal(BAS20$geometry, BAS50$geometry[1:20])
```

BoundingBox *Create a bounding box for a study region.*

Description

Randomly generate a seed from 10,000 possible values in right now 2 dimensions. Note that in van Dam-Bates et al. (2018) we required that the random seed falls into main object shape, such as one of the islands in New Zealand, or within marine environment for BC west coast. However, with a random rotation, we are able to ignore that detail. If this function is used without a random rotation, we recommend running it until the first master sample point does indeed fall within the largest scale of the master sample use.

Usage

```
BoundingBox(shapefile, d = 2, rotate = FALSE, verbose = FALSE)
```

Arguments

shapefile	Spatial feature that defines the boundary of the area to define a bounding box over.
d	Dimension of the new Master Sample, at this stage we only work with d=2.
rotate	Boolean of whether or not to randomly rotate the bounding box. This parameter is not supported at this time.
verbose	Print the rotation and random seed when it is generated.

Value

bounding box for a study area.

Author(s)

This function was first written by Paul van Dam-Bates for the package BASMasterSample and later ported to this package, spbal.

Examples

```
# Create a bounding box for the Gates, North Carolina study area -------------
# Use the North Carolina shapefile supplied in the sf R package.
shp_file <- sf::st_read(system.file("shape/nc.shp", package="sf"))
shp_gates <- shp_file[shp_file$NAME == "Gates",]
# Vertically aligned master sample bounding box.
bb <- spbal::BoundingBox(shapefile = shp_gates)
bb
```

---

cppBASpts *Generate numbers from a Halton Sequence.*

Description

For efficiency, this function can generate points along a random start Halton Sequence for a predefined Halton.

Usage

```
cppBASpts(
  n = 10L,
  seeds = as.integer(c()),
  bases = as.numeric(c()),
  verbose = FALSE
)
```

Arguments

n	Number of points required.
seeds	Random starting point in each dimension.
bases	Co-prime base for the Halton Sequence.
verbose	A boolean indicating whether informational messages are to be issued.

Value

Matrix with the columns, order of points, x in [0,1) and y in [0,1)

Author(s)

This function was first written in R by Blair Robertson, subsequently it was re-written in C/C++ by Phil Davies.

Examples

```
# First 10 points in the Halton Sequence for base 2,3
spbal::cppBASpts(n = 10)
# First 10 points in the Halton Sequence for base 2,3 with
# starting point at the 15th and 22nd index.
spbal::cppBASpts(n = 10, seeds = c(14, 21))
```

---

cppBASptsIndexed *Generate numbers from a Halton Sequence along a specified set of indices.*

Description

For efficiency, this function can generate points along a random start Halton Sequence for a pre-defined set of indices away from the seed. When boxes are provided it will calculate the Halton Sequence only at those boxes and not along the entire sequence.

Usage

```
cppBASptsIndexed(
  n = 10L,
  seeds = as.integer(c()),
  bases = as.numeric(c()),
  boxes = as.integer(c()),
  verbose = FALSE
)
```

Arguments

n	Number of points required.
seeds	Random starting point in each dimension.
bases	Co-prime base for the Halton Sequence.
boxes	Integer vector of indices to sample along the Halton sequence (default 1:n).
verbose	A boolean indicating whether informational messages are to be issued.

Details

When not all points along the Halton sequence are required, this function efficiently generates the points that are needed along a sequence. Taking all points from the random seed equates to boxes = 1:n. However, taking advantage of how the Halton Sequence repeats itself by B = prod(base^J), where J is an integer. We can also select every Bth box to efficiently generate values at specific locations along the sequence. This reduces future computation when bounding boxes are large in comparison to the polygon being sampled.

Value

Matrix with the columns, order of points, x in [0,1) and y in [0,1)

Author(s)

Phil Davies, Paul van Dam-Bates, Blair Robertson.

Examples

```
# First 10 points in the Halton Sequence for base 2,3
spbal::cppBASptsIndexed(n = 10)
# First 10 points in the Halton Sequence for base 2,3 with
# starting point at the 15th and 22nd index.
spbal::cppBASptsIndexed(n = 10, seeds = c(14, 21))
```

---

cppRSHalton_br *Generate numbers from a Halton Sequence with a random start*

Description

For efficiency, this function can generate points along a random start Halton Sequence for a predefined Halton.

Usage

```
cppRSHalton_br(
  n = 10L,
  bases = as.numeric(c()),
  seeds = as.numeric(c()),
  verbose = FALSE
)
```

Arguments

n	Number of points required
bases	Co-prime base for the Halton Sequence
seeds	Random starting point in each dimension
verbose	A boolean indicating whether informational messages are to be issued.

Value

Matrix with the columns, order of point, x in [0,1) and y in [0,1).

Author(s)

This function was first written in R by Blair Robertson, subsequently it was written in C/C++ by Phil Davies.

Examples

```
# First 10 points in the Halton Sequence for base 2,3
 spbal::cppRSHalton_br(n = 10)
# First 10 points in the Halton Sequence for base 2,3 with
# starting point at the 15th and 22nd index.
 spbal::cppRSHalton_br(n = 10, seeds = c(14, 21))
```

DAS *Dynamic Assignment Sampling (DAS).*

Description

DAS draws well-spread master samples over an auxiliary space (Robertson, Price and Reale 2024). Each row of the returned matrix is a candidate sample of size n; the first k columns of any row are a size-k DAS sample.

Usage

```
DAS(pop, n, J1 = NULL, n_threads = 1, verbose = FALSE, cache_W = NULL)
```

Arguments

pop	Numeric matrix of population coordinates, one unit per row.
n	The number of sample points to draw from the population.
J1	Maximum number of candidate samples, `2 <= J1 <= floor(N/2)`. Default is `min(200, floor(N/2))`. Values above 500 are allowed but trigger a warning (large assignment problems dominate run time).
n_threads	Threads used when filling the cost matrix. `0` uses every logical CPU. Parallel fill runs when $J^2 i$ is at least 2048 (so `J1 = 250` does use extra cores). Default is 1.

verbose	Print step timings. Default is FALSE.
cache_W	If TRUE, build one $N \times N$ inverse-distance matrix in C++. If FALSE, compute inverse distances on the fly. Default is TRUE when N <= 2500, otherwise FALSE. At large N the cached matrix is RAM-bound and usually slower.

Value

An integer matrix with one candidate sample per row (1-based population indices).

Author(s)

This function was first written by Blair Robertson in Matlab and later re-written in R/C++ by Phil Davies.

Examples

```
set.seed(511)
pop <- matrix(runif(200), ncol = 2)
sampMx <- spbal::DAS(pop = pop, n = 15)
sampMx[1:3, ]
```

DoubleAssignmentProblem_cpp

Grow a doubly balanced DAS master sample.

Description

Port of MATLAB doubleDAS: successive assignment with cost $\alpha SB + (1 - \alpha)B$, where SB is the inverse-distance spread used by DAS and B is the squared auxiliary-sum balance term. Uses the same candidate growth, cached/on-the-fly weights, and Jonker-Volgenant solver as LinearAssignmentProblem_cpp.

Usage

```
DoubleAssignmentProblem_cpp(
  pop,
  aux,
  alpha,
  target_n,
  initial_J,
  n_threads = 1L,
  verbose = FALSE,
  cache_W = TRUE
)
```

Arguments

pop	Population coordinates, N rows by q columns.
aux	Auxiliary matrix, N rows.
alpha	Mix of spread and balance, in $[0, 1]$. 1 is spatially balanced, 0 is approximately balanced.
target_n	Sample size.
initial_J	Number of candidate samples at the first iteration.
n_threads	Maximum threads for the cost-matrix fill.
verbose	When TRUE, print step timings.
cache_W	When TRUE, build one N by N inverse-distance matrix.

Value

Integer matrix of candidate samples (1-based population indices).

Author(s)

Blair Robertson (Matlab), Phil Davies (R/C++).

---

doubleDAS *Doubly balanced Dynamic Assignment Sampling (doubleDAS).*

Description

Port of the MATLAB doubleDAS sampler. Candidate samples grow by successive linear assignment with cost

$$C = \alpha\, SB + (1 - \alpha)\, B$$

where SB is the inverse-distance spread used by DAS (MATLAB SBMoranIvec) and B is the squared sum of auxiliary variables after appending a candidate unit (MATLAB LinearAssignment).

alpha = 1 is spatially balanced (same objective as DAS). alpha = 0 is approximately balanced on aux. Values in between are doubly balanced.

Reuses arma_dist_al weights, the DAS candidate-growth schedule, and native_lapjv (Jonker-Volgenant) in place of MATLAB matchpairs.

Usage

```
doubleDAS(
  pop,
  aux,
  n,
  alpha = 0.5,
  J1 = NULL,
  n_threads = 1,
  verbose = FALSE,
  cache_W = NULL
)
```

Arguments

pop	Numeric matrix of population coordinates, one unit per row.
aux	Numeric matrix of auxiliary variables, one row per unit.
n	Sample size.
alpha	Mix in $[0, 1]$. Default 0.5.
J1	Maximum number of candidate samples, `2 <= J1 <= floor(N/2)`. Default `min(500, floor(N/2))`, matching the MATLAB.
n_threads	Threads for the cost-matrix fill. See DAS.
verbose	Print step timings. Default FALSE.
cache_W	Cache the $N \times N$ inverse-distance matrix when `alpha > 0`. Default `N <= 2500`. At large `N` this is usually slower than on-the-fly distances.

Value

Integer matrix of candidate samples (1-based population indices). Each row is a sample of size `n`; the first `k` columns of any row are a size-`k` doubleDAS sample.

Author(s)

Blair Robertson (Matlab); R/C++ port by Phil Davies.

Examples

```
set.seed(511)
pop <- matrix(runif(200), ncol = 2)
aux <- matrix(runif(300), ncol = 3)
samp <- spbal::doubleDAS(pop, aux, n = 12, alpha = 0.5)
samp[1, ]
```

---

getPanel *Extract all points with a specified panel id from a sample.*

Description

This is the main function for selecting sites using the BAS master sample. It assumes that you have already defined the master sample using the BoundingBox() function or will be selecting a marine master sample site in BC.

Usage

```
getPanel(shapefile, panelid)
```

Arguments

shapefile	Shape file as a polygon (sp or sf) containing a sample that contains a feature column named panel_id.
panelid	The overlapped panel in the shapefile shp the user wants sample points from.

Value

The sample for the specified panel.

Author(s)

Phil Davies.

Examples

```
# Halton frame overlapping panel design showing use of getPanel.

# Use the North Carolina shapefile supplied in the sf R package.
shp_file <- sf::st_read(system.file("shape/nc.shp", package="sf"))
shp_gates <- shp_file[shp_file$NAME == "Gates",]

# Vertically aligned master sample bounding box.
bb <- spbal::BoundingBox(shapefile = shp_gates)

# Three panels, of 20 samples each.
panels <- c(20, 20, 20)

# second panel overlaps first panel by 5, and third panel
# overlaps second panel by 5.
panel_overlap <- c(0, 5, 5)

# generate the sample.
samp <- spbal::HaltonFrame(J = c(4, 3),
                           boundingbox = bb,
                           panels = panels,
                           panel_overlap = panel_overlap,
                           shapefile = shp_gates)

# get halton frame data from our sample.
samp3 <- samp$hf.pts.shp
samp3

panelid <- 1
olPanel_1 <- spbal::getPanel(samp3, panelid)
```

getSample *Extract a sample of a specified size from a master sample.*

Description

A description of this useful function.

Usage

```
getSample(shapefile, n, randomStart = FALSE, strata = NULL, stratum = NULL)
```

Arguments

shapefile	A MULTIPOINT or POINT object from where to take the sample.
n	The number of sample points to return.
randomStart	Whether a spatially balanced sample will be randomly drawn from the frame or not. Default value is FALSE.
strata	to be added
stratum	The name of a column in the dataframe attached to shapefile that defines the strata of interest.

Value

A list containing the following variable:

- sample The sample from the shapefile POINTS.

Author(s)

Phil Davies.

Examples

```
# Draw a spatially balanced sample of n = 25 from a Halton Frame over Gates --

# Use the North Carolina shapefile supplied in the sf R package.
shp_file <- sf::st_read(system.file("shape/nc.shp", package="sf"))
shp_gates <- shp_file[shp_file$NAME == "Gates",]

# Vertically aligned master sample bounding box.
bb <- spbal::BoundingBox(shapefile = shp_gates)

set.seed(511)
result7 <- spbal::HaltonFrame(shapefile = shp_gates,
                              J = c(6, 4),
                              boundingbox = bb)
Frame <- result7$hf.pts.shp
```

```
# Get the first 25 sites from a B = (2^6) * (3^4) Halton Frame (62,208 grid
# points covering Gates).
n_samples <- 25
FrameSample <-getSample(shapefile = Frame,
                        n = n_samples)
FrameSample <- FrameSample$sample
FrameSample
```

`HaltonFrame` *Create a Halton Frame.*

Description

Halton frames discretize an areal resource into a spatially ordered grid, where samples of consecutive frame points are spatially balanced. To generate Halton Frames, spbal requires a study region `shapefile` and the region's `bounding box`.

Usage

```
HaltonFrame(
  N = 1,
  J = base::c(3, 2),
  bases = base::c(2, 3),
  boundingbox = NULL,
  shapefile = NULL,
  panels = NULL,
  panel_overlap = NULL,
  seeds = NULL,
  stratum = NULL,
  verbose = FALSE
)
```

Arguments

`N`	The number of points in the frame to generate.
`J`	The number of grid cells. A list of 2 values. The default value is c(3, 2).
`bases`	Co-prime base for the Halton Sequence. The default value is c(2, 3).
`boundingbox`	Bounding box around the study area. If a bounding box is not supplied then spbal will generate a bounding box for the shapefile.
`shapefile`	A sf object. If the shapefile parameter is NULL then function HaltonFrameBase is called directly.
`panels`	A list of integers that define the size of each panel in a non-overlapping panels design. The length of the list determines the number of panels required. The sum of the integers in the panels parameter will determine the total number of samples selected, n. The default value for panels is NULL, this indicates that a non-overlapping panel design is not wanted.

panel_overlap	A list of integers that define the overlap into the previous panel. Is only used when the panels parameter is not NULL. The default value for panel_overlap is NULL. The length of panel_overlap must be equal to the length of panels. The first value is always forced to zero as the first panel never overlaps any region.
seeds	A vector of 2 seeds, u1 and u2. If not specified, the default is NULL.
stratum	Name of column in shapefile that makes up the strata.
verbose	Boolean if you want to see any output printed to screen. Helpful if taking a long time. Default is FALSE i.e. no informational messages are displayed.

Value

Returns a list containing five variables:

- J The number of grid cells. A list of 2 values that were used to generate this Halton grid and frame.
- hg.pts.shp Halton grid over the bounding box and study area.
- hf.pts.shp Halton frame, the sample points within the study area.
- bb The bounding box.
- seeds The u1 and u2 seeds used to generate the sample.

The sample points in hf.pts.shp are returned in the form of a simple feature collection of POINT objects. As well as having the features from the original shapefile, the following new attributes have been added:

- spbalSeqID: A unique identifier for every sample point.
- ID: A unique identifier, the Halton frame point order.

Author(s)

Phil Davies.

Examples

```
# we discretize the Gates study region into a coarse grid using
# B = 2^{J_1} * 3^{J_2}= (2^3) * (3^2) (9 by 8 grid) ------------------------

# Use the North Carolina shapefile supplied in the sf R package.
shp_file <- sf::st_read(system.file("shape/nc.shp", package="sf"))
shp_gates <- shp_file[shp_file$NAME == "Gates",]

# Vertically aligned master sample bounding box.
bb <- spbal::BoundingBox(shapefile = shp_gates)

set.seed(511)
result6 <- spbal::HaltonFrame(shapefile = shp_gates,
                              J = c(3, 2),
                              boundingbox = bb)
# get the frame points.
Frame <- result6$hf.pts.shp
```

```
Frame
# get the grid points.
Grid <- result6$hg.pts.shp
Grid
```

HIP *Halton Iterative Partitioning (HIP).*

Description

HIP draws spatially balanced samples and over-samples from point resources by partitioning the resource into boxes with the same nested structure as Halton boxes. The **spbal** parameter **iterations** defines the number of boxes used in the HIP partition and should be larger than the sample size but less than the population size. The **iterations parameter** also defines the number of units available in the HIP over-sample, where the over-sample contains one unit from each box in the HIP partition.

Usage

```
HIP(
  population = NULL,
  n = 20,
  iterations = 7,
  minRadius = NULL,
  panels = NULL,
  panel_overlap = NULL,
  verbose = FALSE
)
```

Arguments

population	A population of point pairs.
n	The number of points to draw from the population. Default 20.
iterations	The levels of partitioning required. Default 7.
minRadius	If specified, the minimum distance, in meters, allowed between sample points. This is applied to the $overSample.
panels	A list of integers that define the size of each panel in a non-overlapping panels design. The length of the list determines the number of panels required. The sum of the integers in the panels parameter will determine the total number of samples selected, n. The default value for panels is NULL, this indicates that a non-overlapping panel design is not wanted.
panel_overlap	A list of integers that define the overlap into the previous panel. Is only used when the panels parameter is not NULL. The default value for panel_overlap is NULL. The length of panel_overlap must be equal to the length of panels. The first value is always forced to zero as the first panel never overlaps any region.
verbose	Boolean if you want to see any output printed to screen. Helpful if taking a long time. Default is FALSE i.e. no informational messages are displayed.

Details

Halton iterative partitioning (HIP) extends Basic acceptance sampling (BAS) to point resources. It partitions the resource into $B >= n$ boxes that have the same nested structure as in BAS, but different sizes. These boxes are then uniquely numbered using a random-start Halton sequence of length B. The HIP sample is obtained by randomly drawing one point from each of the boxes numbered $1, 2, . . . , n$.

Value

Return a list containing the following five variables:

- `Population` Original population point pairs as an sf object.
- `HaltonIndex` The Halton index for the point. Points will be spread equally across all Halton indices.
- `sample` The sample from the population of size n.
- `overSample` The overSample contains one point from each Halton box. All contiguous subsamples from oversample are spatially balanced, and the first n points are identical to sample.
- `minRadius` This result variable contains the sample created using the minRadius parameter. If the minRadius parameter is not specified then the minRadius variable will contain NULL.

Author(s)

Phil Davies, Blair Robertson.

Examples

```
# generating 20 points from a population of 5,000 (random) points with 7
# levels of partitioning (4 in the first dimension and 3 in the second) to
# give (2^4) * (3^3) = 32 * 27, resulting in 864 boxes ----------------------

# set random seed
set.seed(511)

# define HIP parameters.
pop <- matrix(runif(5000*2), nrow = 5000, ncol = 2)
n <- 20
its <- 7

# Convert the population matrix to an sf point object.
sf_points <- sf::st_as_sf(data.frame(pop), coords = c("X1", "X2"))
dim(sf::st_coordinates(sf_points))

# generate HIP sample.
result <- spbal::HIP(population = sf_points,
                     n = n,
                     iterations =  its)

# HaltonIndex
HaltonIndex <- result$HaltonIndex
table(HaltonIndex)
```

```
# Population Sample
HIPsample <- result$sample
HIPsample
```

`LinearAssignmentProblem_cpp` *Grow a DAS master sample by successive linear assignments.*

Description

Implements Algorithm 1 of Robertson, Price and Reale (2024). Costs are inverse Euclidean distances. The unused row-constant $s_j^\top W s_j$ term is omitted. Jonker-Volgenant is called on the Armadillo cost buffer (no wrap through R).

Usage

```
LinearAssignmentProblem_cpp(
  pop,
  target_n,
  initial_J,
  n_threads = 1L,
  verbose = FALSE,
  cache_W = TRUE
)
```

Arguments

`pop`	Population coordinates, N rows by q columns.
`target_n`	The number of sample points required.
`initial_J`	Number of candidate samples at the first iteration.
`n_threads`	Maximum threads for the cost-matrix fill. 0 uses every logical CPU. Parallel fill runs when J^2 * (current sample size) is at least 2048 and n_threads is greater than 1.
`verbose`	When TRUE, print step timings.
`cache_W`	When TRUE, build one N by N inverse-distance matrix in C++. When FALSE, evaluate inverse distances on the fly (less RAM for large N).

Value

Integer matrix of candidate samples (1-based population indices).

Author(s)

Phil Davies.

native_lapjv *Solve the Linear Assignment Problem.*

Description

Solves the linear assignment problem using the Kuhn–Munkres (Hungarian) algorithm. The previous Jonker–Volgenant port could cycle on floating-point DAS costs.

Usage

```
native_lapjv(Cost_R)
```

Arguments

Cost_R	The cost matrix.

Value

A list containing two variables, cost and assignment.

Author(s)

Phil Davies.

Examples

```
# Cost matrix for three 3D points. $cost = 1, $assignment = 2, 1, 0
spbal::native_lapjv(Cost_R = rbind(c(1, 2, 0), c(2, 0, 1), c(1, 4, 19)))
# Cost matrix for three 3D points. $cost = 10, $assignment = 1, 0, 2
spbal::native_lapjv(Cost_R = rbind(c(4, 2, 8), c(2, 3, 7), c(3, 1, 6)))
```

SRS *Simple random sampling.*

Description

This function invokes base::sample() to draw a random sample using a user specified random seed.

Usage

```
SRS(seed = 511, total_rows = 0, sample_size = 0)
```

Arguments

seed	The random seed to be used to draw the current sample.
total_rows	The total number of rows in our input dataset.
sample_size	The number of rows wanted in our random sample.

Details

This function was written by Phil Davies.

Value

A random sample.

Examples

```
# Create a random sample with a seed of 99 ---------------------------------
spbal::SRS(seed = 99, total_rows = 100, sample_size = 20)

# Create a random sample with a seed of 42 ---------------------------------
spbal::SRS(seed = 42, total_rows = 100, sample_size = 20)

# Create a random sample with a seed of 99 ---------------------------------
spbal::SRS(seed = 99, total_rows = 100, sample_size = 25)
```

Index

arma_dist_al, 2

BAS, 3
BoundingBox, 5

cppBASpts, 6
cppBASptsIndexed, 7
cppRSHalton_br, 8

DAS, 9, *11, 12*
DoubleAssignmentProblem_cpp, 10
doubleDAS, 11

getPanel, 12
getSample, 14

HaltonFrame, 15
HIP, 17

LinearAssignmentProblem_cpp, 19

native_lapjv, 20

SRS, 20