Package 'sim2Dpredictr'

October 14, 2022

Title Simulate Outcomes Using Spatially Dependent Design Matrices

Version 0.1.0

Description Provides tools for simulating spatially dependent predictors (continuous or binary), which are used to generate scalar outcomes in a (generalized) linear model framework. Continuous predictors are generated using traditional multivariate normal distributions or Gauss Markov random
fields with several correlation function approaches (e.g., see Rue (2001) <doi:10.1111/1467-9868.00288>
and Furrer and Sain (2010) <doi:10.18637/jss.v036.i10>), while binary predictors are generated using
a Boolean model (see Cressie and Wikle (2011, ISBN: 978-0-471-69274-4)). Parameter vectors exhibiting spatial clustering can also be easily specified by the user.

Depends R (>= 3.5.0)

License GPL-3

Encoding UTF-8

LazyData true

Imports car, ggplot2, MASS, Rdpack, spam (>= 2.2-0), tidyverse, tibble, dplyr, magrittr, matrixcalc

RdMacros Rdpack

RoxygenNote 7.0.2

Suggests knitr, rmarkdown, testthat

URL http://github.com/jmleach-bst/sim2Dpredictr

BugReports http://github.com/jmleach-bst/sim2Dpredictr

NeedsCompilation no

Author Justin Leach [aut, cre, cph]

Maintainer Justin Leach <jleach@uab.edu>

Repository CRAN

Date/Publication 2020-03-14 16:10:02 UTC

R topics documented:

`beta_builder` *Create a Parameter Vector from Lattice Locations*

Description

Specify the locations in the lattice/image that have non-zero parameters as well as the values for those parameters, and the function creates the parameter vector that matches the correct locations in the design matrix.

Usage

```
beta_builder(
  row.index,
  col.index,
  im.res,
  B0 = 0,
  B.values,
  index.type = "manual",
  decay.fn = "gaussian",
  phi = 0.5,
  max.d = Inf,
  h,
  w,
  bayesian = FALSE,
  bayesian.dist = NULL,
  bayesian.scale = NULL,
```

```
  output.indices = TRUE
)
```

Arguments

`row.index, col.index`
: Vectors of row/columns indices for non-zero parameters. If `index.type = "manual"`, each vector should contain specific coordinates. If `index.type = "rectangle"`, each vector should specify rectangle length; e.g. row.index = 1:3 means the rectangle's 'length' is from rows 1 to 3. If `index.type = "ellipse"`, the arguments should be scalar values specifying the row/column coordinates for the center of the ellipse. If `index.type = "decay"`, the arguments should specify the row/column coordinates, respectively, of the peak parameter value.

`im.res`
: A vector specifying the dimension/resolution of the image. The first entry is the number of 'rows' in the lattice/image, and the second entry is the number of 'columns' in the lattice/image.

`B0`
: is the "true" intercept value; default is 0.

`B.values`
: is a vector "true" parameter values for non-zero parameters. The order of assignment is by row. If B.values argument is a single value, then all non-zero parameters are assigned to that value, unless `decay.fn` has been specified, in which case `B.values` is the "peak", and non-zero parameters decay smoothly by distance.

`index.type`
: is one of index.type = c("manual", "rectangle", "ellipse", "decay")

- `index.type = "manual"` uses row.index and col.index arguments to specify manually selected non-zero locations. This setting is good for irregular shaped regions.
- `index.type = "rectangle"` uses row.index and col.index arguments to specify a rectangular region of non-zero parameters.
- `index.type = "ellipse"` uses `w` and `h` arguments to specify elliptical region of non-zero parameters
- `index.type = "decay"` allows the user to specify a peak location with `row.index` and `col.index`, as with `index.type = "ellipse"`. However, the non-zero parameter values decay as a function of distance from the peak.

`decay.fn`
: An argument to specify the decay function of non-zero parameters decay from the peak when `index.type = "decay"`. Options are "exponential" or "gaussian". The rate of decay is given by $B.values * exp(-phi * d)$ and $B.values * exp(-phi * d^2)$ for "exponential" and "gaussian", respectively. The default is `decay.fn = "gaussian"`. Note that d is the Euclidean distance between the peak and a specified location, while phi is the rate of decay and is set by the user with `phi`.

`phi`
: A scalar value greater than 0 that determines the decay rate of non-zero parameters when `index.type = "decay"`. The default is `phi = 0.5`.

`max.d`
: When `index.type = "decay"`, `max.d` determines the maximum Euclidean distance from the peak that is allowed to be non-zero; parameters for locations further than `max.d` from the peak are set to zero. If this argument is not set by the user then all parameter values are determined by the decay function.

w, h	If index.type = "ellipse" then the width and height of the ellipse, respectively.
bayesian	If TRUE, then parameters are drawn from distributions based on initial B.values vector. Default is FALSE.
bayesian.dist	When bayesian = TRUE, specifies the distribution of the parameters. Options are "gaussian" and uniform.
bayesian.scale	A list. When bayesian = TRUE and bayesian.dist = "gaussian", specifies the sd for the distributions of parameters. When bayesian = TRUE and bayesian.dist = "uniform", specifies the width for the uniform distributions for the parameters. The first entry should be one of "unique", "binary". If "unique", then the second entry in the list should be a vector with length equal to B.values + 1 with unique values for the sd's/widths, including B0. B0 can be set to a constant value by setting the first position of bayesian.scale[[2]] to 0. If "binary", then the second entry in the list should be a 3-element vector whose first entry is the sd/width of B0, second entry the sd/width of "non-zero" or "important" parameters, and the third entry is the sd/width of the "zero" or "irrelevant" parameters.
output.indices	If output.indices = TRUE, then the first element of the returned list contains the indices for the non-zero parameter locations (Default). If output.indices = FALSE, then only the parameter vector is returned.

Value

A 2-element list containing (1) indices for the locations of "true" non-zero parameters, and (2) a parameter vector.

Note

The order of the parameters is by row. That is, if the lattice/image is 4x4, then parameters 1-4 make up the first row, 5-8 then second, and so forth.

Examples

```
## example
Bex1 <- beta_builder(row.index = c(5, 5, 6, 6),
                     col.index = c(5, 6, 5, 6),
                     im.res = c(10, 10),
                     B0 = 0, B.values = rep(1, 4))

## True non-zero parameters are locations 45, 46, 55, 56 in B
## i.e. locations (5, 5), (5, 6), (6, 5), (6, 6)

## Suppose that we index rows by i = 1, ... , I
##                       cols by j = 1, ... , J

## The index for a parameter is given by  J * (i - 1) + j
## In this example, I = 10, J = 10; Thus:

## (5, 5) -> 10 * (5 - 1) + 5 = 45
## (5, 6) -> 10 * (5 - 1) + 6 = 46
```

```
## (6, 5) -> 10 * (6 - 1) + 5 = 55
## (6, 6) -> 10 * (6 - 1) + 6 = 45
Bex1
## length 101 (includes B0 w/ 100 variable parameter values)
length(Bex1$B)

## example: index.type = "rectangle"
Bex2 <- beta_builder(row.index = 12:15, col.index = 6:19,
                     im.res = c(20, 20), B0 = 16,
                     B.values = 1:(length(12:15) * length(6:19)),
                     index.type = "rectangle")

Bex2
matrix(Bex2$B[-1], nrow = 20, byrow = TRUE)

## example: index.type = "ellipse"
Bex3 <- beta_builder(row.index = 4, col.index = 5,
                     im.res = c(10, 10),
                     B0 = 16, B.values = 3,
                     index.type = "ellipse",
                     h = 5, w = 4)
Bex3
matrix(Bex3$B[-1], nrow = 10, byrow = TRUE)

## decaying parameter values
Bex4 <- beta_builder(row.index = 10, col.index = 20,
                      im.res = c(30, 30), B0 = 0, B.values = 10,
                      index.type = "decay", max.d = 7,
                      output.indices = FALSE)
inf_2D_image(B = Bex4, im.res = c(30, 30), binarize.B = FALSE)

Bex5 <- beta_builder(row.index = 4, col.index = 5,
                      im.res = c(10, 10),
                      B0 = 16, B.values = 5,
                      index.type = "ellipse",
                      h = 5, w = 4,
                      bayesian = TRUE,
                      bayesian.dist = "gaussian",
                      bayesian.scale = list("binary", c(0, 1, 0.25)))

inf_2D_image(B = Bex5$B, im.res = c(10, 10), binarize.B = FALSE)
```

---

chol_s2Dp *Build and Take the Cholesky Decomposition of a Covariance Matrix*

Description

The function first builds a correlation matrix with correlation.builder, converts that matrix to a covariance matrix if necessary, and then takes the Cholesky decomposition of the matrix using

either base R or the R package spam. Note that spam is particularly effective when the matrix is sparse.

Usage

```
chol_s2Dp(
  matrix.type = "cov",
  im.res,
  use.spam = FALSE,
  corr.structure = "ar1",
  rho = NULL,
  phi = NULL,
  tau = 1,
  alpha = 0.75,
  corr.min = NULL,
  neighborhood = "none",
  w = NULL,
  h = NULL,
  r = NULL,
  print.R = FALSE,
  print.S = FALSE,
  print.Q = FALSE,
  sigma = 1,
  triangle = "upper",
  print.all = FALSE,
  round.d = FALSE,
  return.cov = TRUE,
  return.prec = TRUE
)
```

Arguments

matrix.type	Determines whether to build a covariance matrix, "cov", or a precision matrix, "prec". See correlation_builder{sim2Dpredictr} and precision_builder{sim2Dpredictr} for more details.
im.res	A vector defining the dimension of spatial data. The first entry is the number of rows and the second entry is the number of columns.
use.spam	If use.spam = TRUE then use tools from the R package spam; otherwise, base R functions are employed. For large dimension MVN with sparse correlation structure, spam is recommended; otherwise, base R may be faster. Defaults to FALSE.
corr.structure	One of "ar1", exponential, gaussian, or "CS". Correlations between locations i and j are rho^d for corr.structure = "ar1", $exp(-phi*d)$ for corr.structure = "exponential", $exp(-phi * d^2)$ for corr.structure = "gaussian", and rho when corr.structure = "CS". Note that d is the Euclidean distance between locations i and j.
rho	This is the maximum possible correlation between locations i and j. For all i,j rho MUST be between -1 and 1.

phi — A scalar value greater than 0 that determines the decay rate of correlation. This argument is only utilized when `corr.structure %in% c("exponential", "gaussian")`.

tau — A vector containing precision parameters. If of length 1, then all precisions are assumed equal. Otherwise the length of `tau` should equal the number of variables.

alpha — A scalar value between 0 and 1 that defines the strength of correlations. Note that when `alpha = 0` the data are independent and when `alpha = 1`, the joint distribution is the improper Intrinsic Autoregression (IAR), which cannot be used to generate data. Note also that while `alpha` does control dependence it is not interpretable as a correlation.

corr.min — Scalar value to specify the minimum non-zero correlation. Any correlations below `corr.min` are set to 0. Especially for high image resolution using this option can result in a sparser covariance matrix, which may significantly speed up draws when using `spam`. This option is preferred to using `neighborhood` and associated arguments when the primary concern is to avoid very small correlations and improve computation efficiency. Default is `NULL`, which places no restrictions on the correlations.

neighborhood — Defines the neighborhood within which marginal correlations are non-zero. The default is `"none"`, which allows marginal correlations to extend indefinitely. `neighborhood = "round"` defines a circular neighborhood about locations and `neighborhood = "rectangle"` defines a rectangular neighborhood about locations. Note that this argument differs from that in `precision_builder`, in which `neighborhood` defines conditional non-zero correlations.

w — If `neighborhood = "rectangle"` then w and h are the number of locations to the left/right and above/below a location i that define its neighborhood. Any locations outside this neighborhood have have zero correlation with location i.

h — If `neighborhood = "rectangle"` then w and h are the number of locations to the left/right and above/below a location i that define its neighborhood. Any locations outside this neighborhood have have zero correlation with location i.

r — If `neighborhood = "round"`, then if locations i,j are separated by distance $d \geq r$, the correlation between them is zero.

print.R, print.S, print.Q — Logical. When `TRUE`, then print the correlation, covariance, or precision matrix before taking the Cholesky decomposition. If `sigma = 1`, then S = R.

sigma — Specify the desired standard deviations; the default is 1, in which case the Cholesky decomposition is of a correlation matrix. If `sigma != 1`, then the Cholesky decomposition is of a covariance Matrix.

- If sigma is a vector then length(sigma) must be equal to the total number of locations, i.e. $(n.row * n.col) by (n.row * n.col)$
- sigma can take any scalar value when specifying common standard deviation.

triangle — Determine whether to output an upper (`triangle = "upper"`) or lower (`triangle = "lower"`) triangular matrix.

`print.all` If `print.all = TRUE`, then prints each correlation and allows you to check whether the correlations are as you intended. This option is NOT recommended for large point lattices/images.

`round.d` If `round.d = TRUE`, then d is rounded to the nearest whole number.

`return.cov, return.prec` Logical. When `TRUE`, also return the covariance or precision matrix, respectively. This is recommended when using `spam` to generate draws from the MVN.

Value

Matrix of dimension (n.row x n.col) x (n.row x n.col). If either `return.cov` or `return.prec` is `TRUE`, then returns a list where the first element is the covariance or precision matrix, and the second element is the Cholesky factor.

References

Banerjee S, Carlin BP, Gelfand AE (2015). *Hierarchical Modeling and Analysis for Spatial Data*, Second edition. Chapman \& Hall/CRC, Boca Raton, Florida.

Ripley BD (1987). *Stochastic Simulation*. John Wiley \& Sons. doi: 10.1002/9780470316726.

Rue H (2001). "Fast Sampling of Gaussian Markov Random Fields." *Journal of the Royal Statistical Society B*, **63**, 325-338. doi: 10.1111/14679868.00288.

Furrer R, Sain SR (2010). "spam: A Sparse Matrix R Package with Emphasis on MCMC Methods for Gaussian Markov Random Fields." *Journal of Statistical Software*, **36**(10), 1-25. http://www.jstatsoft.org/v36/i10/.

Examples

```
## Use R package spam for Cholesky decomposition
R <- chol_s2Dp(im.res = c(5, 5), matrix.type = "prec",
               use.spam = TRUE, neighborhood = "ar1",
               triangle = "upper")

## Use base R for Cholesky decomposition
chol_s2Dp(corr.structure = "ar1", im.res = c(3, 3), rho = 0.15,
        neighborhood = "round", r = 3, use.spam = FALSE)

## Specify standard deviations instead of default of sigma = 1.
chol_s2Dp(corr.structure = "ar1",im.res = c(3, 3), rho = 0.15,
        neighborhood = "round", r = 3, sigma = runif(9, 1.1, 4))

## Print options ON
chol_s2Dp(corr.structure = "ar1", im.res = c(3, 3), rho = 0.15,
          sigma = 1:9, neighborhood = "round", r = 3,
          print.R = TRUE, print.S = TRUE)
```

`correlation_builder` *Build a Correlation Matrix for 2D Spatial Data*

Description

This function "builds" a correlation matrix based on user specifications.

Usage

```
correlation_builder(
  corr.structure = "ar1",
  im.res,
  corr.min = NULL,
  neighborhood = "none",
  rho = NULL,
  phi = NULL,
  w = NULL,
  h = NULL,
  r = NULL,
  print.all = FALSE,
  round.d = FALSE
)
```

Arguments

`corr.structure`	One of `"ar1"`, `exponential`, `gaussian`, or `"CS"`. Correlations between locations i and j are rho^d for `corr.structure = "ar1"`, $exp(-phi*d)$ for `corr.structure = "exponential"`, $exp(-phi * d^2)$ for `corr.structure = "gaussian"`, and `rho` when `corr.structure = "CS"`. Note that `d` is the Euclidean distance between locations i and j.
`im.res`	A vector defining the dimension of spatial data. The first entry is the number of rows and the second entry is the number of columns.
`corr.min`	Scalar value to specify the minimum non-zero correlation. Any correlations below `corr.min` are set to 0. Especially for high image resolution using this option can result in a sparser covariance matrix, which may significantly speed up draws when using `spam`. This option is preferred to using `neighborhood` and associated arguments when the primary concern is to avoid very small correlations and improve computation efficiency. Default is `NULL`, which places no restrictions on the correlations.
`neighborhood`	Defines the neighborhood within which marginal correlations are non-zero. The default is `"none"`, which allows marginal correlations to extend indefinitely. `neighborhood = "round"` defines a circular neighborhood about locations and `neighborhood = "rectangle"` defines a rectangular neighborhood about locations. Note that this argument differs from that in `precision_builder`, in which `neighborhood` defines conditional non-zero correlations.

rho	This is the maximum possible correlation between locations i and j. For all i,j rho MUST be between -1 and 1.
phi	A scalar value greater than 0 that determines the decay rate of correlation. This argument is only utilized when corr.structure %in% c("exponential", "gaussian").
w, h	If neighborhood = "rectangle" then w and h are the number of locations to the left/right and above/below a location i that define its neighborhood. Any locations outside this neighborhood have have zero correlation with location i.
r	If neighborhood = "round", then if locations i,j are separated by distance $d \geq r$, the correlation between them is zero.
print.all	If print.all = TRUE, then prints each correlation and allows you to check whether the correlations are as you intended. This option is NOT recommended for large point lattices/images.
round.d	If round.d = TRUE, then d is rounded to the nearest whole number.

Value

Returns $(nr * nc) by (nr * nc)$ correlation matrix.

Note

Caution is recommended when using corr.min or neighborhood to set many correlations to 0, as not all specifications will result in a positive definite matrix. In particular, sharp drop-offs tend to result in non-positive definite matrices.

Examples

```
## examples
correlation_builder(corr.structure = "ar1", im.res = c(3, 3), rho = 0.5,
                    neighborhood = "round", r = 6, print.all = TRUE)

correlation_builder(corr.structure = "exponential", im.res = c(3, 3), phi = 0.5,
                    neighborhood = "round", r = 3, print.all = TRUE)

correlation_builder(corr.structure = "CS", im.res = c(3, 3),
                    rho = 0.5, print.all = TRUE)

## no "true" zeros, but gets close
c.nr <- correlation_builder(corr.structure = "ar1", neighborhood = "none",
                    corr.min = NULL, im.res = c(15, 15), rho = 0.5)
length(c.nr[c.nr > 0])
min(c.nr)

## set corr.min gives many zero entries; sparser structure
c.r <- correlation_builder(corr.structure = "ar1", neighborhood = "none",
                    corr.min = 0.01, im.res = c(15, 15), rho = 0.5)
## raw number > 0
length(c.r[c.r > 0])
## proportion > 0
length(c.r[c.r > 0]) / length(c.nr)
```

corr_fun *Specify the Correlation Function between Two Locations*

Description

This is primarily for use within correlation builder, and may be altered/expanded to handle more complicated correlation functions if desired.

Usage

```
corr_fun(
  corr.structure,
  im.res,
  corr.min = NULL,
  rho = NULL,
  phi = NULL,
  neighborhood = "none",
  round.d = FALSE,
  w = NULL,
  h = NULL,
  r = NULL,
  i,
  j,
  k,
  v
)
```

Arguments

corr.structure	One of "ar1", "exponential", "gaussian", or "CS". Correlations between locations i and j are rho^d for corr.structure = "ar1", $exp(-phi * d)$ for corr.structure = "exponential", $exp(-phi*d^2)$ for corr.structure = "gaussian", and rho when corr.structure = "CS". Note that d is the Euclidean distance between locations i and j.
im.res	A vector defining the dimension of spatial data. The first entry is the number of rows and the second entry is the number of columns.
corr.min	Scalar value to specify the minimum non-zero correlation. Any correlations below corr.min are set to 0. Especially for high image resolution using this option can result in a sparser covariance matrix, which may significantly speed up draws when using spam. This option is preferred to using neighborhood and associated arguments when the primary concern is to avoid very small correlations and improve computation efficiency. Default is NULL, which places no restrictions on the correlations.
rho	This is the maximum possible correlation between locations i and j. For all i,j rho MUST be between -1 and 1.

phi	A scalar value greater than 0 that determines the decay of correlation. This argument is only utilized when corr.structure %in% c("exponential", "gaussian").
neighborhood	Defines the neighborhood within which marginal correlations are non-zero. The default is "none", which allows marginal correlations to extend indefinitely. neighborhood = "round" defines a circular neighborhood about locations and neighborhood = "rectangle" defines a rectangular neighborhood about locations.
round.d	If round.d = TRUE, then d is rounded to the nearest whole number.
w, h	If neighborhood = "rectangle" then w and h are the number of locations to the left/right and above/below a location i that define its neighborhood. Any locations outside this neighborhood have have zero correlation with location i.
r	If neighborhood = "round", then if locations i,j are separated by distance $d \geq r$, the correlation between them is zero.
i, j, k, v	These are the coordinates for the two locations. Location 1 has coordinates (i, j) and location 2 has coordinates (k, v).

Value

A single element vector containing the correlation between spatial locations with indices (i, j) and (k, v).

Examples

```
## examples
corr_fun(corr.structure = "ar1", im.res = c(3, 3), rho = 0.5,
         neighborhood = "round", r = 6, i = 1, j = 2, k = 2, v = 3)

corr_fun(corr.structure = "ar1", im.res = c(3, 3), rho = 0.5,
         neighborhood = "rectangle", w = 1, h = 1, i = 1, j = 2, k = 2, v = 3)
```

---

generate_grid *Convert a 2D Space to Grid Coordinates*

Description

Input the limits of a 2D space and the desired image resolution, then the function outputs the appropriate grid/lattice coordinates.

Usage

```
generate_grid(im.res, xlim = c(0, 1), ylim = c(0, 1))
```

Arguments

`im.res`	A vector specifying the dimension/resolution of the image. The first entry is the number of 'rows' in the lattice/image, and the second entry is the number of 'columns' in the lattice/image.
`xlim, ylim`	These are the 2D image limits. Defaults for both are `c(0, 1)`. It is not recommended to alter these arguments unless changing the limits has a specific practical utility.

Value

A data frame whose first column is x-coordinates and whose second column is y-coordinates.

`inf_2D_image` *Display Inference Results for 2D Predictors*

Description

Provide graphics for spatial extent of predictor parameters, rejections and/or the truth/falsity of the rejections.

Usage

```
inf_2D_image(
  rejections = NULL,
  B = NULL,
  im.res,
  test.statistic = NULL,
  reject.threshold = NULL,
  binarize.B = TRUE,
  grid.color = "grey",
  n.colors = length(unique(B)),
  B.incl.B0 = TRUE,
  plot.title = TRUE
)
```

Arguments

`rejections`	A binary vector; `rejection[i] = 1` means the null hypothesis is rejected for parameter `B[i]`, whereas `rejection[i] = 0` means that the null hypothesis was not rejected for parameter `B[i]`.
`B`	A vector of "true" parameter values. For inference purposes, this can be a vector of actual parameter values, or a binary vector indicating non-zero status.
`im.res`	A vector defining the dimension of spatial data. The first entry is the number of rows and the second entry is the number of columns.
`test.statistic`	A vector of test statistics; e.g., t-statistics or p-values that are used to determine whether or not to reject the null hypothesis.

`reject.threshold`
: A list whose first element is the rejection criteria, e.g., the minimum t-statistic or maximum p-value for which to reject the null hypothesis. The second element is one of `c("greater", "less", "2-tailed")`, which tell the function to reject when the values in `test.statistic` are greater than or less than the threshold, the test is a 2-tailed, respectively. In the latter case the function internally calculates the upper or lower threshold needed for the 2-tailed test.

`binarize.B`
: Either `TRUE` (default) or `FALSE`. When `binarize.B = TRUE` the parameter vector is converted to a binary vector where 1 indicates non-zero parameter and 0 indicates zero-valued parameter.

`grid.color`
: Specify the color for the grid lines.

`n.colors`
: Determines the number of colors in the printed image. Default is `length(unique(B))`, but it is recommended to use trial and error to determine the ideal setting for specific situations.

`B.incl.B0`
: If `B.incl.B0 = TRUE` then the first entry should be the intercept, B0. `B.incl.B0 = FALSE` indicates that the first entry of B is not an intercept.

`plot.title`
: When `plot.title = TRUE` a title accompanies the output graph, and `plot.title = FALSE` suppresses the title.

Value

An image depicting the spatial extent of some image characteristic.

Note

If both `rejections` and `B` are specified then the function provides an image with separate color each for:

- No rejection and `B[i] = 0` (i.e. True Negative).
- No rejection and `B[i] != 0` (i.e. False Negative).
- Rejection and `B[i] = 0` (i.e. False Positive).
- Rejection and `B[i] != 0` (i.e. True Positive).

Examples

```
## parameter vector
Bex <- beta_builder(row.index = c(rep(5, 3), rep(6, 3), rep(7, 3)),
                    col.index = rep(c(5, 6, 7), 3),
                    im.res = c(10, 10), index.type = "manual",
                    B0 = 0, B.values = 1:9,
                    output.indices = FALSE)

## co-opt beta builder to get rejections
rejex <- beta_builder(row.index = c(rep(4, 3), rep(5, 3), rep(6, 3)),
                      col.index = rep(c(4, 5, 6), 3),
                      im.res = c(10, 10), index.type = "manual",
                      B0 = 0, B.values = rep(1, 9),
                      output.indices = FALSE)[-1]
```

```
rejex.sm2 <- beta_builder(row.index = 5:6, col.index = 5:6,
                          im.res = c(10, 10),
                          B0 = 0, B.values = 1,
                          output.indices = FALSE)[-1]

## just B
inf_2D_image(B = Bex, im.res = c(10, 10))
## just rejections
inf_2D_image(rejections = rejex, im.res = c(10, 10))

## both B and rejections
inf_2D_image(rejections = rejex, B = Bex, im.res = c(10, 10))
inf_2D_image(rejections = rejex.sm2, B = Bex, im.res = c(10, 10))

## larger dimension example
Bex2 <- beta_builder(row.index = 5:15, col.index = 16:20,
                     im.res = c(50, 50), B0 = 0,
                     B.values = 1:(length(5:15) * length(16:20)),
                     index.type = "rectangle",
                     output.indices = FALSE)
rejex2 <- beta_builder(row.index = 13:21, col.index = 30:41,
                       im.res = c(50, 50), B0 = 0,
                       B.values = rep(1, (length(13:21) * length(30:41))),
                       index.type = "rectangle",
                       output.indices = FALSE)[-1]
rejex3 <- beta_builder(row.index = 5:20, col.index = 16:30,
                       im.res = c(50, 50), B0 = 0,
                       B.values = rep(1, (length(5:20) * length(16:30))),
                       index.type = "rectangle",
                       output.indices = FALSE)[-1]
rejex4 <- beta_builder(row.index = 5:10, col.index = 16:17,
                       im.res = c(50, 50), B0 = 0,
                       B.values = rep(1, (length(5:10) * length(16:17))),
                       index.type = "rectangle",
                       output.indices = FALSE)[-1]
## images
inf_2D_image(B = Bex2, im.res = c(50, 50))
inf_2D_image(B = Bex2, im.res = c(50, 50), binarize.B = FALSE)
inf_2D_image(rejections = rejex2, im.res = c(50, 50))

## No TP
inf_2D_image(rejections = rejex2, B = Bex2, im.res = c(50, 50))
## ALL TP
inf_2D_image(rejections = Bex2[-1], B = Bex2, im.res = c(50, 50))
## No FN
inf_2D_image(rejections = rejex3, B = Bex2, im.res = c(50, 50))
## No FP, but FN
inf_2D_image(rejections = rejex4, im.res = c(50, 50))
inf_2D_image(B = Bex2, im.res = c(50, 50))
inf_2D_image(rejections = rejex4, B = Bex2, im.res = c(50, 50))
```

make_rejection *Determine rejections*

Description

Determine rejections

Usage

```
make_rejection(B, reject.threshold, test.statistic)
```

Arguments

B	A vector of "true" parameter values. For inference purposes, this can be a vector of actual parameter values, or a binary vector indicating non-zero status.
reject.threshold	A list whose first element is the rejection criteria, e.g., the minimum t-statistic or maximum p-value for which to reject the null hypothesis. The second element is one of `c("greater", "less", "2-tailed")`, which tell the function to reject when the values in `test.statistic` are greater than or less than the threshold, the test is a 2-tailed, respectively. In the latter case the function internally calculates the upper or lower threshold needed for the 2-tailed test.
test.statistic	A vector of test statistics; e.g., t-statistics or p-values that are used to determine whether or not to reject the null hypothesis.

Value

A vector of hypothesis testing rejection indicators, where 1 indicates a rejection and 0 otherwise.

neighbors_by_dist *Determine and store neighbors by Euclidean Distance Constraints*

Description

Determine and store neighbors by Euclidean Distance Constraints

Usage

```
neighbors_by_dist(x, y, coords, im.res, r, print.ring = FALSE)
```

Arguments

x, y	are the row and column coordinates, respectively.
coords	A dataframe containing indices and coordinates for the image.
im.res	A vector containing the number of rows and columns, respectively.
r	A scalar value determining the radius within which other locations are neighbors to the current location (x, y).
print.ring	When print.ring = TRUE, each iteration is shown, with corresponding information regarding the number of neighbors present in each ring. This argument primarily exists to allow the user to test whether the neighborhood structure specified is as desired.

Value

A tibble whose first column contains x indices, second column contains y indices, and third column denotes the current ring about a location.

Note

This function avoids testing all points for being with a certain distance in order to determine neighbor status of a given point by progressively widening a box around the point. Each iteration widens the box by an extra ring, and we only test points in the new ring. If at the end of testing a ring there are no new neighbors then we stop expanding the box and return the neighbors' coordinates. For computational efficiency, this function assumes that all arguments except the current point's coordinates have been specified.

Examples

```
## Necessary pre-specified arguments required for the function to work.

## requires tidyverse (dplyr)
library(tidyverse)

## image resoluation + number of spatial predictors
im.res <- c(5, 5)
J = prod(im.res)

## create predictor indices w/ coordinates
row.id <-rep(1, im.res[2])
for (i in 2:im.res[1]) {
 row.id <- c(row.id, rep(i, im.res[2]))
}
coords <- data.frame(index = 1:J,
                     row.id = row.id,
                     col.id = rep(c(1:im.res[2]), im.res[1]) )

neighbors_by_dist(x = 2, y = 2, im.res = im.res, coords = coords, r = 2)
```

precision_builder *Construct a Precision Matrix*

Description

This function constructs the precision matrix for a Conditional Autoregression (CAR).

Usage

```
precision_builder(
  im.res,
  tau = 1,
  alpha = 0.75,
  neighborhood = "ar1",
  weight = "binary",
  phi = 1,
  r = NULL,
  w = NULL,
  h = NULL,
  digits.Q = 10
)
```

Arguments

im.res	A vector defining the dimension of spatial data. The first entry is the number of rows and the second entry is the number of columns.
tau	A vector containing precision parameters. If of length 1, then all precisions are assumed equal. Otherwise the length of tau should equal the number of variables.
alpha	A scalar value between 0 and 1 that defines the strength of correlations. Note that when alpha = 0 the data are independent and when alpha = 1, the joint distribution is the improper Intrinsic Autoregression (IAR), which cannot be used to generate data. Note also that while alpha does control dependence it is not interpretable as a correlation.
neighborhood	Defines the neighborhood within which conditional correlations are non-zero. This differs from use in correlation_builder, where the neighborhood defines non-zero marginal correlations. The default is "ar1", which creates a neighborhood where the spatial locations directly above, below, left, and right of a location are included in the neighborhood. More complicated neighborhoods can be specified by neighborhood = "round", which defines a circular neighborhood about each location, and neighborhood = "rectangle", which defines a rectangular neighborhood about each location.
weight	Determines how weights are assigned. "distance" assigns weights as the inverse of Euclidean distance times a constant, phi. "binary" assigns weights to 1 for neighbors and 0 otherwise.

phi	When weight = "distance" a constant by which to multiply the inverse of Euclidean distance. Defaults to 1, and must exceed 0.
r	If neighborhood = "round", then locations i,j are separated by distance $d \geq r$ are conditionally independent.
w, h	If neighborhood = "rectangle" then w and h are the number of locations to the left/right and above/below a location i that define its neighborhood. Any locations outside this neighborhood are conditionally independent of the specified location.
digits.Q	Determines the number of digits to round entries in the precision matrix. Default is 10.

Details

This formulation of the CAR model is based on a formulation found in (Banerjee et al. 2015) where the joint distribution of the of the conditionally specified random variables are assumed to be $N(0, [diag(tau^2)(D - alphaW)]^{-1})$ and all neighbors are weighted 1. When weights other than 1 are desired, the joint distribution is $N(0, [diag(tau^2)D(I - alphaD^{-1}W)]^{-1})$, e.g. as in (Jin et al. 2005).

Value

A (precision) matrix.

References

Banerjee S, Carlin BP, Gelfand AE (2015). *Hierarchical Modeling and Analysis for Spatial Data*, Second edition. Chapman \& Hall/CRC, Boca Raton, Florida.

Jin X, Carlin BP, Banerjee S (2005). "Generalized Hierarchical Multivariate CAR Models for Areal Data." *Biometrics*, **61**(4), 950-961. doi: 10.1111/j.15410420.2005.00359.x.

Examples

```
precision_builder(im.res = c(5, 5), tau = 1, alpha = 0.75,
                  neighborhood = "ar1")

## binary weights
precision_builder(im.res = c(5, 5), tau = 1, alpha = 0.75,
                  neighborhood = "round", r = 3)

## weights based on distance
precision_builder(im.res = c(5, 5), tau = 1, alpha = 0.75,
                  weight = "distance", phi = 1,
                  neighborhood = "round", r = 3)

precision_builder(im.res = c(5, 5), tau = 1, alpha = 0.75,
                  neighborhood = "rectangle", w = 2, h = 2)
```

`proximity_builder` *Generate a Proximity Matrix*

Description

Generates a proximity matrix where non-zero entries are the weights associated with neighbors, and zero entries are not neighbors.

Usage

```
proximity_builder(
  im.res,
  neighborhood = "ar1",
  type = c("sparse", "full"),
  weight = "binary",
  phi = 1,
  r = NULL,
  h = NULL,
  w = NULL,
  include.coords = FALSE,
  print.im = FALSE
)
```

Arguments

`im.res`	A vector defining the dimension of spatial data. The first entry is the number of rows and the second entry is the number of columns.
`neighborhood`	Determines how to assign neighbor status to locations; i.e. 1 for neighbors, 0 otherwise. `type = "round"` assigns neighbor status to locations within radius `r`. `type = "ar1"` assigns 1 to locations directly above or beside. `type = "rectangle"` assigns neighbor status to locations within `w` units to the left or right and `h` units up or down.
`type`	Specifies either sparse (`type = "sparse"`) or full (`type = "full"`) proximity matrix.
`weight`	Determines how weights are assigned. `"distance"` assigns weights as the inverse of Euclidean distance times a constant, `phi`. `"binary"` assigns weights to 1 for neighbors and 0 otherwise.
`phi`	When `weight = "distance"` a constant by which to multiply the inverse of Euclidean distance. Defaults to 1.
`r, h, w`	When `neighborhood = "round"`, `r` specifies the radius within which other locations are neighbors. When `neighborhood = "rectangle"`, `w` and `h` specify the number of units to the left/right and above/below the location are to be counted as neighbors.
`include.coords`	If `type = "sparse"` and `include.coords = TRUE`, then the coordinates of neighbors are returned along with their indices.

print.im Allows user to print the 2D "image" matrix with index labels to visually verify that the proximity matrix is as expected.

Value

A (proximity) matrix.

Examples

```
## adjacency matrix with sparse structure (i.e., 2 columns) and ar1 neighborhood
sp.ar1 <- proximity_builder(im.res = c(4, 4),
                            weight = "binary",
                            neighborhood = "ar1",
                            type = "sparse")
## adjacency matrix with full structure (i.e., prod(im.dim) rows & columns) and ar1 neighborhood
full.ar1 <- proximity_builder(im.res = c(4, 4),
                              weight = "binary",
                              neighborhood = "ar1",
                              type = "full")

## proximity matrix weighted by distance (sparse)
sp.rnd <- proximity_builder(im.res = c(5, 5),
                            weight = "distance",
                            neighborhood = "round", r = 2,
                            type = "sparse",
                            include.coords = TRUE)

## proximity matrix weighted by distance (full)
full.rnd <- proximity_builder(im.res = c(5, 5),
                              weight = "distance",
                              neighborhood = "round", r = 2,
                              type = "full")
```

---

sample_FP_Power *Obtain Sample False Positive Rates and Power*

Description

This function calculates sample FDR, FWER, and Power for large numbers of predictors, given a vector of "true" parameter values and a vector of associated rejections. In the case that more than 1 predictor has a "true" non-zero parameter, then Power is defined as the proportion/percentage of those "true" parameters identified.

Usage

```
sample_FP_Power(
  rejections = NULL,
  FP = NULL,
  TP = NULL,
```

```
  test.statistic = NULL,
  reject.threshold = NULL,
  B = NULL,
  B.incl.B0 = TRUE,
  full.summary = FALSE
)
```

Arguments

`rejections`	A binary vector; `rejection[i] = 1` means the null hypothesis is rejected for parameter `B[i]`, whereas `rejection[i] = 0` means that the null hypothesis was not rejected for parameter `B[i]`.
`FP, TP`	Binary vectors of false positive and true positive indicators, respectively. `FP[i] = 1` means the null hypothesis was incorrectly rejected, and `TP[i] = 1` means the null hypothesis was correctly rejected. If either argument is `NULL`, then these vectors are computed; this is the default setting.
`test.statistic`	A vector of test statistics; e.g., t-statistics or p-values that are used to determine whether or not to reject the null hypothesis.
`reject.threshold`	A list whose first element is the rejection criteria, e.g., the minimum t-statistic or maximum p-value for which to reject the null hypothesis. The second element is one of `c("greater", "less", "2-tailed")`, which tell the function to reject when the values in `test.statistic` are greater than or less than the threshold, the test is a 2-tailed, respectively. In the latter case the function internally calculates the upper or lower threshold needed for the 2-tailed test.
`B`	A vector of "true" parameter values. For inference purposes, this can be a vector of actual parameter values, or a binary vector indicating non-zero status.
`B.incl.B0`	If `B.incl.B0 = TRUE` then the first entry should be the intercept, B0. `B.incl.B0 = FALSE` indicates that the first entry of B is not an intercept.
`full.summary`	If `full.summary = TRUE` then the total numbers of rejections, false positives, true positives, and non-zero parameters are output along with FDR, FWER, and Power; otherwise, only FDR, FWER, and Power are output.

Value

A data frame with columns for sample FDR, FWER, and Power.

Note

The default operating approach is that the null hypothesis is `B[i] = 0` for each parameter. If other hypotheses are being tested then `B` should be converted to a binary vector indicating whether the null hypothesis *should* have been rejected.

Examples

```
## example 1

## rejection vector
```

```
rej.ex <- c(0, 1, 1, 0, 0, 1, 0)
## false positive vector
fp.ex  <- c(0, 0, 1, 0, 0, 0, 0)
## true positive vector
tp.ex  <- c(0, 1, 0, 0, 0, 1, 0)
## parameter vector
par.ex <- c(0, 4, 0, 0, 3, 9, 0)

sample_FP_Power(rej.ex, fp.ex, tp.ex, par.ex, B.incl.B0 = FALSE)

## Function can calculate TP and FP vectors
sample_FP_Power(rejections = rej.ex, FP = NULL, TP = NULL, B = par.ex, B.incl.B0 = FALSE)

## example 2: sum(FP, TP) must equal sum(rejections) or function stops execution

rej.ex2 <- c(0, 1, 0, 0, 0, 1, 0)
fp.ex2  <- c(0, 0, 1, 0, 0, 0, 0)
tp.ex2  <- c(0, 1, 0, 0, 0, 1, 0)
par.ex2 <- c(0, 4, 0, 0, 3, 9, 0)

## Not run: sample_FP_Power(rej.ex2, fp.ex2, tp.ex2, par.ex2, B.incl.B0 = FALSE)

## example 3: calculate rejections from vector of test statistics
zstat <- c(-0.5, 1.98, 2.01, 1.45, -1.99)
# 2-tailed
sample_FP_Power(test.statistic = zstat,
                reject.threshold = list(1.96, "2-tailed"),
                B = c(0, 0, 4, 1, -2), B.incl.B0 = FALSE)
# 1-tailed (upper)
sample_FP_Power(test.statistic = zstat,
                reject.threshold = list(1.96, "greater"),
                B = c(0, 0, 4, 1, -2), B.incl.B0 = FALSE)
## p-value
sample_FP_Power(test.statistic = c(0.44, 0.04, 0.01, 0.06, 0.02 ),
                reject.threshold = list(0.05, "less"),
                B = c(0, 0, 4, 1, -2), B.incl.B0 = FALSE)
```

`sim2D_binarymap` *Generate a Binary Map via the Boolean Method*

Description

Use a Homogenous Poisson Process to generate random "events", a uniform distribution to generate circles of random radii about the events, and take the union to obtain a random set. This is mapped onto a lattice to obtain a binary map.

Usage

```
sim2D_binarymap(
  N,
```

```
  xlim = c(0, 1),
  ylim = c(0, 1),
  im.res,
  radius.bounds = c(0.02, 0.1),
  lambda = 50,
  random.lambda = FALSE,
  lambda.sd = 10,
  lambda.bound = NULL,
  prior = "gamma",
  sub.area = FALSE,
  min.sa = c(0.1, 0.1),
  max.sa = c(0.3, 0.3),
  radius.bounds.min.sa = c(0.02, 0.05),
  radius.bounds.max.sa = c(0.08, 0.15),
  print.subj.sa = FALSE,
  print.lambda = FALSE,
  print.iter = FALSE,
  store.type = "list"
)
```

Arguments

N	A scalar value determining the number of images to create.
xlim	These are the 2D image limits. Defaults for both are c(0, 1). It is not recommended to alter these arguments unless changing the limits has a specific practical utility.
ylim	These are the 2D image limits. Defaults for both are c(0, 1). It is not recommended to alter these arguments unless changing the limits has a specific practical utility.
im.res	A vector specifying the dimension/resolution of the image. The first entry is the number of 'rows' in the lattice/image, and the second entry is the number of 'columns' in the lattice/image.
radius.bounds	A 2-element vector whose first and second entries determine the minimum and maximum radius sizes, respectively; these will be the bounds of the uniform distribution used to draw the radii. If sub.area = TRUE, then use radius.bounds.min.sa and radius.bounds.max.sa.
lambda	A scalar value specifying the mean/intensity value of the Poisson process. If random.lambda = FALSE then this is the parameter used to generate the binary image for each subject. If random.lambda = TRUE, then this is the mean parameter in the distribution used to draw subject-specific lambda.
random.lambda	random.lambda = TRUE allows the lambda (mean/intensity) parameter in the Poisson process to vary randomly by subject.
lambda.sd	Only utilized when random.lambda = TRUE, and specifies the standard deviation in the distribution used to draw subject-specific lambda.
lambda.bound	Only utilized when random.lambda = TRUE, and allows the user to specify a lower and upper bound for the subject-specific lambda; if the randomly selected

value is outside of this range, then another draw is taken. This continues until a value is selected within the specified bounds. If no bounds are desired then specify `lambda.bound = NULL`.

`prior` Only utilized when `random.lambda = TRUE`, and specifies the distribution from which to draw the subject-specific lambda. Options are `c("gaussian", "gamma")`.

`sub.area` When `sub.area = TRUE`, a random sub-section of the image is chosen, within which the Poisson process is used to generate the binary image.

`min.sa` Only utilized when `sub.area = TRUE`, and determines the width and height of the minimum and maximum sub-areas; e.g., if `min.sa = c(0.1, 0.1)`, then the smallest possible random sub-area is a 0.1 x 0.1 square.

`max.sa` Only utilized when `sub.area = TRUE`, and determines the width and height of the minimum and maximum sub-areas; e.g., if `min.sa = c(0.1, 0.1)`, then the smallest possible random sub-area is a 0.1 x 0.1 square.

`radius.bounds.min.sa`
Only utilized when `sub.area = TRUE`, and specifies `radius.bounds` for the minimum and maximum sub-areas, respectively. This information is used to adaptively alter the bounds in between the minimum and maximum sub-areas.

`radius.bounds.max.sa`
Only utilized when `sub.area = TRUE`, and specifies `radius.bounds` for the minimum and maximum sub-areas, respectively. This information is used to adaptively alter the bounds in between the minimum and maximum sub-areas.

`print.subj.sa` These arguments are either `TRUE` or `FALSE`, and define print options for checking that the function is working as the user intends. `print.subj.sa = TRUE` prints the x- and y-limits for each subject's sub-area. `print.lambda = TRUE` prints each subject's mean and realized events; the means will be the same unless `random.lambda = TRUE`, but the number of realized events will always vary. `print.iter = TRUE` is only used when `random.lambda = TRUE` and `is.null(lambda.bound) = FALSE`, and shows iterations for re-drawing when the randomly selected intensity is outside the specified bounds.

`print.lambda` These arguments are either `TRUE` or `FALSE`, and define print options for checking that the function is working as the user intends. `print.subj.sa = TRUE` prints the x- and y-limits for each subject's sub-area. `print.lambda = TRUE` prints each subject's mean and realized events; the means will be the same unless `random.lambda = TRUE`, but the number of realized events will always vary. `print.iter = TRUE` is only used when `random.lambda = TRUE` and `is.null(lambda.bound) = FALSE`, and shows iterations for re-drawing when the randomly selected intensity is outside the specified bounds.

`print.iter` These arguments are either `TRUE` or `FALSE`, and define print options for checking that the function is working as the user intends. `print.subj.sa = TRUE` prints the x- and y-limits for each subject's sub-area. `print.lambda = TRUE` prints each subject's mean and realized events; the means will be the same unless `random.lambda = TRUE`, but the number of realized events will always vary. `print.iter = TRUE` is only used when `random.lambda = TRUE` and `is.null(lambda.bound) = FALSE`, and shows iterations for re-drawing when the randomly selected intensity is outside the specified bounds.

store.type — One of c("list", "matrix"). When store.type = "list", the output is a list where each element is a matrix defining a subject image. If store.type = "matrix", then the images are vectorized by row and each row of the output matrix contains an image vector for a single subject.

Value

A list; each element is a matrix of zeroes and ones.

References

Cressie N, Wikle CK (2011). *Statistics for Spatio-Temporal Data*, Wiley Series in Probability and Statistics. John Wiley \& Sons, Hoboken, NJ.

Examples

```
bin_ims <- sim2D_binarymap(N = 5, im.res = c(10, 10), store.type = "list",
                           lambda = 50, sub.area = TRUE,
                           min.sa = c(0.10, 0.10), max.sa = c(0.5, 0.5),
                           radius.bounds.min.sa = c(0.015, 0.04),
                           radius.bounds.max.sa = c(0.041, 0.06))

rotate = function(x){
  t(apply(x, 2, rev))
}

for (i in 1:length(bin_ims)) {
  image(rotate(bin_ims[[i]]),
        col = c("white", "darkgreen"),
        axes = FALSE)
  box()
  grid(nx = 10, ny = 10, col = "black",
       lty = 1)
}
```

---

sim2D_RandSet_HPPP — *Generate a Random Set Using a Poisson Process and Random Radii About Events*

Description

A random set is generated by using a Poisson process in 2D space to choose 'event' locations, about which a circle of random radius is 'drawn'. The union of the circles defines ultimately defines the set.

Usage

```
sim2D_RandSet_HPPP(
  N,
  xlim = c(0, 1),
  ylim = c(0, 1),
  radius.bounds = c(0.05, 0.15),
  lambda = 50,
  lambda.sd = 10,
  lambda.bound = NULL,
  prior = "gamma",
  random.lambda = FALSE,
  sub.area = TRUE,
  min.sa = c(0.1, 0.1),
  max.sa = c(0.3, 0.3),
  radius.bounds.min.sa = c(0.02, 0.05),
  radius.bounds.max.sa = c(0.08, 0.15),
  print.subj.sa = FALSE,
  print.lambda = FALSE,
  print.iter = FALSE
)
```

Arguments

N	A scalar value determining the number of images to create.
xlim, ylim	These are the 2D image limits. Defaults for both are c(0, 1). It is not recommended to alter these arguments unless changing the limits has a specific practical utility.
radius.bounds	A 2-element vector whose first and second entries determine the minimum and maximum radius sizes, respectively; these will be the bounds of the uniform distribution used to draw the radii. If sub.area = TRUE, then use radius.bounds.min.sa and radius.bounds.max.sa.
lambda	A scalar value specifying the mean/intensity value of the Poisson process. If random.lambda = FALSE then this is the parameter used to generate the binary image for each subject. If random.lambda = TRUE, then this is the mean parameter in the distribution used to draw subject-specific lambda.
lambda.sd	Only utilized when random.lambda = TRUE, and specifies the standard deviation in the distribution used to draw subject-specific lambda.
lambda.bound	Only utilized when random.lambda = TRUE, and allows the user to specify a lower and upper bound for the subject-specific lambda; if the randomly selected value is outside of this range, then another draw is taken. This continues until a value is selected within the specified bounds. If no bounds are desired then specify lambda.bound = NULL.
prior	Only utilized when random.lambda = TRUE, and specifies the distribution from which to draw the subject-specific lambda. Options are c("gaussian", "gamma").
random.lambda	random.lambda = TRUE allows the lambda (mean/intensity) parameter in the Poisson process to vary randomly by subject.

`sub.area` When `sub.area = TRUE`, a random sub-section of the image is chosen, within which the Poisson process is used to generate the binary image.

`min.sa, max.sa` Only utilized when `sub.area = TRUE`, and determines the width and height of the minimum and maximum sub-areas; e.g., if `min.sa = c(0.1, 0.1)`, then the smallest possible random sub-area is a 0.1 x 0.1 square.

`radius.bounds.min.sa, radius.bounds.max.sa`
Only utilized when `sub.area = TRUE`, and specifies `radius.bounds` for the minimum and maximum sub-areas, respectively. This information is used to adaptively alter the bounds in between the minimum and maximum sub-areas.

`print.subj.sa, print.lambda, print.iter`
These arguments are either `TRUE` or `FALSE`, and define print options for checking that the function is working as the user intends. `print.subj.sa = TRUE` prints the x- and y-limits for each subject's sub-area. `print.lambda = TRUE` prints each subject's mean and realized events; the means will be the same unless `random.lambda = TRUE`, but the number of realized events will always vary. `print.iter = TRUE` is only used when `random.lambda = TRUE` and `is.null(lambda.bound) = FALSE`, and shows iterations for re-drawing when the randomly selected intensity is outside the specified bounds.

Value

A dataframe with columns for subject ID, x-coordinates, y-coordinates, and associated radii.

References

Cressie N, Wikle CK (2011). *Statistics for Spatio-Temporal Data*, Wiley Series in Probability and Statistics. John Wiley \& Sons, Hoboken, NJ.

---

sim_MVN_X *Simulate Spatially Correlated MVN Data*

Description

Takes N draws from a Multivariate Normal (MVN) distribution using either base R or the R package `spam`. This function requires the Cholesky decomposition of the desired covariance matrix.

Usage

```
sim_MVN_X(
  N,
  mu = 0,
  L = NULL,
  R = NULL,
  S = NULL,
  Q = NULL,
  use.spam = FALSE,
  use.MASS = FALSE,
```

```
  X.categorical = FALSE,
  X.num.categories = 2,
  X.category.type = "percentile",
  X.percentiles = NULL,
  X.manual.thresh = NULL,
  X.cat.names = NULL
)
```

Arguments

N	The number of draws to take from MVN; i.e., the number of subjects.
mu	One of the following: • A single scalar value for common mean. • A vector of length nrow(R) (equivalently nrow(R)) containing means for the MVN.
L, R	L and R are lower and upper triangular matrices, respectively, and are the Cholesky factor(s) of the desired covariance matrix for the MVN. Obtain L or R via chol_s2Dp() with settings triangle = "lower" or triangle = "upper", respectively. Specify either L or R, but NOT both.
S, Q	A covariance or precision matrix respectively. These are for use with spam, and can be extracted from output of chol_s2Dp after choosing return.cov = TRUE or return.prec = TRUE, respectively.
use.spam	Logical. If use.spam = TRUE then use tools from the R package spam; otherwise, base R functions are employed. For large dimension MVN with sparse correlation structure, spam is recommended; otherwise, base R may be faster. Defaults to FALSE. Requires either the covariance matrix S or precision matrix, Q, that corresponds to the Cholesky factor.
use.MASS	Logical. When TRUE draws X from MVN using mvrnorm from MASS. Note that this requires specification of the covariance matrix, S. Specifying the precision matrix instead may slow down the process for large dimensions. Recommended to use spam to generate draws when specifying a precision matrix, Q.
X.categorical	Default is X.categorical = FALSE. If X.categorical = TRUE then thresholds are applied to categorize each predictor/image value.
X.num.categories	A scalar value denoting the number of categories in which to divide the data.
X.category.type	Tells R how to categorize the data. Options are X.category.type = c("percentile", "manual"). If X.category.type = "percentile" then the data are divided into percentiles based on X.num.categories; e.g. if X.num.categories = 4 then the values are divided into quartiles, and values in Q1 equal 0, between Q1 and Q2 equal 1, between Q2 and Q3 equal 2, and greater than Q3 equal 3. If X.category.type = "manual" then specify the cutoff points with X.manual.thresh.
X.percentiles	A vector of percentiles to be used in thresholding when X.categorical = TRUE and X.category.type = "percentile". The length of this vector should equal the number of categories minus one, and all values should be between zero and one.

`X.manual.thresh`
: A vector containing the thresholds for categorizing the values; e.g. if `X.num.categories = 4` and `X.manual.thresh = c(-3, 1, 17)`, then values less than -3 are set to 0, equal or greater than -3 and less than 1 are set to 1, equal or greater than 1 but less than 17 are set to 2, and equal or greater than 17 are set to 3. Note that `length(X.manual.thresh)` must always equal `X.num.categories - 1`.

`X.cat.names`
: A vector of category names. If `X.cat.names = NULL` then then the initial integers assigned are left as the values; the names in `X.cat.names` are assigned in ascending order.

Value

Matrix of dimension `N` x `(nrow(L))` (or equivalently `N` x `(nrow(R))`) where each row is draw from MVN, and each column represents a different "variable"; e.g. location in an image.

Note

This function requires the Cholesky decomposition of the desired covariance matrix for the MVN; this allows for using this function in simulating multiple datasets of `N` MVN draws while only taking the Cholesky decomposition of the covariance matrix once.

References

Furrer R, Sain SR (2010). "spam: A Sparse Matrix R Package with Emphasis on MCMC Methods for Gaussian Markov Random Fields." *Journal of Statistical Software*, **36**(10), 1-25. http://www.jstatsoft.org/v36/i10/.

Ripley BD (1987). *Stochastic Simulation*. John Wiley \& Sons. doi: 10.1002/9780470316726.

Rue H (2001). "Fast Sampling of Gaussian Markov Random Fields." *Journal of the Royal Statistical Society B*, **63**, 325-338. doi: 10.1111/14679868.00288.

Examples

```
## verify MVN with base R
set.seed(732)
Lex <- chol_s2Dp(corr.structure = "ar1", im.res = c(3, 3), rho = 0.25,
                 sigma = 1, use.spam = FALSE, corr.min = 0.02,
                 triangle = "lower", return.cov = TRUE)
XbR = sim_MVN_X(N = 1000, mu = 0, L = Lex$L)

apply(XbR, 2, mean)
cov(XbR)
Lex$S

## verify MVN with \code{spam}
set.seed(472)
Rex <- chol_s2Dp(im.res = c(3, 3), matrix.type = "prec",
                use.spam = TRUE, neighborhood = "ar1",
                triangle = "upper", return.prec = TRUE)

Xspam = sim_MVN_X(N = 1000, mu = 0, R = Rex$R, Q = Rex$Q)
```

```
apply(Xspam, 2, mean)
solve(cov(Xspam))
as.matrix(Rex$Q)

## Categories
set.seed(832)
Xtest <- sim_MVN_X(N = 30, mu = 0, L = Lex$L,
                   X.categorical = TRUE,
                   X.num.categories = 3,
                   X.category.type = "percentile",
                   X.cat.names = c("A", "B", "C"))
Xtest
```

`sim_Y_Binary_X` *Simulate Scalar Outcomes from Simulated Spatially Dependent Binary Predictors*

Description

N spatially dependent binary design vectors are simulated using `sim2D_binarymap`. These design vectors are used to then simulate scalar outcomes that have one of Gaussian, Binomial, or Poisson distributions.

Usage

```
sim_Y_Binary_X(
  N,
  B,
  rand.err = 1,
  dist,
  incl.subjectID = TRUE,
  binomial.method = "traditional",
  count.method = "traditional",
  Y.thresh = NULL,
  print.out = FALSE,
  xlim = c(0, 1),
  ylim = c(0, 1),
  im.res,
  radius.bounds = c(0.02, 0.1),
  lambda = 50,
  random.lambda = FALSE,
  lambda.sd = 10,
  lambda.bound = NULL,
  prior = "gamma",
  sub.area = FALSE,
  min.sa = c(0.1, 0.1),
```

```
  max.sa = c(0.3, 0.3),
  radius.bounds.min.sa = c(0.02, 0.05),
  radius.bounds.max.sa = c(0.08, 0.15),
  print.subj.sa = FALSE,
  print.lambda = FALSE,
  print.iter = FALSE
)
```

Arguments

N	A scalar value determining the number of images to create.
B	A vector parameter values; i.e. "betas". Note that `length(B)` must equal `p + 1 = n.row * n.col + 1`; e.g. for normal outcomes $Y = XB + e$ with `Y` a scalar outcome and `e` the random error.
rand.err	A scalar for the random error variance when `dist = "gaussian"`.
dist	The distribution of the scalar outcome. • `dist = "gaussian"` has $Y = XB + e$, where $e\ N(0, rand.err)$. • `dist = "binomial"` is drawn from eqnBin(XB, XB(1-XB)) using `rbinom()` when `binary.method = "Traditional"`. If `binary.method = "Gaussian"`, then simulation is based on a cutoff using `binary.cutoff`. • `dist = "poisson"` is drawn from $Poisson(XB)$ using `rpois()`.
incl.subjectID	When `incl.subjectID = TRUE` a column of subject indices is generated. `Y.thresh = NULL` (default). If `binomial.method = "gaussian manual"`, then `Y.thresh` should be any scalar real number; values equal or above this cutoff are assigned 1 and values below are assigned 0. If `binomial.method = "gaussian percentile"`, then values equal or above this percentile are assigned 1, and other wise 0; in this case values should be between 0 and 1. For example, if `Y.thresh = 0.9`, then the cutoff is the 90th percentile.
binomial.method	One of `c("traditional", "gaussian manual", "gaussian percentile")`. Only specified when `dist = "binomial"`, and determines whether draws are directly taken from a binomial distribution or if draws are taken from a Multivariate Normal Distribution (in the manner of `dist = "gaussian"`) and thresholds imposed to binarize the outcomes. `binomial.method = "gaussian manual"` allows the user to specify specific values for categorizing outcomes. `binomial.method = "gaussian percentile"` allows the user to specify percentiles for binarizing the data. Both approaches use `Y.thresh` to specify the cutoff value(s). If `binomial.method = "gaussian percentile"` and `Y.thresh = NULL` then the median is used as the threshold. If `binomial.method = "gaussian manual"` and `Y.thresh = NULL`, then 0 is used as the threshold. Default is `binomial.method = "traditional"`.
count.method	One of `c("traditional", "rounding")`. When `count.method = "traditional"`, the outcomes are drawn sequentially using `rpois()`. When `count.method = "traditional"`, the outcomes are drawn from an MVN, then values less than or equal to 0 are set to 0, and all other values are rounded to the nearest whole number.

`Y.thresh` When `binomial.method = "traditional"`

`print.out` If `print.out = TRUE` then print the following for each subject, indexed y:

- `X[y] %*% B`
- `p[y]`, `lambda[y]` for Binomial, Poisson, respectively.

This is useful to see the effect of image parameter selection and beta parameter selection on distributional parameters for the outcome of interest.

`xlim` These are the 2D image limits. Defaults for both are `c(0, 1)`. It is not recommended to alter these arguments unless changing the limits has a specific practical utility.

`ylim` These are the 2D image limits. Defaults for both are `c(0, 1)`. It is not recommended to alter these arguments unless changing the limits has a specific practical utility.

`im.res` A vector specifying the dimension/resolution of the image. The first entry is the number of 'rows' in the lattice/image, and the second entry is the number of 'columns' in the lattice/image.

`radius.bounds` A 2-element vector whose first and second entries determine the minimum and maximum radius sizes, respectively; these will be the bounds of the uniform distribution used to draw the radii. If `sub.area = TRUE`, then use `radius.bounds.min.sa` and `radius.bounds.max.sa`.

`lambda` A scalar value specifying the mean/intensity value of the Poisson process. If `random.lambda = FALSE` then this is the parameter used to generate the binary image for each subject. If `random.lambda = TRUE`, then this is the mean parameter in the distribution used to draw subject-specific lambda.

`random.lambda` `random.lambda = TRUE` allows the lambda (mean/intensity) parameter in the Poisson process to vary randomly by subject.

`lambda.sd` Only utilized when `random.lambda = TRUE`, and specifies the standard deviation in the distribution used to draw subject-specific lambda.

`lambda.bound` Only utilized when `random.lambda = TRUE`, and allows the user to specify a lower and upper bound for the subject-specific lambda; if the randomly selected value is outside of this range, then another draw is taken. This continues until a value is selected within the specified bounds. If no bounds are desired then specify `lambda.bound = NULL`.

`prior` Only utilized when `random.lambda = TRUE`, and specifies the distribution from which to draw the subject-specific lambda. Options are `c("gaussian", "gamma")`.

`sub.area` When `sub.area = TRUE`, a random sub-section of the image is chosen, within which the Poisson process is used to generate the binary image.

`min.sa` Only utilized when `sub.area = TRUE`, and determines the width and height of the minimum and maximum sub-areas; e.g., if `min.sa = c(0.1, 0.1)`, then the smallest possible random sub-area is a 0.1 x 0.1 square.

`max.sa` Only utilized when `sub.area = TRUE`, and determines the width and height of the minimum and maximum sub-areas; e.g., if `min.sa = c(0.1, 0.1)`, then the smallest possible random sub-area is a 0.1 x 0.1 square.

`radius.bounds.min.sa`
: Only utilized when `sub.area = TRUE`, and specifies `radius.bounds` for the minimum and maximum sub-areas, respectively. This information is used to adaptively alter the bounds in between the minimum and maximum sub-areas.

`radius.bounds.max.sa`
: Only utilized when `sub.area = TRUE`, and specifies `radius.bounds` for the minimum and maximum sub-areas, respectively. This information is used to adaptively alter the bounds in between the minimum and maximum sub-areas.

`print.subj.sa`
: These arguments are either `TRUE` or `FALSE`, and define print options for checking that the function is working as the user intends. `print.subj.sa = TRUE` prints the x- and y-limits for each subject's sub-area. `print.lambda = TRUE` prints each subject's mean and realized events; the means will be the same unless `random.lambda = TRUE`, but the number of realized events will always vary. `print.iter = TRUE` is only used when `random.lambda = TRUE` and `is.null(lambda.bound) = FALSE`, and shows iterations for re-drawing when the randomly selected intensity is outside the specified bounds.

`print.lambda`
: These arguments are either `TRUE` or `FALSE`, and define print options for checking that the function is working as the user intends. `print.subj.sa = TRUE` prints the x- and y-limits for each subject's sub-area. `print.lambda = TRUE` prints each subject's mean and realized events; the means will be the same unless `random.lambda = TRUE`, but the number of realized events will always vary. `print.iter = TRUE` is only used when `random.lambda = TRUE` and `is.null(lambda.bound) = FALSE`, and shows iterations for re-drawing when the randomly selected intensity is outside the specified bounds.

`print.iter`
: These arguments are either `TRUE` or `FALSE`, and define print options for checking that the function is working as the user intends. `print.subj.sa = TRUE` prints the x- and y-limits for each subject's sub-area. `print.lambda = TRUE` prints each subject's mean and realized events; the means will be the same unless `random.lambda = TRUE`, but the number of realized events will always vary. `print.iter = TRUE` is only used when `random.lambda = TRUE` and `is.null(lambda.bound) = FALSE`, and shows iterations for re-drawing when the randomly selected intensity is outside the specified bounds.

Value

A data frame where each row consists of a single subject's data. Col 1 is the outcome, Y, and each successive column contains the subject predictor values.

Note

Careful parameter selection, i.e. `B`, is necessary to ensure that simulated outcomes are reasonable; in particular, counts arising from the Poisson distribution can be unnaturally large.

References

Cressie N, Wikle CK (2011). *Statistics for Spatio-Temporal Data*, Wiley Series in Probability and Statistics. John Wiley \& Sons, Hoboken, NJ.

Ripley BD (1987). *Stochastic Simulation*. John Wiley \& Sons. doi: 10.1002/9780470316726.

Examples

```
## Define non-zero beta values
Bex <- beta_builder(row.index = c(3, 3, 4), col.index = c(3, 4, 3),
                    im.res = c(5, 5),
                    B0 = 0, B.values = rep(1/3, 3),
                    output.indices = FALSE)
## Simulate Datasets
## parameter values
Nex = 10
set.seed(28743)

Gauss.ex <- sim_Y_Binary_X(N = Nex, B = Bex, dist = "gaussian", im.res = c(5, 5))
hist(Gauss.ex$Y)

## direct draws from binomial
Bin.ex <- sim_Y_Binary_X(N = Nex, B = Bex, im.res = c(5, 5),
                         dist = "binomial", print.out = TRUE)
table(Bin.ex$Y)
```

`sim_Y_MVN_X` *Simulate Scalar Outcomes from Simulated Spatially Correlated Predictors*

Description

N spatially correlated design vectors are simulated from an MVN. These design vectors are used to then simulate scalar outcomes that have one of Gaussian, Binomial, or Poisson distributions.

Usage

```
sim_Y_MVN_X(
  N,
  B,
  L = NULL,
  R = NULL,
  S = NULL,
  Q = NULL,
  use.spam = TRUE,
  mu = 0,
  rand.err = 1,
  dist = "gaussian",
  incl.subjectID = TRUE,
  threshold.method = "none",
  Y.thresh = NULL,
  X.categorical = FALSE,
  X.num.categories = 2,
  X.category.type = "percentile",
```

```
  X.manual.thresh = NULL,
  X.cat.names = NULL,
  print.out = FALSE
)
```

Arguments

N	The number of draws to take from MVN; i.e., the number of subjects.
B	A vector parameter values; i.e. "betas". Note that `length(B)` must equal `p + 1 = n.row * n.col + 1`; e.g. for normal outcomes $Y = XB + e$ with `Y` a scalar outcome and `e` the random error.
L	`L` and `R` are lower and upper triangular matrices, respectively, and are the Cholesky factor(s) of the desired covariance matrix for the MVN. Obtain `L` or `R` via `chol_s2Dp()` with settings `triangle = "lower"` or `triangle = "upper"`, respectively. Specify either `L` or `R`, but NOT both.
R	`L` and `R` are lower and upper triangular matrices, respectively, and are the Cholesky factor(s) of the desired covariance matrix for the MVN. Obtain `L` or `R` via `chol_s2Dp()` with settings `triangle = "lower"` or `triangle = "upper"`, respectively. Specify either `L` or `R`, but NOT both.
S	A covariance or precision matrix respectively. These are for use with `spam`, and can be extracted from output of `chol_s2Dp` after choosing `return.cov = TRUE` or `return.prec = TRUE`, respectively.
Q	A covariance or precision matrix respectively. These are for use with `spam`, and can be extracted from output of `chol_s2Dp` after choosing `return.cov = TRUE` or `return.prec = TRUE`, respectively.
use.spam	Logical. If `use.spam = TRUE` then use tools from the R package `spam`; otherwise, base R functions are employed. For large dimension MVN with sparse correlation structure, `spam` is recommended; otherwise, base R may be faster. Defaults to `FALSE`. Requires either the covariance matrix `S` or precision matrix, `Q`, that corresponds to the Cholesky factor.
mu	One of the following: • A single scalar value for common mean. • A vector of length `nrow(R)` (equivalently `nrow(R)`) containing means for the MVN.
rand.err	A vector for the random error standard deviation when `dist = "gaussian"`, or thresholding is used to obtain non-Normal draws. Must have length 1 or length N.
dist	The distribution of the scalar outcome. • `dist = "gaussian"` has $Y = XB + e$, where $e\ N(0, rand.err)$. • `dist = "binomial"` is drawn from eqnBin(XB, XB(1-XB)) using `rbinom()` when `binary.method = "traditional"`. If `binary.method = "gaussian"`, then simulation is based on a cutoff using `binary.cutoff`. • `dist = "poisson"` is drawn from $Poisson(XB)$ using `rpois()`.
incl.subjectID	When `incl.subjectID = TRUE` a column of subject indices is generated.

`threshold.method`
: One of `"none"`, `"manual"`, `"percentile"`, `"round"`. When `"none"` draws from Binomial or Poisson distributions are taken subject-wise using base R functions. For the remaining options, draws are first taken from a Normal distribution and then thresholded. `"manual"` uses `Y.thresh` to manually select a cutoff, `"percentile"` uses `Y.thresh` to select percentiles used to bin outcomes, and `"round"` sets values equal or less than 0 to 0, and rounds all positive values to the nearest whole number.

`Y.thresh`
: A manual value used to threshold when `threshold.method = "manual"`; values equal or greater than the cutoff are assigned 1 and all others 0. When `threshold.method = "percentile"`, a percentile to use to bin outcomes.

`X.categorical`
: Default is `X.categorical = FALSE`. If `X.categorical = TRUE` then thresholds are applied to categorize each predictor/image value.

`X.num.categories`
: A scalar value denoting the number of categories in which to divide the data.

`X.category.type`
: Tells R how to categorize the data. Options are `X.category.type = c("percentile", "manual")`. If `X.category.type = "percentile"` then the data are divided into percentiles based on `X.num.categories`; e.g. if `X.num.categories = 4` then the values are divided into quartiles, and values in Q1 equal 0, between Q1 and Q2 equal 1, between Q2 and Q3 equal 2, and greater than Q3 equal 3. If `X.category.type = "manual"` then specify the cutoff points with `X.manual.thresh`.

`X.manual.thresh`
: A vector containing the thresholds for categorizing the values; e.g. if `X.num.categories = 4` and `X.manual.thresh = c(-3, 1, 17)`, then values less than -3 are set to 0, equal or greater than -3 and less than 1 are set to 1, equal or greater than 1 but less than 17 are set to 2, and equal or greater than 17 are set to 3. Note that `length(X.manual.thresh)` must always equal `X.num.categories - 1`.

`X.cat.names`
: A vector of category names. If `X.cat.names = NULL` then then the initial integers assigned are left as the values; the names in `X.cat.names` are assigned in ascending order.

`print.out`
: If `print.out = TRUE` then print the following for each subject, indexed y:

  - `X[y] %*% B`
  - `p[y]`, `lambda[y]` for Binomial, Poisson, respectively.

  This is useful to see the effect of image parameter selection and beta parameter selection on distributional parameters for the outcome of interest.

Value

A data frame where each row consists of a single subject's data. Col 1 is the outcome, Y, and each successive column contains the subject predictor values.

Note

Careful parameter selection, i.e. `B`, is necessary to ensure that simulated outcomes are reasonable; in particular, counts arising from the Poisson distribution can be unnaturally large.

References

Furrer R, Sain SR (2010). "spam: A Sparse Matrix R Package with Emphasis on MCMC Methods for Gaussian Markov Random Fields." *Journal of Statistical Software*, **36**(10), 1-25. http://www.jstatsoft.org/v36/i10/.

Ripley BD (1987). *Stochastic Simulation*. John Wiley \& Sons. doi: 10.1002/9780470316726.

Rue H (2001). "Fast Sampling of Gaussian Markov Random Fields." *Journal of the Royal Statistical Society B*, **63**, 325-338. doi: 10.1111/14679868.00288.

Examples

```
## generate precision matrix and take Cholesky decomposition
Rpre <- chol_s2Dp(im.res = c(5, 5), matrix.type = "prec",
                  use.spam = TRUE, neighborhood = "ar1",
                  triangle = "upper", return.prec = TRUE)
## Generate correlation matrix & take Cholesky decomposition
Rcov <- chol_s2Dp(corr.structure = "ar1", im.res = c(5, 5), rho = 0.5,
                  triangle = "upper",
                  use.spam = FALSE, neighborhood = "none")

## Define non-zero beta values
Bex <- beta_builder(row.index = c(3, 3, 4, 4), col.index = c(3, 4, 3, 4),
                    im.res = c(5, 5),
                    B0 = 0, B.values = rep(1, 4),
                    output.indices = FALSE)
## Simulate Datasets
## parameter values
Nex = 100
set.seed(28743)

## with precision matrix
Gauss.exp <- sim_Y_MVN_X(N = Nex, B = Bex,
                         R = Rpre$R, Q = Rpre$Q,
                         dist = "gaussian")
hist(Gauss.exp$Y)

## with covariance matrix
Gauss.exc <- sim_Y_MVN_X(N = Nex, B = Bex,
                         R = Rcov$R, S = Rcov$S,
                         dist = "gaussian")
hist(Gauss.exc$Y)

## direct draws from binomial
Bin.ex <- sim_Y_MVN_X(N = Nex, B = Bex, R = Rcov$R, S = Rcov$S,
                      dist = "binomial", print.out = TRUE)
table(Bin.ex$Y)

## manual cutoff
Bin.ex2 <- sim_Y_MVN_X(N = Nex, B = Bex,
                       R = Rcov$R, S = Rcov$S,
                       dist = "binomial",
                       threshold.method = "manual",
```

```
                        Y.thresh = 1.25)
table(Bin.ex2$Y)

## percentile cutoff
Bin.ex3 <- sim_Y_MVN_X(N = Nex, B = Bex,
                        R = Rcov$R, S = Rcov$S,
                        dist = "binomial",
                        threshold.method = "percentile",
                        Y.thresh = 0.75)
table(Bin.ex3$Y)

## Poisson Example - note the large counts
Pois.ex <- sim_Y_MVN_X(N = Nex, B = Bex,
                        R = Rcov$R, S = Rcov$S,
                        dist = "poisson", print.out = TRUE)
mean(Pois.ex$Y)
quantile(Pois.ex$Y, probs = c(0, 0.1, 0.25, 0.45, 0.5, 0.75, 0.9, 0.95, 0.99, 1))
hist(Pois.ex$Y)
```

`within_area` *Determine Whether Lattice Points are Within or Without a Random Set*

Description

Determine whether locations in the image/lattice (from `generate.grid`) are within or without the union of a random set generated by `sim2D_HPPP_coords()`. If the Euclidean distance between a lattice location and any 'event' is less than the radius about the 'event', then the location is said to be within the random set. Otherwise, it is without the random set.

Usage

```
within_area(grid.centers, radii, event.xcoord, event.ycoord)
```

Arguments

`grid.centers`	Output from `generate.grid()` that specifies the coordinates of the lattice locations in native space.
`radii`	A vector of radii values.
`event.xcoord, event.ycoord`	Paired vectors specifying the x- and y- coordinates, respectively, of each 'event' from the Poisson process.

Value

A data frame with lattice x- and y- coordinates, and a binary vector where 1 indicates the location is within the random set, and 0 indicates the location is without the random set.

Index