# Package ‘sdsfun’

September 22, 2025

**Title** Spatial Data Science Complementary Features

**Version** 0.8.1

**Description** Wrapping and supplementing commonly used functions in the R ecosystem related to spatial data science,
while serving as a basis for other packages maintained by Wenbo Lv.

**License** GPL-3

**Encoding** UTF-8

**URL** https://stscl.github.io/sdsfun/, https://github.com/stscl/sdsfun

**BugReports** https://github.com/stscl/sdsfun/issues

**RoxygenNote** 7.3.3

**Depends** R (>= 4.1.0)

**LinkingTo** Rcpp, RcppArmadillo

**Imports** dplyr, geosphere, magrittr, pander, purrr, sf, spdep, stats,
tibble, utils

**Suggests** ggplot2, Rcpp, RcppArmadillo, terra, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**NeedsCompilation** yes

**Author** Wenbo Lv [aut, cre, cph] (ORCID:
<https://orcid.org/0009-0002-6003-3800>)

**Maintainer** Wenbo Lv <lyu.geosocial@gmail.com>

**Repository** CRAN

**Date/Publication** 2025-09-22 10:30:02 UTC

## Contents

---

`check_tbl_na` *check for NA values in a tibble*

---

## Description

check for NA values in a tibble

## Usage

```
check_tbl_na(tbl)
```

## Arguments

| | |
|---|---|
| `tbl` | A `tibble` |

### Value

A logical value.

### Examples

```
demotbl = tibble::tibble(x = c(1,2,3,NA,1),
                         y = c(NA,NA,1:3),
                         z = 1:5)
demotbl
check_tbl_na(demotbl)
```

---

## cor_test *(partial) correlation test*

---

### Description

(partial) correlation test

### Usage

```
cor_test(x, y, z = NULL, level = 0.05)
```

### Arguments

| | |
|---|---|
| x | A numeric vector representing the first variable. |
| y | A numeric vector representing the second variable. |
| z | An optional numeric vector or matrix of control variables. If provided, partial correlation is computed. |
| level | (optional) Significance level. Default is 0.05. |

### Value

A numeric vector

### Examples

```
gzma = sf::read_sf(system.file('extdata/gzma.gpkg',package = 'sdsfun'))
cor_test(gzma$PS_Score,gzma$EL_Score)
cor_test(gzma$PS_Score,gzma$EL_Score,gzma$OH_Score)
```

## discretize_vector *discretization*

### Description

discretization

### Usage

```
discretize_vector(
  x,
  n,
  method = "natural",
  breakpoint = NULL,
  sampleprob = 0.15,
  thr = 0.4,
  seed = 123456789
)
```

### Arguments

| | |
|---|---|
| x | A continuous numeric vector. |
| n | (optional) The number of discretized classes. |
| method | (optional) The method of discretization, default is `natural`. |
| breakpoint | (optional) Break points for manually splitting data. When `method` is `manual`, `breakpoint` is required. |
| sampleprob | (optional) When the data size exceeds `3000`, perform sampling for discretization, applicable only to natural breaks. Default is `0.15`. |
| thr | (optional) Threshold for controlling iteration, applicable only to headtails breaks. Default is `0.4`. |
| seed | (optional) Random seed number, default is `123456789`. |

### Value

A discretized integer vector

### Examples

```
xvar = c(22361, 9573, 4836, 5309, 10384, 4359, 11016, 4414, 3327, 3408,
         17816, 6909, 6936, 7990, 3758, 3569, 21965, 3605, 2181, 1892,
         2459, 2934, 6399, 8578, 8537, 4840, 12132, 3734, 4372, 9073,
         7508, 5203)
discretize_vector(xvar, n = 5, method = 'natural')
```

`dummy_tbl` *transforming a category tibble into the corresponding dummy variable tibble*

## Description

transforming a category tibble into the corresponding dummy variable tibble

## Usage

```
dummy_tbl(tbl)
```

## Arguments

| | |
|---|---|
| `tbl` | A `tibble` or `data.frame`. |

## Value

A `tibble`

## Examples

```
a = tibble::tibble(x = 1:3,y = 4:6)
dummy_tbl(a)
```

`dummy_vec` *transforming a categorical variable into dummy variables*

## Description

transforming a categorical variable into dummy variables

## Usage

```
dummy_vec(x)
```

## Arguments

| | |
|---|---|
| `x` | An integer vector or can be converted into an integer vector. |

## Value

A matrix.

## Examples

```
dummy_vec(c(1,1,3,2,4,6))
```

## formula_varname *get variable names in a formula and data*

### Description

get variable names in a formula and data

### Usage

```
formula_varname(formula, data)
```

### Arguments

| | |
|---|---|
| formula | A formula. |
| data | A `data.frame`, `tibble` or `sf` object of observation data. |

### Value

A list.

| | |
|---|---|
| yname | Independent variable name |
| xname | Dependent variable names |

### Examples

```
gzma = sf::read_sf(system.file('extdata/gzma.gpkg',package = 'sdsfun'))
formula_varname(PS_Score ~ EL_Score + OH_Score, gzma)
formula_varname(PS_Score ~ ., gzma)
```

## fuzzyoverlay *spatial fuzzy overlay*

### Description

spatial fuzzy overlay

### Usage

```
fuzzyoverlay(formula, data, method = "and")
```

### Arguments

| | |
|---|---|
| formula | A formula of spatial fuzzy overlay. |
| data | A data.frame or tibble of discretized data. |
| method | (optional) Overlay methods. When `method` is `and`, use `min` to do fuzzy overlay; and when `method` is `or`,use `max` to do fuzzy overlay. Default is `and`. |

### Value

A numeric vector.

### Note

Independent variables in the data provided to fuzzyoverlay() must be discretized variables, and dependent variable are continuous variable.

### Examples

```
set.seed(42)
sim = tibble::tibble(y = stats::runif(7,0,10),
                     x1 = c(1,rep(2,3),rep(3,3)),
                     x2 = c(rep(1,2),rep(2,2),rep(3,3)))
fo1 = fuzzyoverlay(y~x1+x2,data = sim, method = 'and')
fo1
fo2 = fuzzyoverlay(y~x1+x2,data = sim, method = 'or')
fo2
```

---

## generate_subsets *generate subsets of a set*

---

### Description

generate subsets of a set

### Usage

```
generate_subsets(set, empty = TRUE, self = TRUE)
```

### Arguments

| | |
|---|---|
| set | A vector. |
| empty | (optional) When empty is TRUE, the generated subset includes the empty set, otherwise the empty set is removed. Default is TRUE. |
| self | (optional) When self is TRUE, the resulting subset includes the set itself, otherwise the set itself is removed. Default is TRUE. |

### Value

A list.

### Examples

```
generate_subsets(letters[1:3])
generate_subsets(letters[1:3],empty = FALSE)
generate_subsets(letters[1:3],self = FALSE)
generate_subsets(letters[1:3],empty = FALSE,self = FALSE)
```

## geodetector_q — *only geodetector q-value*

### Description

only geodetector q-value

### Usage

```
geodetector_q(y, hs)
```

### Arguments

| | |
|---|---|
| y | Dependent variable |
| hs | Independent variable |

### Value

A numeric value

### Examples

```
geodetector_q(y = 1:7, hs = c('x',rep('y',3),rep('z',3)))
```

## hclustgeo_disc — *hierarchical clustering with spatial soft constraints*

### Description

hierarchical clustering with spatial soft constraints

### Usage

```
hclustgeo_disc(
  data,
  n,
  alpha = 0.5,
  D1 = NULL,
  hclustm = "ward.D2",
  scale = TRUE,
  wt = NULL,
  ...
)
```

### Arguments

- `data` — An `sf` object, `tibble`, `data.frame`, `matrix` or `vector` of observations data.
- `n` — The number of hierarchical clustering classes, which can be a numeric value or vector.
- `alpha` — (optional) A positive value between `0` and `1`. This mixing parameter gives the relative importance of "feature" space and "constraint" space. Default is `0.5`.
- `D1` — (optional) A `matrix` with other dissimilarities between the same observations data. if `data` is an `sf` object and alpha is not `0`, the `D1` will be generated by `sdsfun::sf_distance_matrix()`, others will use a `matrix` with all elements equal to `0`.
- `hclustm` — (optional) The agglomeration method to be used, default is `ward.D2`. For more details, please see `stats::hclust()`.
- `scale` — (optional) Whether to scaled the dissimilarities matrix, default is `TRUE`.
- `wt` — (optional) Vector with the weights of the observations. By default, `wt` is `NULL`.
- `...` — (optional) Other arguments passed to `stats::dist()`.

### Value

The grouped membership: a `vector` if `n` is a scalar, a `matrix` (columns correspond to elements of `n`) if not.

### Note

This is a `C++` enhanced implementation of the `hclustgeo` function in `ClustGeo` package.

### Examples

```
gzma = sf::read_sf(system.file('extdata/gzma.gpkg',package = 'sdsfun'))
gzma$group = hclustgeo_disc(gzma,5,alpha = 0.75)
plot(gzma["group"])
```

---

## `inverse_distance_swm` *construct inverse distance weight*

---

### Description

Function for constructing inverse distance weight.

### Usage

```
inverse_distance_swm(sfj, power = 1, bandwidth = NULL)
```

### Arguments

- `sfj` Vector object that can be converted to `sf` by `sf::st_as_sf()`.
- `power` (optional) Default is 1. Set to 2 for gravity weights.
- `bandwidth` (optional) When the distance is bigger than bandwidth, the corresponding part of the weight matrix is set to 0. Default is `NULL`, which means not use the bandwidth.

### Details

The inverse distance weight formula is $w_{ij} = 1/d_{ij}^{\alpha}$

### Value

A inverse distance weight matrices with class of `matrix`.

### Examples

```
library(sf)
pts = read_sf(system.file('extdata/pts.gpkg',package = 'sdsfun'))
wt = inverse_distance_swm(pts)
wt[1:5,1:5]
```

---

## loess_optnum *determine optimal spatial data discretization for individual variables*

---

### Description

Function for determining optimal spatial data discretization for individual variables based on locally estimated scatterplot smoothing (LOESS) model.

### Usage

```
loess_optnum(qvec, discnumvec, increase_rate = 0.05)
```

### Arguments

- `qvec` A numeric vector of q statistics.
- `discnumvec` A numeric vector of break numbers corresponding to `qvec`.
- `increase_rate` (optional) The critical increase rate of the number of discretization. Default is `0.05`.

### Value

A two element numeric vector.

- `discnum` optimal number of spatial data discretization
- `increase_rate` the critical increase rate of the number of discretization

## Note

When `increase_rate` is not satisfied by the calculation, the discrete number corresponding to the highest `q statistic` is selected as a return.

Note that `sdsfun` sorts `discnumvec` from smallest to largest and keeps `qvec` in one-to-one correspondence with `discnumvec`.

## Examples

```
qv = c(0.26045642,0.64120405,0.43938704,0.95165535,0.46347836,
       0.25385338,0.78778726,0.95938330,0.83247885,0.09285196)
loess_optnum(qv,3:12)
```

---

# `moran_test` *global spatial autocorrelation test*

---

## Description

global spatial autocorrelation test

## Usage

```
moran_test(sfj, wt = NULL, alternative = "greater", symmetrize = FALSE)
```

## Arguments

| | |
|---|---|
| `sfj` | An `sf` object or can be converted to `sf` by `sf::st_as_sf()`. |
| `wt` | (optional) Spatial weight matrix. Must be a `matrix` class. If `wt` is not provided, `sdsfun` will use a first-order queen adjacency binary matrix. |
| `alternative` | (optional) Specification of alternative hypothesis as `greater` (default), `lower`, or `two.sided`. |
| `symmetrize` | (optional) Whether or not to symmetrize the asymmetrical spatial weight matrix **wt** by: 1/2 * (**wt** + **wt**'). Default is `FALSE`. |

## Value

A list utilizing a `result` tibble to store the following information for each variable:

`MoranI` observed value of the Moran coefficient

`EI` expected value of Moran's I

`VarI` variance of Moran's I (under normality)

`ZI` standardized Moran coefficient

`PI` *p*-value of the test statistic

## Examples

```
gzma = sf::read_sf(system.file('extdata/gzma.gpkg',package = 'sdsfun'))
moran_test(gzma)
```

---

`normalize_vector` *normalization*

---

## Description

normalization

## Usage

```
normalize_vector(x, to_left = 0, to_right = 1)
```

## Arguments

| | |
|---|---|
| x | A continuous numeric vector. |
| to_left | (optional) Specified minimum. Default is `0`. |
| to_right | (optional) Specified maximum. Default is `1`. |

## Value

A continuous vector which has normalized.

## Examples

```
normalize_vector(c(-5,1,5,0.01,0.99))
```

---

`rm_lineartrend` *remove variable linear trend based on covariate*

---

## Description

remove variable linear trend based on covariate

## Usage

```
rm_lineartrend(formula, data, method = c("cpp", "r"))
```

### Arguments

| | |
|---|---|
| `formula` | A formula. |
| `data` | The observation data. |
| `method` | (optional) The method for using, which can be chosen as either `cpp` or `r`. Default is `cpp`. |

### Value

A numeric vector.

### Examples

```
gzma = sf::read_sf(system.file('extdata/gzma.gpkg',package = 'sdsfun'))
rm_lineartrend(PS_Score ~ ., gzma)
rm_lineartrend(PS_Score ~ ., gzma, method = "r")
```

---

## `sf_coordinates` *extract locations*

---

### Description

Extract locations of sf objects.

### Usage

```
sf_coordinates(sfj)
```

### Arguments

| | |
|---|---|
| `sfj` | An `sf` object or can be converted to `sf` by `sf::st_as_sf()`. |

### Value

A matrix.

### Examples

```
pts = sf::read_sf(system.file('extdata/pts.gpkg',package = 'sdsfun'))
sf_coordinates(pts)
```

## sf_distance_matrix *generates distance matrix*

### Description

Generates distance matrix for sf object

### Usage

```
sf_distance_matrix(sfj)
```

### Arguments

| | |
|---|---|
| sfj | An sf object or can be converted to sf by sf::st_as_sf(). |

### Value

A matrix.

### Examples

```
pts = sf::read_sf(system.file('extdata/pts.gpkg',package = 'sdsfun'))
pts_distm = sf_distance_matrix(pts)
pts_distm[1:5,1:5]
```

## sf_geometry_name *sf object geometry column name*

### Description

Get the geometry column name of an sf object

### Usage

```
sf_geometry_name(sfj)
```

### Arguments

| | |
|---|---|
| sfj | An sf object. |

### Value

A character.

### Examples

```
gzma = sf::read_sf(system.file('extdata/gzma.gpkg',package = 'sdsfun'))
sf_geometry_name(gzma)
```

---

## sf_geometry_type    *sf object geometry type*

---

### Description

Get the geometry type of an sf object

### Usage

```
sf_geometry_type(sfj)
```

### Arguments

| | |
|---|---|
| sfj | An sf object. |

### Value

A lowercase character vector

### Examples

```
gzma = sf::read_sf(system.file('extdata/gzma.gpkg',package = 'sdsfun'))
sf_geometry_type(gzma)
```

---

## sf_gk_proj_cgcs2000    *generates cgcs2000 Gauss-Kruger projection epsg coding character*

---

### Description

Generates a Gauss-Kruger projection epsg coding character corresponding to an `sfj` object under the CGCS2000 spatial reference.

### Usage

```
sf_gk_proj_cgcs2000(sfj, degree = 6L)
```

### Arguments

| | |
|---|---|
| sfj | An `sf` object or can be converted to `sf` by `sf::st_as_sf()`. |
| degree | (optional) `3`-degree or `6`-degree zonal projection, default is `6L`. |

## Value

A character.

## Examples

```
gzma = sf::read_sf(system.file('extdata/gzma.gpkg',package = 'sdsfun')) |>
  sf::st_transform(4490)
sf_gk_proj_cgcs2000(gzma,3)
sf_gk_proj_cgcs2000(gzma,6)
```

---

`sf_utm_proj_wgs84` *generates wgs84 utm projection epsg coding character*

---

## Description

Generates a utm projection epsg coding character corresponding to an `sfj` object under the WGS84 spatial reference.

## Usage

```
sf_utm_proj_wgs84(sfj)
```

## Arguments

| | |
|---|---|
| `sfj` | An `sf` object or can be converted to `sf` by `sf::st_as_sf()`. |

## Value

A character.

## Examples

```
gzma = sf::read_sf(system.file('extdata/gzma.gpkg',package = 'sdsfun'))
sf_utm_proj_wgs84(gzma)
```

## sf_voronoi_diagram *generates voronoi diagram*

### Description

Generates Voronoi diagram (Thiessen polygons) for sf object

### Usage

```
sf_voronoi_diagram(sfj)
```

### Arguments

| | |
|---|---|
| sfj | An `sf` object. |

### Value

An `sf` object of polygon geometry type or can be converted to this by `sf::st_as_sf()`.

### Note

Only sf objects of (multi-)point type are supported to generate voronoi diagram and the returned result includes only the geometry column.

### Examples

```
pts = sf::read_sf(system.file('extdata/pts.gpkg',package = 'sdsfun'))
pts_v = sf_voronoi_diagram(pts)

library(ggplot2)
ggplot() +
  geom_sf(data = pts_v, color = 'red',
          fill = 'transparent') +
  geom_sf(data = pts, color = 'blue', size = 1.25) +
  theme_void()
```

## spade_psd *only spade power of spatial determinant*

### Description

only spade power of spatial determinant

### Usage

```
spade_psd(y, hs, wt)
```

### Arguments

| | |
|---|---|
| y | Dependent variable |
| hs | Independent variable |
| wt | Spatial weight matrix |

### Value

A numeric value

### Examples

```
gzma = sf::read_sf(system.file('extdata/gzma.gpkg',package = 'sdsfun'))
wt1 = inverse_distance_swm(gzma)
spade_psd(y = gzma$PS_Score,
          hs = discretize_vector(gzma$PS_Score,5),
          wt = wt1)
```

---

## spdep_contiguity_swm *constructs spatial weight matrices based on contiguity*

---

### Description

Constructs spatial weight matrices based on contiguity via spdep package.

### Usage

```
spdep_contiguity_swm(
  sfj,
  queen = TRUE,
  k = NULL,
  order = 1L,
  cumulate = TRUE,
  style = "W",
  zero.policy = TRUE
)
```

### Arguments

| | |
|---|---|
| sfj | An sf object or can be converted to sf by sf::st_as_sf(). |
| queen | (optional) if TRUE, using queen contiguity, otherwise rook contiguity. Default is TRUE. |
| k | (optional) The number of nearest neighbours. Ignore this parameter when not using distance based neighbours to construct spatial weight matrices. |
| order | (optional) The order of the adjacency object. Default is 1. |
| cumulate | (optional) Whether to accumulate adjacency objects. Default is TRUE. |

| | |
|---|---|
| style | (optional) style can take values W, B, C, and S. More to see spdep::nb2mat(). Default is W. |
| zero.policy | (optional) if FALSE stop with error for any empty neighbour sets, if TRUE permit the weights list to be formed with zero-length weights vectors. Default is TRUE. |

### Value

A matrix

### Note

When k is set to a positive value, using K-Nearest Neighbor Weights.

### Examples

```
gzma = sf::read_sf(system.file('extdata/gzma.gpkg',package = 'sdsfun'))
wt1 = spdep_contiguity_swm(gzma, k = 6, style = 'B')
wt2 = spdep_contiguity_swm(gzma, queen = TRUE, style = 'B')
wt3 = spdep_contiguity_swm(gzma, queen = FALSE, order = 2, style = 'B')
```

---

## spdep_distance_swm *constructs spatial weight matrices based on distance*

---

### Description

Constructs spatial weight matrices based on distance via spdep package.

### Usage

```
spdep_distance_swm(
  sfj,
  kernel = NULL,
  k = NULL,
  bandwidth = NULL,
  power = 1,
  style = "W",
  zero.policy = TRUE
)
```

### Arguments

| | |
|---|---|
| sfj | An sf object or can be converted to sf by sf::st_as_sf(). |
| kernel | (optional) The kernel function, can be one of uniform, triangular,quadratic(epanechnikov),quartic and gaussian. Default is NULL. |
| k | (optional) The number of nearest neighbours. Default is NULL. Only useful when kernel is provided. |

| | |
|---|---|
| `bandwidth` | (optional) The bandwidth, default is `NULL`. When the spatial reference of sf object is the geographical coordinate system, the unit of `bandwidth` is `km`. The unit used in the projection coordinate system are consistent with those used in the sf object coordinate system. |
| `power` | (optional) Default is `1`. Useful when `kernel` is not provided. |
| `style` | (optional) `style` can take values `W`, `B`, `C`, and `S`. More to see `spdep::nb2mat()`. Default is `W`. For spatial weights based on distance functions, a style of `B` means using the original value of the calculated distance function. |
| `zero.policy` | (optional) if `FALSE` stop with error for any empty neighbour sets, if `TRUE` permit the weights list to be formed with zero-length weights vectors. Default is `TRUE`. |

## Details

five different kernel weight functions:

- uniform: $K_{(z)} = 1/2$,for $|z| < 1$
- triangular $K_{(z)} = 1 - |z|$,for $|z| < 1$
- quadratic (epanechnikov) $K_{(z)} = \frac{3}{4}\left(1 - z^2\right)$,for $|z| < 1$
- quartic $K_{(z)} = \frac{15}{16}\left(1 - z^2\right)^2$,for $|z| < 1$
- gaussian $K_{(z)} = \frac{1}{\sqrt{2\pi}}e^{-\frac{z^2}{2}}$

For the equation above, $z = d_{ij}/h_i$ where $h_i$ is the bandwidth

## Value

A matrix

## Note

When `kernel` is setting, using distance weight based on kernel function, Otherwise the inverse distance weight will be used.

## Examples

```
pts = sf::read_sf(system.file('extdata/pts.gpkg',package = 'sdsfun'))
wt1 = spdep_distance_swm(pts, style = 'B')
wt2 = spdep_distance_swm(pts, kernel = 'gaussian')
wt3 = spdep_distance_swm(pts, k = 3, kernel = 'gaussian')
wt4 = spdep_distance_swm(pts, k = 3, kernel = 'gaussian', bandwidth = 10000)
```

## spdep_lmtest — *spatial linear models selection*

### Description

spatial linear models selection

### Usage

```
spdep_lmtest(formula, data, listw = NULL)
```

### Arguments

| | |
|---|---|
| `formula` | A formula for linear regression model. |
| `data` | An `sf` object of observation data. |
| `listw` | (optional) A listw. See `spdep::mat2listw()` and `spdep::nb2listw()` for details. |

### Value

A list

### Examples

```
gzma = sf::read_sf(system.file('extdata/gzma.gpkg',package = 'sdsfun'))
spdep_lmtest(PS_Score ~ ., gzma)
```

## spdep_nb — *construct neighbours list*

### Description

construct neighbours list

### Usage

```
spdep_nb(sfj, queen = TRUE, k = NULL, order = 1L, cumulate = TRUE)
```

### Arguments

- `sfj` — An `sf` object or can be converted to `sf` by `sf::st_as_sf()`.
- `queen` — (optional) if `TRUE`, using queen contiguity, otherwise rook contiguity. Default is `TRUE`.
- `k` — (optional) The number of nearest neighbours. Ignore this parameter when not using distance based neighbours.
- `order` — (optional) The order of the adjacency object. Default is `1`.
- `cumulate` — (optional) Whether to accumulate adjacency objects. Default is `TRUE`.

### Value

A neighbours list with class `nb`

### Note

When `k` is set to a positive value, using K-Nearest Neighbor

### Examples

```
pts = sf::read_sf(system.file('extdata/pts.gpkg',package = 'sdsfun'))
nb1 = spdep_nb(pts, k = 6)
nb2 = spdep_nb(pts, queen = TRUE)
nb3 = spdep_nb(pts, queen = FALSE, order = 2)
```

---

## spdep_skater — *spatial c(k)luster analysis by tree edge removal*

---

### Description

SKATER forms clusters by spatially partitioning data that has similar values for features of interest.

### Usage

```
spdep_skater(sfj, k = 6, nb = NULL, ini = 5, ...)
```

### Arguments

- `sfj` — An `sf` object of observation data. Please ensure that the attribute columns are included in the SKATER analysis.
- `k` — (optional) The number of clusters. Default is `6`.
- `nb` — (optional) A neighbours list with class nb. If the input `nb` is NULL, it will be constructed automatically using `spdep_nb()`.
- `ini` — (optional) The initial node in the minimal spanning tree. Defaul is `5`.
- `...` — (optional) Other parameters passed to spdep::skater().

### Value

A numeric vector of clusters.

### Examples

```
gzma = sf::read_sf(system.file('extdata/gzma.gpkg',package = 'sdsfun'))
gzma_c = spdep_skater(gzma,8)
gzma$group = gzma_c
plot(gzma["group"])
```

---

## spvar *spatial variance*

---

### Description

spatial variance

### Usage

```
spvar(x, wt, method = c("cpp", "r"))
```

### Arguments

| | |
|---|---|
| x | A numerical vector . |
| wt | The spatial weight matrix. |
| method | (optional) The method for calculating spatial variance, which can be chosen as either cpp or r. Default is cpp. |

### Details

The spatial variance formula is $\Gamma = \frac{\sum_i \sum_{j \neq i} \omega_{ij} \frac{(y_i - y_j)^2}{2}}{\sum_i \sum_{j \neq i} \omega_{ij}}$

### Value

A numerical value.

### Examples

```
gzma = sf::read_sf(system.file('extdata/gzma.gpkg',package = 'sdsfun'))
wt1 = inverse_distance_swm(gzma)
spvar(gzma$PS_Score,wt1)
```

`ssh_test` *spatial stratified heterogeneity test*

## Description

spatial stratified heterogeneity test

## Usage

```
ssh_test(y, hs)
```

## Arguments

| | |
|---|---|
| `y` | Variable Y, continuous numeric vector. |
| `hs` | Spatial stratification or classification of each explanatory variable. `factor`, `character`, `integer` or `data.frame`, `tibble` and `sf` object. |

## Value

A `tibble`

## Examples

```
ssh_test(y = 1:7, hs = c('x',rep('y',3),rep('z',3)))
```

`standardize_vector` *standardization*

## Description

To calculate the Z-score using variance normalization, the formula is as follows:
$Z = \frac{(x - mean(x))}{sd(x)}$

## Usage

```
standardize_vector(x)
```

## Arguments

| | |
|---|---|
| `x` | A numeric vector |

## Value

A standardized numeric vector

## Examples

```
standardize_vector(1:10)
```

---

tbl_all2int *convert discrete variables in a tibble to integers*

---

## Description

convert discrete variables in a tibble to integers

## Usage

```
tbl_all2int(tbl)
```

## Arguments

| | |
|---|---|
| tbl | A `tibble`,`data.frame` or `sf` object. |

## Value

A converted `tibble`,`data.frame` or `sf` object.

## Examples

```
demotbl = tibble::tibble(x = c(1,2,3,3,1),
                         y = letters[1:5],
                         z = c(1L,1L,2L,2L,3L),
                         m = factor(letters[1:5],levels = letters[5:1]))
tbl_all2int(demotbl)
```

---

tbl_xyz2mat *convert xyz tbl to matrix*

---

## Description

convert xyz tbl to matrix

## Usage

```
tbl_xyz2mat(tbl, x = 1, y = 2, z = 3)
```

## Arguments

| | |
|---|---|
| tbl | A `tibble`,`data.frame` or `sf` object. |
| x | (optional) The x-axis coordinates column number, default is `1`. |
| y | (optional) The y-axis coordinates column number, default is `2`. |
| z | (optional) The z (attribute) coordinates column number, default is `3`. |

## Value

A `list`.

**z_attrs_matrix** A matrix with attribute information.

**x_coords_matrix** A matrix with the x-axis coordinates.

**y_coords_matrix** A matrix with the y-axis coordinates.

## Examples

```
set.seed(42)
lon = rep(1:3,each = 3)
lat = rep(1:3,times = 3)
zattr = rnorm(9, mean = 10, sd = 1)
demodf = data.frame(x = lon, y = lat, z = zattr)
demodf
tbl_xyz2mat(demodf)
```

# Index