# Package: leastcostpath (via r-universe)

September 3, 2024

**Title** Modelling Pathways and Movement Potential Within a Landscape

**Version** 2.0.12

**Date** 2023-10-09

**Maintainer** Joseph Lewis <josephlewis1992@gmail.com>

**URL** Available at: https://CRAN.R-project.org/package=leastcostpath

**Description** Calculates cost surfaces based on slope to be used when modelling pathways and movement potential within a landscape (Lewis, 2021) <doi:10.1007/s10816-021-09522-w>.

**Depends** R (>= 3.4.0)

**Imports** terra (>= 1.5-34), sf (>= 1.0-8), igraph (>= 1.3.0), foreach (>= 1.5.2), gstat (>= 2.0-9), methods, stats, Matrix (>= 1.4-1), parallel, doParallel

**License** GPL-3

**Encoding** UTF-8

**RoxygenNote** 7.2.3

**Repository** https://josephlewis.r-universe.dev

**RemoteUrl** https://github.com/josephlewis/leastcostpath

**RemoteRef** HEAD

**RemoteSha** 86ee715ddec685afefecd963a4393a6cb8adb569

# Contents

---

add_dem_error                 *Incorporate vertical error into a Digital Elevation Model*

---

### Description

Incorporate vertical error into a Digital Elevation Model

### Usage

```
add_dem_error(x, rmse, type = "u", samples = NULL)
```

### Arguments

| | |
|---|---|
| x | spatRaster |
| rmse | numeric. Vertical Root Mean Square Error of the Digital Elevation Model |
| type | character type 'u' (unfiltered), 'n' (neighbourhood autocorrelation), and 'd' (mean spatial dependence) implemented. See details for more information |
| samples | numeric number of random spatial data locations sampled when using type 'd'. This can be used to overcome issues with computing time and memory limits |

### Details

Digital Elevation Models (DEMs) are representations of the earth's surface and are subject to error (Wechsler and Kroll, 2006)

The add_dem_error function incorporates vertical error into the supplied DEM. Three methods are implemented:

Unfiltered: Random error based on DEM RMSE range. Autocorrelation between random error is not accounted for. This can be interpreted as the worst case scenario

Neighbourhood autocorrelation: Random error is spatially autocorrelated by passing a mean low pass filter in a 3x3 neighbourhood over the surface

Mean Spatial Dependence: Random error is spatially autocorrelated by passing a DxD kernel over each cell. The centre cell of each kernel is replaced by the mean of the surrounding DxD cells. Distance of spatial dependence (D) is estimated by calcualting the semi-variogram nugget using the gstat package

Examples of RMSE for various datasets:

Shuttle Radar Topography Mission (SRTM) has a RMSE of 9.73m

Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER) Global Digital Elevation Model (GDEM) has a RMSE of 10.20m

Ordnance Survey OS Terrain 5 has a maximum RMSE of 2.5m

Ordnance Survey OS Terrain 50 has a maximum RMSE of 4m

TINITALY DEM has a RMSE of 4.3m

### Author(s)

Joseph Lewis

### References

Wechsler, S. P., & Kroll, C. N. (2006). Quantifying DEM Uncertainty and its Effect on Topographic Parameters. Photogrammetric Engineering & Remote Sensing, 72(9), 1081-1090. https://doi.org/10.14358/PERS.72.9.1081

Fisher, P., & Tate, N. J. (2006). Causes and consequences of error in digital elevation models. Progress in Physical Geography: Earth and Environment, 30(4), 467-489. https://doi.org/10.1191/0309133306pp492ra

### Examples

```
r <- terra::rast(system.file("extdata/SICILY_1000m.tif", package="leastcostpath"))

r2 <- add_dem_error(x = r, rmse = 4.3, type = "u")
```

---

add_global_stochasticity

*Add global stochasticity to a conductanceMatrix*

---

### Description

Add stochasticity to a conductanceMatrix based on a global value. This method is based on 'Multiple shortest paths (MSPs)' as proposed by Pinto and Keitt (2009)

### Usage

```
add_global_stochasticity(x, percent_quantile = 1)
```

### Arguments

| | |
|---|---|
| x | conductanceMatrix |
| percent_quantile | |
| | numeric value between 0 and 1. See details for more information |

**Details**

The add_global_stochasticity to a conductanceMatrix is based on the method proposed by Pinto and Keitt (2009). Rather than using a static neighbourhood (for example as supplied in the neighbours function in the create_slope_cs), the neighbourhood is redefined such that the adjacency is non-deterministic and is instead determined randomly based on the threshold value.

The algorithm proceeds as follows:

1. With a percent_quantile supplied, draw a random value between the minimum value in the conductanceMatrix and the supplied percent quantile

2. Replace values in conductanceMatrix below this random value with 0. This ensures that the conductance between the neighbours are 0, and thus deemed non-adjacent

Supplying a percent_quantile of 0 is equivalent to incorporating no stochasticity into the conductanceMatrix. That is, if the supplied percent_quantile is 0, then no values are below this value and thus no values will be replaced with 0 (see step 2). This therefore does not change the neigbourhood adjacency

The closer the percent_quantile is to 0, the less stochasticity is incorporated. For example, a percent_quantile value of 0.2 will result in the threshold being a random value between the minimum value in the conductanceMatrix and the 0.2 percent quantile of the values in the conductanceMatrix. All values in the conductanceMatrix below the random value will be replaced with 0 (i.e. the neighbours are no longer adjacent). In contrast, a percent_quantile value of 0.8 will result in the threshold being a random value between the minimum value in the conductanceMatrix and the 0.8 percent quantile of the values in the conductanceMatrix. In this case, there is greater probability that the random value will result in an increased number of values in the conductanceMatrix being replaced with 0.

**Author(s)**

Joseph Lewis

**References**

Pinto, N., & Keitt, T. H. (2009). Beyond the least-cost path: evaluating corridor redundancy using a graph-theoretic approach. Landscape Ecology, 24(2), 253-266. https://doi.org/10.1007/s10980-008-9303-y

**Examples**

```
r <- terra::rast(system.file("extdata/SICILY_1000m.tif", package="leastcostpath"))

slope_cs <- create_slope_cs(x = r, cost_function = "tobler", neighbours = 4)

slope_cs2 <- add_global_stochasticity(slope_cs, percent_quantile = 0.2)
```

---

buffer_validation          *Calculate the similarity of a least-cost path to a known route*

---

### Description

Calculates the similarity of a least-cost path to a known route using the buffer method proposed by Goodchild and Hunter (1997)

### Usage

```
buffer_validation(lcp, comparison, dist)
```

### Arguments

| | |
|---|---|
| lcp | sf or spatVector |
| comparison | sf or spatVector |
| dist | numeric buffer distances to assess similarity |

### Value

data.frame

### Author(s)

Joseph Lewis

### Examples

```
r <- terra::rast(system.file("extdata/SICILY_1000m.tif", package="leastcostpath"))

slope_cs <- create_slope_cs(x = r, cost_function = "tobler", neighbours = 4)

locs <- sf::st_sf(geometry = sf::st_sfc(
sf::st_point(c(839769, 4199443)),
sf::st_point(c(1038608, 4100024)),
crs = terra::crs(r)))

lcp1 <- create_lcp(x = slope_cs, origin = locs[1,], destination = locs[2,])

lcp2 <- create_lcp(x = slope_cs, origin = locs[2,], destination = locs[1,])

buffer_validation(lcp = lcp1, comparison = lcp2, dist = c(1000, 2500, 5000, 10000))
```

---

calculate_distance        *calculate distance between adjacent cells*

---

### Description

calculate distance between adjacent cells

### Usage

```
calculate_distance(x, adj)
```

### Arguments

| | |
|---|---|
| x | spatRaster |
| adj | matrix of adjacent cells |

### Details

calculate_distance function allows for both projected and geographic coordinate systems. If the coordinate system is geographic (e.g. wgs84) the distance is calculated using the sf::st_distance function else distance calculated using Pythagorean theorem

### Value

matrix euclidean distances between adjacent cells

### Author(s)

Joseph Lewis

---

check_locations        *check supplied locations*

---

### Description

checks that locations can be reached when calculating least-cost paths

### Usage

```
check_locations(x, locations)
```

### Arguments

| | |
|---|---|
| x | conductanceMatrix |
| locations | sf 'POINT' or 'MULTIPOINT', SpatVector, data.frame or matrix containing the locations coordinates |

## Details

Using the supplied conductanceMatrix and locations, the function checks whether: (1) the supplied locations are traversable from at least one adjacent cell (2) the supplied locations are within the extent of the supplied conductanceMatrix

## Value

```
message
```

## Author(s)

Joseph Lewis

## Examples

```
r <- terra::rast(system.file("extdata/SICILY_1000m.tif", package="leastcostpath"))

slope_cs <- create_slope_cs(x = r, cost_function = "tobler")

locs <- sf::st_sf(geometry = sf::st_sfc(
sf::st_point(c(861534, 4173726)),
sf::st_point(c(897360, 4155813)),
sf::st_point(c(928364, 4138588)),
crs = terra::crs(r)))

check_locations(x = slope_cs, locations = locs)
```

---

create_accum_cost *creates an accumulated cost surface*

---

## Description

Creates an accumulated cost surfaces from one or more origins

## Usage

```
create_accum_cost(
  x,
  origins,
  FUN = mean,
  rescale = FALSE,
  check_locations = FALSE
)
```

## Arguments

| | |
|---|---|
| x | `SpatRaster` |
| origins | sf 'POINT' or 'MULTIPOINT', `SpatVector`, `data.frame` or `matrix` containing the origins coordinates. If multiple origins are supplied then the multiple accumulated cost surfaces will be summarised using the FUN argument |
| FUN | `function` Apply a function to the cells of a SpatRaster. Default applied function is 'mean'. See terra::app() for more information |
| rescale | `logical`. if TRUE, values scaled to between 0 and 1. FALSE (default) |
| check_locations | |
| | `logical` if TRUE checks if origins are traversable. FALSE (default) |

## Value

`SpatRaster`

## Author(s)

Joseph Lewis

## Examples

```
r <- terra::rast(system.file("extdata/SICILY_1000m.tif", package="leastcostpath"))

slope_cs <- create_slope_cs(x = r, cost_function = "tobler", neighbours = 4)

locs <- sf::st_sf(geometry = sf::st_sfc(
sf::st_point(c(839769, 4199443)),
sf::st_point(c(1038608, 4100024)),
sf::st_point(c(907695, 4145478)),
crs = terra::crs(r)))

cc <- create_accum_cost(x = slope_cs, origins = locs, FUN = mean, rescale = FALSE)
```

---

create_cost_corridor      *creates a cost corridor*

---

## Description

Combines and averages the accumulated cost surfaces from origin-to-destination and destination-to-origin to identify areas of preferential movement that takes into account both directions of movement

## Usage

```
create_cost_corridor(x, origin, destination, rescale = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | `SpatRaster` |
| `origin` | `sf` 'POINT' or 'MULTIPOINT', `SpatVector`, `data.frame` or `matrix` containing the origin coordinates. Only the first row of the supplied object is used as the origin. |
| `destination` | `sf` 'POINT' or 'MULTIPOINT', `SpatVector`, `data.frame` or `matrix` containing the destination coordinates. Only the first row of the supplied object is used as the destination. |
| `rescale` | `logical`. if TRUE, values scaled to between 0 and 1. FALSE (default) |

## Value

`SpatRaster`

## Author(s)

Joseph Lewis

## Examples

```
r <- terra::rast(system.file("extdata/SICILY_1000m.tif", package="leastcostpath"))

slope_cs <- create_slope_cs(x = r, cost_function = "tobler", neighbours = 4)

locs <- sf::st_sf(geometry = sf::st_sfc(
sf::st_point(c(839769, 4199443)),
sf::st_point(c(1038608, 4100024)),
crs = terra::crs(r)))

cc <- create_cost_corridor(x = slope_cs, origin = locs[1,], destination = locs[2,], rescale = TRUE)
```

---

create_cs                          *Creates a cost surface from a SpatRaster*

---

## Description

Creates a cost surface using the values in the supplied SpatRaster. This function also provides for the inclusion of maximum slope traversable

The supplied 'spatRaster' object must have a projected CRS

## Usage

```
create_cs(
  x,
  neighbours = 16,
  dem = NULL,
  max_slope = NULL,
  exaggeration = FALSE
)
```

## Arguments

| | |
|---|---|
| `x` | SpatRaster |
| `neighbours` | numeric value. Number of directions used in the conductance matrix calculation. Expected numeric values are 4, 8, 16, 32, 48, or matrix object. 16 (default) |
| `dem` | SpatRaster Digital Elevation Model (DEM) |
| `max_slope` | numeric value. Maximum percentage slope that is traversable. Slope values that are greater than the specified max_slope are given a conductivity value of 0. If cost_function argument is 'campbell 2019' then max_slope is fixed at 30 degrees slope to reflect the maximum slope that the cost function is parametised to. NULL (default) |
| `exaggeration` | logical. if TRUE, positive slope values (up-hill movement) multiplied by 1.99 and negative slope values (down-hill movement) multiplied by 2.31 |

## Value

conductanceMatrix that numerically expresses the difficulty of moving across a surface based on the provided SpatRaster

## Author(s)

Joseph Lewis

## Examples

```
r <- terra::rast(system.file("extdata/SICILY_1000m.tif", package="leastcostpath"))

cs1 <- create_cs(x = r, neighbours = 16, dem = NULL, max_slope = NULL)
cs2 <- create_cs(x = r, neighbours = 16, dem = r, max_slope = 10)
```

---

| create_distance_cs | *Creates a distance-based conductance matrix* |
|---|---|

---

## Description

Creates a conductance matrix based on the distance between neighbouring cells. Distance corrected for if neighbours value is greater than 4.

## Usage

```
create_distance_cs(x, neighbours = 16, max_slope = NULL, exaggeration = FALSE)
```

## Arguments

| | |
|---|---|
| x | SpatRaster. Digital Elevation Model (DEM) |
| neighbours | numeric value. Number of directions used in the conductance matrix calculation. Expected numeric values are 4, 8, 16, 32, 48, or matrix object. 16 (default) |
| max_slope | numeric value. Maximum percentage slope that is traversable. Slope values that are greater than the specified max_slope are given a conductivity value of 0. If cost_function argument is 'campbell 2019' then max_slope is fixed at 30 degrees slope to reflect the maximum slope that the cost function is parametised to. NULL (default) |
| exaggeration | logical. if TRUE, positive slope values (up-hill movement) multiplied by 1.99 and negative slope values (down-hill movement) multiplied by 2.31 |

## Value

conductanceMatrix that numerically expresses the difficulty of moving across slope based on the provided cost function

## Author(s)

Joseph Lewis

## Examples

```
r <- terra::rast(system.file("extdata/SICILY_1000m.tif", package="leastcostpath"))

distance_cs <- create_distance_cs(x = r, neighbours = 4)
```

---

create_FETE_lcps            *Calculate Least-cost Paths from each location to all other locations*

---

## Description

Calculates Least-cost paths from-everywhere-to-everywhere. This is based on the approach proposed by White and Barber (2012).

## Usage

```
create_FETE_lcps(x, locations, cost_distance = FALSE, ncores = 1)
```

## Arguments

| | |
|---|---|
| x | conductanceMatrix |
| locations | sf 'POINT' or 'MULTIPOINT', SpatVector, data.frame or matrix containing the locations coordinates |
| cost_distance | logical if TRUE computes total accumulated cost from origin to destination. FALSE (default) |
| ncores | numeric Number of cores used when calculating least-cost paths from-everywhere-to-everywhere. 1 (default) |

## Value

sf or spatVector Least-cost paths from-everywhere-to-everywhere based on the supplied `conductanceMatrix`. If supplied `locations` is a `spatVector` object then `spatVector` object returned else `sf` object

## Author(s)

Joseph Lewis

## Examples

```
r <- terra::rast(system.file("extdata/SICILY_1000m.tif", package="leastcostpath"))

slope_cs <- create_slope_cs(x = r, cost_function = "tobler", neighbours = 4)

locs <- sf::st_sf(geometry = sf::st_sfc(
sf::st_point(c(839769, 4199443)),
sf::st_point(c(1038608, 4100024)),
sf::st_point(c(907695, 4145478)),
sf::st_point(c(907695, 4145478)),
crs = terra::crs(r)))

lcps <- create_FETE_lcps(x = slope_cs, locations = locs)
```

---

create_lcp                    *Calculate Least-cost Path from Origin to Destinations*

---

## Description

Calculates the Least-cost path from an origin location to one or more destination locations. Applies Dijkstra's algorithm as implemented in the igraph R package.

## Usage

```
create_lcp(
  x,
  origin,
  destination,
  cost_distance = FALSE,
  check_locations = FALSE
)
```

## Arguments

| | |
|---|---|
| x | `conductanceMatrix` |
| origin | sf 'POINT' or 'MULTIPOINT', `SpatVector`, `data.frame` or `matrix` containing the origin coordinates. Only the first row of the supplied object is used as the origin. |

| | |
|---|---|
| destination | sf 'POINT' or 'MULTIPOINT', SpatVector, data.frame or matrix containing the destination coordinates. If the object contains multiple coordinates then least-cost paths will be calculated from the origin to all destinations |
| cost_distance | logical if TRUE computes total accumulated cost from origin to the destinations. FALSE (default) |
| check_locations | |
| | logical if TRUE checks if origin and destination are traversable by the least-cost path. FALSE (default) |

## Value

sf Least-cost path from origin and destinations based on the supplied conductanceMatrix

## Author(s)

Joseph Lewis

## Examples

```
r <- terra::rast(system.file("extdata/SICILY_1000m.tif", package="leastcostpath"))

slope_cs <- create_slope_cs(x = r, cost_function = "tobler", neighbours = 4)

locs <- sf::st_sf(geometry = sf::st_sfc(
sf::st_point(c(839769, 4199443)),
sf::st_point(c(1038608, 4100024)),
sf::st_point(c(1017819, 4206255)),
sf::st_point(c(1017819, 4206255)),
crs = terra::crs(r)))

lcps <- create_lcp(x = slope_cs, origin = locs[1,], destination = locs)
```

---

| create_lcp_density | *creates a cumulative least-cost path raster* |
|---|---|

---

## Description

Cumulatively combines least-cost paths to idenify routes of preferential movement

## Usage

```
create_lcp_density(x, lcps, rescale = FALSE)
```

## Arguments

| | |
|---|---|
| x | SpatRaster |
| lcps | sf or spatVector |
| rescale | logical. if TRUE, values scaled to between 0 and 1. FALSE (default) |

## Value

```
SpatRaster
```

## Author(s)

Joseph Lewis

## Examples

```
r <- terra::rast(system.file("extdata/SICILY_1000m.tif", package="leastcostpath"))

slope_cs <- create_slope_cs(x = r, cost_function = "tobler", neighbours = 4)

locs <- sf::st_sf(geometry = sf::st_sfc(
sf::st_point(c(839769, 4199443)),
sf::st_point(c(1038608, 4100024)),
sf::st_point(c(907695, 4145478)),
crs = terra::crs(r)))

lcps <- create_FETE_lcps(x = slope_cs, locations = locs)

lcps_dens <- create_lcp_density(x = r, lcps = lcps)
```

---

create_slope_cs            *Creates a slope-based cost surface*

---

## Description

Creates a cost surface based on the difficulty of moving up and down slope. This function imple-
ments multiple isotropic and anisotropic cost functions that estimate the 'cost' of human movement
when traversing a landscape

The supplied 'spatRaster' object can have a projected or geographic coordinate system

## Usage

```
create_slope_cs(
  x,
  cost_function = "tobler",
  neighbours = 16,
  crit_slope = 12,
  max_slope = NULL,
  exaggeration = FALSE
)
```

## Arguments

| | |
|---|---|
| x | SpatRaster Digital Elevation Model (DEM) |
| cost_function | character or function. Cost function applied to slope values. See details for implemented cost functions. tobler (default) |
| neighbours | numeric value. Number of directions used in the conductance matrix calculation. Expected numeric values are 4, 8, 16, 32, 48, or matrix object. 16 (default) |
| crit_slope | numeric value. Critical Slope (in percentage) is 'the transition where switchbacks become more effective than direct uphill or downhill paths'. Cost of climbing the critical slope is twice as high as those for moving on flat terrain and is used for estimating the cost of using wheeled vehicles. Default value is 12, which is the postulated maximum gradient traversable by ancient transport (Verhagen and Jeneson, 2012). Critical slope only used in 'wheeled transport' cost function |
| max_slope | numeric value. Maximum percentage slope that is traversable. Slope values that are greater than the specified max_slope are given a conductivity value of 0. If cost_function argument is 'campbell 2019' or 'campbell' then max_slope is fixed at 30 degrees slope to reflect the maximum slope that the cost function is parametised to. NULL (default) |
| exaggeration | logical. if TRUE, positive slope values (up-hill movement) multiplied by 1.99 and negative slope values (down-hill movement) multiplied by 2.31 |

## Details

The following cost functions have been implemented however users may also supply their own cost function (see Examples):

"tobler", "tobler offpath", "davey", 'rees', "irmischer-clarke male", "irmischer-clarke offpath male", "irmischer-clarke female", "irmischer-clarke offpath female", "modified tobler", 'garmy', 'kondo-saino', "wheeled transport", "herzog", "llobera-sluckin", "naismith", "minetti", "campbell","campbell 2019 1","campbell 2019 5" ,"campbell 2019 10","campbell 2019 15","campbell 2019 20","campbell 2019 25","campbell 2019 30","campbell 2019 35","campbell 2019 40","campbell 2019 45","campbell 2019 50","campbell 2019 55","campbell 2019 60","campbell 2019 65","campbell 2019 70","campbell 2019 75","campbell 2019 80","campbell 2019 85","campbell 2019 90","campbell 2019 95","campbell 2019 99", "sullivan 167","sullivan 5", "sullivan 833"

Multiple travel rate percentiles implemented for campbell 2019 and sullivan, e.g. "campbell 2019 50" is the 50th percentile

## Value

conductanceMatrix that numerically expresses the difficulty of moving across slope based on the provided cost function

## Author(s)

Joseph Lewis

**References**

Tobler, W. 1993. Three Presentations on Geographical Analysis and Modeling. Technical Report 93-1 (Santa Barbara, CA)

Davey, R.C., M. Hayes and J.M. Norman 1994. "Running Uphill: An Experimental Result and Its Applications," The Journal of the Operational Research Society 45, 25

Rees, W.G. 2004. "Least-cost paths in mountainous terrain," Computers & Geosciences 30, 203–09

Irmischer, I.J. and K.C. Clarke 2018. "Measuring and modeling the speed of human navigation," Cartography and Geographic Information Science 45, 177–86

Márquez-Pérez, J., I. Vallejo-Villalta and J.I. Álvarez-Francoso 2017. "Estimated travel time for walking trails in natural areas," Geografisk Tidsskrift-Danish Journal of Geography 117, 53–62

Garmy, P. et al. 2005. "Logiques spatiales et 'systèmes de villes' en Lodévois de l'Antiquité à la période moderne," Temps et espaces de l'homme en société, analyses et modèles spatiaux en archéologie 335–46

Kondo, Y. and Y. Seino 2010. "GPS-aided walking experiments and data-driven travel cost modeling on the historical road of Nakasendō-Kisoji (Central Highland Japan)," Making History Interactive (Proceedings of the 37th International Conference, Williamsburg, Virginia, United States of America) 158–65

Herzog, I. 2013. "The potential and limits of Optimal Path Analysis," in Bevan, A. and M. Lake (edd.), Computational approaches to archaeological spaces (Publications of the Institute of Archaeology, University College London) 179–211

Llobera, M. and T.J. Sluckin 2007. "Zigzagging: Theoretical insights on climbing strategies," Journal of Theoretical Biology 249, 206–17

Naismith, W. 1892. "Excursions: Cruach Ardran, Stobinian, and Ben More," Scottish Mountaineering club journal 2, 136

Minetti, A.E. et al. 2002. "Energy cost of walking and running at extreme uphill and downhill slopes," Journal of Applied Physiology 93, 1039–46

Campbell, M.J., P.E. Dennison and B.W. Butler 2017. "A LiDAR-based analysis of the effects of slope, vegetation density, and ground surface roughness on travel rates for wildland firefighter escape route mapping," Int. J. Wildland Fire 26, 884

Campbell, M.J. et al. 2019. "Using crowdsourced fitness tracker data to model the relationship between slope and travel rates," Applied Geography 106, 93–107

Sullivan, P.R. et al. 2020. "Modeling Wildland Firefighter Travel Rates by Terrain Slope: Results from GPS-Tracking of Type 1 Crew Movement," Fire 3, 52

**Examples**

```
r <- terra::rast(system.file("extdata/SICILY_1000m.tif", package="leastcostpath"))

slope_cs <- create_slope_cs(x = r, cost_function = "tobler", neighbours = 4)
slope_cs <- create_slope_cs(x = r, cost_function = "campbell 2019 50", neighbours = 4)
slope_cs2 <- create_slope_cs(x = r,
cost_function = function(x) {(6 * exp(-3.5 * abs(x + 0.05))) / 3.6}, neighbours = 4)
```

---

| crop_cs | *Crop conductanceMatrix to extent* |

---

### Description

Crop conductanceMatrix to extent

### Usage

```
crop_cs(x, extent)
```

### Arguments

| | |
|---|---|
| x | spatRaster |
| extent | sf object or terra SpatRaster. Extent obtained from object using terra::ext |

### Details

conductanceMatrix cropped to extent of supplied Sf object or terra SpatRaster. conductance-Matrix spatRaster dimensions and Matrix dimensions update to reflect cropped extent

### Author(s)

Joseph Lewis

### Examples

```
r <- terra::rast(system.file("extdata/SICILY_1000m.tif", package="leastcostpath"))

slope_cs <- create_slope_cs(x = r, cost_function = "tobler", neighbours = 4)

ext <- sf::st_as_sfc(sf::st_bbox(rasterise(slope_cs)))
ext <- sf::st_buffer(ext, dist = -75000)
ext <- sf::st_as_sf(ext)

slope_cs_cropped <- crop_cs(slope_cs, extent = ext)
```

---

| force_isotropy | *Coerce an anisotropic cost surface to an isotropic cost surface* |

---

### Description

Averages conductance values from-to adjacent cells

### Usage

```
force_isotropy(x)
```

**Arguments**

x             conductanceMatrix

**Value**

conductanceMatrix

r <- terra::rast(system.file("extdata/SICILY_1000m.tif", package="leastcostpath"))

slope_cs_aniso <- create_slope_cs(x = r, cost_function = "tobler", neighbours = 4)

slope_cs_iso <- force_isotropy(x = slope_cs_aniso)

---

get_coordinates          *get coordinates from a variety of different object classes*

---

**Description**

get coordinates from a variety of different object classes

**Usage**

```
get_coordinates(x)
```

**Arguments**

x             coordinates. sf 'POINT' or 'MULTIPOINT', SpatVector, data.frame or matrix containing the locations coordinates

**Value**

matrix matrix of coordinates

**Author(s)**

Joseph Lewis

---

neighbourhood *neighbourhood matrix to represent adjacent cells*

---

### Description

neighbourhood matrix to represent adjacent cells

### Usage

```
neighbourhood(neighbours)
```

### Arguments

neighbours    numeric or matrix Expected numeric values are 4, 8, 16, 32, or 48. A user-
              supplied matrix of 0 and 1s can be supplied. Here, 0 denotes non-adjaency and
              1 denotes adjacency

### Value

matrix neighbourhood matrix

### Author(s)

Joseph Lewis

---

PDI_validation *Calculate the similarity of a least-cost path to a known route*

---

### Description

Calculates the similarity of a least-cost path to a known route using the path deviation index method
proposed by Jan et al. (1999)

### Usage

```
PDI_validation(lcp, comparison)
```

### Arguments

lcp           sf

comparison    sf

**Details**

The Path Deviation Index (pdi) measures the spatial separation between a pair of paths and aims to overcome the shortcomings of measuring the percentage of coverage of a least cost path from a comparison path (e.g. as implemented in the buffer_validation function).

The pdi index is defined as the area between paths divided by the Euclidean distance of the shortest path between the origin and destination of the paths. The index can be interpreted as the average distance between the paths.

`pdi = area / length`

The value of the pdi depends on the length of the path and makes comparison of pdis difficult for paths with different origins and destinations. This is overcome by normalising the pdi by the Euclidean distance of the shortest path between the origin and destination of the paths

`Normalised pdi = pdi / length * 100`

The normalised pdi is the percent of spatial separation between the two paths over the shortest path. For example, if a normalised pdi is 30 percent, it means that the average distance between two paths is 30 percent of the length of the shortest path. With normalised pdi, the spatial separations of all paths can be compared regardless of the length of the shortest path.

Note: If the lcp path has a different origin and destination than the comparison path, the origin and destination of the lcp path are replaced with the origin and destination of the comparison path. This to ensure that a polygon can be created between the two paths which is required for calculating the area of spatial separation.

**Value**

`sf` POLYGON of the area between the lcp and comparison with data.frame of area, pdi, max distance, and normalised pdi

**Author(s)**

Joseph Lewis

**Examples**

```
r <- terra::rast(system.file("extdata/SICILY_1000m.tif", package="leastcostpath"))

slope_cs <- create_slope_cs(x = r, cost_function = "tobler", neighbours = 4)

locs <- sf::st_sf(geometry = sf::st_sfc(
sf::st_point(c(839769, 4199443)),
sf::st_point(c(1038608, 4100024)),
crs = terra::crs(r)))

lcp1 <- create_lcp(x = slope_cs, origin = locs[1,], destination = locs[2,],
cost_distance = TRUE)

lcp2 <- create_lcp(x = slope_cs, origin = locs[2,], destination = locs[1,],
cost_distance = TRUE)

pdi_val <- PDI_validation(lcp = lcp1, comparison = lcp2)
```

---

```
plot.conductanceMatrix
```
*plot conductanceMatrix*

---

### Description

plot conductanceMatrix for visualisation. Conductivity values are the mean conductivity for each cell

### Usage

```
## S3 method for class 'conductanceMatrix'
plot(x, ...)
```

### Arguments

| | |
|---|---|
| x | conductanceMatrix |
| ... | arguments passed to `terra::plot` |

---

```
rasterise
```
*converts conductanceMatrix to SpatRaster*

---

### Description

converts conductanceMatrix to SpatRaster

### Usage

```
rasterise(x)
```

### Arguments

| | |
|---|---|
| x | conductanceMatrix |

### Value

`spatRaster`

### Author(s)

Joseph Lewis

## Examples

```
r <- terra::rast(system.file("extdata/SICILY_1000m.tif", package="leastcostpath"))

slope_cs <- create_slope_cs(x = r, cost_function = "tobler", neighbours = 4)

cs_rast <- rasterise(slope_cs)
```

---

replace_values          *replace values with values from another object*

---

### Description

Replace values of conductanceMatrix x with the values of conductanceMatrix y that coincide with the supplied sf object

### Usage

```
replace_values(x, y, sf)
```

### Arguments

| | |
|---|---|
| x | conductanceMatrix |
| y | conductanceMatrix |
| sf | sf |

### Details

The values of conductanceMatrix x are replaced with the values from conductanceMatrix y that coincide with the supplied sf object

### Value

conductanceMatrix

### Author(s)

Joseph Lewis

### Examples

```
r <- terra::rast(system.file("extdata/SICILY_1000m.tif", package="leastcostpath"))

x <- create_slope_cs(x = r, cost_function = "tobler", neighbours = 4)

locs <- sf::st_sf(geometry = sf::st_sfc(
sf::st_point(c(960745, 4166836)),
crs = terra::crs(r)))
locs <- sf::st_buffer(x = locs, dist = 25000)
```

```
y <- update_values(x = x, sf = locs, FUN = function(j) { j + 10})

output <- replace_values(x = x, y = y, sf = locs)

identical(y$conductanceMatrix, output$conductanceMatrix)
```

---

| update_values | *update values in a conductanceMatrix* |
|---|---|

---

### Description

Apply a function to values in the `conductanceMatrix` that coincide with the supplied sf object

### Usage

```
update_values(x, sf, FUN)
```

### Arguments

```
x              conductanceMatrix
sf             sf
FUN            function
```

### Details

An updated conductanceMatrix is produced by assessing which areas of the conductanceMatrix coincide with the supplied sf object. The values within the areas that coincide with the sf object are modified based on the supplied function

### Value

```
conductanceMatrix
```

### Author(s)

Joseph Lewis

### Examples

```
r <- terra::rast(system.file("extdata/SICILY_1000m.tif", package="leastcostpath"))

slope_cs <- create_slope_cs(x = r, cost_function = "tobler", neighbours = 4)

locs <- sf::st_sf(geometry = sf::st_sfc(
sf::st_point(c(960745, 4166836)),
crs = terra::crs(r)))

locs <- sf::st_buffer(x = locs, dist = 25000)
```

```
slope_cs2 <- update_values(x = slope_cs, sf = locs,
FUN = function(j) { j * 0.6})

slope_cs3 <- update_values(x = slope_cs, sf = locs,
FUN = function(j) { j + 10})

slope_cs4 <- update_values(x = slope_cs, sf = locs,
FUN = function(j) { replace(x = j, values = 0)})
```

# Index