Package: locdiff (via r-universe)

October 23, 2026

Type Package

Title Local Diffusion Exponent

Version 1.4

Maintainer Chun-Xiao Nie <niechunxiao2009@163.com>

Description The local diffusion exponent for undirected, unweighted networks characterizes the speed of local diffusion from a specified node and depends on the resistance distance.

License GPL-3

Encoding UTF-8

Depends igraph, MASS

RoxygenNote 7.3.2

NeedsCompilation no

Author Chun-Xiao Nie [aut, cre] (ORCID: <https://orcid.org/0000-0002-7790-0803>)

Config/pak/sysreqs libglpk-dev libxml2-dev

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

Date/Publication 2026-06-24 09:50:07 UTC

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

RemoteRef HEAD

RemoteSha 17fc1a9a07a12dd0f251c68bea98a1bae175ec9c

Contents

adaptivelocaldiffexp 2
exactlocaldiffexp 3
generalized_sierpinski 4
localdiffexp 5
renyiindex 6
resistance_distance_matrix 7

Index **8**

`adaptivelocaldiffexp` *Compute local diffusion exponents with adaptive fitting range*

Description

For each node, the mean squared resistance displacement is computed exactly via transition matrix powers. Then, for a fixed starting time t_min, we try all possible end times t_max from t_max_min to t_max_max, perform a log-log linear regression, and select the t_max that maximizes the coefficient of determination (R^2). The corresponding H_i is returned.

Usage

```
adaptivelocaldiffexp(adj, max_time, t_min, t_max_min, t_max_max)
```

Arguments

`adj`	Symmetric adjacency matrix (0/1, no self-loops) of a connected graph.
`max_time`	Maximum number of steps (t = 1, ..., max_time) for MSD calculation.
`t_min`	Fixed starting time for fitting (>= 1).
`t_max_min`	Minimum end time to try (must be >= t_min + 2 to have at least 3 points).
`t_max_max`	Maximum end time to try (must be <= max_time).

Value

A list with components:

`H`	Numeric vector of length n, estimated local diffusion exponents.
`best_R2`	Numeric vector of length n, the best R^2 achieved.
`best_tmax`	Integer vector of length n, the t_max that gave the best R^2.
`msd`	Matrix of size max_time x n, where rows are times 1:max_time and columns are nodes.
`fit_objects`	List of lm objects for the best fit (one per node).

References

Nie, Chun-Xiao. "Vertex-level diffusion scaling from resistance distance." .Chaos, Under Review.

Examples

```
# Path graph with 4 nodes
adj <- matrix(c(0,1,0,0,
                1,0,1,0,
                0,1,0,1,
                0,0,1,0), nrow=4, byrow=TRUE)
res <- adaptivelocaldiffexp(adj, max_time = 50,
                                          t_min = 10, t_max_min = 20, t_max_max = 40)
print(res$H)
```

`exactlocaldiffexp` *Compute local diffusion exponents for all nodes using exact analytical method*

Description

For each node i, the mean squared resistance displacement is computed exactly via the transition matrix powers: MSD_i(t), for t = 1, 2, ..., max_time. Then a log-log linear regression is performed over the specified time window to estimate the local diffusion exponent H_i.

Usage

```
exactlocaldiffexp(adj, max_time, fit_range)
```

Arguments

`adj`	Symmetric adjacency matrix (0/1, no self-loops) of a connected graph.
`max_time`	Maximum number of steps (starting from t=1).
`fit_range`	Numeric vector of length 2, e.g. c(t_min, t_max), specifying the time window for fitting (1 <= t_min < t_max <= max_time).

Value

A list with components:

`H`	Numeric vector of length n, the estimated local diffusion exponents.
`msd`	Matrix of size max_time x n, where rows correspond to times 1:max_time and columns correspond to nodes. Entry [t, i] = MSD_i(t).
`fit`	List of lm objects (one per node), or NULL if fit failed.

References

Nie, Chun-Xiao. "Vertex-level diffusion scaling from resistance distance." .Chaos, Under Review.

Examples

```
# Example: path graph with 4 nodes
adj <- matrix(c(0,1,0,0,
                1,0,1,0,
                0,1,0,1,
                0,0,1,0), nrow=4, byrow=TRUE)
res <- exactlocaldiffexp(adj, max_time = 50, fit_range = c(10, 40))
print(res$H)
```

`generalized_sierpinski` *Generalized Sierpiński graph*

Description

Constructs the generalized Sierpiński graph $S(n, G)$ as defined in "Generalized Sierpiński graphs" (Gravier et al.). The vertex set is $\{1, \ldots, k\}^n$ where $k = |V(G)|$, and edges are defined recursively based on the edges of G.

Usage

```
generalized_sierpinski(G, n)
```

Arguments

`G`	An igraph object representing the base graph. Its vertices must be labelled `1`, `2`, `...`, `k` (as characters or integers).
`n`	Integer, dimension ($n \geq 1$). `n = 1` returns `G`.

Value

An igraph object representing $S(n, G)$.

References

Gravier, Sylvain, Matjaž Kovše, and Aline Parreau. "Generalized Sierpiński graphs." Posters at EuroComb 11 (2011): 2016-0216.

Examples

```
library(igraph)

# Example 1: Base graph is a 4-cycle
G <- make_ring(4)
S2 <- generalized_sierpinski(G, 2)
plot(S2, vertex.size = 8, vertex.label = NA)

# Example 2: Base graph is a triangle (classical Sierpiński gasket)
G_tri <- make_graph(~ 1-2-3-1)
S3_tri <- generalized_sierpinski(G_tri, 3)
summary(S3_tri)  # vertices: 27, edges: 63

# Example 3: Base graph is a complete graph K4
G_K4 <- make_full_graph(4)
S2_K4 <- generalized_sierpinski(G_K4, 2)
summary(S2_K4)  # vertices: 16, edges: 96
```

`localdiffexp` *Compute the local diffusion exponent for a given starting node*

Description

Simulates random walks from a specified node on a graph and estimates the local diffusion exponent from the scaling of the mean squared resistance distance.

Usage

```
localdiffexp(
  R,
  start_node,
  adj,
  fit_range,
  num_paths = 100,
  max_time = 100,
  seed = NULL
)
```

Arguments

`R`	Resistance distance matrix (symmetric, zero diagonal).
`start_node`	Index or name of the starting node (must match row/column names of `R`).
`adj`	Adjacency matrix (symmetric, 0/1, zero diagonal).
`fit_range`	Numeric vector of length 2 specifying the time range for fitting, e.g. `c(t_start, t_end)` with `1 <= t_start < t_end <= max_time`.
`num_paths`	Number of random walk paths to simulate. Default 100.
`max_time`	Maximum number of steps (from 0 to `max_time`). Default 100.
`seed`	Optional random seed for reproducibility.

Value

A list containing:

`H`	Estimated local diffusion exponent \(H_i\).
`msd`	Data frame with columns `t` (time) and `msd` (mean squared resistance distance).
`fit`	`lm` object from the log-log fit.
`paths`	Matrix of resistance distances for each path (rows = time steps, columns = paths).

References

Nie, Chun-Xiao. "Vertex-level diffusion scaling from resistance distance." .Chaos, Under Review.

Examples

```
g<-make_lattice(length = 6, dim = 2)
adj <- as_adjacency_matrix(g,sparse = FALSE)
R <- resistance_distance_matrix(adj)
res <- localdiffexp(R, start_node = 1, adj = adj,fit_range = c(3, 30), max_time = 30)
print(res$H)
```

`renyiindex` *Compute Rényi's index for a wealth vector*

Description

Compute Rényi's index for a wealth vector

Usage

```
renyiindex(w, alpha)
```

Arguments

`w`	Numeric vector of wealth values (non-negative, at least one positive).
`alpha`	Positive parameter controlling the sensitivity of the index.

Value

The Rényi index, a number between 0 and 1.

References

Eliazar, Iddo. "Randomness, evenness, and Rényi's index." Physica A: Statistical Mechanics and its Applications 390.11 (2011): 1982-1990.

Examples

```
renyiindex(runif(100,0.1,1), 2)
```

`resistance_distance_matrix` *Compute the resistance distance matrix of a connected undirected graph*

Description

For a connected undirected graph, the resistance distance between two vertices is defined as the effective resistance when each edge is replaced by a unit resistor. It can be computed from the Moore–Penrose pseudoinverse of the graph Laplacian.

Usage

```
resistance_distance_matrix(adj_matrix)
```

Arguments

`adj_matrix`	A symmetric adjacency matrix (0/1, no self-loops) representing a connected undirected graph.

Value

A symmetric matrix `R` where `R[i,j]` is the resistance distance between vertices `i` and `j`. Diagonal entries are 0.

References

D.J.Klein and M.Randic, “Resistance distance,” Journal of Mathematical." Chemistry 12, 81–95 (1993).

Examples

```
# Example: a path graph with 3 vertices
adj <- matrix(c(0,1,0,
                1,0,1,
                0,1,0), nrow = 3, byrow = TRUE)
R <- resistance_distance_matrix(adj)
print(R)
```

Index

adaptivelocaldiffexp, 2

exactlocaldiffexp, 3

generalized_sierpinski, 4

localdiffexp, 5

renyiindex, 6
resistance_distance_matrix, 7