

Package ‘NetOrigin’

April 1, 2021

Title Origin Estimation for Propagation Processes on Complex Networks

Description Performs network-based source estimation. Different approaches are available: effective distance median, recursive backtracking, and centrality-based source estimation. Additionally, we provide public transportation network data as well as methods for data preparation, source estimation performance analysis and visualization.

Version 1.1-2

Depends R (>= 3.2.2)

Imports igraph, Hmisc, colorspace, mvtnorm, corpcor

License GPL-3

LazyData true

RoxygenNote 7.1.1

Collate '0_helper_net.r' 'NetOrigin.r' 'compute_mu_lambda.R'
'origin_helper.r' 'origin_methods.r' 'distance.r' 'data.r'
'data_handling.r' 'initial_condition_sib_model.R'
'robustness.r' 'stochastic_sib_model.R'

NeedsCompilation no

Author Juliane Manitz [aut, cre],
Jonas Harbering [ctb],
Jun Li [ctb]

Maintainer Juliane Manitz <r@manitz.org>

Repository CRAN

URL <https://netorigin.manitz.org/>

BugReports <https://github.com/jmanitz/NetOrigin/issues>

Date/Publication 2021-04-01 13:40:02 UTC

R topics documented:

| | |
|-------------------|---|
| aggr_data | 2 |
| analyze_ptn | 3 |
| compute_mu_lambda | 4 |

| | |
|---------------------------------------|----|
| delay-data | 5 |
| eff_dist | 6 |
| initial_condition_sib_model | 8 |
| NetOrigin | 9 |
| origin | 10 |
| origin-methods | 13 |
| origin_multiple | 15 |
| performance | 16 |
| plot_performance | 16 |
| plot_ptn | 18 |
| ptn-data | 19 |
| read_DB_data | 20 |
| robustness | 20 |
| robustness-methods | 22 |
| stochastic_sib_model | 23 |
| var_wtd_mean_cochran | 25 |

Index 26

| | |
|-----------|--|
| aggr_data | <i>convert individual event information to aggregated information per network node</i> |
|-----------|--|

Description

convert individual event information to aggregated information per network node

Usage

```
aggr_data(dat, from = NULL, cumsum = TRUE)
```

Arguments

| | |
|--------|---|
| dat | data.frame with variables 'node', 'time', 'delay', events data with single events with count magnitude |
| from | character in strftime format, e.g. "2014-06-12 16:15", data is subsetted accordingly before aggregation |
| cumsum | logical indicating whether data is aggregated by cumulative sum, default is TRUE |

Value

data.frame of dimension (T×K), where T is the number of observation times and K the number of network nodes. Thus, each row represents a snapshot of the spreading process at a specific observation time with the event magnitude observed at the network nodes. Rownames are observation times, colnames are node names.

See Also

Other data_handling: [read_DB_data\(\)](#)

| | |
|-------------|--|
| analyze_ptn | <i>analyze public transportation network characteristics</i> |
|-------------|--|

Description

analyze public transportation network characteristics

Usage

```
analyze_ptn(g)
```

Arguments

`g` [igraph](#) object, network graph representing the public transportation network, vertices represent stations, which are linked by an edge if there is a direct transfer between them

Value

'data.frame': 1 obs. of 7 variables:

- `vcount` number of nodes,
- `ecount` number of edges,
- `density` network graph density,
- `av_deg` average degree,
- `av_cent` average unit betweenness,
- `av_spl` average shortest path length,
- `diam` diameter, and
- `trans` transitivity.

References

Details to the computation and interpretation can be found in:

- Kolaczyk, E. D. (2009). Statistical analysis of network data: methods and models. Springer series in statistics. Springer. <DOI: 10.1007/978-0-387-88146-1>
- Manitz, J. (2014): Statistical Inference for Propagation Processes on Complex Networks. Ph.D. thesis, Georg-August-University Goettingen. Verlag Dr.-Hut, ISBN 978-3-8439-1668-4. Available online: <https://ediss.uni-goettingen.de/handle/11858/00-1735-0000-0022-5F38-B>.

See Also

Other network helper: [plot_ptn\(\)](#)

Examples

```
data(ptnAth)
analyze_ptn(ptnAth)

data(ptnGoe)
analyze_ptn(ptnGoe)
```

| | |
|-------------------|--|
| compute_mu_lambda | <i>Compute Mu and Lambda for Source Detection Function</i> |
|-------------------|--|

Description

compute_mu_lambda computes 'mu' and 'lambda' from training data and selected observers, for Gaussian source estimation with prior information

Usage

```
compute_mu_lambda(train.data, obs.vec, candidate.thres)
```

Arguments

| | |
|-----------------|--|
| train.data | training data for 'mu' and 'lambda' list computation format-list, length-number of cities/nodes format of train.data[[i]]- number of simulated scenarios x number of cities/nodes, each entry is minimum arrival time |
| obs.vec | list of cities ids used as observers |
| candidate.thres | threshold to determine if a node/city could be a candidate for source e.g. if we set this number to be 0.2, if in [x] simulated scenarios, there are only 10 percent scenarios a node [a] is infected, we do not think [a] is a potential source |

Value

a list, consisting of 3 variables: mu.mat, lambda.list, poss.candidate.vec
 mu.mat: matrix- number of cities/nodes x number of observers, each row represents- if this node is the source, the mean of arrival time vector;
 lambda.list: a length-number of cities/nodes list, each element is a number of observers x number of observers matrix- if a node is the source, the covariance matrix for arrival time vector;
 poss.candidate.vec: a boolean vector indicating if a node has the potential to be the source

Author(s)

Jun Li

References

Li, J., Manitz, J., Bertuzzo, E. and Kolaczyk, E.D. (2020). Sensor-based localization of epidemic sources on human mobility networks. arXiv preprint Available online: <https://arxiv.org/abs/2011.00138>.

Examples

```
# fake training data, indicating format
nnodes <- 851
max.day <- 1312
nsimu <- 20
train.data.fake <- list()
for (j in 1:nnodes) {
  train.data.fake[[j]] <- matrix(sample.int(max.day,
    size = nsimu*nnodes, replace = TRUE), nrow = nsimu, ncol = nnodes)
}
obs.vec <- (1:9)
candidate.thres <- 0.3
mu.lambda.list <- compute_mu_lambda(train.data.fake, obs.vec, candidate.thres)
```

delay-data

Delay propagation data examples simulated by LinTim software

Description

Delay propagation data examples simulated by LinTim software

delayAth Delay propagation data generated on the Athens metro network by LinTim software

delayGoe Delay propagation data generated on the Goettingen bus system by LinTim software

Details

delayAth Delay data on the Athens metro network. Propagation simulation under consideration of security distances and fixed-waiting time delay management. 'data.frame' with 510 observations (10 sequential time pictures for delay spreading pattern from 51 stations) of 53 variables (k0 true source, time, delays at 51 stations).

delayGoe Delay data on the directed Goettingen bus system. Propagation simulation under consideration of security distances and fixed-waiting time delay management. 'data.frame' with 2570 observations (10 sequential time pictures for delay spreading pattern from 257 stations) of 259 variables (k0 true source, time, delays at 257 stations).

Author(s)

Jonas Harbering

Source

Public transportation network datasets are generated by LinTim software (Integrated Optimization in Public Transportation; <https://www.lintim.net/index.php?go=data&lang=en>).

References

Manitz, J., J. Harbering, M. Schmidt, T. Kneib, and A. Schoebel (2017): Source Estimation for Propagation Processes on Complex Networks with an Application to Delays in Public Transportation Systems. *Journal of Royal Statistical Society C (Applied Statistics)*, 66: 521-536.

See Also[ptn-data](#)**Examples**

```
## Not run:
# compute effective distance
data(ptnAth)
athnet <- igraph::as_adjacency_matrix(ptnAth, sparse=FALSE)
p <- athnet/rowSums(athnet)
eff <- eff_dist(p)
# apply source estimation
if (requireNamespace("aplyr", quietly = TRUE)) {
  data(delayAth)
  res <- alply(.data=delayAth[, -c(1:2)], .margins=1, .fun=origin_edm, distance=eff,
              silent=TRUE, .progress='text')
  perfAth <- ldply(Map(performance, x = res, start = as.list(delayAth$k0),
                      list(graph = ptnAth)))
}

## End(Not run)
## Not run:
# compute effective distance
data(ptnGoe)
goenet <- igraph::as_adjacency_matrix(ptnGoe, sparse=FALSE)
p <- goenet/rowSums(goenet)
eff <- eff_dist(p)
# apply source estimation
if (requireNamespace("aplyr", quietly = TRUE)) {
  data(delayGoe)
  res <- alply(.data=delayGoe[, -c(1:2)], .margins=1, .fun=origin_edm, distance=eff,
              silent=TRUE, .progress='text')
  perfGoe <- ldply(Map(performance, x = res, start = as.list(delayGoe$k0),
                      list(graph = ptnGoe)))
}

## End(Not run)
```

 eff_dist

Computation of effective path distance

Description

eff_dist computes the effective distance between all nodes in the network

Usage

```
eff_dist(p)
```

```
eff_dijkstra(p, start)
```

```
spd_dijkstra(p, start)
```

Arguments

| | |
|-------|--|
| p | numeric matrix, representing the transition probability matrix for the network graph |
| start | start of path |

Value

A numeric matrix, representing the effective distance between all nodes in the network graph.

References

- Dijkstra, E. W. (1959): A note on two problems in connexion with graphs. *Numerische Mathematik*, 1, 269-271. <DOI: 10.1007/BF01386390>
- Brockmann, D. and Helbing, D. (2013): The hidden geometry of complex, network-driven contagion phenomena. *Science*, 342, 1337-1342. <DOI: 10.1126/science.1245200>
- Manitz, J. (2014): *Statistical Inference for Propagation Processes on Complex Networks*. Ph.D. thesis, Georg-August-University Goettingen. Verlag Dr. Hut, ISBN 978-3-8439-1668-4. Available online: <https://ediss.uni-goettingen.de/handle/11858/00-1735-0000-0022-5F38-B>.

Examples

```
# compute effective shortest path distance
data(ptnAth)
require(igraph)
net <- igraph::as_adjacency_matrix(ptnAth, sparse=FALSE)
p <- net/rowSums(net)
eff <- eff_dist(p)

# compute shortest path distance
data(ptnAth)
athnet <- as_adj(ptnAth, sparse=FALSE)
spd <- spd_dijkstra(athnet, start=1)

# compare calculations with the one from igraph
spd_igraph <- igraph::distances(ptnAth, v=1, algorithm='dijkstra')
all(spd[[1]] == spd_igraph)
```

```
initial_condition_sib_model
```

Provide Initial Condition for Function SIB_SS

Description

initial_condition_sib_model Compute Initial Condition for Function SIB_SS

Usage

```
initial_condition_sib_model(  
  POP_node,  
  sigma,  
  mu_B,  
  theta,  
  node_in,  
  in_prevalence = 0.001  
)
```

Arguments

| | |
|---------------|---|
| POP_node | vector, length represents number of cities/nodes; vector represents population at each node |
| sigma | symptomatic ratio, i.e., fraction of infected people that develop symptoms and are infective. (The remaining fraction enters directly the recovered compartment.) |
| mu_B | death rate of <i>V.cholerae</i> in the aquatic environment (day ⁻¹) |
| theta | contamination rate |
| node_in | index/indices for initial infected node(s) |
| in_prevalence | initial prevalence of symptomatic infected in a node, default is 0.1% |

Value

a 5 x number of nodes matrix, each row represents the following for all the nodes: Row 1: number of susceptible people, i.e., population expect infected and recovered for each node; Row 2: number of infected people; Row 3: number of recovered people; Row 4: bacteria concentration in equilibrium with infected individuals; Row 2: number of infected people, but representing cumulative cases

Author(s)

Jun Li

Examples

```
set.seed(2020)
popu <- rep(20000, 10)
sigma <- 0.05
mu_B <- 0.2
theta_max <- 16
theta <- runif(10, 0.1, 0.9) * theta_max
y0 <- initial_condition_sib_model(popu, sigma, mu_B, theta, c(3))
```

NetOrigin

Origin Estimation for Propagation Processes on Complex Networks

Description

Performs different approaches for network-based source estimation: effective distance median, recursive backtracking, and centrality-based source estimation. Additionally, we provide public transportation network data as well as methods for data preparation, source estimation performance analysis and visualization.

Details

The main function for origin estimation of propagation processes on complex network is [origin](#). Different methods are available: effective distance median ('edm'), recursive backtracking ('backtracking'), and centrality-based source estimation ('centrality'). For more details on the methodological background, we refer to the corresponding publications.

Author(s)

Juliane Manitz with contributions by Jonas Harbering

References

- Manitz, J., J. Harbering, M. Schmidt, T. Kneib, and A. Schoebel (2017): Source Estimation for Propagation Processes on Complex Networks with an Application to Delays in Public Transportation Systems. *Journal of Royal Statistical Society C (Applied Statistics)*, 66: 521-536.
- Manitz, J., Kneib, T., Schlather, M., Helbing, D. and Brockmann, D. (2014) Origin detection during food-borne disease outbreaks - a case study of the 2011 EHEC/HUS outbreak in Germany. *PLoS Currents Outbreaks*, 1. <DOI: 10.1371/currents.outbreaks.f3fdeb08c5b9de7c09ed9cbcef5f01f2>
- Comin, C. H. and da Fontoura Costa, L. (2011) Identifying the starting point of a spreading process in complex networks. *Physical Review E*, 84. <DOI: 10.1103/PhysRevE.84.056105>

Description

This is the main function for origin estimation for propagation processes on complex networks. Different methods are available: effective distance median ('edm'), recursive backtracking ('backtracking'), and centrality-based source estimation ('centrality'). For details on the methodological background, we refer to the corresponding publications.

`origin_edm` for effective distance-median origin estimation (Manitz et al., 2016)

Usage

```
origin(events, type = c("edm", "backtracking", "centrality", "bayesian"), ...)

origin_edm(events, distance, silent = TRUE)

origin_backtracking(events, graph, start_with_event_node = TRUE, silent = TRUE)

origin_centrality(events, graph, silent = TRUE)

origin_bayesian(
  events,
  thres.vec,
  obs.vec,
  mu.mat,
  lambda.list,
  poss.candidate.vec,
  prior,
  use.prior = TRUE
)
```

Arguments

| | |
|-----------------------|---|
| <code>events</code> | numeric vector of event counts at a specific time point; if type is 'bayesian', 'events' is a matrix, number of nodes x time points; entries represent number of cases |
| <code>type</code> | character specifying the method, 'edm', 'backtracking', 'centrality' and 'bayesian' are available. |
| <code>...</code> | parameters to be passed to origin methods origin_edm , origin_backtracking , origin_centrality or origin_centrality |
| <code>distance</code> | numeric matrix specifying the distance matrix (for type='edm') |
| <code>silent</code> | logical, should the messages be suppressed? |
| <code>graph</code> | igraph object specifying the underlying network graph (for type='backtracking' and type='centrality') |

| | |
|-----------------------|---|
| start_with_event_node | logical specifying whether backtracking only starts from nodes that experienced events (for type='backtracking') |
| thres.vec | vector, length represents number of cities/nodes, representing thresholds for cities/nodes that they are infected |
| obs.vec | list of cities ids used as observers |
| mu.mat | matrix- number of cities/nodes x number of observers, each row represents - if this node is the source, the mean of arrival time vector |
| lambda.list | a length-number of cities/nodes list, each element is a number of observers x number of observers matrix - if a node is the source, the covariance matrix for arrival time vector |
| poss.candidate.vec | a boolean vector indicating if a node has the potential to be the source |
| prior | vector, length - number of cities/nodes, prior for cities |
| use.prior | boolean, TRUE or FALSE, if use prior, default TRUE |

Value

origin_edm returns an object of class origin, list with

- est origin estimate
- aux data.frame with auxiliary variables
 - id as node identifier,
 - events for event magnitude,
 - wmean for weighted mean,
 - wvar for weighted variance, and
 - mdist mean distance from a node to all other nodes.
- type = 'edm' effective distance median origin estimation

origin_backtracking returns an object of class origin, list with

- est origin estimate
- aux data.frame with auxiliary variables
 - id as node identifier,
 - events for event magnitude, and
 - bcount for backtracking counts, how often backtracking identifies this source node.
- type = 'backtracking' backtracking origin estimation

origin_centrality returns an object of class origin, list with

- est origin estimate
- aux data.frame with auxiliary variables
 - id as node identifier,
 - events for event magnitude, and
 - cent for node centrality (betweenness divided degree).
- type = 'centrality' centrality-based origin estimation

a dataframe with columns 'nodes' and 'probab', indicating nodes indices and their posteriors

Author(s)

Juliane Manitz with contributions by Jonas Harbering
Jun Li

References

- Comin, C. H. and da Fontoura Costa, L. (2011). Identifying the starting point of a spreading process in complex networks. *Physical Review E*, 84. <doi: 10.1103/PhysRevE.84.056105>
- Manitz, J., J. Harbering, M. Schmidt, T. Kneib, and A. Schoebel (2017): Source Estimation for Propagation Processes on Complex Networks with an Application to Delays in Public Transportation Systems. *Journal of Royal Statistical Society C (Applied Statistics)*, 66: 521-536. <doi: 10.1111/rssc.12176>
- Manitz, J. (2014). *Statistical Inference for Propagation Processes on Complex Networks*. Ph.D. thesis, Georg-August-University Goettingen. Verlag Dr.-Hut, ISBN 978-3-8439-1668-4. Available online: <https://ediss.uni-goettingen.de/handle/11858/00-1735-0000-0022-5F38-B>.
- Manitz, J., Kneib, T., Schlather, M., Helbing, D. and Brockmann, D. (2014). Origin detection during food-borne disease outbreaks - a case study of the 2011 EHEC/HUS outbreak in Germany. *PLoS Currents Outbreaks*, 1. <doi: 10.1371/currents.outbreaks.f3fdeb08c5b9de7c09ed9cbcef5f01f2>
- Li, J., Manitz, J., Bertuzzo, E. and Kolaczyk, E.D. (2020). Sensor-based localization of epidemic sources on human mobility networks. arXiv preprint Available online: <https://arxiv.org/abs/2011.00138>.

See Also

Other origin-est: [origin_multiple\(\)](#)

Examples

```
data(delayGoe)

# compute effective distance
data(ptnGoe)
goenet <- igraph::as_adjacency_matrix(ptnGoe, sparse=FALSE)
p <- goenet/rowSums(goenet)
eff <- eff_dist(p)
# apply effective distance median source estimation
om <- origin(events=delayGoe[10,-c(1:2)], type='edm', distance=eff)
summary(om)
plot(om, 'mdist',start=1)
plot(om, 'wvar',start=1)
performance(om, start=1, graph=ptnGoe)

# backtracking origin estimation (Manitz et al., 2016)
ob <- origin(events=delayGoe[10,-c(1:2)], type='backtracking', graph=ptnGoe)
summary(ob)
plot(ob, start=1)
performance(ob, start=1, graph=ptnGoe)

# centrality-based origin estimation (Comin et al., 2011)
```

```

oc <- origin(events=delayGoe[10,-c(1:2)], type='centrality', graph=ptnGoe)
summary(oc)
plot(oc, start=1)
performance(oc, start=1, graph=ptnGoe)

# fake training data, indicating format
nnodes <- 851
max.day <- 1312
nsimu <- 20
max.case.per.day <- 10
train.data.fake <- list()
for (j in 1:nnodes) {
  train.data.fake[[j]] <- matrix(sample.int(max.day,
    size = nsimu*nnodes, replace = TRUE), nrow = nsimu, ncol = nnodes)
}
obs.vec <- (1:9)
candidate.thres <- 0.3
mu.lambda.list <- compute_mu_lambda(train.data.fake, obs.vec, candidate.thres)
# matrix representing number of cases per node per day
cases.node.day <- matrix(sample.int(max.case.per.day,
  size = nnodes*max.day, replace = TRUE), nrow = nnodes, ncol = max.day)
nnodes <- dim(cases.node.day)[1] # number of nodes
# fixed threshold for all nodes - 10 infected people
thres.vec <- rep(10, nnodes)
# flat/non-informative prior
prior <- rep(1, nnodes)
result2.df <- origin(events = cases.node.day, type = "bayesian",
  thres.vec = thres.vec,
  obs.vec = obs.vec,
  mu.mat=mu.lambda.list$mu.mat, lambda.list = mu.lambda.list$lambda.list,
  poss.candidate.vec=mu.lambda.list$poss.candidate.vec,
  prior=prior, use.prior=TRUE)

```

origin-methods

methods for origin estimation objects of class origin

Description

print produces an output for objects of class origin.

Usage

```

## S3 method for class 'origin'
print(x, ...)

## S3 method for class 'origin'
summary(object, x = object, ...)

## S3 method for class 'origin'

```

```
plot(x, y = "id", start, ...)

## S3 method for class 'origin'
performance(x, start, graph = NULL, ...)
```

Arguments

| | |
|--------|--|
| x | object of class origin , origin estimation object from function <code>origin_xxx</code> |
| ... | further arguments to be passed to default plot function |
| object | object of class origin , origin estimation object from function <code>origin_xxx</code> ; passed to x |
| y | character specifying the variable being plotted at the y-axis; options are 'id' for node identifier (default), 'mdist' for mean distance (only available for origin_edm) or 'wvar' for weighted variance (only available for origin_edm) |
| start | numeric, giving the node of the true origin |
| graph | igraph object specifying the underlying network graph with attribute 'length' on edges for calculation of distance to the correct origin |

Value

`performance.origin` returns a data.frame with variables

- `origin = start` representing the true origin,
- `est` the estimated node of origin,
- `hitt` logical indicating whether origin estimation is correct or not,
- `rank` rank of correct detection,
- `spj` number of segments from estimated origin to true origin (requires an [igraph](#) object),
- `dist` distance along the shortest path from estimated origin to true origin ([igraph](#) edge attribute `length`)

See Also

[origin plot_performance](#)

Examples

```
data(ptnGoe)
data(delayGoe)

res <- origin(events=delayGoe[10,-c(1:2)], type='centrality', graph=ptnGoe)
res

summary(res)
plot(res, start=1)
performance(res, start=1, graph=ptnGoe)
```

| | |
|-----------------|--|
| origin_multiple | <i>Multiple origin estimation using community partitioning</i> |
|-----------------|--|

Description

Multiple origin estimation using community partitioning

Usage

```
origin_multiple(  
  events,  
  type = c("edm", "backtracking", "centrality"),  
  graph,  
  no = 2,  
  distance,  
  fast = TRUE,  
  ...  
)
```

Arguments

| | |
|----------|---|
| events | numeric vector of event counts at specific time point |
| type | character specifying the method, 'edm', 'backtracking' and 'centrality' are available. |
| graph | igraph object specifying the underlying network graph |
| no | numeric specifying the number of supposed origins |
| distance | numeric matrix specifying the distance matrix |
| fast | logical specifying community partitioning algorithm, default is 'TRUE' that uses fastgreedy.community , 'FALSE' refers to leading.eigenvector.community |
| ... | parameters to be passed to origin methods origin_edm , origin_backtracking or origin_centrality |

Value

origin_multiple returns a list object with objects of class [origin](#) of length no

References

Zang, W., Zhang, P., Zhou, C. and Guo, L. (2014) Discovering Multiple Diffusion Source Nodes in Social Networks. *Procedia Computer Science*, 29, 443-452. <DOI: 10.1016/j.procs.2014.05.040>

See Also

Other origin-est: [origin\(\)](#)

Examples

```

data(ptnAth)
# backtracking
origin_multiple(events=delayAth[10,-c(1:2)], type='backtracking', graph=ptnAth, no=2)
# edm
athnet <- igraph::as_adjacency_matrix(ptnAth, sparse=FALSE)
p <- athnet/rowSums(athnet)
eff <- eff_dist(p)
origin_multiple(events=delayAth[10,-c(1:2)], type='edm', graph=ptnAth, no=2, distance=eff)

```

performance

generic method for performance evaluation

Description

generic method for performance evaluation

Usage

```
performance(x, ...)
```

Arguments

| | |
|-----|-------------------|
| x | object |
| ... | further arguments |

See Also

[origin-methods](#) [plot_performance](#)

plot_performance

A plot method combining a time series of performance results.

Description

A plot method combining a time series of performance results.

Usage

```
plot_performance(
  x,
  var = "rank",
  add = FALSE,
  offset = NULL,
  log = FALSE,
  col = 1,
  ylim = NULL,
  text.padding = 0.9,
  ...
)
```

Arguments

| | |
|--------------|---|
| x | data.frame obtained by combined results from <code>performance.origin</code> with variables X1 for time point, start for true origin, est for estimated origin, and performance variables |
| var | character, variable to be plotted, <code>performance.origin</code> returns rank, spj, and dist, default is 'rank' |
| add | logical, should be added to another performance plot |
| offset | POSIXct, starting time of spreading |
| log | logical, should y-axis be logarithmized? |
| col | numeric or character, color of lines |
| ylim | numeric vector, range of y axis |
| text.padding | a numeric value specifying the factor for the text position relative to the y values |
| ... | further graphical parameters passed to default plot function |

Examples

```
## Not run:
### delays on Goettingen bus network
# compute effective distance
data(ptnGoe)
goenet <- igraph::as_adjacency_matrix(ptnGoe, sparse=FALSE)
p <- goenet/rowSums(goenet)
eff <- eff_dist(p)
# apply source estimation
data(delayGoe)
if (requireNamespace("aply", quietly = TRUE)) {
  res <- alply(.data=delayGoe[11:20,-c(1:2)], .margins=1, .fun=origin_edm,
             distance=eff, silent=TRUE, .progress='text')
  perfGoe <- ldply(Map(performance, x = res, start = 2, list(graph = ptnGoe)))
  # performance plots
  plot_performance(perfGoe, var='rank', ylab='rank of correct detection', text.padding=0.5)
  plot_performance(perfGoe, var='dist', ylab='distance to correct detection')
}
```

```

### delays on Athens metro network
# compute effective distance
data(ptnAth)
athnet <- igraph::as_adjacency_matrix(ptnAth, sparse=FALSE)
p <- athnet/rowSums(athnet)
eff <- eff_dist(p)
# apply source estimation
data(delayAth)
if (requireNamespace("aplyr", quietly = TRUE)) {
  res <- alply(.data=delayAth[11:20,-c(1:2)], .margins=1, .fun=origin_edm,
              distance=eff, silent=TRUE, .progress='text')
  perfAth <- ldply(Map(performance, x = res, start = as.list(delayAth$k0),
                      list(graph = ptnAth)))
  # performance plots
  plot_performance(perfAth, var='rank', ylab='rank of correct detection',text.padding=0.5)
  plot_performance(perfAth, var='dist', ylab='distance to correct detection')
}

## End(Not run)

```

plot_ptn

A plot method for public transportation networks (PTNs).

Description

A plot method for public transportation networks (PTNs).

Usage

```

plot_ptn(
  g,
  color.coding = NULL,
  color.scheme = rev(sequential_hcl(5)),
  legend = FALSE,
  ...
)

```

Arguments

| | |
|---------------------------|--|
| <code>g</code> | igraph object, network graph representing the public transportation network, vertices represent stations, which are linked by an edge if there is a direct transfer between them |
| <code>color.coding</code> | numeric vector with length equal to the number of network nodes |
| <code>color.scheme</code> | character vector of length 5 indicating the vertex.color, default is <code>rev(sequential_hcl(5))</code> |
| <code>legend</code> | logical indicating whether legend for color-coding should be added or not. |
| <code>...</code> | further arguments to be passed to plot.igraph |

See Also

Other network helper: [analyze_ptn\(\)](#)

Examples

```
data(ptnAth)
plot_ptn(ptnAth)
```

```
data(ptnGoe)
plot_ptn(ptnGoe)
```

| | |
|----------|---|
| ptn-data | <i>Public transportation network datasets from LinTim software (Integrated Optimization in Public Transportation)</i> |
|----------|---|

Description

Public transportation network datasets from LinTim software (Integrated Optimization in Public Transportation)

ptnAth The data of the Athens Metro, consisting of 51 nodes and 52 edges.

- Vertex attributes: station name, additional station info.
- Edge attributes: track length (in meter), minimal and maximal time required to pass the track (in minutes).

ptnGoe The data of the Goettingen bus network, consisting of 257 nodes and 548 edges.

- Vertex attributes: station name.
- Edge attributes: track length (in meter), minimal and maximal time required to pass the track (in minutes).

Author(s)

Juliane Manitz and Jonas Harbering

Source

Public transportation network datasets are extracted from LinTim software (Integrated Optimization in Public Transportation; <https://www.lintim.net/index.php?go=data&lang=en>). Special thanks to Anita Schoebel for making the data available.

The Athens Metro data was collected by Konstantinos Gkoumas.

The Goettingen bus network data was collected by Barbara Michalski.

See Also

[delay-data](#)

Examples

```
# Athens metro system
data(ptnAth)
plot_ptn(ptnAth)

# Goettingen bus system
data(ptnGoe)
plot_ptn(ptnGoe)
```

| | |
|--------------|---|
| read_DB_data | <i>Reads a data file as provided by 'Deutsche Bahn' (for internal use).</i> |
|--------------|---|

Description

Reads a data file as provided by 'Deutsche Bahn' (for internal use).

Usage

```
read_DB_data(file)
```

Arguments

| | |
|------|--|
| file | character with path and file name containing the variables for 'stationID', 'date', 'hour', 'minutes', and 'delay' |
|------|--|

Value

data.frame with variables 'node', 'time', 'delay'

See Also

Other data_handling: [aggr_data\(\)](#)

| | |
|------------|--|
| robustness | <i>run robustness analysis for a source estimate by subsampling individual events.</i> |
|------------|--|

Description

run robustness analysis for a source estimate by subsampling individual events.

Usage

```
robustness(
  x,
  type = c("edm", "backtracking", "centrality"),
  prop,
  n = 100,
  ...
)
```

Arguments

| | |
|------|---|
| x | data.frame, dataset with individual events and their magnitude, to be passed to aggr_data |
| type | character, specifying the method, 'edm', 'backtracking' and 'centrality' are available. |
| prop | numeric, value between zero and one, proportion of events to be sampled |
| n | numeric, number of resamplings |
| ... | parameters to be passed to origin methods origin_edm , origin_backtracking or origin_centrality |

Details

We create subsamples of individual events and their magnitude using a sampling proportion p in $[0, 1]$. After aggregating the data, we apply the source estimation approach. Using this result, we deduce the relative frequency of how often the source estimate obtained with the complete data set can be recovered by source estimation based on the subsample. Thus, the estimate robustness is assessed by the proportion of estimate recovery.

Value

data.frame with columns

- est origin estimated when all data is evaluated
- rob estimate uncertainty, computed as the proportion of resamplings when origin estimate was recovered

See Also

[robustness-methods](#)

Examples

```
# generate random delay data
data(ptnAth)
require(igraph)
dat <- data.frame(node = sample(size = 500, make.names(V(ptnAth)$name), replace = TRUE),
                  time = sample(size = 500, 1:10, replace = TRUE),
                  delay = rexp(500, rate=10))
```

```

# compute effective distance
net <- igraph::as_adjacency_matrix(ptnAth, sparse=FALSE)
p <- net/rowSums(net)
eff <- eff_dist(p)
colnames(eff) <- paste('x.',colnames(eff),sep='')

# run robustness analysis
r5 <- robustness(x=dat, type='edm', prop=0.5, n=10, distance=eff)
summary(r5)
plot(r5)

# compare results
r9 <- robustness(x=dat, type='edm', prop=0.9, n=10, distance=eff)
plot(r9, add=TRUE, col='gray')

```

robustness-methods *methods for robustness estimation objects of class robustness*

Description

print produces an output for objects of class robustness

Usage

```

## S3 method for class 'robustness'
print(x, ...)

## S3 method for class 'robustness'
summary(object, x = object, ...)

## S3 method for class 'robustness'
plot(x, y = NULL, add = FALSE, ...)

```

Arguments

| | |
|--------|---|
| x | data.frame obtained by robustness , robustness estimation object for source estimation from function robustness |
| ... | further arguments passed to the default print method |
| object | object of class origin , origin estimation object from function origin_xxx ; passed to x |
| y | not used; default NULL |
| add | logical specifying whether this should be added to another robustness plot |

See Also

[robustness](#)

stochastic_sib_model *Stochastic SIB model for infected cases simulation*

Description

stochastic_sib_model Stochastic SIB model for infected cases simulation

Usage

```
stochastic_sib_model(
  mu,
  beta,
  rho,
  sigma,
  gamma,
  alpha,
  mu_B,
  m = 0.3,
  theta,
  nnodes,
  POP_node,
  fluxes,
  time_sim,
  y0
)
```

Arguments

| | |
|----------|---|
| mu | population natality and mortality rate (day ⁻¹) |
| beta | contact rate |
| rho | immunity loss rate (day ⁻¹) |
| sigma | symptomatic ratio, i.e., fraction of infected people that develop symptoms and are infective. (The remaining fraction enters directly the recovered compartment.) |
| gamma | rate at which people recover from cholera (day ⁻¹) |
| alpha | cholera induced mortality rate (day ⁻¹) |
| mu_B | death rate of <i>V.cholerae</i> in the aquatic environment (day ⁻¹) |
| m | parameter for infection force, default value is 0.3 |
| theta | contamination rate |
| nnodes | number of nodes/cities |
| POP_node | vector, length represents number of cities/nodes; vector represents population at each node |

| | |
|----------|--|
| fluxes | matrix, number of nodes x number of nodes where each row contains the probabilities a person travels from the given city (by Row Index) to another city (by Column Index). |
| time_sim | time steps for simulation, e.g., seq(0, 100, 0.1) |
| y0 | initial condition for stochastic_sib_model, output of 'initial_condition_sib_model' |

Value

a matrix, nnodes x number of time steps, representing number of new cases at each node, each time step

Author(s)

Jun Li

References

Li, J., Manitz, J., Bertuzzo, E. and Kolaczyk, E.D. (2020). Sensor-based localization of epidemic sources on human mobility networks. arXiv preprint Available online: <https://arxiv.org/abs/2011.00138>.

Examples

```
set.seed(2020)
popu <- rep(20000, 10)
sigma <- 0.05
mu_B <- 0.2
theta_max <- 16
theta <- runif(10, 0.1, 0.9) * theta_max
y0 <- initial_condition_sib_model(popu, sigma, mu_B, theta, c(3))
time_sim <- seq(0, 1, by=0.1)
mu <- 4e-05
beta_max <- 1
rho <- 0
beta <- runif(10, 0.1, 0.9) * beta_max
gamma <- 0.2
alpha <- 0
humanmob.mass <- matrix(runif(100, 0.1, 0.9), 10, 10)
diag(humanmob.mass) <- 0
for (j in 1:10) {
  humanmob.mass[j, ] <- humanmob.mass[j, ]/sum(humanmob.mass[j, ])
}
simu.list = stochastic_sib_model(mu = mu, beta = beta, rho = rho, sigma = sigma, gamma = gamma,
  alpha = alpha, mu_B = mu_B, theta = theta, nnodes = 10, POP_node = popu,
  fluxes = humanmob.mass, time_sim = time_sim, y0 = y0)
```

var_wtd_mean_cochran *Computes the variance of a weighted mean following the definition by Cochran (1977; see Gatz and Smith, 1995)*

Description

This is a helper method for weighted variance computation in `origin_edm`, which is the closest to the bootstrap.

Usage

```
var_wtd_mean_cochran(x, w)
```

Arguments

| | |
|---|---------------------------|
| x | numeric vector of values |
| w | numeric vector of weights |

Value

numeric value of weighted variance

References

- Gatz, D. F., and Smith, L. (1995). The standard error of a weighted mean concentration-I. Bootstrapping vs other methods. *Atmospheric Environment*, 29(11), 1185-1193. <DOI: 10.1016/1352-2310(94)00210-C>
- Gatz, D. F., and Smith, L. (1995). The standard error of a weighted mean concentration-II. Estimating confidence intervals. *Atmospheric Environment*, 29(11), 1195-1200. <DOI: 10.1016/1352-2310(94)00209-4>
- <https://r.789695.n4.nabble.com/Problem-with-Weighted-Variance-in-Hmisc-td826437.html>

Index

- * **data_handling**
 - aggr_data, 2
 - read_DB_data, 20
- * **data**
 - delay-data, 5
 - ptn-data, 19
- * **distance**
 - eff_dist, 6
- * **network helper**
 - analyze_ptn, 3
 - plot_ptn, 18
- * **origin-est**
 - origin, 10
 - origin_multiple, 15
- aggr_data, 2, 20, 21
- analyze_ptn, 3, 19
- compute_mu_lambda, 4
- delay-data, 5
- delayAth (delay-data), 5
- delayGoe (delay-data), 5
- eff_dijkstra (eff_dist), 6
- eff_dist, 6
- fastgreedy.community, 15
- igraph, 3, 14, 18
- initial_condition_sib_model, 8
- leading.eigenvector.community, 15
- NetOrigin, 9
- origin, 9, 10, 14, 15, 22
- origin-methods, 13
- origin_backtracking, 10, 15, 21
- origin_backtracking (origin), 10
- origin_bayesian (origin), 10
- origin centrality, 10, 15, 21
- origin centrality (origin), 10
- origin_edm, 10, 14, 15, 21, 25
- origin_edm (origin), 10
- origin_multiple, 12, 15
- performance, 16
- performance.origin, 17
- performance.origin (origin-methods), 13
- plot (robustness-methods), 22
- plot.igraph, 18
- plot.origin (origin-methods), 13
- plot_performance, 14, 16, 16
- plot_ptn, 3, 18
- print.origin (origin-methods), 13
- print.robustness (robustness-methods), 22
- ptn-data, 19
- ptnAth (ptn-data), 19
- ptnGoe (ptn-data), 19
- read_DB_data, 2, 20
- robustness, 20, 22
- robustness-methods, 22
- spd_dijkstra (eff_dist), 6
- stochastic_sib_model, 23
- strftime, 2
- summary (robustness-methods), 22
- summary.origin (origin-methods), 13
- var_wtd_mean_cochran, 25