Package: blvim (via r-universe)

October 24, 2026

Title Boltzmann–Lotka–Volterra Interaction Model

Version 0.1.1

Description Estimates Boltzmann–Lotka–Volterra (BLV) interaction model efficiently. Enables programmatic and graphical exploration of the solution space of BLV models when parameters are varied. See Wilson, A. (2008) <dx.doi.org/10.1098/rsif.2007.1288>.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.3

LinkingTo Rcpp, RcppArmadillo

Imports cli, collapse, Rcpp, rlang, stats

Suggests callr, covr, dplyr, ggplot2, ggrepel, knitr, pkgload, rmarkdown, sf, sloop, testthat (>= 3.0.0), vctrs, vdiffr, withr

Config/testthat/edition 3

Config/testthat/parallel true

URL https://fabrice-rossi.github.io/blvim/
https://fabrice-rossi.r-universe.dev/blvim

BugReports https://github.com/fabrice-rossi/blvim/issues

VignetteBuilder knitr

Depends R (>= 3.5)

LazyData true

NeedsCompilation yes

Author Fabrice Rossi [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-4638-1286>)

Maintainer Fabrice Rossi <Fabrice.Rossi@apiacoa.org>

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

Date/Publication 2026-01-14 18:00:13 UTC

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

RemoteRef HEAD

RemoteSha 9270fe1f189adfd63f6d492a678667c3facd3f22

Contents

as.data.frame.sim_list . . . 3
attractiveness . . . 4
autoplot.sim . . . 5
autoplot.sim_df . . . 9
autoplot.sim_list . . . 10
blvim . . . 13
c.sim_list . . . 16
costs . . . 17
costs.sim_list . . . 18
destination_flow . . . 19
destination_names . . . 19
destination_positions . . . 20
diversity . . . 21
flows . . . 24
flows_df . . . 25
fortify.sim . . . 26
fortify.sim_list . . . 29
french_cities . . . 31
french_cities_distances . . . 32
french_departments . . . 33
french_regions . . . 33
grid_attractiveness . . . 34
grid_autoplot . . . 35
grid_blvim . . . 37
grid_destination_flow . . . 39
grid_diversity . . . 40
grid_is_terminal . . . 41
grid_sim_converged . . . 42
grid_sim_iterations . . . 43
grid_var_autoplot . . . 44
inverse_cost . . . 46
is_terminal . . . 47
location_names . . . 48
location_positions . . . 49
median.sim_list . . . 50
names<-.sim_df . . . 52
nd_graph . . . 53
origin_names . . . 54
origin_positions . . . 55
production . . . 56
quantile.sim_list . . . 57
return_to_scale . . . 58
sim_column . . . 59
sim_converged . . . 60
sim_df . . . 61
sim_df_extract . . . 62

sim_distance 64
sim_is_bipartite 65
sim_iterations 66
sim_list 67
static_blvim 67
summary.sim_list 70
terminals 71

Index **74**

`as.data.frame.sim_list` *Coerce to a Data Frame*

Description

This function creates a data frame with a single column storing its collection of spatial interaction models. The default name of the column is `"sim"` (can be modified using the `sim_column` parameter). An more flexible alternative is provided by the `sim_df()` function.

Usage

```
## S3 method for class 'sim_list'
as.data.frame(x, ..., sim_column = "sim")
```

Arguments

`x`	a collection of spatial interaction models, an object of class `sim_list`
`...`	additional parameters (not used currently)
`sim_column`	the name of the `sim_list` column (default `"sim"`)

Value

a data frame

See Also

`sim_df()`

Examples

```
distances <- french_cities_distances[1:15, 1:15] / 1000 ## convert to km
production <- log(french_cities$population[1:15])
attractiveness <- log(french_cities$area[1:15])
all_flows_log <- grid_blvim(
  distances, production, c(1.1, 1.25, 1.5),
  c(1, 2, 3, 4) / 500, attractiveness,
  epsilon = 0.1,
  bipartite = FALSE,
```

```
  iter_max = 750
)
as.data.frame(all_flows_log, sim_column = "log flows")
```

`attractiveness` *Extract the attractivenesses from a spatial interaction model object*

Description

Extract the attractivenesses from a spatial interaction model object

Usage

```
attractiveness(sim, ...)
```

Arguments

`sim`	a spatial interaction model object
`...`	additional parameters

Value

a vector of attractivenesses at the destination locations

See Also

`production()`, `destination_flow()`

Examples

```
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- log(french_cities$population[1:10])
attractiveness <- log(french_cities$area[1:10])
model <- static_blvim(distances, production, 1.5, 1 / 250, attractiveness)
attractiveness(model)
## the names of the attractiveness vector are set from the distance matrix
## we remove them for testing equality
all.equal(as.numeric(attractiveness(model)), attractiveness)
```

`autoplot.sim` *Create a complete ggplot for a spatial interaction model*

Description

This function represents graphical the flows of a spatial interaction model, in different direct or aggregated forms.

Usage

```
## S3 method for class 'sim'
autoplot(
  object,
  flows = c("full", "destination", "attractiveness"),
  with_names = FALSE,
  with_positions = FALSE,
  show_destination = FALSE,
  show_attractiveness = FALSE,
  show_production = FALSE,
  cut_off = 100 * .Machine$double.eps^0.5,
  adjust_limits = FALSE,
  with_labels = FALSE,
  ...
)
```

Arguments

`object`	a spatial interaction model object
`flows`	`"full"` (default), `"destination"` or `"attractiveness"`, see details.
`with_names`	specifies whether the graphical representation includes location names (`FALSE` by default)
`with_positions`	specifies whether the graphical representation is based on location positions (`FALSE` by default)
`show_destination`	specifies whether the position based `"full"` flow figure includes a representation of the destination flows (`FALSE` by default)
`show_attractiveness`	specifies whether the position based `"full"` flow figure includes a representation of the attractivenesses (`FALSE` by default)
`show_production`	specifies whether the position based `"full"` flow figure includes a representation of the productions (`FALSE` by default)
`cut_off`	cut off limit for inclusion of a graphical primitive when `with_positions = TRUE`. In the full flow matrix representation, segments are removed when their flow is smaller than the cut off. In the attractiveness or destination representation, disks are removed when the corresponding value is below the cut off.

adjust_limits	if FALSE (default value), the limits of the position based graph are not adjusted after removing graphical primitives. This eases comparison between graphical representations with different cut off value. If TRUE, limits are adjusted to the data using the standard ggplot2 behaviour.
with_labels	if FALSE (default value) names are displayed using plain texts. If TRUE, names are shown using labels.
...	additional parameters, see details

Details

The graphical representation depends on the values of flows and with_positions. It is based on the data frame representation produced by fortify.sim().

If with_position is FALSE (default value), the graphical representations are "abstract". Depending on flows we have the following representations:

- "full": this is the default case for which the full flow matrix is represented. It is extracted from the spatial interaction model with flows() and displayed using a matrix representation with origin locations in rows and destination locations in columns. The colour of a cell corresponds to the intensity of a flow between the corresponding locations. To mimic the standard top to bottom reading order of a flow matrix, the top row of the graphical representation corresponds to the first origin location.
- "destination": the function computes the incoming flows for destination locations (using destination_flow()) and represents them with a bar plot (each bar is proportional to the incoming flow);
- "attractiveness": the function uses a bar plot to represent the attractivenesses of the destination locations (as given by attractiveness()). This is interesting for dynamic models where those values are updated during the iterations (see blvim() for details). When the calculation has converged (see sim_converged()), both "destination" and "attractiveness" graphics should be almost identical.

When the with_names parameter is TRUE, the location names (location_names()) are used to label the axis of the graphical representation. If names are not specified, they are replaced by indexes.

When the with_positions parameter is TRUE, the location positions (location_positions()) are used to produce more "geographically informed" representations. Notice that if no positions are known for the locations, the use of with_positions = TRUE is an error. Depending on flows we have the following representations:

- "full": this is the default case for which the full flow matrix is represented. Positions for both origin and destination locations are needed. The representation uses arrows from origin location positions to destination location positions. The thickness of the lines (linewidth aesthetics) is proportional to the flows. Only segments that carry a flow above the cut_off value are included. When the spatial interaction model is not bipartite (see sim_is_bipartite()), zero length segments corresponding to self exchange are removed. Additional parameters in ... are submitted to ggplot2::geom_segment(). This can be used to override defaults parameters used for the arrow shapes, for instance. Those parameters must be named. In addition to the individual flows, the representation can include location based information. If show_production is TRUE, the production constraints (obtained by production()) are shown

as disks centred on the origin locations. If show_destination is TRUE, incoming flows (obtained by destination_flow()) are shown as disks centred on the destination locations. If show_attractiveness is TRUE, attractivenesses (obtained by attractiveness()) are shown as disks centred on the destination locations. show_destination and show_attractiveness are not compatible (only one can be TRUE). show_production is compatible with show_destination or show_attractiveness for bipartite models only (see sim_is_bipartite()). When disks are used, segments are drawn without arrows, while the default drawing uses arrows. This can be modified with the additional parameters.

- "destination": the function draws a disk at each destination location using for the size aesthetics the incoming flow at this destination location (using destination_flow()). Only destinations with an incoming flow above the cut_off value are included.
- "attractiveness": the function draws a disk at each destination location using for the size aesthetics the attractiveness of the destination. When the calculation has converged (see sim_converged()), both "destination" and "attractiveness" graphics should be almost identical. Only destinations with an attractiveness above the cut_off value are included.

For the position based representations and when with_names is TRUE, the names of the destinations are added to the graphical representation . If with_labels is TRUE the names are represented as labels instead of plain texts (see ggplot2::geom_label()). If the ggrepel package is installed, its functions are used instead of ggplot2 native functions. When disks are used to show aggregated flows, the names match the chosen locations: for destination flow and attractiveness, destination locations are named, while for production, origin locations are named (they can be both named when the model is bipartite).

Value

a ggplot object

See Also

fortify.sim()

Examples

```
positions <- as.matrix(french_cities[1:10, c("th_longitude", "th_latitude")])
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- log(french_cities$area[1:10])
flows <- blvim(distances, production, 1.5, 1 / 150, attractiveness,
  origin_data = list(
    names = french_cities$name[1:10],
    positions = positions
  ),
  destination_data = list(
    names = french_cities$name[1:10],
    positions = positions
  ),
  bipartite = FALSE
)
ggplot2::autoplot(flows)
```

```
## bar plots should be almost identical if convergence occurred
sim_converged(flows)
ggplot2::autoplot(flows, "destination")
ggplot2::autoplot(flows, "attractiveness")
## names inclusion
ggplot2::autoplot(flows, "destination", with_names = TRUE) +
  ggplot2::coord_flip()
ggplot2::autoplot(flows, with_names = TRUE) +
  ggplot2::theme(axis.text.x = ggplot2::element_text(angle = 90))
## positions
ggplot2::autoplot(flows, "attractiveness", with_positions = TRUE) +
  ggplot2::scale_size_continuous(range = c(0, 6)) +
  ggplot2::coord_sf(crs = "epsg:4326")
ggplot2::autoplot(flows, "destination",
  with_positions = TRUE,
  with_names = TRUE
) +
  ggplot2::scale_size_continuous(range = c(0, 6)) +
  ggplot2::coord_sf(crs = "epsg:4326")
ggplot2::autoplot(flows, "destination",
  with_positions = TRUE,
  with_names = TRUE, with_labels = TRUE
) +
  ggplot2::scale_size_continuous(range = c(0, 6)) +
  ggplot2::coord_sf(crs = "epsg:4326")
ggplot2::autoplot(flows, with_positions = TRUE) +
  ggplot2::scale_linewidth_continuous(range = c(0, 2)) +
  ggplot2::coord_sf(crs = "epsg:4326")
ggplot2::autoplot(flows,
  with_positions = TRUE,
  arrow = ggplot2::arrow(length = ggplot2::unit(0.025, "npc"))
) +
  ggplot2::scale_linewidth_continuous(range = c(0, 2)) +
  ggplot2::coord_sf(crs = "epsg:4326")
## individual flows combined with destination flows
## no arrows
ggplot2::autoplot(flows,
  with_positions = TRUE,
  show_destination = TRUE
) +
  ggplot2::scale_linewidth_continuous(range = c(0, 2)) +
  ggplot2::coord_sf(crs = "epsg:4326")
## readding arrows
ggplot2::autoplot(flows,
  with_positions = TRUE,
  show_destination = TRUE,
  arrow = ggplot2::arrow(length = ggplot2::unit(0.025, "npc"))
) +
  ggplot2::scale_linewidth_continuous(range = c(0, 2)) +
  ggplot2::coord_sf(crs = "epsg:4326")
```

`autoplot.sim_df` *Create a complete ggplot for a spatial interaction models data frame*

Description

This function uses a tile plot from ggplot2 to display a single value for each of the parameter pairs used to produce the collection of spatial interaction models.

Usage

```
## S3 method for class 'sim_df'
autoplot(object, value, inverse = TRUE, ...)
```

Arguments

`object`	a data frame of spatial interaction models, an object of class `sim_df`
`value`	the value to display, default to `diversity` if unspecified
`inverse`	whether to use the cost scale parameter (default)
`...`	additional parameters (not used currently)

Details

The value to display is specified via an expression evaluated in the context of the data frame. It defaults to the diversity as computed by `diversity()`. See the below for examples of use.

The horizontal axis is used by default for the cost scale parameter, that is $1/\beta$. This is in general easier to read than using the inverse cost scale. The `inverse` parameter can be used to turn off this feature. The vertical axis is used by default for the return to scale parameter.

Value

a ggplot object

See Also

`sim_df()`, `diversity()`

Examples

```
positions <- as.matrix(french_cities[1:10, c("th_longitude", "th_latitude")])
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- log(french_cities$area[1:10])
all_flows <- grid_blvim(distances, production, seq(1.05, 1.45, by = 0.1),
  seq(1, 3, by = 0.5) / 400,
  attractiveness,
  bipartite = FALSE,
  epsilon = 0.1, iter_max = 1000,
```

```
)
all_flows_df <- sim_df(all_flows)

## default display: Shannon diversity
ggplot2::autoplot(all_flows_df)
## iterations
ggplot2::autoplot(all_flows_df, iterations)
## we leverage non standard evaluation to compute a different diversity
ggplot2::autoplot(all_flows_df, diversity(sim, "RW"))
## or to refer to columns of the data frame, either default ones
ggplot2::autoplot(all_flows_df, converged)
ggplot2::autoplot(all_flows_df, iterations)
## or added ones
all_flows_df["Nystuen & Dacey"] <- diversity(sim_column(all_flows_df), "ND")
ggplot2::autoplot(all_flows_df, `Nystuen & Dacey`)
```

`autoplot.sim_list` *Create a complete variability for a collection of spatial interaction models*

Description

This function represents graphically the variability of the flows represented by the spatial interaction models contained in a collection (a `sim_list` object).

Usage

```
## S3 method for class 'sim_list'
autoplot(
  object,
  flows = c("full", "destination", "attractiveness"),
  with_names = FALSE,
  with_positions = FALSE,
  cut_off = 100 * .Machine$double.eps^0.5,
  adjust_limits = FALSE,
  with_labels = FALSE,
  qmin = 0.05,
  qmax = 0.95,
  normalisation = c("none", "origin", "full"),
  ...
)
```

Arguments

`object`	a collection of spatial interaction models, a `sim_list`
`flows`	`"full"` (default), `"destination"` or `"attractiveness"`, see details.

with_names	specifies whether the graphical representation includes location names (FALSE by default)
with_positions	specifies whether the graphical representation is based on location positions (FALSE by default)
cut_off	cut off limit for inclusion of a graphical primitive when with_positions = TRUE. In the attractiveness or destination representation, circles are removed when the corresponding upper quantile value is below the cut off.
adjust_limits	if FALSE (default value), the limits of the position based graph are not adjusted after removing graphical primitives. This eases comparison between graphical representations with different cut off value. If TRUE, limits are adjusted to the data using the standard ggplot2 behaviour.
with_labels	if FALSE (default value) names are displayed using plain texts. If TRUE, names are shown using labels.
qmin	lower quantile, see details (default: 0.05)
qmax	upper quantile, see details (default: 0.95)
normalisation	when flows="full", the flows can be reported without normalisation (normalisation="none", the default value) or they can be normalised, either to sum to one for each origin location (normalisation="origin") or to sum to one globally (normalisation="full").
...	additional parameters, not used currently

Details

The graphical representation depends on the values of flows and with_positions. It is based on the data frame representation produced by fortify.sim_list(). In all cases, the variations of the flows are represented via quantiles of their distribution over the collection of models (computed with quantile.sim_list()). For instance, when flows is "destination", the function computes the quantiles of the incoming flows observed in the collection at each destination. We consider three quantiles:

- a lower quantile qmin defaulting to 0.05;
- the median;
- a upper quantile qmax defaulting to 0.95.

If with_position is FALSE (default value), the graphical representations are "abstract". Depending on flows we have the following representations:

- "full": the function displays the quantiles over the collection of models of the flows using nested squares (flows()). The graph is organised as matrix with origin locations on rows and destination locations on columns. At each row and column intersection, three nested squares represent respectively the lower quantile, the median and the upper quantile of the distribution of the flows between the corresponding origin and destination locations over the collection of models. The median square borders are thicker than the other two squares. The area of each square is proportional to the represented value.
- "destination": the function displays the quantiles over the collection of models of the incoming flows for each destination location (using destination_flow()). Quantiles are represented using ggplot2::geom_crossbar(): each location is represented by a rectangle that spans from its lower quantile to its upper quantile. An intermediate thicker bar represents the median quantile.

- "attractiveness": the function displays the quantiles over the collection of models of the attractiveness of each destination location (as given by attractiveness()). The graphical representation is the same as for "destination". This is interesting for dynamic models where those values are updated during the iterations (see blvim() for details). When the calculation has converged (see sim_converged()), both "destination" and "attractiveness" graphics should be almost identical.

When the with_names parameter is TRUE, the location names (location_names()) are used to label the axis of the graphical representation. If names are not specified, they are replaced by indexes.

When the with_positions parameter is TRUE, the location positions (location_positions()) are used to produce more "geographically informed" representations. Notice that if no positions are known for the locations, the use of with_positions = TRUE is an error. Moreover, flows = "full" is not supported: the function will issue a warning and revert to the position free representation if this value is used.

The representations for flows="destination" and flows="attractiveness" are based on the same principle. Each destination location is represented by a collection of three nested circles centred on the corresponding location position, representing respectively the lower quantile, the median and the upper quantile of the incoming flows or of the attractivenesses. The diameters of the circles are proportional to the quantities they represent. The border ot the median circle is thicker than the ones of the other circles.

When both with_positions and with_names are TRUE, the names of the destinations are added to the graphical representation. If with_labels is TRUE the names are represented as labels instead of plain texts (see ggplot2::geom_label()). If the ggrepel package is installed, its functions are used instead of ggplot2 native functions.

Value

a ggplot object

See Also

fortify.sim_list(), quantile.sim_list()

Examples

```
positions <- as.matrix(french_cities[1:10, c("th_longitude", "th_latitude")])
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- log(french_cities$area[1:10])
all_flows <- grid_blvim(distances, production, seq(1.05, 1.45, by = 0.1),
  seq(1, 3, by = 0.5) / 400,
  attractiveness,
  bipartite = FALSE,
  epsilon = 0.1, iter_max = 1000,
  destination_data = list(
    names = french_cities$name[1:10],
    positions = positions
  ),
  origin_data = list(
    names = french_cities$name[1:10],
```

```
    positions = positions
  )
)
ggplot2::autoplot(all_flows, with_names = TRUE) +
  ggplot2::theme(axis.text.x = ggplot2::element_text(angle = 90))
ggplot2::autoplot(all_flows, with_names = TRUE, normalisation = "none") +
  ggplot2::theme(axis.text.x = ggplot2::element_text(angle = 90))
ggplot2::autoplot(all_flows,
  flow = "destination", with_names = TRUE,
  qmin = 0, qmax = 1
) +
  ggplot2::theme(axis.text.x = ggplot2::element_text(angle = 90))
ggplot2::autoplot(all_flows,
  flow = "destination", with_positions = TRUE,
  qmin = 0, qmax = 1
) + ggplot2::scale_size_continuous(range = c(0, 6)) +
  ggplot2::coord_sf(crs = "epsg:4326")
ggplot2::autoplot(all_flows,
  flow = "destination", with_positions = TRUE,
  qmin = 0, qmax = 1,
  cut_off = 1.1
) +
  ggplot2::coord_sf(crs = "epsg:4326")
ggplot2::autoplot(all_flows,
  flow = "destination", with_positions = TRUE,
  with_names = TRUE,
  with_labels = TRUE,
  qmin = 0, qmax = 1,
  cut_off = 1.1
) +
  ggplot2::coord_sf(crs = "epsg:4326")
```

`blvim` *Compute an equilibrium solution of the Boltzmann-Lotka-Volterra model*

Description

This function computes flows between origin locations and destination locations at an equilibrium solution of A. Wilson's Boltzmann-Lotka-Volterra (BLV) interaction model. The BLV dynamic model is initialised with the production constraints at the origin locations and the initial values of the the attractiveness of destination locations. Iterations update the attractivenesses according the received flows.

Usage

```
blvim(
  costs,
  X,
```

```
  alpha,
  beta,
  Z,
  bipartite = TRUE,
  origin_data = NULL,
  destination_data = NULL,
  epsilon = 0.01,
  iter_max = 50000,
  conv_check = 100,
  precision = 1e-06,
  quadratic = FALSE
)
```

Arguments

costs	a cost matrix
X	a vector of production constraints
alpha	the return to scale parameter
beta	the inverse cost scale parameter
Z	a vector of initial destination attractivenesses
bipartite	when TRUE (default value), the origin and destination locations are considered to be distinct. When FALSE, a single set of locations plays the both roles. This has only consequences in functions specific to this latter case such as terminals().
origin_data	NULL or a list of additional data about the origin locations (see details)
destination_data	NULL or a list of additional data about the destination locations (see details)
epsilon	the update intensity
iter_max	the maximal number of steps of the BLV dynamic
conv_check	number of iterations between to convergence test
precision	convergence threshold
quadratic	selects the update rule, see details.

Details

In a step of the BLV model, flows are computed according to the production constrained entropy maximising model proposed by A. Wilson and implemented in static_blvim(). Then the flows received at a destination are computed as follows

$$\forall j, \quad D_j = \sum_{i=1}^{n} Y_{ij},$$

for destination j. This enables updating the attractivenesses by making them closer to the received flows, i.e. trying to reduce $|D_j - Z_j|$.

A. Wilson and co-authors proposed two different update strategies:

1. The original model proposed in Harris & Wilson (1978) updates the Z_j as follows

$$Z_j^{t+1} = Z_j^t + \epsilon(D_j^t - Z_j^t)$$

2. In Wilson (2008), the update is given by

$$Z_j^{t+1} = Z_j^t + \epsilon(D_j^t - Z_j^t)Z_j^t$$

In both cases, ϵ is given by the `epsilon` parameter. It should be smaller than 1. The first update is used when the `quadratic` parameter is `FALSE` which is the default value. The second update is used when `quadratic` is `TRUE`.

Updates are performed until convergence or for a maximum of `iter_max` iterations. Convergence is checked every `conv_check` iterations. The algorithm is considered to have converged if

$$\|Z^{t+1} - Z^t\| < \delta(\|Z^{t+1}\| + \delta),$$

where δ is given by the `precision` parameter.

Value

an object of class `sim`(and `sim_blvim`) for spatial interaction model that contains the matrix of flows between the origin and the destination locations as well as the final attractivenesses computed by the model.

Location data

While models in this package do not use location data beyond `X` and `Z`, additional data can be stored and used when analysing spatial interaction models.

Origin and destination location names:

Spatial interaction models can store names for origin and destination locations, using `origin_names<-()` and `destination_names<-()`. Names are taken by default from names of the cost matrix `costs`. More precisely, `rownames(costs)` is used for origin location names and `colnames(costs)` for destination location names.

Origin and destination location positions:

Spatial interaction models can store the positions of the origin and destination locations, using `origin_positions<-()` and `destination_positions<-()`.

Specifying location data:

In addition to the functions mentioned above, location data can be specified directly using the `origin_data` and `destination_data` parameters. Data are given by a list whose components are not interpreted excepted the following ones:

- `names` is used to specify location names and its content has to follow the restrictions documented in `origin_names<-()` and `destination_names<-()`
- `positions` is used to specify location positions and its content has to follow the restrictions documented in `origin_positions<-()` and `destination_positions<-()`

References

Harris, B., & Wilson, A. G. (1978). "Equilibrium Values and Dynamics of Attractiveness Terms in Production-Constrained Spatial-Interaction Models", Environment and Planning A: Economy and Space, 10(4), 371-388. doi:10.1068/a100371

Wilson, A. (2008), "Boltzmann, Lotka and Volterra and spatial structural evolution: an integrated methodology for some dynamical systems", J. R. Soc. Interface.5865-871 doi:10.1098/rsif.2007.1288

See Also

`grid_blvim()` for systematic exploration of parameter influence, `static_blvim()` for the static model.

Examples

```
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- log(french_cities$area[1:10])
## rescale to production
attractiveness <- attractiveness / sum(attractiveness) * sum(production)
flows <- blvim(distances, production, 1.5, 1 / 250, attractiveness)
flows
```

---

c.sim_list *Combine multiple sim_list objects into a single one*

Description

This function combines the `sim_list` and `sim` objects use as arguments in a single `sim_list`, provided they are compatible. Compatibility is defined as in `sim_list()`: all spatial interaction models must share the same costs as well as the same origin and destination data.

Usage

```
## S3 method for class 'sim_list'
c(...)
```

Arguments

`...`	`sim_list` and `sim` to be concatenated.

Value

A combined object of class `sim_list`.

Examples

```
distances <- french_cities_distances[1:15, 1:15] / 1000 ## convert to km
production <- log(french_cities$population[1:15])
attractiveness <- log(french_cities$area[1:15])
all_flows_log <- grid_blvim(
  distances, production, c(1.1, 1.25, 1.5),
  c(1, 2, 3, 4) / 500, attractiveness,
  epsilon = 0.1,
  bipartite = FALSE,
  iter_max = 750
)
production <- rep(1, 15)
attractiveness <- rep(1, 15)
all_flows_unit <- grid_blvim(
  distances, production, c(1.1, 1.25, 1.5),
  c(1, 2, 3, 4) / 500, attractiveness,
  epsilon = 0.1,
  bipartite = FALSE,
  iter_max = 750
)
all_flows <- c(all_flows_log, all_flows_unit)
```

---

costs *Extract the cost matrix used to compute this model*

Description

Extract the cost matrix used to compute this model

Usage

```
costs(sim, ...)
```

Arguments

sim	a spatial interaction model with a cost matrix
...	additional parameters

Value

the cost matrix

Examples

```
positions <- matrix(rnorm(10 * 2), ncol = 2)
distances <- as.matrix(dist(positions))
production <- rep(1, 10)
attractiveness <- c(2, rep(1, 9))
model <- static_blvim(distances, production, 1.5, 1, attractiveness)
```

```
costs(model) ## should be equal to distances above
all.equal(costs(model), distances)
```

`costs.sim_list` *Extract the common cost matrix from a collection of spatial interaction models*

Description

Extract the common cost matrix from a collection of spatial interaction models

Usage

```
## S3 method for class 'sim_list'
costs(sim, ...)
```

Arguments

`sim`	a collection of spatial interaction models, an object of class `sim_list`
`...`	additional parameters

Value

the cost matrix

See Also

`costs()`, `grid_blvim()`

Examples

```
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- log(french_cities$area[1:10])
all_flows <- grid_blvim(
  distances, production, c(1.25, 1.5),
  c(1, 2, 3, 4) / 500, attractiveness
)
## should be TRUE
identical(distances, costs(all_flows))
```

destination_flow *Compute the flows incoming at each destination location*

Description

Compute the flows incoming at each destination location

Usage

```
destination_flow(sim, ...)
```

Arguments

sim	a spatial interaction model object
...	additional parameters

Value

a vector of flows incoming at destination locations

See Also

production(), attractiveness()

Examples

```
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- log(french_cities$population[1:10])
attractiveness <- log(french_cities$area[1:10])
model <- static_blvim(distances, production, 1.5, 1 / 250, attractiveness)
destination_flow(model)
## should be different from the attractiveness as the model is static
attractiveness(model)
```

destination_names *Names of destination locations in a spatial interaction model*

Description

Functions to get or set the names of the destination locations in a spatial interaction model (or in a collection of spatial interaction models).

Usage

```
destination_names(sim)

destination_names(sim) <- value
```

Arguments

sim	a spatial interaction model object (an object of class sim) or a collection of spatial interaction models (an object of class sim_list)
value	a character vector of length equal to the number of destination locations, or NULL (vectors of adapted length are converted to character vectors)

Value

for destination_names NULL or a character vector with one name per destination locations in the model. for destination_names<- the modified sim object or sim_list object.

See Also

location_names(), origin_names()

Examples

```
distances <- french_cities_distances[1:10, 1:10]
production <- rep(1, 10)
attractiveness <- rep(1, 10)
## the row/column names of the cost matrix are used for the location
model <- static_blvim(distances, production, 1.5, 1 / 250000, attractiveness)
destination_names(model)
destination_names(model) <- french_cities$name[1:10]
destination_names(model)
```

---

destination_positions *positions of destination locations in a spatial interaction model*

Description

Functions to get or set the positions of the destination locations in a spatial interaction model.

Usage

```
destination_positions(sim)

destination_positions(sim) <- value
```

Arguments

sim	a spatial interaction model object
value	a matrix with as many rows as the number of destination locations and 2 or 3 columns, or NULL

Value

for `destination_positions` `NULL` or coordinate matrix for the destination locations. for `destination_positions<-` the modified `sim` object

Positions

Location positions are given by numeric matrices with 2 or 3 columns. The first two columns are assumed to be geographical coordinates while the 3rd column can be used for instance to store altitude. Coordinates are interpreted as is in graphical representations (see `autoplot.sim()`). They are not matched to the costs as those can be derived from complex movement models and other non purely geographic considerations.

See Also

`location_positions()`, `origin_positions()`

Examples

```
positions <- as.matrix(french_cities[1:10, c("th_longitude", "th_latitude")])
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- rep(1, 10)
model <- static_blvim(distances, production, 1.5, 1, attractiveness)
destination_positions(model) <- positions
destination_positions(model)
```

---

diversity *Compute the diversity of the destination locations in a spatial interaction model*

Description

This function computes the diversity of the destination locations according to different definitions that all aim at estimating a number of active destinations, i.e., the number of destination locations that receive a "significant fraction" of the total flow or that are attractive enough. The function applies also to a collection of spatial interaction models as represented by a `sim_list`.

Usage

```
diversity(
  sim,
  definition = c("shannon", "renyi", "ND", "RW"),
  order = 1L,
  activity = c("destination", "attractiveness"),
  ...
)

## S3 method for class 'sim'
```

```
diversity(
  sim,
  definition = c("shannon", "renyi", "ND", "RW"),
  order = 1L,
  activity = c("destination", "attractiveness"),
  ...
)

## S3 method for class 'sim_list'
diversity(
  sim,
  definition = c("shannon", "renyi", "ND", "RW"),
  order = 1L,
  activity = c("destination", "attractiveness"),
  ...
)
```

Arguments

`sim`	a spatial interaction model object (an object of class `sim`) or a collection of spatial interaction models (an object of class `sim_list`)
`definition`	diversity definition `"shannon"` (default), `"renyi"` (see details) or a definition supported by `terminals()`
`order`	order of the Rényi entropy, used only when `definition="renyi"`
`activity`	specifies whether the diversity is computed based on the destination flows (for `activity="destination"`, the default case) or on the attractivenesses (for `activity="attractiveness`
`...`	additional parameters

Details

In general, the activity of a destination location is measured by its incoming flow a.k.a. its destination flow. If Y is a flow matrix, the destination flows are computed as follows

$$\forall j, \quad D_j = \sum_{i=1}^{n} Y_{ij},$$

for each destination j (see `destination_flow()`). This is the default calculation mode in this function (when the parameter `activity` is set to `"destination"`).

For dynamic models produced by `blvim()`, the destination attractivenesses can be also considered as activity measures. When convergence occurs, the values are identical, but prior convergence they can be quite different. When `activity` is set to `"attractiveness"`, the diversity measures are computed using the same formula as below but with D_j replaced by Z_j (as given by `attractiveness()`).

To compute their diversity using entropy based definitions, the activities are first normalised to be interpreted as a probability distribution over the destination locations. For instance for destination flows, we use

$$\forall j, \quad p_j = \frac{D_j}{\sum_{k=1}^{n} D_k}.$$

The most classic diversity index is given by the exponential of Shannon's entropy (parameter `definition="shannon"`). This gives

$$\text{diversity}(p, \text{Shannon}) = \exp\left(-\sum_{k=1}^{n} p_k \ln p_k\right).$$

Rényi generalized entropy can be used to define a collection of other diversity metrics. The Rényi diversity of order γ is the exponential of the Rényi entropy of order γ of the p distribution, that is

$$\text{diversity}(p, \text{Rényi}, \gamma) = \exp\left(\frac{1}{1-\gamma} \ln\left(\sum_{k=1}^{n} p_k^{\gamma}\right)\right).$$

This is defined directly only for $\gamma \in]0, 1[\cup]1, \infty[$, but extensions to the limit case are straightforward:

- $\gamma = 1$ is Shannon's entropy/diversity
- $\gamma = 0$ is the max-entropy, here $\ln(n)$ and thus the corresponding diversity is the number of locations
- $\gamma = \infty$ is the min-entropy, here $-\log \max_k p_k$ and thhe corresponding diversity is $\frac{1}{\max_k p_k}$

The `definition` parameter specifies the diversity used for calculation. The default value is `shannon` for Shannon's entropy (in this case the `order` parameter is not used). Using `renyi` gives access to all Rényi diversities as specified by the `order` parameter. Large values of `order` tend to generate underflows in the calculation that will trigger the use of the min-entropy instead of the exact Rényi entropy.

In addition to those entropy based definition, terminal based calculations are also provided. Using any definition supported by the `terminals()` function, the diversity is the number of terminals identified. Notice this applies only to interaction models in which origin and destination locations are identical, i.e. when the model is not bipartite. In addition, the notion of terminals is based on destination flows and cannot be used with activities based on attractivenesses. `definition` can be:

- `"ND"` for the original Nystuen and Dacey definition
- `"RW"` for the variant by Rihll and Wilson

See `terminals()` for details.

When applied to a collection of spatial interaction models (an object of class `sim_list`) the function uses the same parameters (`definition` and `order`) for all models and returns a vector of diversities. This is completely equivalent to `grid_diversity()`.

Value

the diversity of destination flows (one value per spatial interaction model)

References

Jost, L. (2006), "Entropy and diversity", Oikos, 113: 363-375. doi:10.1111/j.2006.00301299.14714.x

See Also

destination_flow(), attractiveness(), terminals(), sim_is_bipartite()

Examples

```
distances <- french_cities_distances[1:15, 1:15] / 1000 ## convert to km
production <- log(french_cities$population[1:15])
attractiveness <- rep(1, 15)
flows <- blvim(distances, production, 1.5, 1 / 100, attractiveness,
  bipartite = FALSE
)
diversity(flows)
sim_converged(flows)
## should be identical because of convergence
diversity(flows, activity = "attractiveness")
diversity(flows, "renyi", 2)
diversity(flows, "RW")
```

---

flows *Extract the flow matrix from a spatial interaction model object*

Description

Extract the flow matrix from a spatial interaction model object

Usage

```
flows(sim, ...)
```

Arguments

sim	a spatial interaction model object
...	additional parameters

Value

a matrix of flows between origin locations and destination locations

See Also

flows_df() for a data frame version of the flows, destination_flow() for destination flows.

Examples

```
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- log(french_cities$population[1:10])
attractiveness <- log(french_cities$area[1:10])
## rescale to production
attractiveness <- attractiveness / sum(attractiveness) * sum(production)
model <- static_blvim(distances, production, 1.5, 1 / 500, attractiveness)
flows(model)
```

`flows_df` *Extract the flow matrix from a spatial interaction model object in data frame format*

Description

Extract the flow matrix from a spatial interaction model object in data frame format

Usage

```
flows_df(sim, ...)
```

Arguments

`sim`	a spatial interaction model object
`...`	additional parameters (not used currently)

Details

This function extracts the flow matrix in a long format. Each row contains the flow between an origin location and a destination location. The resulting data frame has at least three columns:

- `origin_idx`: identifies the origin location by its index from 1 to the number of origin locations
- `destination_idx`: identifies the destination location by its index from 1 to the number of destination locations
- `flow`: the flow between the corresponding location

In addition, if location information is available, it will be included in the data frame as follows:

- location names are included using columns `origin_name` or `destination_name`
- positions are included using 2 or 3 columns (per location type, origin or destination) depending on the number of dimensions used for the location. The names of the columns are by default `origin_x`, `origin_y` and `origin_z` (and equivalent names for destination location) unless coordinate names are specified in the location positions. In this latter case, the names are prefixed by `origin_` or `destination_`. For instance, if the destination location position coordinates are named `"longitude"` and `"latitude"`, the resulting columns will be `destination_longitude` and `destination_latitude`.

Value

a data frame of flows between origin locations and destination locations with additional content if available (see Details).

See Also

location_positions(), location_names(), flows()

Examples

```
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- log(french_cities$population[1:10])
attractiveness <- log(french_cities$area[1:10])
## rescale to production
attractiveness <- attractiveness / sum(attractiveness) * sum(production)
## simple case (no positions and default names)
model <- static_blvim(distances, production, 1.5, 1 / 500, attractiveness)
head(flows_df(model))
## with location data
positions <- as.matrix(french_cities[1:10, c("th_longitude", "th_latitude")])
model <- static_blvim(distances, production, 1.5, 1 / 500, attractiveness,
  origin_data = list(positions = positions),
  destination_data = list(positions = positions)
)
head(flows_df(model))
## with names
origin_names(model) <- french_cities$name[1:10]
destination_names(model) <- french_cities$name[1:10]
head(flows_df(model))
```

---

fortify.sim *Turn a spatial interaction model into a data frame*

Description

This function extracts from a spatial interaction model different types of data frame that can be used to produce graphical representations. autoplot.sim() leverages this function to produce its graphical representations.

Usage

```
## S3 method for class 'sim'
fortify(
  model,
  data,
  flows = c("full", "destination", "attractiveness"),
  with_names = FALSE,
  with_positions = FALSE,
```

```
  cut_off = 100 * .Machine$double.eps^0.5,
  ...
)
```

Arguments

model	a spatial interaction model object
data	not used
flows	"full" (default), "destination" or "attractiveness", see details.
with_names	specifies whether the extracted data frame includes location names (FALSE by default)
with_positions	specifies whether the extracted data frame is based on location positions (FALSE by default)
cut_off	cut off limit for inclusion of a flow row in the final data frame.
...	additional parameters, not used currently

Details

The data frame produced by the method depends on the values of flows and with_positions. The general principal is to have one row per flow, either a single flow from an origin location to a destination location, or an aggregated flow to a destination location. Flows are stored in one column of the data frame, while the other columns are used to identify origin and destination.

If with_position is FALSE (default value), data frames are simple. Depending on flows, the function extracts different data frames:

- "full": this is the default case for which the full flow matrix is extracted. The data frame has three variables:
  - origin: identifies the origin location by its index from 1 to the number of origin locations
  - destination: identifies the destination location by its index from 1 to the number of destination locations
  - flow: the flow between the corresponding location It is recommend to use flows_df() for more control over the extraction outside of simple graphical representations.
- "destination": the data frame has only two or three columns:
  - destination: identifies the destination location by its index from 1 to the number of destination locations
  - flow: the incoming flows (see destination_flow())
  - name: the name of the destination location if with_names is TRUE
- "attractiveness": the data frame has also two ot three columns, destination and name as in the previous case and attractiveness which contains the attractivenesses of the destinations (see attractiveness()).

When the with_positions parameter is TRUE, the location positions (location_positions()) are used to produce more "geographically informed" extractions. Notice that if no positions are known for the locations, the use of with_positions = TRUE is an error. Depending on flows we have the following representations:

- `"full"`: this is the default case for which the full flow matrix is extracted. Positions for both origin and destination locations are needed. The data frame contains five columns:
  - the first two columns are used for the coordinates of the origin locations (see below for the names of the columns)
  - `xend` and `yend` are used for the coordinates of the destination locations
  - `flow` is used for the flows
- `"destination"` and `"attractiveness"` produce both a data frame with three or four columns. As when `with_positions` is `FALSE`, one column is dedicated either to the incoming flows (`destination_flow()`) for `flows="destination"` (the name of the column is `destination`) or to the attractivenesses (`attractiveness()`), in which case its name is `attractiveness`. The other two columns are used for the positions of the destination locations. Their names are the names of the columns of the positions (`colnames(destination_location(object))`) or `"x"` and `"y"`, when such names are not specified. If `with_names` is `TRUE`, a `name` column is included and contains the names of the destination locations.

In the position based data frames, rows are excluded from the returned data frames when the flow they represent are small, i.e. when they are smaller than the `cut_off` value.

Value

a data frame, see details

See Also

`autoplot.sim()`, `flows_df()`

Examples

```
positions <- matrix(rnorm(10 * 2), ncol = 2)
colnames(positions) <- c("X", "Y")
distances <- as.matrix(dist(positions))
production <- rep(1, 10)
attractiveness <- c(2, rep(1, 9))
flows <- blvim(distances, production, 1.5, 4, attractiveness,
  origin_data =
    list(names = LETTERS[1:10], positions = positions),
  destination_data =
    list(names = LETTERS[1:10], positions = positions)
)
ggplot2::fortify(flows)
ggplot2::fortify(flows, flows = "destination")
ggplot2::fortify(flows, flows = "attractiveness")
## positions
ggplot2::fortify(flows, flows = "attractiveness", with_positions = TRUE)
## names and positions
ggplot2::fortify(flows,
  flows = "destination", with_positions = TRUE,
  with_names = TRUE
)
ggplot2::fortify(flows, with_positions = TRUE, cut_off = 0.1)
```

`fortify.sim_list` *Turn a collection of spatial interaction models into a data frame*

Description

This function extracts from a collection of spatial interaction models (represented by a `sim_list`) a data frame in a long format, with one flow per row. This can be seen a collection oriented version of `fortify.sim()`. The resulting data frame is used by `autoplot.sim_list()` to produce summary graphics.

Usage

```
## S3 method for class 'sim_list'
fortify(
  model,
  data,
  flows = c("full", "destination", "attractiveness"),
  with_names = FALSE,
  normalisation = c("none", "origin", "full"),
  ...
)
```

Arguments

`model`	a collection of spatial interaction models, a `sim_list`
`data`	not used
`flows`	`"full"` (default), `"destination"` or `"attractiveness"`, see details.
`with_names`	specifies whether the extracted data frame includes location names (`FALSE` by default), see details.
`normalisation`	when `flows="full"`, the flows can be reported without normalisation (`normalisation="none"`, the default value) or they can be normalised, either to sum to one for each origin location (`normalisation="origin"`) or to sum to one globally (`normalisation="full"`).
`...`	additional parameters, not used currently

Details

The data frame produced by the method depends on the values of `flows` and to a lesser extent on the value of `with_names`. In all cases, the data frame has a `configuration` column that identify from which spatial interaction model the other values have been extracted: this is the index of the model in the original `sim_list`. Depending on `flows` we have the following representations:

- if `flows="full"`: this is the default case for which the full flow matrix of each spatial interaction model is extracted. The data frame contains 4 columns:
  - `origin_idx`: identifies the origin location by its index from 1 to the number of origin locations

  - `destination_idx`: identifies the destination location by its index from 1 to the number of destination locations
  - `flow`: the flow between the corresponding location. By default, flows are normalised by origin location (when `normalisation="origin"`): the total flows originating from each origin location is normalised to 1. If `normalisation="full"`, this normalisation is global: the sum of all flows in each model is normalised to 1. If `normalisation="none"` flows are not normalised.
  - `configuration`: the spatial interaction model index

- if `flows="destination"` or `flows="attractiveness"`, the data frame contains 3 or 4 columns:
  - `destination`: identifies the destination location by its index from 1 to the number of destination locations
  - `flow` or `attractiveness` depending on the value of `"flows"`: this contains either the `destination_flow()` or the `attractiveness()` of the destination location
  - `configuration`: the spatial interaction model index
  - `name`: the destination location names if `with_names=TRUE` (the column is not present if `with_names=FALSE`)

The normalisation operated when `flows="full"` can improve the readability of the graphical representation proposed in `autoplot.sim_list()` when the production constraints differ significantly from one origin location to another.

Value

a data frame, see details

See Also

`autoplot.sim_list()`

Examples

```
positions <- matrix(rnorm(10 * 2), ncol = 2)
distances <- as.matrix(dist(positions))
production <- rep(1, 10)
attractiveness <- c(2, rep(1, 9))
flows_1 <- blvim(distances, production, 1.5, 1, attractiveness)
flows_2 <- blvim(distances, production, 1.25, 2, attractiveness)
all_flows <- sim_list(list(flows_1, flows_2))
ggplot2::fortify(all_flows) ## somewhat similar to a row bind of sim_df results
ggplot2::fortify(all_flows, flows = "destination")
destination_names(all_flows) <- letters[1:10]
ggplot2::fortify(all_flows, flows = "attractiveness", with_names = TRUE)
```

`french_cities` *French cities*

Description

French cities with 50,000 inhabitants or more.

Usage

```
french_cities
```

Format

A `data.frame` with 121 rows and 9 columns

id The INSEE code of the city according to the official geographical code of 2025 (OGC)

name The name of the city

department The code of the department of the city in the OGC (see french_departments)

region The code of the region of the city in the OGC (see french_regions)

population The population of the city in 2022

area Area of the city in squared kilometers

th_latitude The latitude of the of town hall the city (epsg:4326)

th_longitude The longitude of the town hall of the city (epsg:4326)

center_latitude The latitude of the centre of the city (epsg:4326)

center_longitude The longitude of the centre of the city (epsg:4326)

Details

This data set describes all Metropolitan France cities with 50,000 or more inhabitants in 2022, excluding Corsican cities. It contains 121 cities described by 8 variables. The data frame is sorted in decreasing population order, making it straightforward to select the most populated cities. The same order is used for rows and columns in distance matrices french_cities_distances and french_cities_times.

The population and administrative information was collected from the INSEE open data in November 2025. These data are distributed under the French "Open Licence". Geographical coordinates and areas have been obtained from the Geo API in November 2025 and are also available under the French "Open Licence".

Source

INSEE Population census - Main extraction (2022) https://catalogue-donnees.insee.fr/en/catalogue/recherche/DS_RP_POPULATION_PRINC

INSEE Official Geographical Code (2025) https://www.data.gouv.fr/datasets/code-officiel-geographique-cog/ https://www.data.gouv.fr/api/1/datasets/r/91a95bee-c7c8-45f9-a8aa-f14cc4697545

Geo API (2025) https://geo.api.gouv.fr/

See Also

french_departments, french_regions, french_cities_distances and french_cities_times

---

`french_cities_distances`
French cities distances

Description

Distances in meters and in minutes between the French cities with at least 50,000 inhabitants (french_cities).

Usage

```
french_cities_distances

french_cities_times
```

Format

matrices with 121 rows and 121 columns

An object of class `matrix` (inherits from `array`) with 121 rows and 121 columns.

Details

Both data sets are square matrices of dimension (121, 121). If `D` is one of the matrix, `D[i,j]` is the distance from city of id `rownames(D)[i]` to city id `colnames(D)[j]` expressed in meters (`french_cities_distances`) or in minutes (`french_cities_times`). The distance is measured along the fastest path from `i` to `j` on the French road networks as computed using OSRM engine (see details below). Ids in column and row names can be used to get information on the cities in the french_cities data set (column `id`). Rows and columns are sorted in decreasing population order, as in french_cities. Note that the matrices are not symmetric.

Distance calculation

The distances and durations have been computed using the OSRM engine in version 6.0.0.

Calculations are based on the `car` profile included in OSRM sources for the 6.0.0 version.

France roads are provided by OpenStreetMap under the Open Data Commons Open Database License (ODbL) using the GeoFabrik export dated 2025-11-07T21:20:50Z was used.

Source

Geo API (2025) `https://geo.api.gouv.fr/`

OpenStreetMap `https://www.openstreetmap.org`

GeoFabrik `https://download.geofabrik.de/europe/france.html`

OSRM `https://project-osrm.org/`

french_departments *French departments*

Description

This data set contains the list of all French departments. It can be joined with the french_regions or the french_cities data set. The data set was extracted from the INSEE open data in November 2025. These data are distributed under the French "Open Licence".

Usage

```
french_departments
```

Format

A `data.frame` with 101 rows and 3 columns

department The code of the department in the official geographical code of 2025 (OGC)

region The code of the region of the city in the OGC (see french_regions)

name The name of the department

Source

INSEE Official Geographical Code (2025) https://www.data.gouv.fr/datasets/code-officiel-geographique-cog/
https://www.data.gouv.fr/api/1/datasets/r/91a95bee-c7c8-45f9-a8aa-f14cc4697545

See Also

french_cities and french_regions

french_regions *French regions*

Description

This data set contains the list of all French regions It can be joined with the french_departments or the french_cities data set. The data set was extracted from the INSEE open data in November 2025. These data are distributed under the French "Open Licence".

Usage

```
french_regions
```

Format

A `data.frame` with 18 rows and 2 columns

region The code of the region in the official geographical code of 2025 (OGC)

name The name of the region

Source

INSEE Official Geographical Code (2025) https://www.data.gouv.fr/datasets/code-officiel-geographique-cog/
https://www.data.gouv.fr/api/1/datasets/r/91a95bee-c7c8-45f9-a8aa-f14cc4697545

See Also

french_departments and french_cities

`grid_attractiveness` *Extract all the attractivenesses from a collection of spatial interaction models*

Description

The function extract attractivenesses from all the spatial interaction models of the collection and returns them in a matrix in which each row corresponds to a model and each column to a destination location.

Usage

```
grid_attractiveness(sim_list, ...)
```

Arguments

`sim_list`	a collection of spatial interaction models, an object of class `sim_list`
`...`	additional parameters for the `attractiveness()` function

Value

a matrix of attractivenesses at the destination locations

See Also

`attractiveness()` and `grid_blvim()`

Examples

```
distances <- french_cities_distances[1:15, 1:15] / 1000 ## convert to km
production <- log(french_cities$population[1:15])
attractiveness <- log(french_cities$area[1:15])
all_flows <- grid_blvim(
  distances, production, c(1.1, 1.25, 1.5),
  c(1, 2, 3, 4) / 500, attractiveness,
  epsilon = 0.1
)
g_Z <- grid_attractiveness(all_flows)
## should be 12 rows (3 times 4 parameter pairs) and 15 columns (15
## destination locations)
dim(g_Z)
```

`grid_autoplot` *Create a complete ggplot for spatial interaction models in a data frame*

Description

This function combines spatial interaction model representations similar to the ones produced by `autoplot.sim()` into a single ggplot. It provides an alternative graphical representation to the one produced by `autoplot.sim_df()` for collection of spatial interaction models in a `sim_df` object.

Usage

```
grid_autoplot(
  sim_df,
  key,
  flows = c("full", "destination", "attractiveness"),
  with_names = FALSE,
  with_positions = FALSE,
  show_destination = FALSE,
  show_attractiveness = FALSE,
  show_production = FALSE,
  cut_off = 100 * .Machine$double.eps^0.5,
  adjust_limits = FALSE,
  with_labels = FALSE,
  max_sims = 25,
  fw_params = NULL,
  ...
)
```

Arguments

`sim_df`	a data frame of spatial interaction models, an object of class `sim_df`
`key`	the wrapping variable which acts as an identifier for spatial interaction models
`flows`	`"full"` (default), `"destination"` or `"attractiveness"`, see details.

with_names	specifies whether the graphical representation includes location names (FALSE by default)
with_positions	specifies whether the graphical representation is based on location positions (FALSE by default)
show_destination	specifies whether the position based "full" flow figure includes a representation of the destination flows (FALSE by default)
show_attractiveness	specifies whether the position based "full" flow figure includes a representation of the attractivenesses (FALSE by default)
show_production	specifies whether the position based "full" flow figure includes a representation of the productions (FALSE by default)
cut_off	cut off limit for inclusion of a graphical primitive when with_positions = TRUE. In the full flow matrix representation, segments are removed when their flow is smaller than the cut off. In the attractiveness or destination representation, disks are removed when the corresponding value is below the cut off.
adjust_limits	if FALSE (default value), the limits of the position based graph are not adjusted after removing graphical primitives. This eases comparison between graphical representations with different cut off value. If TRUE, limits are adjusted to the data using the standard ggplot2 behaviour.
with_labels	if FALSE (default value) names are displayed using plain texts. If TRUE, names are shown using labels.
max_sims	the maximum number of spatial interaction models allowed in the sim_df data frame
fw_params	parameters for the ggplot2::facet_wrap call (if non NULL)
...	additional (named) parameters passed to autoplot.sim()

Details

The rationale of autoplot.sim_df() is to display a single value for each spatial interaction model (SIM) in the sim_df data frame. On the contrary, this function produces a full graphical representation of each SIM. It is therefore limited to small collection of SIMs (as specified by the max_sims parameter which default to 25).

Under the hood, the function uses fortify.sim() and shares code with autoplot.sim() to have identical representations. It is simply based on facet wrapping facility of ggplot2. In particular the key parameter is used as the wrapping variable in the call to ggplot2::facet_wrap(). If not specified, the function generates an id variable which ranges from 1 to the number of SIMs in the sim_df data frame. If specified, it is evaluated in the context of the data frame and used for wrapping. Notice that if the expression evaluates to identical values for different SIMs, they will be drawn on the same panel of the final figure, which may end up with meaningless representations. Parameters of ggplot2::facet_wrap() can be set using the fw_params parameter (in a list).

Value

a ggplot object

Examples

```
positions <- as.matrix(french_cities[1:10, c("th_longitude", "th_latitude")])
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- log(french_cities$area[1:10])
all_flows <- grid_blvim(distances, production, seq(1.05, 1.35, by = 0.1),
  seq(1, 2.5, by = 0.5) / 400,
  attractiveness,
  bipartite = FALSE,
  epsilon = 0.1, iter_max = 1000,
  destination_data = list(
    names = french_cities$name[1:10],
    positions = positions
  ),
  origin_data = list(
    names = french_cities$name[1:10],
    positions = positions
  )
)
all_flows_df <- sim_df(all_flows)
## default display: flows as matrices
grid_autoplot(all_flows_df)
## custom wrapping variable
grid_autoplot(all_flows_df, paste(alpha, "~", beta))
## bar plots
grid_autoplot(all_flows_df, flows = "attractiveness")
## with positions
grid_autoplot(all_flows_df, with_positions = TRUE, show_destination = TRUE) +
  ggplot2::scale_linewidth_continuous(range = c(0, 1)) +
  ggplot2::scale_size_continuous(range = c(0, 2)) +
  ggplot2::coord_sf(crs = "epsg:4326")
```

`grid_blvim` *Compute a collection of Boltzmann-Lotka-Volterra model solutions*

Description

This function computes a collection of flows between origin locations and destination locations using `blvim()` on a grid of parameters. The flows use the same costs, same production constraints and same attractivenesses. Each flow is computed using one of all the pairwise combinations between the alpha values given by `alphas` and the beta values given by `betas`. The function returns an object of class `sim_list` which contains the resulting flows.

Usage

```
grid_blvim(
  costs,
  X,
```

```
  alphas,
  betas,
  Z,
  bipartite = TRUE,
  origin_data = NULL,
  destination_data = NULL,
  epsilon = 0.01,
  iter_max = 50000,
  conv_check = 100,
  precision = 1e-06,
  quadratic = FALSE,
  progress = FALSE
)
```

Arguments

costs	a cost matrix
X	a vector of production constraints
alphas	a vector of return to scale parameters
betas	a vector of cost inverse scale parameters
Z	a vector of initial destination attractivenesses
bipartite	when TRUE (default value), the origin and destination locations are considered to be distinct. When FALSE, a single set of locations plays the both roles. This has only consequences in functions specific to this latter case such as terminals().
origin_data	NULL or a list of additional data about the origin locations (see details)
destination_data	NULL or a list of additional data about the destination locations (see details)
epsilon	the update intensity
iter_max	the maximal number of steps of the BLV dynamic
conv_check	number of iterations between to convergence test
precision	convergence threshold
quadratic	selects the update rule, see details.
progress	if TRUE, a progress bar is shown during the calculation (defaults to FALSE)

Value

an object of class sim_list

Location data

While models in this package do not use location data beyond X and Z, additional data can be stored and used when analysing spatial interaction models.

Origin and destination location names:

Spatial interaction models can store names for origin and destination locations, using origin_names<-() and destination_names<-(). Names are taken by default from names of the cost matrix costs.

More precisely, `rownames(costs)` is used for origin location names and `colnames(costs)` for destination location names.

Origin and destination location positions:

Spatial interaction models can store the positions of the origin and destination locations, using `origin_positions<-()` and `destination_positions<-()`.

Specifying location data:

In addition to the functions mentioned above, location data can be specified directly using the `origin_data` and `destination_data` parameters. Data are given by a list whose components are not interpreted excepted the following ones:

- `names` is used to specify location names and its content has to follow the restrictions documented in `origin_names<-()` and `destination_names<-()`
- `positions` is used to specify location positions and its content has to follow the restrictions documented in `origin_positions<-()` and `destination_positions<-()`

Examples

```
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- log(french_cities$area[1:10])
all_flows <- grid_blvim(
  distances, production, c(1.25, 1.5),
  c(1, 2, 3, 4) / 500, attractiveness
)
all_flows
length(all_flows)
all_flows[[2]]
```

---

grid_destination_flow *Extract all the destination flows from a collection of spatial interaction models*

Description

The function extract destination flows from all the spatial interaction models of the collection and returns them in a matrix in which each row corresponds to a model and each column to a destination location.

Usage

```
grid_destination_flow(sim_list, ...)
```

Arguments

`sim_list`	a collection of spatial interaction models, an object of class `sim_list`
`...`	additional parameters for the `destination_flow()` function

Value

a matrix of destination flows at the destination locations

See Also

destination_flow() and grid_blvim()

Examples

```
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- log(french_cities$area[1:10])
all_flows <- grid_blvim(
  distances, production, c(1.1, 1.25, 1.5),
  c(1, 2, 3, 4) / 500, attractiveness,
  epsilon = 0.1
)
g_df <- grid_destination_flow(all_flows)
## should be 12 rows (3 times 4 parameter pairs) and 10 columns (10
## destination locations)
dim(g_df)
```

---

grid_diversity *Compute diversities for a collection of spatial interaction models*

Description

The function computes for each spatial interaction model of its sim_list parameter the diversity() of the corresponding destination flows and returns the values as a vector. The type of diversity and the associated parameters are identical for all models.

Usage

```
grid_diversity(
  sim,
  definition = c("shannon", "renyi", "ND", "RW"),
  order = 1L,
  activity = c("destination", "attractiveness"),
  ...
)
```

Arguments

sim	a collection of spatial interaction models, an object of class sim_list
definition	diversity definition "shannon" (default), "renyi" (see details) or a definition supported by terminals()
order	order of the Rényi entropy, used only when definition="renyi"

activity	specifies whether the diversity is computed based on the destination flows (for activity="destination", the default case) or on the attractivenesses (for activity="attractiveness
...	additional parameters

Details

See diversity() for the definition of the diversities. Notice that diversity() is generic and can be applied directly to sim_list objects. The current function is provided to be explicit in R code about what is a unique model and what is a collection of models (using function names that start with "grid_")

Value

a vector of diversities, one per spatial interaction model

See Also

diversity() and grid_blvim()

Examples

```
distances <- french_cities_distances[1:15, 1:15] / 1000 ## convert to km
production <- log(french_cities$population[1:15])
attractiveness <- log(french_cities$area[1:15])
all_flows <- grid_blvim(
  distances, production, c(1.1, 1.25, 1.5),
  c(1, 2, 3, 4) / 500, attractiveness,
  epsilon = 0.1,
  bipartite = FALSE
)
diversities <- grid_diversity(all_flows)
diversities ## should be a length 12 vector
grid_diversity(all_flows, "renyi", 3)
```

---

grid_is_terminal *Extract all terminal status from a collection of spatial interaction models*

Description

The function extract terminal status from all the spatial interaction models of the collection and returns them in a matrix in which each row corresponds to a model and each column to a destination location. The value at row i and column j is TRUE if destination j is a terminal in model i. This function applies only to non bipartite models.

Usage

```
grid_is_terminal(sim_list, definition = c("ND", "RW"), ...)
```

Arguments

sim_list	a collection of non bipartite spatial interaction models, an object of class sim_list
definition	terminal definition, either "ND" (for Nystuen & Dacey, default) or "RW" (for Rihll & Wilson), see details.
...	additional parameters for the is_terminal() function

Details

See terminals() for the definition of terminal locations.

Value

a matrix of terminal status at the destination locations

See Also

is_terminal() and grid_blvim()

Examples

```
distances <- french_cities_distances[1:15, 1:15] / 1000 ## convert to km
production <- log(french_cities$population[1:15])
attractiveness <- log(french_cities$area[1:15])
all_flows <- grid_blvim(
  distances, production, c(1.1, 1.25, 1.5),
  c(1, 2, 3, 4) / 500, attractiveness,
  epsilon = 0.1,
  bipartite = FALSE
)
g_df <- grid_is_terminal(all_flows)
## should be 12 rows (3 times 4 parameter pairs) and 15 columns (15
## destination locations)
dim(g_df)
```

---

grid_sim_converged — *Reports the convergence statuses of a collection of spatial interaction models*

Description

The function reports for each spatial interaction model of its sim_list parameter its convergence status, as defined in sim_converged().

Usage

```
grid_sim_converged(sim, ...)
```

Arguments

sim	a collection of spatial interaction models, an object of class sim_list
...	additional parameters

Details

Notice that sim_converged() is generic and can be applied directly to sim_list objects. The current function is provided to be explicit in R code about what is a unique model and what is a collection of models (using function names that start with "grid_")

Value

a vector of convergence status, one per spatial interaction model

See Also

sim_converged(), grid_sim_iterations() and grid_blvim()

Examples

```
distances <- french_cities_distances[1:15, 1:15] / 1000 ## convert to km
production <- log(french_cities$population[1:15])
attractiveness <- log(french_cities$area[1:15])
all_flows <- grid_blvim(
  distances, production, c(1.1, 1.25, 1.5),
  c(1, 2, 3, 4) / 500, attractiveness,
  epsilon = 0.1,
  bipartite = FALSE,
  iter_max = 750
)
grid_sim_converged(all_flows)
```

---

grid_sim_iterations *Returns the number of iterations used to produce of a collection of spatial interaction models*

Description

The function reports for each spatial interaction model of its sim_list parameter the number of iterations used to produce it (see sim_iterations())

Usage

```
grid_sim_iterations(sim, ...)
```

Arguments

sim	a collection of spatial interaction models, an object of class sim_list
...	additional parameters

Details

Notice that sim_iterations() is generic and can be applied directly to sim_list objects. The current function is provided to be explicit in R code about what is a unique model and what is a collection of models (using function names that start with "grid_")

Value

a vector of numbers of iteration, one per spatial interaction model

See Also

sim_iterations(), grid_sim_converged() and grid_blvim()

Examples

```
distances <- french_cities_distances[1:15, 1:15] / 1000 ## convert to km
production <- log(french_cities$population[1:15])
attractiveness <- log(french_cities$area[1:15])
all_flows <- grid_blvim(
  distances, production, c(1.1, 1.25, 1.5),
  c(1, 2, 3, 4) / 500, attractiveness,
  epsilon = 0.1,
  bipartite = FALSE,
  iter_max = 750
)
grid_sim_iterations(all_flows)
```

---

grid_var_autoplot *Create a complete variability plot for spatial interaction models in a data frame*

Description

This function combines spatial variability interaction model representations similar to the ones produced by autoplot.sim_list() into a single ggplot. It provides an alternative graphical representation to the one produced by autoplot.sim_df() and by grid_autoplot() for collection of spatial interaction models in a sim_df object.

Usage

```
grid_var_autoplot(
  sim_df,
  key,
  flows = c("full", "destination", "attractiveness"),
  with_names = FALSE,
  with_positions = FALSE,
  cut_off = 100 * .Machine$double.eps^0.5,
  adjust_limits = FALSE,
```

```
  with_labels = FALSE,
  qmin = 0.05,
  qmax = 0.95,
  normalisation = c("origin", "full", "none"),
  fw_params = NULL,
  ...
)
```

Arguments

sim_df	a data frame of spatial interaction models, an object of class sim_df
key	the wrapping variable which acts as group identifier for spatial interaction models
flows	"full" (default), "destination" or "attractiveness", see details.
with_names	specifies whether the graphical representation includes location names (FALSE by default)
with_positions	specifies whether the graphical representation is based on location positions (FALSE by default)
cut_off	cut off limit for inclusion of a graphical primitive when with_positions = TRUE. In the attractiveness or destination representation, circles are removed when the corresponding upper quantile value is below the cut off.
adjust_limits	if FALSE (default value), the limits of the position based graph are not adjusted after removing graphical primitives. This eases comparison between graphical representations with different cut off value. If TRUE, limits are adjusted to the data using the standard ggplot2 behaviour.
with_labels	if FALSE (default value) names are displayed using plain texts. If TRUE, names are shown using labels.
qmin	lower quantile, see details (default: 0.05)
qmax	upper quantile, see details (default: 0.95)
normalisation	when flows="full", the flows can be reported without normalisation (normalisation="none", the default value) or they can be normalised, either to sum to one for each origin location (normalisation="origin") or to sum to one globally (normalisation="full").
fw_params	parameters for the ggplot2::facet_wrap call (if non NULL)
...	additional parameters passed to autoplot.sim_list()

Details

The rationale of autoplot.sim_df() is to display a single value for each spatial interaction model (SIM) in the sim_df data frame. On the contrary, this function produces a graphical representation of the variability of a partition of the SIMs in the data frame, using autoplot.sim_list() as the graphical engine.

The key parameter is used to partition the collection of SIMs. It can be any expression which can be evaluated in the context of the sim_df parameter. The function uses this parameter as the wrapping variable in a call to ggplot2::facet_wrap(). It also uses it as a way to specific a partition of the SIMs: each panel of the final figure is essentially the variability graph generated by

autoplot.sim_list() for the subset of the SIMs in sim_df that match the corresponding value of key.

Parameters of ggplot2::facet_wrap() can be set using the fw_params parameter (in a list).

Value

a ggplot object

Examples

```
positions <- as.matrix(french_cities[1:10, c("th_longitude", "th_latitude")])
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- log(french_cities$area[1:10])
all_flows <- grid_blvim(distances, production, seq(1.05, 1.45, by = 0.1),
  seq(1, 3, by = 0.5) / 400,
  attractiveness,
  bipartite = FALSE,
  epsilon = 0.1, iter_max = 1000,
  destination_data = list(
    names = french_cities$name[1:10],
    positions = positions
  ),
  origin_data = list(
    names = french_cities$name[1:10],
    positions = positions
  )
)
all_flows_df <- sim_df(all_flows)
## group models by iteration number
grid_var_autoplot(all_flows_df, iterations)
## or by convergence status (showing destination)
grid_var_autoplot(all_flows_df, converged,
  flow = "destination",
  with_names = TRUE
) + ggplot2::coord_flip()
## using positions
grid_var_autoplot(all_flows_df, iterations,
  flow = "destination",
  with_positions = TRUE
) +
  ggplot2::scale_size_continuous(range = c(0, 3)) +
  ggplot2::coord_sf(crs = "epsg:4326")
```

---

inverse_cost *Extract the inverse cost scale parameter used to compute this model*

Description

Extract the inverse cost scale parameter used to compute this model

Usage

```
inverse_cost(sim, ...)
```

Arguments

sim	a spatial interaction model with a inverse cost scale parameter
...	additional parameters

Value

the inverse cost scale parameter

Examples

```
positions <- matrix(rnorm(10 * 2), ncol = 2)
distances <- as.matrix(dist(positions))
production <- rep(1, 10)
attractiveness <- c(2, rep(1, 9))
model <- static_blvim(distances, production, 1.5, 1, attractiveness)
inverse_cost(model) ## should be 1
```

---

is_terminal *Report whether locations are terminal sites or not*

Description

This function returns a logical vector whose length equals the number of locations. The value in position `i` is `TRUE` if location number `i` is a terminal and `FALSE` if it is not. For the definition of terminals, see `terminals()`.

Usage

```
is_terminal(sim, definition = c("ND", "RW"), ...)
```

Arguments

sim	a spatial interaction model object
definition	terminal definition, either `"ND"` (for Nystuen & Dacey, default) or `"RW"` (for Rihll & Wilson), see details.
...	additional parameters

Value

a logical vector with `TRUE` at the positions of locations that are terminals and `FALSE` for other locations.

See Also

terminals()

Examples

```
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- rep(1, 10)
model <- blvim(distances, production, 1.3, 1 / 500, attractiveness,
  bipartite = FALSE
)
destination_names(model) <- french_cities$name[1:10]
is_terminal(model)
dist_times <- french_cities_times[1:10, 1:10]
tmodel <- blvim(dist_times, production, 1.3, 1 / 10000, attractiveness,
  bipartite = FALSE
)
destination_names(tmodel) <- french_cities$name[1:10]
is_terminal(tmodel)
```

---

location_names *Names of origin and destination locations in a spatial interaction model*

Description

Those functions provide low level access to origin and destination local names. It is recommended to use origin_names() and destination_names() instead of location_names and location_names<-.

Usage

```
location_names(sim)

location_names(sim) <- value
```

Arguments

sim	a spatial interaction model object (an object of class sim) or a collection of spatial interaction models (an object of class sim_list)
value	a list with two components (see the returned value) or NULL

Value

for location_names NULL or a list with two components: origin for the origin location names and destination for the destination location names. For location_names<-() the modified sim object or sim_list object.

See Also

origin_names(), destination_names()

Examples

```
distances <- french_cities_distances[1:10, 1:10]
production <- rep(1, 10)
attractiveness <- rep(1, 10)
## the row/column names of the cost matrix are used for the location
model <- static_blvim(distances, production, 1.5, 1 / 250000, attractiveness)
location_names(model)
location_names(model) <- NULL
location_names(model)
location_names(model) <- list(
  origin = french_cities$name[1:10],
  destination = LETTERS[1:10]
)
destination_names(model)
origin_names(model)
```

---

location_positions *Positions of origin and destination locations in a spatial interaction model*

Description

These functions provide low level access to origin and destination local positions. It is recommended to use origin_positions() and destination_positions() instead of location_positions and location_positions<-.

Usage

```
location_positions(sim)

location_positions(sim) <- value
```

Arguments

sim	a spatial interaction model object
value	a list with two components (see the returned value) or NULL

Value

for location_positions NULL or a list with two components: origin for the origin location positions and destination for the destination location positions. For location_positions<-() the modified sim object.

Positions

Location positions are given by numeric matrices with 2 or 3 columns. The first two columns are assumed to be geographical coordinates while the 3rd column can be used for instance to store altitude. Coordinates are interpreted as is in graphical representations (see `autoplot.sim()`). They are not matched to the costs as those can be derived from complex movement models and other non purely geographic considerations.

See Also

`origin_positions()`, `destination_positions()`

Examples

```
positions <- as.matrix(french_cities[1:10, c("th_longitude", "th_latitude")])
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- rep(1, 10)
model <- static_blvim(distances, production, 1.5, 1 / 250, attractiveness)
## No positions
location_positions(model) <- list(
  origin = positions,
  destination = positions
)
destination_positions(model)
origin_positions(model)
```

---

median.sim_list *Compute the "median" of a collection of spatial interaction models*

Description

This function computes all pairwise distances between spatial interaction models (SIMs) of its `x` parameter, using `sim_distance()` with the specified distance parameters. Then it returns the "median" of the collection defined as the SIM that is in average the closest to all the other SIMs. Tie breaking uses the order of the SIMs in the collection.

Usage

```
## S3 method for class 'sim_list'
median(
  x,
  na.rm = FALSE,
  flows = c("full", "destination", "attractiveness"),
  method = c("euclidean"),
  return_distances = FALSE,
  ...
)
```

Arguments

x	a collection of spatial interaction models, an object of class sim_list
na.rm	not used
flows	"full" (default), "destination" or "attractiveness", see details.
method	the distance measure to be used. Currently only "euclidean" is supported
return_distances	should the distances computed to find the median be returned as a distances attribute of the resulting object? (defaults to FALSE)
...	additional parameters (not used currently)

Details

As distance calculation can be slow in a large collection of SIMs, the distance object can be returned as a distances attribute of the median SIM by setting the return_distances parameter to TRUE. In addition, the returned SIM has always two attributes:

- index gives the index of the mode in the original sim_list
- distortion gives the mean of the distances from the median SIM to all the other SIMs

Value

a spatial interaction model, an object of class sim with additional attributes

Examples

```
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- log(french_cities$area[1:10])
all_flows <- grid_blvim(distances, production, seq(1.05, 1.45, by = 0.1),
  seq(1, 3, by = 0.5) / 400,
  attractiveness,
  bipartite = FALSE,
  epsilon = 0.1, iter_max = 1000,
)
all_flows_median <- median(all_flows)
attr(all_flows_median, "index")
attr(all_flows_median, "distortion")
median(all_flows, flows = "destination")
median(all_flows, flows = "attractiveness")
```

`names<-.sim_df` *Set the column names of a SIM data frame*

Description

Set the column names of a SIM data frame. Renaming the `sim_list` column is supported and tracked, but renaming any core column turns the `sim_df` into a standard `data.frame`.

Usage

```
## S3 replacement method for class 'sim_df'
names(x) <- value
```

Arguments

`x`	data frame of spatial interaction models, an object of class `sim_df`
`value`	unique names for the columns of the data frame

Value

a `sim_df` data frame if possible, a standard `data.frame` when this is not possible.

Examples

```
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- log(french_cities$area[1:10])
all_flows <- grid_blvim(distances, production, seq(1.05, 1.45, by = 0.2),
  seq(1, 3, by = 0.5) / 400,
  attractiveness,
  bipartite = FALSE,
  epsilon = 0.1, iter_max = 1000,
)
all_flows_df <- sim_df(all_flows)
names(all_flows_df)
names(all_flows_df)[6] <- "my_sim"
names(all_flows_df)
## still a sim_df
class(all_flows_df)
names(all_flows_df)[1] <- "return to scale"
names(all_flows_df)
## not a sim_df
class(all_flows_df)
```

nd_graph *Compute the Nystuen and Dacey graph for a spatial interaction model*

Description

This function computes the most important flows in a spatial interaction model according to the approach outlined by J. D. Nystuen and M. F. Dacey (Nystuen & Dacey, 1961. In this work, a *nodal flow* is the largest flow sent from a non terminal location (based on the definition of terminals recalled in `terminals()`). The *nodal structure* is the collection of those flows. They form an oriented graph that has interesting properties. In particular each weakly connected component contains a single terminal location which can be seen as the dominant location of the component. Notice that because nodal flows are based on terminals, this function applies only to the non bipartite setting.

Usage

```
nd_graph(sim, definition = c("ND", "RW"), ...)

## S3 method for class 'sim'
nd_graph(sim, definition = c("ND", "RW"), ...)
```

Arguments

`sim`	a spatial interaction model object
`definition`	terminal definition, either `"ND"` (for Nystuen & Dacey, default) or `"RW"` (for Rihll & Wilson), see details.
`...`	additional parameters

Details

In practice, the function computes first the terminals and non terminals according to either Nystuen & Dacey (1961) or Rihll and Wilson (1991). Then it extracts the nodal flows. The result of the analysis is returned as a data frame with three columns:

- `from`: the index of the non terminal origin location
- `to`: the index of destination location of the nodal flow of `from`
- `flow`: the value of the nodal flow

An important aspect of the node structure is that is does not contain isolated terminals. If a location is a terminal but is never the receiver of a nodal flow it will not appear in the collection of nodal flows. It constitutes a a trivial connected component in itself.

Value

a data frame describing the Nystuen and Dacey graph a.k.a. the nodal structure of a spatial interaction model

References

Nystuen, J.D. and Dacey, M.F. (1961), "A graph theory interpretation of nodal regions", Papers and Proceedings of the Regional Science Association 7: 29-42. doi:10.1007/bf01969070

Rihll, T., and Wilson, A. (1991), "Modelling settlement structures in ancient Greece: new approaches to the polis", In City and Country in the Ancient World, Vol. 3, Edited by J. Rich and A. Wallace-Hadrill, 58-95. London: Routledge.

See Also

`sim_is_bipartite()`, `is_terminal()`, `terminals()`

Examples

```
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- rep(1, 10)
model <- blvim(distances, production, 1.3, 1 / 250, attractiveness,
  bipartite = FALSE
)
destination_names(model) <- french_cities$name[1:10]
nd_graph(model)
dist_times <- french_cities_times[1:15, 1:15]
tmodel <- blvim(dist_times, rep(1, 15), 1.3, 1 / 5000, rep(1, 15),
  bipartite = FALSE
)
destination_names(tmodel) <- french_cities$name[1:15]
terminals(tmodel, definition = "RW")
nd_graph(tmodel, "RW")
```

---

origin_names *Names of origin locations in a spatial interaction model*

Description

Functions to get or set the names of the origin locations in a spatial interaction model (or in a collection of spatial interaction models).

Usage

```
origin_names(sim)

origin_names(sim) <- value
```

Arguments

`sim`	a spatial interaction model object (an object of class `sim`) or a collection of spatial interaction models (an object of class `sim_list`)
`value`	a character vector of length equal to the number of origin locations, or `NULL` (vectors of adapted length are converted to character vectors)

Value

for `origin_names` `NULL` or a character vector with one name per origin locations in the model. for `origin_names<-` the modified `sim` object or `sim_list` object.

See Also

`location_names()`, `destination_names()`

Examples

```
distances <- french_cities_distances[1:10, 1:10]
production <- rep(1, 10)
attractiveness <- rep(1, 10)
## the row/column names of the cost matrix are used for the location
model <- static_blvim(distances, production, 1.5, 1 / 250000, attractiveness)
origin_names(model)
origin_names(model) <- french_cities$name[1:10]
origin_names(model)
```

---

origin_positions *Positions of origin locations in a spatial interaction model*

Description

Functions to get or set the positions of the origin locations in a spatial interaction model.

Usage

```
origin_positions(sim)

origin_positions(sim) <- value
```

Arguments

`sim`	a spatial interaction model object
`value`	a matrix with as many rows as the number of origin locations and 2 or 3 columns, or `NULL`

Value

for `origin_positions` `NULL` or the coordinate matrix for the origin locations. for `origin_positions<-` the modified `sim` object

Positions

Location positions are given by numeric matrices with 2 or 3 columns. The first two columns are assumed to be geographical coordinates while the 3rd column can be used for instance to store altitude. Coordinates are interpreted as is in graphical representations (see `autoplot.sim()`). They are not matched to the costs as those can be derived from complex movement models and other non purely geographic considerations.

See Also

location_positions(), destination_positions()

Examples

```
positions <- as.matrix(french_cities[1:10, c("th_longitude", "th_latitude")])
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- rep(1, 10)
model <- static_blvim(distances, production, 1.5, 1 / 250, attractiveness)
origin_positions(model) <- positions
origin_positions(model)
```

---

production *Extract the production constraints from a spatial interaction model object*

Description

Extract the production constraints from a spatial interaction model object

Usage

```
production(sim, ...)
```

Arguments

sim	a spatial interaction model object
...	additional parameters

Value

a vector of production constraints at the origin locations

See Also

attractiveness(), destination_flow()

Examples

```
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- log(french_cities$population[1:10])
attractiveness <- log(french_cities$area[1:10])
model <- static_blvim(distances, production, 1.5, 1 / 250, attractiveness)
production(model)
## the names of the production vector are set from the distance matrix
## we remove them for testing equality
all.equal(as.numeric(production(model)), production)
```

quantile.sim_list *Compute quantiles of the flows in a collection of spatial interaction models*

Description

The function computes the specified quantiles for individual or aggregated flows in a collection of spatial interaction models.

Usage

```
## S3 method for class 'sim_list'
quantile(
  x,
  flows = c("full", "destination", "attractiveness"),
  probs = seq(0, 1, 0.25),
  normalisation = c("none", "origin", "full"),
  ...
)
```

Arguments

x	a collection of spatial interaction models, a sim_list
flows	"full" (default), "destination" or "attractiveness", see details.
probs	numeric vector of probabilities with values in $[0, 1]$.
normalisation	when flows="full", the flows are used as is, that without normalisation (normalisation="none", default case) or they can be normalised prior the calculation of the quantiles, either to sum to one for each origin location (normalisation="origin") or to sum to one globally (normalisation="full"). This is useful for autoplot.sim_list().
...	additional parameters, not used currently

Details

The exact behaviour depends on the value of flows. In all cases, the function returns a data frame. The data frame has one column per quantile, named after the quantile using a percentage based named, as in the default stats::quantile(). For a graphical representation of those quantiles, see autoplot.sim_list().

- if flows="full": this is the default case in which the function computes for each pair of origin o and destination d locations the quantiles of the distribution of the flow from o to d in the collection of models (the flows maybe normalised before quantile calculations, depending on the value of normalisation). In addition to the quantiles, the data frame has two columns:
  - origin_idx: identifies the origin location by its index from 1 to the number of origin locations;
  - destination_idx: identifies the destination location by its index from 1 to the number of destination locations.

- if `flows="destination"`, the function computes quantiles for each destination d location of the distribution of its incoming flow (`destination_flow()`) in the collection of models. In addition to the quantiles, the data frame has one column `destination` that identifies the destination location by its index from 1 to the number of destination locations.
- if `flows="attractiveness"`, the function computes quantiles for each destination d location of the distribution of its attractiveness (`attractiveness()`) in the collection of models. In addition to the quantiles, the data frame has one column `destination` that identifies the destination location by its index from 1 to the number of destination locations.

Value

a date frame, see details

See Also

`fortify.sim_list()` `autoplot.sim_list()`

Examples

```
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- log(french_cities$area[1:10])
all_flows <- grid_blvim(distances, production, seq(1.05, 1.45, by = 0.1),
  seq(1, 3, by = 0.5) / 400,
  attractiveness,
  bipartite = FALSE,
  epsilon = 0.1, iter_max = 1000
)
head(quantile(all_flows))
head(quantile(all_flows, flows = "destination"))
head(quantile(all_flows,
  flows = "attractiveness",
  probs = c(0.1, 0.3, 0.6, 0.9)
))
```

---

return_to_scale *Extract the return to scale parameter used to compute this model*

Description

Extract the return to scale parameter used to compute this model

Usage

```
return_to_scale(sim, ...)
```

Arguments

`sim`	a spatial interaction model with a return to scale parameter
`...`	additional parameters

Value

the return to scale parameter

Examples

```
positions <- matrix(rnorm(10 * 2), ncol = 2)
distances <- as.matrix(dist(positions))
production <- rep(1, 10)
attractiveness <- c(2, rep(1, 9))
model <- static_blvim(distances, production, 1.5, 1, attractiveness)
return_to_scale(model) ## should be 1.5
```

---

sim_column *Get the collection of spatial interaction models from a SIM data frame*

Description

Get the collection of spatial interaction models from a SIM data frame

Usage

```
sim_column(sim_df)
```

Arguments

sim_df	a data frame of spatial interaction models, an object of class sim_df

Value

the collection of spatial interaction models in the sim_df object, as a sim_list object

Examples

```
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- log(french_cities$area[1:10])
all_flows <- grid_blvim(distances, production, seq(1.05, 1.45, by = 0.2),
  seq(1, 3, by = 0.5) / 400,
  attractiveness,
  bipartite = FALSE,
  epsilon = 0.1, iter_max = 1000,
)
all_flows_df <- sim_df(all_flows, sim_colum = "my_col")
names(all_flows_df)
sim_column(all_flows_df)
```

sim_converged *Reports whether the spatial interaction model construction converged*

Description

Some spatial interaction models are the result of an iterative calculation, see for instance blvim(). This calculation may have been interrupted before convergence. The present function returns TRUE if the calculation converged, FALSE if this was not the case and NA if the spatial interaction model is not the result of an iterative calculation. The function applies also to a collection of spatial interaction models as represented by a sim_list.

Usage

```
sim_converged(sim, ...)
```

Arguments

sim	a spatial interaction model object (an object of class sim) or a collection of spatial interaction models (an object of class sim_list)
...	additional parameters

Value

TRUE, FALSE or NA, as described above. In the case of a sim_list the function returns a logical vector with one value per model.

See Also

sim_iterations(), blvim(), grid_blvim()

Examples

```
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- log(french_cities$population[1:10])
attractiveness <- log(french_cities$area[1:10])
model <- static_blvim(distances, production, 1.5, 1 / 250, attractiveness)
destination_flow(model)
sim_converged(model) ## must be NA
```

sim_df *Create a spatial interaction models data frame from a collection of interaction models*

Description

This function build a data frame from a collection of spatial interaction models. The data frame has a list column `sim` of type `sim_list` which stores the collection of models and classical columns that contain characteristics of the models. The name of the list column can be set to something else than `sim` (but not a name used by other default columns). See details for the default columns.

Usage

```
sim_df(x, sim_column = "sim")
```

Arguments

`x`	a collection of spatial interaction models, an object of class `sim_list`
`sim_column`	the name of the `sim_list` column (default `"sim"`)

Details

The data frame has one row per spatial interaction model and the following columns:

- `sim` (default name): the last column that contains the models
- `alpha`: the return to scale parameter used to build the model
- `beta`: the cost inverse scale parameter used to build the model
- `diversity`: model default `diversity()` (Shannon's diversity)
- `iterations`: the number of iterations used to produce the model (1 for a static model)
- `converged`: `TRUE` is the iterative calculation of the model converged (for models produced by `blvim()` and related approaches), `FALSE` for no convergence and `NA` for static models

The resulting object behaves mostly like a `data.frame` and support standard extraction and replacement operators. The object tries to keep its `sim_df` class during modifications. In particular, `names<-.sim_df()` tracks name change for the `sim_list` column. If a modification or an extraction operation changes the type of the `sim_list` column or drops it, the resulting object is a standard `data.frame`. See `[.sim_df` and `names<-.sim_df()` for details.

Value

a data frame representation of the spatial interaction model collection with classes `sim_df` and `data.frame`

Examples

```
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- log(french_cities$area[1:10])
all_flows <- grid_blvim(distances, production, seq(1.05, 1.45, by = 0.2),
  seq(1, 3, by = 0.5) / 400,
  attractiveness,
  bipartite = FALSE,
  epsilon = 0.1, iter_max = 1000,
)
all_flows_df <- sim_df(all_flows)
all_flows_df$converged
## change the name of the sim column
names(all_flows_df)[6] <- "models"
## still a sim_df
class(all_flows_df)
## get the models
sim_column(all_flows_df)
```

`sim_df_extract` *Extract or replace parts of a SIM data frame*

Description

Extract or replace subsets of SIM data frames. The behaviour of the functions is very close to the one of the corresponding `data.frame` functions, see `[.data.frame`. However, modifications of the SIM columns or suppression of core columns will turn the object into a standard `data.frame` to void issues in e.g. graphical representations, see below for details.

Usage

```
## S3 replacement method for class 'sim_df'
x$name <- value

## S3 replacement method for class 'sim_df'
x[[i, j]] <- value

## S3 method for class 'sim_df'
x[i, j, ..., drop]

## S3 replacement method for class 'sim_df'
x[i, j] <- value
```

Arguments

`x`	data frame of spatial interaction models, an object of class `sim_df`
`name`	a literal character string

value	a suitable replacement value: it will be repeated a whole number of times if necessary and it may be coerced: see the Coercion section in [.data.frame. If NULL, deletes the column if a single column is selected.
i, j, ...	elements to extract or replace. For [and [[, these are numeric or character or, for [only, empty or logical. Numeric values are coerced to integer as if by as.integer. For replacement by [, a logical matrix is allowed.
drop	If TRUE the result is coerced to the lowest possible dimension. The default is to drop if only one column is left, but not to drop if only one row is left.

Details

In a sim_df, the core columns are derived from the sim_list column. Replacement functions maintain this property by updating the columns after any modification of the sim_list column. Modifications of the core columns are rejected (removing a core column is accepted but this turns the sim_df into a standard data.frame).

In addition, the sim_list column obeys to restriction on sim_list modifications (i.e, a sim_list contains a homogeneous collection of spatial interaction models).

Extraction functions keep the sim_df class only if the result is a data frame with a sim_list column, the core columns and potentially additional columns.

Value

For [a sim_df, a data.frame or a single column depending on the values of i and j.

For [[a column of the sim_df (or NULL) or an element of a column when two indices are used.

For $ a column of the sim_df (or NULL).

For [<-, [[<-, and $<- a sim_df or a data frame (see details).

Examples

```
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- log(french_cities$area[1:10])
all_flows <- grid_blvim(distances, production, seq(1.05, 1.45, by = 0.2),
  seq(1, 3, by = 0.5) / 400,
  attractiveness,
  bipartite = FALSE,
  epsilon = 0.1, iter_max = 1000,
)
all_flows_df <- sim_df(all_flows)
## the models as a sim_list
all_flows_df[, "sim"]
## sub data frame, a sim_df
all_flows_df[1:5, ]
## sub data frame, not a sim_df (alpha is missing)
all_flows_df[6:10, 2:6]
all_flows_2 <- grid_blvim(distances, log(french_cities$population[1:10]),
  seq(1.05, 1.45, by = 0.2),
  seq(1, 3, by = 0.5) / 400,
  attractiveness,
```

```
  bipartite = FALSE,
  epsilon = 0.1, iter_max = 1000,
)
## replace the sim_list column by the new models
## before
all_flows_df$diversity
all_flows_df$sim <- all_flows_2
## after (all core columns have been updated)
all_flows_df$diversity
```

`sim_distance` *Compute all pairwise distances between the spatial interaction models in a collection*

Description

This function extracts from each spatial interaction model of the collection a vector representation derived from its flow matrix (see details). This vector is then used to compute distances between the models.

Usage

```
sim_distance(
  sim_list,
  flows = c("full", "destination", "attractiveness"),
  method = c("euclidean"),
  ...
)
```

Arguments

`sim_list`	a collection of spatial interaction models, an object of class `sim_list`
`flows`	`"full"` (default), `"destination"` or `"attractiveness"`, see details.
`method`	the distance measure to be used. Currently only `"euclidean"` is supported
`...`	additional parameters (not used currently)

Details

The vector representation is selected using the `flows` parameters. Possible values are

- `"full"` (default value): the representation is obtained by considering the matrix of `flows()` as a vector (with the standard `as.vector()` function);
- `"destination"`: the representation is the `destination_flow()` vector associated to each spatial interaction model;
- `"attractiveness"`: the representation is the `attractiveness()` vector associated to each spatial interaction model.

Value

an object of class "dist"

See Also

dist()

Examples

```
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- log(french_cities$area[1:10])
all_flows <- grid_blvim(
  distances, production, c(1.25, 1.5),
  c(1, 2, 3, 4) / 500, attractiveness,
  epsilon = 0.1
)
flows_distances <- sim_distance(all_flows)
inflows_distances <- sim_distance(all_flows, "destination")
```

---

sim_is_bipartite *Reports whether the spatial interaction model is bipartite*

Description

The function returns TRUE is the spatial interaction model (SIM) is bipartite, that is if the origin locations are distinct from the destination locations (at least from the analysis point of view). The function return FALSE when the SIM uses the same locations for origin and destination.

Usage

```
sim_is_bipartite(sim)
```

Arguments

sim	a spatial interaction model object

Value

TRUE if the spatial interaction model is bipartite, FALSE if not.

Examples

```
positions <- matrix(rnorm(10 * 2), ncol = 2)
distances <- as.matrix(dist(positions))
production <- rep(1, 10)
attractiveness <- c(2, rep(1, 9))
model <- static_blvim(distances, production, 1.5, 1, attractiveness)
## returns TRUE despite the use of a single set of positions
```

```
sim_is_bipartite(model)
## now we are clear about the non bipartite nature of the model
model <- static_blvim(distances, production, 1.5, 1, attractiveness,
  bipartite = FALSE
)
sim_is_bipartite(model)
```

`sim_iterations` *Returns the number of iterations used to produce this spatial interaction model*

Description

Returns the number of iterations used to produce this spatial interaction model

Usage

```
sim_iterations(sim, ...)
```

Arguments

`sim`	a spatial interaction model object (an object of class `sim`) or a collection of spatial interaction models (an object of class `sim_list`)
`...`	additional parameters

Value

a number of iterations that may be one if the spatial interaction model has been obtained using a static model (see `static_blvim()`). In the case of a `sim_list` the function returns a vector with iteration number per model.

See Also

`sim_converged()`

Examples

```
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- log(french_cities$population[1:10])
attractiveness <- log(french_cities$area[1:10])
model <- static_blvim(distances, production, 1.5, 1 / 250, attractiveness)
destination_flow(model)
sim_iterations(model) ## must be one
```

sim_list *Create a sim_list object from a list of spatial interaction objects*

Description

The collection of sim objects represented by a sim_list object is assumed to be homogeneous, that is to correspond to a fix set of origin and destination locations, associated to a fixed cost matrix.

Usage

```
sim_list(sims, validate = TRUE)
```

Arguments

sims	a list of homogeneous spatial interaction objects
validate	should the function validate the homogeneity of the list of spatial interaction objects (defaults to TRUE)

Value

a sim_list object

Examples

```
positions <- matrix(rnorm(10 * 2), ncol = 2)
distances <- as.matrix(dist(positions))
production <- rep(1, 10)
attractiveness <- c(2, rep(1, 9))
flows_1 <- blvim(distances, production, 1.5, 1, attractiveness)
flows_2 <- blvim(distances, production, 1.25, 2, attractiveness)
all_flows <- sim_list(list(flows_1, flows_2))
```

static_blvim *Compute flows between origin and destination locations*

Description

This function computes flows between origin locations and destination locations according to the production constrained entropy maximising model proposed by A. Wilson.

Usage

```
static_blvim(
  costs,
  X,
  alpha,
  beta,
  Z,
  bipartite = TRUE,
  origin_data = NULL,
  destination_data = NULL
)
```

Arguments

costs	a cost matrix
X	a vector of production constraints
alpha	the return to scale parameter
beta	the inverse cost scale parameter
Z	a vector of destination attractivenesses
bipartite	when TRUE (default value), the origin and destination locations are considered to be distinct. When FALSE, a single set of locations plays the both roles. This has only consequences in functions specific to this latter case such as terminals().
origin_data	NULL or a list of additional data about the origin locations (see details)
destination_data	NULL or a list of additional data about the destination locations (see details)

Details

The model computes flows using the following parameters:

- costs (c) is a $n \times p$ matrix whose (i, j) entry is the cost of having a "unitary" flow from origin location i to destination location j
- X (X) is a vector of size n containing non negative production constraints for the n origin locations
- alpha (α) is a return to scale parameter that enhance (or reduce if smaller that 1) the attractivenesses of destination locations when they are larger than 1
- beta (β) is the inverse of a cost scale parameter, i.e., costs are multiplied by beta in the model
- Z (Z) is a vector of size p containing the positive attractivenesses of the p destination locations

According to Wilson's model, the flow from origin location i to destination location j, Y_{ij}, is given by

$$Y_{ij} = \frac{X_i Z_j^\alpha \exp(-\beta c_{ij})}{\sum_{k=1}^{p} Z_k^\alpha \exp(-\beta c_{ik})}.$$

The model is production constrained because

$$\forall i, \quad X_i = \sum_{j=1}^{p} Y_{ij},$$

that is the origin location i sends a total flow of exactly X_i.

Value

an object of class `sim` (and `sim_wpc`) for spatial interaction model that contains the matrix of flows from the origin locations to the destination locations (see $(Y_{ij})_{1 \leq i \leq n, 1 \leq j \leq p}$ above) and the attractivenesses of the destination locations.

Location data

While models in this package do not use location data beyond `X` and `Z`, additional data can be stored and used when analysing spatial interaction models.

Origin and destination location names:

Spatial interaction models can store names for origin and destination locations, using `origin_names<-()` and `destination_names<-()`. Names are taken by default from names of the cost matrix `costs`. More precisely, `rownames(costs)` is used for origin location names and `colnames(costs)` for destination location names.

Origin and destination location positions:

Spatial interaction models can store the positions of the origin and destination locations, using `origin_positions<-()` and `destination_positions<-()`.

Specifying location data:

In addition to the functions mentioned above, location data can be specified directly using the `origin_data` and `destination_data` parameters. Data are given by a list whose components are not interpreted excepted the following ones:

- `names` is used to specify location names and its content has to follow the restrictions documented in `origin_names<-()` and `destination_names<-()`
- `positions` is used to specify location positions and its content has to follow the restrictions documented in `origin_positions<-()` and `destination_positions<-()`

References

Wilson, A. (1971), "A family of spatial interaction models, and associated developments", Environment and Planning A: Economy and Space, 3(1), 1-32 doi:10.1068/a030001

See Also

`origin_names()`, `destination_names()`, `origin_positions()`, `destination_positions()`

Examples

```
positions <- as.matrix(french_cities[1:10, c("th_longitude", "th_latitude")])
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- log(french_cities$area[1:10])
model <- static_blvim(distances, production, 1.5, 1 / 500, attractiveness,
  origin_data = list(
    names = french_cities$name[1:10],
    positions = positions
  ),
  destination_data = list(
    names = french_cities$name[1:10],
    positions = positions
  )
)
model
location_names(model)
location_positions(model)
```

---

summary.sim_list *Summary of a collection of spatial interaction models*

Description

This function computes summary statistics on a collection of spatial interaction models (in a `sim_list`).

Usage

```
## S3 method for class 'sim_list'
summary(object, ...)

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

Arguments

`object`	a collection of spatial interaction models, an object of class `sim_list` `summary.sim_list()`
`...`	additional parameters (not used currently)
`x`	an object of class `summary_sim_list` produced by

Details

The list returned by the function contains the following elements:

- `median`: the median of the collection, as return by the `median.sim_list()` function
- `distortion`: the average distance of all elements of the collection to the median model
- `withinss`: the sum of all pairwise distances between the elements of the collection

- nb_sims: the size of the collection

In addition, if the collection contains non bipartite models, the result has another element, nb_configurations which gives the number of distinct terminal sets in the collection, where the terminals are computed by terminals(), using the "RW" definition.

Value

an object of class summary_sim_list and list with a set of summary statistics computed on the collection of spatial interaction models

See Also

median.sim_list(), terminals()

Examples

```
positions <- matrix(rnorm(15 * 2), ncol = 2)
distances <- as.matrix(dist(positions))
production <- rep(1, 15)
attractiveness <- rep(1, 15)
all_flows <- grid_blvim(distances,
  production,
  c(1.1, 1.25, 1.5),
  c(1, 2, 3),
  attractiveness,
  epsilon = 0.1,
  bipartite = FALSE,
)
summary(all_flows)
```

---

terminals *Compute terminals for a spatial interaction model*

Description

This function identifies terminals in the locations underlying the given spatial interaction model. Terminals are locally dominating locations that essentially send less to other locations than they receive (see details for formal definitions). As we compare incoming flows to outgoing flows, terminal computation is restricted to interaction models in which origin and destination locations are identical, i.e. models that are not bipartite.

Usage

```
terminals(sim, definition = c("ND", "RW"), ...)
```

Arguments

`sim`	a spatial interaction model object
`definition`	terminal definition, either `"ND"` (for Nystuen & Dacey, default) or `"RW"` (for Rihll & Wilson), see details.
`...`	additional parameters

Details

The notion of terminal used in this function is based on seminal work by J. D. Nystuen and M. F. Dacey (Nystuen & Dacey, 1961), as well as on the follow up variation from Rihll & Wislon (1987 and 1991). We assume given a square flow matrix $(Y_{ij})_{1\leq i\leq n, 1\leq j\leq n}$. The incoming flow at location j is given by

$$D_j = \sum_{j=i}^{p} Y_{ij},$$

and is used as a measure of importance of this location. Then in Nystuen & Dacey (1961), location j is a "terminal point" (or a "central city") if

$$D_j \geq D_{m(j)},$$

where $m(j)$ is such that

$$\forall l, \quad Y_{jl} \leq Y_{jm(j)}.$$

In words, j is a terminal if the location $m(j)$ to which it sends its largest flow is less important than j itself, in terms of incoming flows. This is the definition used by the function when `definition` is `"ND"`.

Rihll & Wilson (1987) use a modified version of this definition described in details in Rihll and Wilson (1991). With this relaxed version, location j is a terminal if

$$\forall i, \quad D_j \geq Y_{ij}.$$

In words, j is a terminal if it receives more flows than it is sending to each other location. It is easy to see that each Nystuen & Dacey terminal is a Rihll & Wilson terminal, but the reverse is false in general. The function use the Rihll & Wilson definition when `definition` is `"RW"`

Value

a vector containing the indexes of the terminals identified from the flow matrix of the interaction model.

References

Nystuen, J.D. and Dacey, M.F. (1961), "A graph theory interpretation of nodal regions", Papers and Proceedings of the Regional Science Association 7: 29-42. doi:10.1007/bf01969070

Rihll, T.E., and Wilson, A.G. (1987). "Spatial interaction and structural models in historical analysis: some possibilities and an example", Histoire & Mesure 2: 5-32. doi:10.3406/hism.1987.1300

Rihll, T., and Wilson, A. (1991), "Modelling settlement structures in ancient Greece: new approaches to the polis", In City and Country in the Ancient World, Vol. 3, Edited by J. Rich and A. Wallace-Hadrill, 58-95. London: Routledge.

See Also

`sim_is_bipartite()`, `is_terminal()`, `grid_is_terminal()`

Examples

```
distances <- french_cities_distances[1:10, 1:10] / 1000 ## convert to km
production <- rep(1, 10)
attractiveness <- rep(1, 10)
model <- blvim(distances, production, 1.3, 1 / 250, attractiveness,
  bipartite = FALSE
)
destination_names(model) <- french_cities$name[1:10]
terminals(model)
dist_times <- french_cities_times[1:10, 1:10]
tmodel <- blvim(dist_times, production, 1.3, 1 / 5000, attractiveness,
  bipartite = FALSE
)
destination_names(tmodel) <- french_cities$name[1:10]
terminals(tmodel)
terminals(tmodel, definition = "RW")
```

Index

∗ **datasets**
    french_cities, 31
    french_cities_distances, 32
    french_departments, 33
    french_regions, 33
[.data.frame, *62, 63*
[.sim_df, *61*
[.sim_df *(*sim_df_extract*)*, 62
[<-.sim_df *(*sim_df_extract*)*, 62
[[<-.sim_df *(*sim_df_extract*)*, 62
$<-.sim_df *(*sim_df_extract*)*, 62

as.data.frame.sim_list, 3
as.vector(), *64*
attractiveness, 4
attractiveness(), *6, 7*, *12*, *19*, *22*, *24*, *27, 28*, *30*, *34*, *56*, *58*, *64*
autoplot.sim, 5
autoplot.sim(), *21*, *26*, *28*, *35, 36*, *50*, *55*
autoplot.sim_df, 9
autoplot.sim_df(), *35, 36*, *44, 45*
autoplot.sim_list, 10
autoplot.sim_list(), *29, 30*, *44–46*, *57, 58*

blvim, 13
blvim(), *6*, *12*, *22*, *37*, *60, 61*

c.sim_list, 16
costs, 17
costs(), *18*
costs.sim_list, 18

destination_flow, 19
destination_flow(), *4*, *6, 7*, *11*, *22*, *24*, *27, 28*, *30*, *39, 40*, *56*, *58*, *64*
destination_names, 19
destination_names(), *48, 49*, *55*, *69*
destination_names<- *(*destination_names*)*, 19
destination_positions, 20
destination_positions(), *49, 50*, *56*, *69*
destination_positions<- *(*destination_positions*)*, 20
dist(), *65*
diversity, 21
diversity(), *9*, *40, 41*, *61*

flows, 24
flows(), *6*, *11*, *26*, *64*
flows_df, 25
flows_df(), *24*, *27, 28*
fortify.sim, 26
fortify.sim(), *6, 7*, *29*, *36*
fortify.sim_list, 29
fortify.sim_list(), *11, 12*, *58*
french_cities, 31, *32–34*
french_cities_distances, *31*, 32, *32*
french_cities_times, *31, 32*
french_cities_times *(*french_cities_distances*)*, 32
french_departments, *31, 32*, 33, *33, 34*
french_regions, *31, 32*, 33, *33*

ggplot2::facet_wrap, *36*, *45*
ggplot2::facet_wrap(), *36*, *45, 46*
ggplot2::geom_crossbar(), *11*
ggplot2::geom_label(), *7*, *12*
ggplot2::geom_segment(), *6*
grid_attractiveness, 34
grid_autoplot, 35
grid_autoplot(), *44*
grid_blvim, 37
grid_blvim(), *16*, *18*, *34*, *40–44*, *60*
grid_destination_flow, 39
grid_diversity, 40
grid_diversity(), *23*
grid_is_terminal, 41
grid_is_terminal(), *73*
grid_sim_converged, 42
grid_sim_converged(), *44*

grid_sim_iterations, 43
grid_sim_iterations(), *43*
grid_var_autoplot, 44

inverse_cost, 46
is_terminal, 47
is_terminal(), *42*, *54*, *73*

location_names, 48
location_names(), *6*, *12*, *20*, *26*, *55*
location_names<- *(*location_names*)*, 48
location_positions, 49
location_positions(), *6*, *12*, *21*, *26*, *27*, *56*
location_positions<-
*(*location_positions*)*, 49

median.sim_list, 50
median.sim_list(), *71*

names<-.sim_df, 52
nd_graph, 53

origin_names, 54
origin_names(), *20*, *48*, *49*, *69*
origin_names<- *(*origin_names*)*, 54
origin_positions, 55
origin_positions(), *21*, *49*, *50*, *69*
origin_positions<- *(*origin_positions*)*,
55

print.summary_sim_list
*(*summary.sim_list*)*, 70
production, 56
production(), *4*, *6*, *19*

quantile.sim_list, 57
quantile.sim_list(), *11*, *12*

return_to_scale, 58

sim_column, 59
sim_converged, 60
sim_converged(), *6*, *7*, *12*, *42*, *43*, *66*
sim_df, 61
sim_df(), *3*, *9*
sim_df_extract, 62
sim_distance, 64
sim_is_bipartite, 65
sim_is_bipartite(), *6*, *7*, *24*, *54*, *73*
sim_iterations, 66
sim_iterations(), *43*, *44*, *60*
sim_list, 67
static_blvim, 67
static_blvim(), *14*, *16*, *66*
stats::quantile(), *57*
summary.sim_list, 70
summary.sim_list(), *70*

terminals, 71
terminals(), *14*, *22–24*, *38*, *40*, *42*, *47*, *48*,
53, *54*, *68*, *71*