Package 'accessibility'

October 30, 2026

Type Package

Title Transport Accessibility Measures

Version 1.6.0

Description A set of fast and convenient functions to help conduct accessibility analyses. Given a pre-computed travel cost matrix and a land use dataset (containing for example the location of jobs, healthcare and population), the package allows one to calculate accessibility levels, and accessibility poverty and inequality. The package covers the majority of the most commonly used accessibility measures (such as cumulative opportunities, gravity-based and floating catchment area methods), some cutting edge measures proposed in the literature (e.g. balancing cost and constrained accessibility) as well as the most frequently used inequality and poverty metrics (such as the Palma ratio, the concentration and Theil indices and the FGT family of measures).

License MIT + file LICENSE

URL https://github.com/ipea/accessibility, https://ipea.github.io/accessibility/, https://github.com/ipea/accessibility

BugReports https://github.com/ipea/accessibility/issues

Depends R (>= 3.5.0)

Imports checkmate, cli (>= 3.0.0), data.table, Hmisc, Rdpack (>= 0.7), rlang (>= 1.0.0), stats, utils

Suggests covr, ggplot2, knitr, rmarkdown, sf, testthat, tibble

VignetteBuilder knitr

RdMacros Rdpack

Encoding UTF-8

NeedsCompilation no

Config/roxygen2/version 8.1.0

Author Rafael H. M. Pereira [aut, cre] (ORCID:
<https://orcid.org/0000-0003-2125-7465>),
Daniel Herszenhut [aut] (ORCID:
<https://orcid.org/0000-0001-8066-1105>),
Anastasia Soukhov [aut],
Christopher Higgins [ctb],
Joey Reid [ctb],
Ipea - Institute for Applied Economic Research [cph, fnd]

Maintainer Rafael H. M. Pereira <rafa.pereira.br@gmail.com>

Repository CRAN

Date/Publication 2026-10-30 09:50:50 UTC

Contents

balancing_cost . . . 2
concentration_index . . . 4
constrained_accessibility . . . 6
cost_to_closest . . . 13
cumulative_cutoff . . . 15
cumulative_interval . . . 17
decay_binary . . . 20
decay_exponential . . . 21
decay_linear . . . 22
decay_logistic . . . 23
decay_power . . . 24
decay_stepped . . . 25
fgt_poverty . . . 26
floating_catchment_area . . . 28
gini_index . . . 31
gravity . . . 32
palma_ratio . . . 34
spatial_availability . . . 36
theil_t . . . 39

Index **42**

`balancing_cost` *Balancing cost accessibility measure*

Description

Calculates the balancing cost measure, which is defined as the travel cost required to reach as many opportunities as the number of people in a given origin. Originally proposed by Barboza et al. (2021), under the name "balancing time".

This function is generic over any kind of numeric travel cost, such as distance, time and money.

Usage

```
balancing_cost(
  travel_matrix,
  land_use_data,
  opportunity,
  travel_cost,
  demand,
  cost_increment = 1,
  group_by = character(0),
  fill_missing_ids = TRUE
)
```

Arguments

travel_matrix	A data frame. The travel matrix describing the costs (i.e. travel time, distance, monetary cost, etc.) between the origins and destinations in the study area. Must contain the columns from_id, to_id and any others specified in travel_cost.
land_use_data	A data frame. The distribution of opportunities within the study area cells. Must contain the columns id and any others specified in opportunity. NA values, and ids in travel_matrix that are missing from this table, are treated as 0.
opportunity	A string. The name of the column in land_use_data with the number of opportunities/resources/services to be considered when calculating accessibility levels.
travel_cost	A string. The name of the column in travel_matrix with the travel cost between origins and destinations. The notion of cost here is generic over any kind of numeric travel cost, such as distance, time and money.
demand	A string. The name of the column in land_use_data with the number of opportunity-demanders at each origin (e.g., people) that will be considered.
cost_increment	A number. The cost increment that should be used when defining the travel cost distribution from which the potential balancing costs will be picked. For example, an increment of 1 tends to be suitable for travel time distributions, meaning that the function will first check if any origins reach their balancing cost with a travel time of 0 minutes, then 1 minute, 2 minutes, 3, 4, ..., etc. An increment of 1 might be too big for a distribution of monetary costs, on the other hand, which could possibly benefit from a smaller increment of 0.05, for example, resulting in the function looking for balancing costs first at a cost of 0, then 0.05, 0.10, ..., etc. Defaults to 1.
group_by	A character vector. When not character(0) (the default), indicates the travel_matrix columns that should be used to group the accessibility estimates by. For example, if travel_matrix includes a departure time column, that specifies the departure time of each entry in the data frame, passing "departure_time" to this parameter results in accessibility estimates grouped by origin and by departure time. Accessibility is calculated separately for each unique combination of grouping variables. For constrained accessibility measures, each group is treated as an independent land-use-transportation system.

`fill_missing_ids`
: A `logical`. Origins that never reach as many opportunities as there are people residing in them, as well as (when `group_by` is not empty) combinations of groups and origins missing from the travel matrix, would not be included in the output. When `TRUE` (the default), the function identifies which combinations would be left out and fills their respective balancing costs with `NA`, which incurs a performance penalty.

Value

A data frame containing the balancing cost of each origin in the travel matrix. Origins with a `NA` balancing cost never reach as many opportunities as there are people residing in them, given the specified travel matrix.

References

Barboza MH, Carneiro MS, Falavigna C, Luz G, Orrico R (2021). "Balancing Time: Using a New Accessibility Measure in Rio de Janeiro." *Journal of Transport Geography*, **90**, 102924. ISSN 09666923. doi:10.1016/j.jtrangeo.2020.102924.

See Also

Other accessibility measures: `cost_to_closest()`, `gravity()`

Examples

```
data_dir <- system.file("extdata", package = "accessibility")
travel_matrix <- readRDS(file.path(data_dir, "travel_matrix.rds"))
land_use_data <- readRDS(file.path(data_dir, "land_use_data.rds"))

bc <- balancing_cost(
  travel_matrix,
  land_use_data,
  opportunity = "jobs",
  travel_cost = "travel_time",
  demand = "population"
)
head(bc)
```

`concentration_index` *Concentration Index*

Description

Calculates the Concentration Index (CI) of a given accessibility distribution. This measure estimates the extent to which accessibility inequalities are systematically associated with individuals' socioeconomic levels. CI values can theoretically vary between -1 and +1 (when all accessibility

is concentrated in the most or in the least disadvantaged person, respectively). Negative values indicate that inequalities favor the poor, while positive values indicate a pro-rich bias. The function supports calculating the standard relative CI and the corrected CI, as proposed by Erreygers (2009).

Usage

```
concentration_index(
  accessibility_data,
  sociodemographic_data,
  opportunity,
  population,
  income,
  type,
  group_by = character(0)
)
```

Arguments

`accessibility_data`	A data frame. The accessibility levels whose inequality should be calculated. Must contain the columns `id` and any others specified in `opportunity`.
`sociodemographic_data`	A data frame. The distribution of sociodemographic characteristics of the population in the study area cells. Must contain the columns `id` and any others specified in `population` and `income`.
`opportunity`	A string. The name of the column in `accessibility_data` with the accessibility levels to be considered when calculating inequality levels.
`population`	A string. The name of the column in `sociodemographic_data` with the number of people in each cell. Used to weigh accessibility levels when calculating inequality.
`income`	A string. The name of the column in `sociodemographic_data` with the income variable that should be used to sort the population from the least to the most privileged. Please note that this variable should describe income per capita (e.g. mean income per capita, household income per capita, etc), instead of the total amount of income in each cell. Also note that, while income is generally used to rank population groups, any variable that can be used to describe one's socioeconomic status, such as education level, can be passed to this argument, as long as it can be numerically ordered (in which higher values denote higher socioeconomic status).
`type`	A string. Which type of Concentration Index to calculate. Current available options are `"standard"` and `"corrected"`.
`group_by`	A `character` vector. When not `character(0)` (the default), indicates the `accessibility_data` columns that should be used to group the inequality estimates by. For example, if `accessibility_data` includes a `scenario` column that identifies distinct scenarios that each accessibility estimates refer to (e.g. before and after a transport policy intervention), passing `"scenario"` to this parameter results in inequality estimates grouped by scenario.

Value

A data frame containing the inequality estimates for the study area.

References

Erreygers G (2009). "Correcting the Concentration Index." *Journal of Health Economics*, **28**(2), 504–515. ISSN 0167-6296. doi:10.1016/j.jhealeco.2008.02.003.

See Also

Other equity: `fgt_poverty()`, `gini_index()`, `palma_ratio()`, `theil_t()`

Examples

```
data_dir <- system.file("extdata", package = "accessibility")
travel_matrix <- readRDS(file.path(data_dir, "travel_matrix.rds"))
land_use_data <- readRDS(file.path(data_dir, "land_use_data.rds"))

access <- cumulative_cutoff(
  travel_matrix,
  land_use_data,
  cutoff = 30,
  opportunity = "jobs",
  travel_cost = "travel_time"
)

ci <- concentration_index(
  access,
  sociodemographic_data = land_use_data,
  opportunity = "jobs",
  population = "population",
  income = "income_per_capita",
  type = "corrected"
)
ci
```

`constrained_accessibility` *Constrained accessibility*

Description

Calculates accessibility using constraints, as proposed in Soukhov et al. (2025). Accessibility is conceptualised as potential spatial interaction. This function covers three constraint cases. Please see the Details section for more information.

Usage

```
constrained_accessibility(
  travel_matrix,
  land_use_data,
  travel_cost,
  demand = NULL,
  supply = NULL,
  constraint,
  decay_function,
  active = NULL,
  error_threshold = 0.001,
  improvement_threshold = 1e-06,
  max_iterations = 1000,
  group_by = character(0),
  fill_missing_ids = TRUE,
  detailed_results = FALSE
)
```

Arguments

travel_matrix	A data frame. The travel matrix describing the costs (i.e. travel time, distance, monetary cost, etc.) between the origins and destinations in the study area. Must contain the columns from_id, to_id and any others specified in travel_cost.
land_use_data	A data frame. The distribution of opportunities within the study area cells. Must contain the columns id and any others specified in opportunity. NA values, and ids in travel_matrix that are missing from this table, are treated as 0.
travel_cost	A string. The name of the column in travel_matrix with the travel cost between origins and destinations. The notion of cost here is generic over any kind of numeric travel cost, such as distance, time and money.
demand	A string. The name of the column in land_use_data with the number of opportunity-demanders at each origin (e.g., people) that will be considered.
supply	A string. The name of the column in land_use_data with the number of opportunity supply at each destination (e.g., jobs, school-seats) that will be considered.
constraint	A string. One of "total", "singly", or "doubly". See Details section for more information.
decay_function	A function that converts travel cost into an impedance factor used to weight opportunities. This function should take a numeric vector and also return a numeric vector as output, with the same length as the input. For convenience, the package currently includes the following functions: decay_binary(), decay_exponential(), decay_linear(), decay_logistic(), decay_power() and decay_stepped(). See the documentation of each decay function for more details.
active	A logical. When TRUE, the function calculates active accessibility (the quantity of opportunities that can be reached from a given origin). When FALSE, it calculates passive accessibility (by how many people each destination can be

reached), which is equivalent to the notion of market potential. This parameter only works for constraint types "total" and "singly". Ignored for constraint = "doubly".

error_threshold
: Numeric. Convergence criterion used only for calibration in the doubly-constrained case (constraint = "doubly").

improvement_threshold
: Numeric. Convergence criterion for improvement used only for calibration in the doubly-constrained case (constraint = "doubly").

max_iterations
: Integer. Maximum iterations used only for calibration in the doubly-constrained case (constraint = "doubly").

group_by
: A character vector. When not character(0) (the default), indicates the travel_matrix columns that should be used to group the accessibility estimates by. For example, if travel_matrix includes a departure time column, that specifies the departure time of each entry in the data frame, passing "departure_time" to this parameter results in accessibility estimates grouped by origin and by departure time.

  Accessibility is calculated separately for each unique combination of grouping variables. For constrained accessibility measures, each group is treated as an independent land-use-transportation system.

fill_missing_ids
: A logical. When calculating grouped accessibility estimates (i.e. when group_by is not empty), some combinations of groups and origins may be missing. For example, if a single trip can depart from origin A at 7:15am and reach destination B within 55 minutes, but no trips departing from A at 7:30am can be completed at all, this second combination will not be included in the output. When TRUE (the default), the function identifies which combinations would be left out and fills their respective accessibility values with 0, which incurs a performance penalty.

detailed_results
: Logical. Whether to return detailed OD-level results for constraint = "total"- and constraint = "singly" accessibility. Currently ignored for constraint = "doubly" accessibility, which always returns OD-level flows.

Value

A data frame of the same class as travel_matrix (data.table, data.frame or tibble). Its columns depend on constraint, active and detailed_results:

- constraint = "total" or "singly", detailed_results = FALSE: one row per origin (from_id, when active = TRUE) or destination (to_id, when active = FALSE), plus the group_by columns. Accessibility is in a column named supply (active) or demand (passive).
- constraint = "total", detailed_results = TRUE: one row per origin-destination pair, with the columns from_id, to_id, the weighted opportunities (weighted_supply or weighted_demand), the share of the regional total allocated to the pair (kappa_total or hatkappa_total), the allocated flow (supply or demand) and the scaling constant (K_total or hatK_total).
- constraint = "singly", detailed_results = TRUE: one row per origin-destination pair, with the columns from_id, to_id, the allocation share (kappa_singly or hatkappa_singly), the allocated flow (supply or demand) and the balancing factor (B_j or A_i).

- `constraint = "doubly"`: one row per origin-destination pair, with the columns `from_id`, `to_id`, `flow`, the balancing factors `A_i` and `B_j`, `kappa_doubly` and the final balancing `error`. `detailed_results` and `active` are not read.

Detailed results include the `group_by` columns and, when the decay function has several values, `decay_function_arg`.

Details

This function covers the family of constrained accessibility measures proposed in Soukhov et al. (2025).

For these measures, each group or decay value is treated as an independent land-use-transportation system. This is not a multimodal model that splits demand or supply across groups. Instead, each group is calculated as if the function were run separately on that group's rows of `travel_matrix`. The total allocated within a group is the sum of the opportunities (or population) associated with the ids that appear in that group's rows.

As well, when `fill_missing_ids = TRUE`, ids that are absent from a particular group's rows are still returned with accessibility equal to 0. This doesn't apply to `constraint = "doubly"`, which ignores `fill_missing_ids` and returns every origin-destination pair of the ids in each group.

Total Constrained Accessibility:

Allocates the system-wide total proportionally based on travel impedance between origins and destinations. This measure uses the logic of a total (or 'unconstrained' by Wilson's terms) constraint.

Use this measure when the total quantity of **supply** OR **demand** in the system is known and representing accessibility as a proportion of this total is meaningful.

Requirement::

- `supply` must be provided when `active = TRUE`, and `demand` when `active = FALSE`. The other argument is ignored, so the same call can be used with `constraint = "singly"`, which requires both.

Interpretation::

- `active = TRUE` (*active accessibility*): Results represent the total number of **opportunities** (supply) accessible from each origin based on region-relative travel impedance. The units are in 'supply' (e.g., jobs, school seats).
  - If `detailed_results = FALSE`, outputs are aggregated and returned by origin.
  - If `detailed_results = TRUE`, OD-level flows are returned. Summing flows by origin equals the aggregated result.
- `active = FALSE` (*passive accessibility*, the notion of market potential): Results represent the total number of **population** (demand) that can reach each destination based on region-relative travel impedance. The units are in 'demand' (e.g., population).
  - If `detailed_results = FALSE`, outputs are aggregated by destination.
  - If `detailed_results = TRUE`, OD-level flows are returned. Summing flows by destination equals the aggregated result.

Use cases::

- Active accessibility (aggregated): "How many jobs can be reached from origin zone A given its region-relative travel impedance?"

- Active accessibility (flow-level): "How many jobs can be reached by flow A->1 given A->1's region-relative travel impedance?"
- Passive accessibility (aggregated): "How many people can reach destination zone 1 given its region-relative travel impedance?"
- Passive accessibility (flow-level): "How many people are reached by flow 1->A given 1->A's region-relative travel impedance?"

Singly Constrained Accessibility:

Allocates opportunities at each destination (or population at each origin) proportionally based on travel impedance and the opposite marginal. This measure uses the logic of single constraint from Wilson (1971).

Use this measure when modeling **competition**, where both demand and supply are conceptualised to influence accessibility but only one side is fixed. The measure distributes flows so that totals match the constrained side while weighting by travel impedance and the unconstrained side.

Requirements::

- Both `demand` and `supply` must be provided (the logical for `active` determines if either demand or supply is constrained).

Interpretation::

- `active = TRUE` (*active accessibility*): constrains supply. Results represent the total number of **opportunities** (supply) accessible from each origin based on region-relative travel impedance and population at the origin. The units are in 'supply' (e.g., jobs, school seats).
  - If `detailed_results = FALSE`, outputs are aggregated and returned by origin.
  - If `detailed_results = TRUE`, OD-level flows are returned. Summing flows by origin equals the aggregated result.
- `active = FALSE` (*passive accessibility*, the notion of market potential): constrains demand. Results represent the total number of **population** (demand) that can reach each destination based on region-relative travel impedance and opportunities at the destination. The units are in 'demand' (e.g., population).
  - If `detailed_results = FALSE`, outputs are aggregated by destination.
  - If `detailed_results = TRUE`, OD-level flows are returned. Summing flows by destination equals the aggregated result.

Use cases::

- Active accessibility (aggregated): "How many jobs can be reached from origin zone A given its region-relative travel impedance and demand?"
- Active accessibility (flow-level): "How many jobs can be reached by flow A->1 given A->1's region-relative travel impedance and demand?"
- Passive accessibility (aggregated): "How many people can reach destination zone 1 given its region-relative travel impedance and supply?"
- Passive accessibility (flow-level): "How many people are reached by flow 1->A given 1->A's region-relative travel impedance and supply?"

NOTE: the active form of this measure yields equivalent results to the `spatial_availability()` function, through different logic.

Doubly Constrained Accessibility:

Allocates flows so supply at each destination matches demand at each origin. It uses Wilson's *doubly-constrained* gravity model Wilson (1971).

The model uses iterative proportional fitting to update balancing factors in order to calibrate OD flows on both margins (`A_i` for origins and `B_j` for destinations) until convergence (i.e. the sum of `demand` and `supply` match). This guarantees that flows satisfy both marginals while being weighted by travel impedance.

Requirements::

- Both `demand` and `supply` must be provided.
- Unlike `total` and `singly`, `doubly` requires the sum of demand and supply to match; otherwise, an error is raised. When `group_by` or multiple decay values are used, the totals must match within each independently balanced system.
- `active` is ignored. Since supply must match demand, their units are the same and there is no distinction between 'active' and 'passive' notions.
- `detailed_results` is currently ignored for doubly-constrained accessibility. OD-level flows are always returned.

Interpretation::

- Results include OD-level flows (`flow`) along with balancing factors (`A_i`, `B_j`) and travel impedance weights. The resulting flows represent the distribution of demand and supply across all origin-destination pairs. NOTE: OD flows are in flow units (jointly determined by demand and supply).

Use cases::

- flow-level: "What is the count of A->1 flows given A->1's region-relative travel impedance, demand and supply?"

References

Soukhov A, Pereira RHM, Higgins CD, Páez A (2025). "A family of accessibility measures derived from spatial interaction principles." *PLoS One*, **20**(11), e0335951. doi:10.1371/journal.pone.0335951.

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

See Also

Other Constrained accessibility: `spatial_availability()`

Examples

```
# Load demo data shipped with the package
data_dir <- system.file("extdata", package = "accessibility")
travel_matrix <- readRDS(file.path(data_dir, "travel_matrix.rds"))
land_use_data <- readRDS(file.path(data_dir, "land_use_data.rds"))

# Total-constrained (active accessibility, aggregated): returns units of
# accessible supply by origin (requires supply)
constrained_accessibility(
  travel_matrix   = travel_matrix,
  land_use_data   = land_use_data,
  travel_cost     = "travel_time",
  constraint      = "total",
```

```
  decay_function  = decay_exponential(0.1),
  demand          = NULL,
  supply          = "jobs",
  active          = TRUE,
  detailed_results = FALSE
)

# Total-constrained (passive accessibility, aggregated): returns units of
# accessible demand by destination  (requires demand)
constrained_accessibility(
  travel_matrix  = travel_matrix,
  land_use_data  = land_use_data,
  travel_cost    = "travel_time",
  constraint     = "total",
  decay_function = decay_exponential(0.1),
  demand         = "population",
  supply         = NULL,
  active         = FALSE,
  detailed_results = FALSE
)

# Singly-constrained (active accessibility, aggregated): returns units of
# accessible supply by origin (requires supply and demand)
constrained_accessibility(
  travel_matrix   = travel_matrix,
  land_use_data   = land_use_data,
  travel_cost     = "travel_time",
  constraint      = "singly",
  decay_function  = decay_exponential(0.1),
  demand          = "population",
  supply          = "jobs",
  active          = TRUE,
  detailed_results = FALSE
)

# Doubly-constrained: returns units of flow (requires both demand and supply
# with matching totals)

# Using a small toy dataset with matching totals.
tm_small <- data.table::data.table(
  expand.grid(from_id = c("1","2","3"), to_id = c("1","2","3"))
)
tm_small[, travel_time := c(10, 30, 15, 30, 10, 25, 15, 25, 10)]
lu_small <- data.table::data.table(
  id         = c("1","2","3"),
  population = c(4, 10, 6),   # sum = 20
  jobs       = c(7,  5,  8)   # sum = 20
)

constrained_accessibility(
  travel_matrix   = tm_small,
  land_use_data   = lu_small,
  travel_cost     = "travel_time",
```

```
  constraint      = "doubly",
  decay_function  = decay_exponential(0.1),
  demand          = "population",
  supply          = "jobs"
)

# With several decay values, each one is balanced separately
constrained_accessibility(
  travel_matrix   = tm_small,
  land_use_data   = lu_small,
  travel_cost     = "travel_time",
  constraint      = "doubly",
  decay_function  = decay_exponential(c(0.1, 0.5)),
  demand          = "population",
  supply          = "jobs"
)
```

`cost_to_closest` *Minimum travel cost to closest N number of opportunities*

Description

Calculates the minimum travel cost to the closest N number of opportunities.

This function is generic over any kind of numeric travel cost, such as distance, time and money.

Usage

```
cost_to_closest(
  travel_matrix,
  land_use_data,
  opportunity,
  travel_cost,
  n = 1,
  group_by = character(0),
  active = TRUE,
  fill_missing_ids = TRUE
)
```

Arguments

`travel_matrix`	A data frame. The travel matrix describing the costs (i.e. travel time, distance, monetary cost, etc.) between the origins and destinations in the study area. Must contain the columns `from_id`, `to_id` and any others specified in `travel_cost`.
`land_use_data`	A data frame. The distribution of opportunities within the study area cells. Must contain the columns `id` and any others specified in `opportunity`. `NA` values, and ids in `travel_matrix` that are missing from this table, are treated as 0.

`opportunity` A string. The name of the column in `land_use_data` with the number of opportunities/resources/services to be considered when calculating accessibility levels.

`travel_cost` A string. The name of the column in `travel_matrix` with the travel cost between origins and destinations. The notion of cost here is generic over any kind of numeric travel cost, such as distance, time and money.

`n` A `numeric` vector. The minimum number of opportunities that should be considered. Defaults to 1. If more than one value is provided, the output includes an extra column specifying the number of opportunities that the minimum travel cost refers to.

`group_by` A `character` vector. When not `character(0)` (the default), indicates the `travel_matrix` columns that should be used to group the accessibility estimates by. For example, if `travel_matrix` includes a departure time column, that specifies the departure time of each entry in the data frame, passing `"departure_time"` to this parameter results in accessibility estimates grouped by origin and by departure time.

Accessibility is calculated separately for each unique combination of grouping variables. For constrained accessibility measures, each group is treated as an independent land-use-transportation system.

`active` A logical. When `TRUE`, the function calculates active accessibility (the quantity of opportunities that can be reached from a given origin). When `FALSE`, it calculates passive accessibility (by how many people each destination can be reached), which is equivalent to the notion of market potential.

`fill_missing_ids`

A `logical`. Calculating minimum travel cost to closest N number of opportunities may result in missing ids in the output if they cannot reach the specified amount of opportunities across all destinations they can reach. For example, estimating the minimum travel time that an origin that can only reach 4 opportunities takes to reach 5 opportunities results in such origin not being included in the output. When `TRUE` (the default), the function identifies which ids would be left out from the output and fills their respective minimum travel costs with `Inf`, which incurs a performance penalty.

Value

A data frame containing the accessibility estimates for each origin/destination (depending if `active` is `TRUE` or `FALSE`) in the travel matrix.

See Also

Other accessibility measures: `balancing_cost()`, `gravity()`

Examples

```
data_dir <- system.file("extdata", package = "accessibility")
travel_matrix <- readRDS(file.path(data_dir, "travel_matrix.rds"))
land_use_data <- readRDS(file.path(data_dir, "land_use_data.rds"))
```

```
df <- cost_to_closest(
  travel_matrix,
  land_use_data,
  n = 1,
  opportunity = "schools",
  travel_cost = "travel_time"
)
head(df)

df <- cost_to_closest(
  travel_matrix,
  land_use_data,
  n = c(1, 2),
  opportunity = "schools",
  travel_cost = "travel_time"
)
head(df)
```

`cumulative_cutoff` *Cumulative access based on a travel cost cutoff*

Description

Calculates the number of opportunities accessible under a given specified travel cost cutoff.

This function is generic over any kind of numeric travel cost, such as distance, time and money.

Usage

```
cumulative_cutoff(
  travel_matrix,
  land_use_data,
  opportunity,
  travel_cost,
  cutoff,
  group_by = character(0),
  active = TRUE,
  fill_missing_ids = TRUE
)
```

Arguments

`travel_matrix`	A data frame. The travel matrix describing the costs (i.e. travel time, distance, monetary cost, etc.) between the origins and destinations in the study area. Must contain the columns `from_id`, `to_id` and any others specified in `travel_cost`.
`land_use_data`	A data frame. The distribution of opportunities within the study area cells. Must contain the columns `id` and any others specified in `opportunity`. `NA` values, and ids in `travel_matrix` that are missing from this table, are treated as 0.

`opportunity` A string. The name of the column in `land_use_data` with the number of opportunities/resources/services to be considered when calculating accessibility levels.

`travel_cost` A `character` vector. The name of the columns in `travel_matrix` with the travel costs between origins and destinations to be considered in the calculation.

`cutoff` Either a `numeric` vector or a list of `numeric` vectors, one for each cost specified in `travel_cost`. The travel cost cutoffs to consider when calculating accessibility levels. If a list, the function finds every single possible cutoff combination and uses them to calculate accessibility (e.g. if one specifies that travel time cutoffs should be 30 and 60 minutes and that monetary cost cutoffs should be 5 and 10 dollars, the output includes accessibility estimates limited at 30 min & 5 dollars, 30 min & 10 dollars, 60 min & 5 dollars and 60 min & 10 dollars). In these cases, cost constraints are considered simultaneously - i.e. only trips that take 30 minutes or less AND 5 dollars or less to be completed, for example, are included in the accessibility output. The cutoff parameter is not included in the final output if the input includes only a single cutoff for a single travel cost.

`group_by` A `character` vector. When not `character(0)` (the default), indicates the `travel_matrix` columns that should be used to group the accessibility estimates by. For example, if `travel_matrix` includes a departure time column, that specifies the departure time of each entry in the data frame, passing `"departure_time"` to this parameter results in accessibility estimates grouped by origin and by departure time.

Accessibility is calculated separately for each unique combination of grouping variables. For constrained accessibility measures, each group is treated as an independent land-use-transportation system.

`active` A logical. When `TRUE`, the function calculates active accessibility (the quantity of opportunities that can be reached from a given origin). When `FALSE`, it calculates passive accessibility (by how many people each destination can be reached), which is equivalent to the notion of market potential.

`fill_missing_ids` A `logical`. Calculating cumulative accessibility may result in missing ids if they cannot reach any of the destinations within the specified travel cost cutoff. For example, using a travel time cutoff of 20 minutes, when estimating the accessibility of origin `A` that can only reach destinations with more than 40 minutes results in id `A` not being included in the output. When `TRUE` (the default), the function identifies which origins would be left out and fills their respective accessibility values with 0, which incurs a performance penalty.

Value

A data frame containing the accessibility estimates for each origin/destination (depending if `active` is `TRUE` or `FALSE`) in the travel matrix.

See Also

Other cumulative access: `cumulative_interval()`

Examples

```
data_dir <- system.file("extdata", package = "accessibility")
travel_matrix <- readRDS(file.path(data_dir, "travel_matrix.rds"))
land_use_data <- readRDS(file.path(data_dir, "land_use_data.rds"))

# active accessibility: number of schools accessible from each origin
df <- cumulative_cutoff(
  travel_matrix = travel_matrix,
  land_use_data = land_use_data,
  cutoff = 30,
  opportunity = "schools",
  travel_cost = "travel_time"
)
head(df)

df <- cumulative_cutoff(
  travel_matrix = travel_matrix,
  land_use_data = land_use_data,
  cutoff = c(30, 60),
  opportunity = "schools",
  travel_cost = "travel_time"
)
head(df)

# passive accessibility: number of people that can reach each destination
df <- cumulative_cutoff(
  travel_matrix = travel_matrix,
  land_use_data = land_use_data,
  cutoff = 30,
  opportunity = "population",
  travel_cost = "travel_time",
  active = FALSE
)
head(df)

# using multiple travel costs
pareto_frontier <- readRDS(file.path(data_dir, "pareto_frontier.rds"))

df <- cumulative_cutoff(
  pareto_frontier,
  land_use_data = land_use_data,
  opportunity = "jobs",
  travel_cost = c("travel_time", "monetary_cost"),
  cutoff = list(c(20, 30), c(0, 5, 10))
)
head(df)
```

`cumulative_interval` *Cumulative access based on maximum travel time interval*

Description

Calculates the average or median number of opportunities that can be reached considering multiple maximum travel cost thresholds within a given travel cost interval specified by the user. The time interval cumulative accessibility measure was originally proposed by Tomasiello et al. (2023).

This function is generic over any kind of numeric travel cost, such as distance, time and money.

Usage

```
cumulative_interval(
  travel_matrix,
  land_use_data,
  opportunity,
  travel_cost,
  interval,
  interval_increment = 1,
  summary_function = stats::median,
  group_by = character(0),
  active = TRUE,
  fill_missing_ids = TRUE
)
```

Arguments

`travel_matrix`	A data frame. The travel matrix describing the costs (i.e. travel time, distance, monetary cost, etc.) between the origins and destinations in the study area. Must contain the columns `from_id`, `to_id` and any others specified in `travel_cost`.
`land_use_data`	A data frame. The distribution of opportunities within the study area cells. Must contain the columns `id` and any others specified in `opportunity`. `NA` values, and ids in `travel_matrix` that are missing from this table, are treated as 0.
`opportunity`	A string. The name of the column in `land_use_data` with the number of opportunities/resources/services to be considered when calculating accessibility levels.
`travel_cost`	A string. The name of the column in `travel_matrix` with the travel cost between origins and destinations. The notion of cost here is generic over any kind of numeric travel cost, such as distance, time and money.
`interval`	A `numeric` vector of length 2. Indicates the start and end points of the interval of travel cost thresholds to be used. The first entry must be lower than the second.
`interval_increment`	A `numeric`. How many travel cost units separate the cutoffs used to calculate the accessibility estimates which will be used to calculate the summary estimate within the specified interval. Should be thought as the resolution of the distribution of travel costs within the interval. Defaults to 1.
`summary_function`	A function. This function is used to summarize a distribution of accessibility estimates within a travel cost interval as a single value. Can be any function that takes an arbitrary number of numeric values as input and returns a single number as output. Defaults to `stats::median()`.

group_by
: A character vector. When not character(0) (the default), indicates the travel_matrix columns that should be used to group the accessibility estimates by. For example, if travel_matrix includes a departure time column, that specifies the departure time of each entry in the data frame, passing "departure_time" to this parameter results in accessibility estimates grouped by origin and by departure time.

  Accessibility is calculated separately for each unique combination of grouping variables. For constrained accessibility measures, each group is treated as an independent land-use-transportation system.

active
: A logical. When TRUE, the function calculates active accessibility (the quantity of opportunities that can be reached from a given origin). When FALSE, it calculates passive accessibility (by how many people each destination can be reached), which is equivalent to the notion of market potential.

fill_missing_ids
: A logical. Origins (or destinations, when active is FALSE) that cannot reach any opportunity within the interval would not be included in the output. When TRUE (the default), the function includes them with an accessibility of 0.

Value

A data frame containing the accessibility estimates for each origin/destination (depending if active is TRUE or FALSE) in the travel matrix.

References

Tomasiello DB, Herszenhut D, Oliveira JLA, Braga CKV, Pereira RHM (2023). "A Time Interval Metric for Cumulative Opportunity Accessibility." *Applied Geography*, **157**, 103007. ISSN 0143-6228. doi:10.1016/j.apgeog.2023.103007.

See Also

Other cumulative access: cumulative_cutoff()

Examples

```
data_dir <- system.file("extdata", package = "accessibility")
travel_matrix <- readRDS(file.path(data_dir, "travel_matrix.rds"))
land_use_data <- readRDS(file.path(data_dir, "land_use_data.rds"))

df <- cumulative_interval(
  travel_matrix = travel_matrix,
  land_use_data = land_use_data,
  interval = c(20, 30),
  opportunity = "schools",
  travel_cost = "travel_time"
)
head(df)

df <- cumulative_interval(
  travel_matrix = travel_matrix,
```

```
  land_use_data = land_use_data,
  interval = c(40, 80),
  opportunity = "jobs",
  travel_cost = "travel_time"
)
head(df)
```

`decay_binary` *Binary (a.k.a. step) decay function*

Description

Returns a binary weighting function (frequently used to calculate cumulative opportunities measures) to be used inside accessibility calculating functions.

This function is generic over any kind of numeric travel cost, such as distance, time and money.

Usage

```
decay_binary(cutoff)
```

Arguments

`cutoff`	A `numeric` vector. The numbers indicating the travel cost cutoffs.

Value

A `function` that takes a generic travel cost vector (`numeric`) as an input and returns a `list` of weight vectors (a list of `numeric` vectors, named after the arguments passed to the decay function).

See Also

Other decay functions: `decay_exponential()`, `decay_linear()`, `decay_logistic()`, `decay_power()`, `decay_stepped()`

Examples

```
weighting_function <- decay_binary(cutoff = 30)

weighting_function(c(20, 35))

weighting_function <- decay_binary(cutoff = c(30, 45))

weighting_function(c(20, 35))
```

decay_exponential *Negative exponential decay function*

Description

Returns a negative exponential weighting function to be used inside accessibility calculating functions.

This function is generic over any kind of numeric travel cost, such as distance, time and money.

Usage

```
decay_exponential(decay_value)
```

Arguments

decay_value	A numeric vector. The calibration parameters that, when multiplied by the travel cost, are used as the exponent of e in the negative exponential function.

Value

A function that takes a generic travel cost vector (numeric) as an input and returns a list of weight vectors (a list of numeric vectors, named after the arguments passed to the decay function).

See Also

Other decay functions: decay_binary(), decay_linear(), decay_logistic(), decay_power(), decay_stepped()

Examples

```
weighting_function <- decay_exponential(decay_value = 0.1)

weighting_function(c(20, 30))

weighting_function <- decay_exponential(decay_value = c(0.1, 0.2))

weighting_function(c(20, 30))
```

decay_linear *Linear decay function*

Description

Returns a linear weighting function to be used inside accessibility calculating functions.

This function is generic over any kind of numeric travel cost, such as distance, time and money.

Usage

```
decay_linear(cutoff)
```

Arguments

cutoff	A numeric vector. Indicates the travel cost cutoffs until which the weighting factor decays linearly. From this point onward the weight is equal to 0. Must be greater than 0.

Value

A function that takes a generic travel cost vector (numeric) as an input and returns a list of weight vectors (a list of numeric vectors, named after the arguments passed to the decay function).

See Also

Other decay functions: decay_binary(), decay_exponential(), decay_logistic(), decay_power(), decay_stepped()

Examples

```
weighting_function <- decay_linear(cutoff = 30)

weighting_function(c(20, 35))

weighting_function <- decay_linear(cutoff = c(30, 45))

weighting_function(c(20, 35))
```

`decay_logistic` *Logistic decay function*

Description

Returns a logistic weighting function (in which the weights follow the distribution of a reversed cumulative logistic curve) to be used inside accessibility calculating functions. The logistic curve is parameterized with the cutoff that sets its inflection point and the standard deviation that sets its steepness.

This function is generic over any kind of numeric travel cost, such as distance, time and money.

Usage

```
decay_logistic(cutoff, sd)
```

Arguments

`cutoff`	A `numeric` vector. The cost value that serves as the inflection point of the cumulative logistic curve.
`sd`	A `numeric` vector with same length as `cutoff`. The standard deviation of the logistic curve. Values near 0 result in weighting curves that approximate binary decay, while higher values tend to linearize the decay.

Details

When using a function created with `decay_logistic()`, the output is named after the combination of cutoffs (`"c"`) and standard deviations (`"sd"`) - e.g. given the cutoffs `c(30, 40)` and the standard deviations `c(10, 20)`, the first element of the output will be named `"c30;sd10"` and the second will be named `"c40;sd20"`. This function uses the adjusted logistic decay curve proposed by Bauer and Groneberg (2016), in which the condition f(0) = 1 is met (i.e. the weight of an opportunity whose cost to reach is 0 is 1).

Value

A `function` that takes a generic travel cost vector (`numeric`) as an input and returns a `list` of weight vectors (a list of `numeric` vectors, named after the arguments passed to the decay function).

References

Bauer J, Groneberg DA (2016). "Measuring Spatial Accessibility of Health Care Providers – Introduction of a Variable Distance Decay Function within the Floating Catchment Area (FCA) Method." *PLOS ONE*, **11**(7), e0159148. ISSN 1932-6203. doi:10.1371/journal.pone.0159148.

See Also

Other decay functions: `decay_binary()`, `decay_exponential()`, `decay_linear()`, `decay_power()`, `decay_stepped()`

Examples

```
weighting_function <- decay_logistic(cutoff = 30, sd = 5)

weighting_function(c(0, 30, 45, 60))

weighting_function <- decay_logistic(cutoff = c(30, 45), sd = c(5, 10))

weighting_function(c(0, 30, 45, 60))
```

---

decay_power *Inverse power decay function*

Description

Returns an inverse power weighting function to be used inside accessibility calculating functions.

This function is generic over any kind of numeric travel cost, such as distance, time and money.

Usage

```
decay_power(decay_value)
```

Arguments

decay_value	A numeric vector. The calibration parameters to be used as the exponents in the inverse power function.

Value

A function that takes a generic travel cost vector (numeric) as an input and returns a list of weight vectors (a list of numeric vectors, named after the arguments passed to the decay function).

See Also

Other decay functions: decay_binary(), decay_exponential(), decay_linear(), decay_logistic(), decay_stepped()

Examples

```
weighting_function <- decay_power(decay_value = 0.1)

weighting_function(c(20, 35))

weighting_function <- decay_power(decay_value = c(0.1, 0.2))

weighting_function(c(20, 35))
```

decay_stepped *Stepped decay function*

Description

Returns a stepped weighting function to be used inside accessibility calculating functions.

This function is generic over any kind of numeric travel cost, such as distance, time and money.

Usage

```
decay_stepped(steps, weights)
```

Arguments

steps	A numeric vector or a list of numeric vectors. The travel cost steps, in ascending order. Please do not include travel cost 0 as a step: this is already handled by the function.
weights	A numeric vector with same length as steps or a list of numeric vectors whose lengths are equal to the lengths of the elements of same index in steps. The values, between 0 and 1, that the function assumes at each step. Please do not include weight 1 as the first value: this is already handled by the function. The function considers the steps' intervals "open on the right", meaning that the function assumes the step value at the actual step, not right after it. Please see the illustrative examples for effects of this assumption on the results.

Details

When both steps and weights parameters are given lists, their content are matched element-wise to define each stepped weighting function

- i.e. the first element of steps is matched to the first element of weights, the second element of steps is matched to the second of weights, etc. When using a function created with decay_stepped(), the output is named after the combination of steps ("s") and weights ("w")
- e.g. given the steps c(10, 20, 30) and the weights c(0.66, 0.33, 0), the output will be named "s(10,20,30);w(0.66,0.33,0)".

Value

A function that takes a generic travel cost vector (numeric) as an input and returns a list of weight vectors (a list of numeric vectors, named after the arguments passed to the decay function).

See Also

Other decay functions: decay_binary(), decay_exponential(), decay_linear(), decay_logistic(), decay_power()

Examples

```
weighting_function <- decay_stepped(
  c(10, 20, 30, 40),
  weights = c(0.75, 0.5, 0.25, 0)
)

weighting_function(c(5, 25, 35, 45))

weighting_function <- decay_stepped(
  list(c(10, 20, 30, 40), c(10, 20, 30, 40)),
  weights = list(c(0.75, 0.5, 0.25, 0), c(0.8, 0.6, 0.4, 0.2))
)

weighting_function(c(5, 25, 35, 45))

# intervals are open on the right, so the values change exactly at each step
weighting_function(c(0, 10, 20, 30, 40))
```

---

fgt_poverty *Foster-Greer-Thorbecke (FGT) poverty measures*

Description

Calculates the FGT metrics, a family of poverty measures originally proposed by Foster et al. (1984) that capture the extent and severity of poverty within an accessibility distribution. The FGT family is composed of three measures that differ based on the α parameter used to calculate them (either 0, 1 or 2) and which also changes their interpretation. Please see the details section for more information on the interpretation of the measures.

Usage

```
fgt_poverty(
  accessibility_data,
  sociodemographic_data,
  opportunity,
  population,
  poverty_line,
  poor_below_threshold = TRUE,
  group_by = character(0)
)
```

Arguments

`accessibility_data`
: A data frame. The accessibility levels whose poverty levels should be calculated. Must contain the columns `id` and any others specified in `opportunity`.

`sociodemographic_data`
: A data frame. The distribution of sociodemographic characteristics of the population in the study area cells. Must contain the columns `id` and any others specified in `population`.

`opportunity`
: A string. The name of the column in `accessibility_data` with the accessibility levels to be considered when calculating accessibility poverty.

`population`
: A string. The name of the column in `sociodemographic_data` with the number of people in each cell. Used to weigh accessibility levels when calculating poverty.

`poverty_line`
: A `numeric`. The poverty line below which individuals are considered to be in accessibility poverty.

`poor_below_threshold`
: A logical. If `TRUE` (default) the observations below the poverty line are considered to be poor. This is the correct approach for primal accessibility measures (e.g. cumulative accessibility). If `FALSE`, then observations above the poverty line are considered to be poor. This is the correct approach for dual accessibility measures (e.g. travel time to the closest facility). When set to `FALSE`, FGT 1 and 2 do not have an upper bound.

`group_by`
: A `character` vector. When not `character(0)` (the default), indicates the `accessibility_data` columns that should be used to group the poverty estimates by. For example, if `accessibility_data` includes a `race` column that specifies the racial category of the population (e.g. `"black"` and `"white"`) that each entry refers to, passing `"race"` to this parameter results in poverty estimates grouped by race.

Value

A data frame containing the three poverty estimates (FGT0, FGT1 and FGT2) for the study area.

Interpretation of FGT measures

With a poverty line z, the FGT measures are calculated as the population-weighted average of g_i^α, in which g_i is the normalised shortfall of each observation i with accessibility y_i: $g_i = (z - y_i)/z$ if $y_i < z$ and 0 otherwise. When `poor_below_threshold` is `FALSE`, g_i is instead the normalised excess $(y_i - z)/z$ if $y_i > z$, and 0 otherwise. The interpretation of each measure depends on the α parameter:

- with $\alpha = 0$ (FGT0) the measure captures the *incidence* (or extent) of poverty as a simple headcount - i.e. the proportion of people in poverty;
- with $\alpha = 1$ (FGT1) the measure, also known as the "poverty gap index", captures the *depth* (or intensity) of poverty as the average normalised shortfall from the poverty line, calculated over the whole population (people not in poverty have a shortfall of 0);
- with $\alpha = 2$ (FGT2) the measure, also known as the "squared poverty gap index", captures the *severity* of poverty. Squaring the shortfalls gives more weight to the people furthest from the poverty line, which makes the measure sensitive to inequality among the poor.

A value of 0 indicates that no one is in poverty. When `poor_below_threshold` is `TRUE`, FGT values range from 0 to 1: FGT0 is 1 when everyone is below the poverty line, and FGT1 and FGT2 are 1 only when everyone has an accessibility of 0. When `poor_below_threshold` is `FALSE`, FGT0 still

ranges from 0 to 1, but FGT1 and FGT2 have no upper bound: they can be greater than 1 (e.g. when travel costs are more than twice the poverty line) and are `Inf` if any accessibility value is `Inf`. This happens, for example, with the travel costs of origins that can't reach the required opportunities, as returned by `cost_to_closest()`. The function raises a warning in this case, and these values should be replaced by a maximum travel cost (e.g. the highest cost in the travel matrix) before calculating the FGT measures.

References

Foster J, Greer J, Thorbecke E (1984). "A Class of Decomposable Poverty Measures." *Econometrica*, **52**(3), 761–766. ISSN 0012-9682. doi:10.2307/1913475. 1913475.

See Also

Other equity: `concentration_index()`, `gini_index()`, `palma_ratio()`, `theil_t()`

Examples

```
data_dir <- system.file("extdata", package = "accessibility")
travel_matrix <- readRDS(file.path(data_dir, "travel_matrix.rds"))
land_use_data <- readRDS(file.path(data_dir, "land_use_data.rds"))

access <- cumulative_cutoff(
  travel_matrix,
  land_use_data,
  cutoff = 30,
  opportunity = "jobs",
  travel_cost = "travel_time"
)

poverty <- fgt_poverty(
  accessibility_data = access,
  opportunity = "jobs",
  sociodemographic_data = land_use_data,
  population = "population",
  poverty_line = 50000
)
poverty
```

---

floating_catchment_area

Floating catchment area accessibility

Description

Calculates accessibility accounting for the competition of resources using a measure from the floating catchment area (FCA) family. Please see the details for the available FCA measures.

This function is generic over any kind of numeric travel cost, such as distance, time and money.

Usage

```
floating_catchment_area(
  travel_matrix,
  land_use_data,
  opportunity,
  travel_cost,
  demand,
  method,
  decay_function,
  group_by = character(0),
  fill_missing_ids = TRUE
)
```

Arguments

travel_matrix	A data frame. The travel matrix describing the costs (i.e. travel time, distance, monetary cost, etc.) between the origins and destinations in the study area. Must contain the columns from_id, to_id and any others specified in travel_cost.
land_use_data	A data frame. The distribution of opportunities within the study area cells. Must contain the columns id and any others specified in opportunity. NA values, and ids in travel_matrix that are missing from this table, are treated as 0.
opportunity	A string. The name of the column in land_use_data with the number of opportunities/resources/services to be considered when calculating accessibility levels.
travel_cost	A string. The name of the column in travel_matrix with the travel cost between origins and destinations. The notion of cost here is generic over any kind of numeric travel cost, such as distance, time and money.
demand	A string. The name of the column in land_use_data with the number of opportunity-demanders at each origin (e.g., people) that will be considered.
method	A string. Which floating catchment area measure to use. Current available options are "2sfca" and "bfca". More info in the details.
decay_function	A function that converts travel cost into an impedance factor used to weight opportunities. This function should take a numeric vector and also return a numeric vector as output, with the same length as the input. For convenience, the package currently includes the following functions: decay_binary(), decay_exponential(), decay_linear(), decay_logistic(), decay_power() and decay_stepped(). See the documentation of each decay function for more details.
group_by	A character vector. When not character(0) (the default), indicates the travel_matrix columns that should be used to group the accessibility estimates by. For example, if travel_matrix includes a departure time column, that specifies the departure time of each entry in the data frame, passing "departure_time" to this parameter results in accessibility estimates grouped by origin and by departure time. Accessibility is calculated separately for each unique combination of grouping variables. For constrained accessibility measures, each group is treated as an independent land-use-transportation system.

`fill_missing_ids`
: A `logical`. When calculating grouped accessibility estimates (i.e. when `group_by` is not empty), some combinations of groups and origins may be missing. For example, if a single trip can depart from origin `A` at 7:15am and reach destination `B` within 55 minutes, but no trips departing from `A` at 7:30am can be completed at all, this second combination will not be included in the output. When `TRUE` (the default), the function identifies which combinations would be left out and fills their respective accessibility values with 0, which incurs a performance penalty.

Value

A data frame containing the accessibility estimates for each origin in the travel matrix.

Details

The package currently includes two built-in FCA measures:

- 2SFCA - the 2-Step Floating Catchment Area measure was the first accessibility metric in the FCA family. It was originally proposed by Luo and Wang (2003).
- BFCA - the Balanced Floating Catchment Area measure calculates accessibility accounting for competition effects while simultaneously correcting for issues of inflation of demand and service levels that are present in other FCA measures. It was originally proposed by Paez et al. (2019) and named in Pereira et al. (2021).

References

Luo W, Wang F (2003). "Measures of Spatial Accessibility to Health Care in a GIS Environment: Synthesis and a Case Study in the Chicago Region." *Environment and Planning B: Planning and Design*, **30**(6), 865–884. ISSN 0265-8135, 1472-3417. doi:10.1068/b29120.

Paez A, Higgins CD, Vivona SF (2019). "Demand and Level of Service Inflation in Floating Catchment Area (FCA) Methods." *PLOS ONE*, **14**(6), e0218773. ISSN 1932-6203. doi:10.1371/journal.pone.0218773.

Pereira RHM, Braga CKV, Servo LM, Serra B, Amaral P, Gouveia N, Paez A (2021). "Geographic Access to COVID-19 Healthcare in Brazil Using a Balanced Float Catchment Area Approach." *Social Science & Medicine*, **273**, 113773. ISSN 0277-9536. doi:10.1016/j.socscimed.2021.113773.

Examples

```
data_dir <- system.file("extdata", package = "accessibility")
travel_matrix <- readRDS(file.path(data_dir, "travel_matrix.rds"))
land_use_data <- readRDS(file.path(data_dir, "land_use_data.rds"))

# 2SFCA with a step decay function
df <- floating_catchment_area(
  travel_matrix,
  land_use_data,
  method = "2sfca",
  decay_function = decay_binary(cutoff = 50),
```

```
  opportunity = "jobs",
  travel_cost = "travel_time",
  demand = "population"
)
head(df)


# BFCA with an exponential decay function
df <- floating_catchment_area(
  travel_matrix,
  land_use_data,
  method = "bfca",
  decay_function = decay_exponential(decay_value = 0.5),
  opportunity = "jobs",
  travel_cost = "travel_time",
  demand = "population"
)
head(df)
```

`gini_index` *Gini Index*

Description

Calculates the Gini Index of a given accessibility distribution.

Usage

```
gini_index(
  accessibility_data,
  sociodemographic_data,
  opportunity,
  population,
  group_by = character(0)
)
```

Arguments

`accessibility_data`	A data frame. The accessibility levels whose inequality should be calculated. Must contain the columns `id` and any others specified in `opportunity`.
`sociodemographic_data`	A data frame. The distribution of sociodemographic characteristics of the population in the study area cells. Must contain the columns `id` and any others specified in `population`.
`opportunity`	A string. The name of the column in `accessibility_data` with the accessibility levels to be considered when calculating inequality levels.

population	A string. The name of the column in sociodemographic_data with the number of people in each cell. Used to weigh accessibility levels when calculating inequality.
group_by	A character vector. When not character(0) (the default), indicates the accessibility_data columns that should be used to group the inequality estimates by. For example, if accessibility_data includes a scenario column that identifies distinct scenarios that each accessibility estimates refer to (e.g. before and after a transport policy intervention), passing "scenario" to this parameter results in inequality estimates grouped by scenario.

Value

A data frame containing the inequality estimates for the study area.

See Also

Other equity: concentration_index(), fgt_poverty(), palma_ratio(), theil_t()

Examples

```
data_dir <- system.file("extdata", package = "accessibility")
travel_matrix <- readRDS(file.path(data_dir, "travel_matrix.rds"))
land_use_data <- readRDS(file.path(data_dir, "land_use_data.rds"))

access <- cumulative_cutoff(
  travel_matrix,
  land_use_data,
  cutoff = 30,
  opportunity = "jobs",
  travel_cost = "travel_time"
)

gini <- gini_index(
  access,
  sociodemographic_data = land_use_data,
  opportunity = "jobs",
  population = "population"
)
gini
```

---

gravity *Gravity-based accessibility measures*

Description

Calculates gravity-based accessibility using a decay function specified by the user.

This function is generic over any kind of numeric travel cost, such as distance, time and money.

Usage

```
gravity(
  travel_matrix,
  land_use_data,
  opportunity,
  travel_cost,
  decay_function,
  group_by = character(0),
  active = TRUE,
  fill_missing_ids = TRUE
)
```

Arguments

travel_matrix	A data frame. The travel matrix describing the costs (i.e. travel time, distance, monetary cost, etc.) between the origins and destinations in the study area. Must contain the columns from_id, to_id and any others specified in travel_cost.
land_use_data	A data frame. The distribution of opportunities within the study area cells. Must contain the columns id and any others specified in opportunity. NA values, and ids in travel_matrix that are missing from this table, are treated as 0.
opportunity	A string. The name of the column in land_use_data with the number of opportunities/resources/services to be considered when calculating accessibility levels.
travel_cost	A string. The name of the column in travel_matrix with the travel cost between origins and destinations. The notion of cost here is generic over any kind of numeric travel cost, such as distance, time and money.
decay_function	A function that converts travel cost into an impedance factor used to weight opportunities. This function should take a numeric vector and also return a numeric vector as output, with the same length as the input. For convenience, the package currently includes the following functions: decay_binary(), decay_exponential(), decay_linear(), decay_logistic(), decay_power() and decay_stepped(). See the documentation of each decay function for more details.
group_by	A character vector. When not character(0) (the default), indicates the travel_matrix columns that should be used to group the accessibility estimates by. For example, if travel_matrix includes a departure time column, that specifies the departure time of each entry in the data frame, passing "departure_time" to this parameter results in accessibility estimates grouped by origin and by departure time. Accessibility is calculated separately for each unique combination of grouping variables. For constrained accessibility measures, each group is treated as an independent land-use-transportation system.
active	A logical. When TRUE, the function calculates active accessibility (the quantity of opportunities that can be reached from a given origin). When FALSE, it calculates passive accessibility (by how many people each destination can be reached), which is equivalent to the notion of market potential.

fill_missing_ids
: A logical. When calculating grouped accessibility estimates (i.e. when group_by is not empty), some combinations of groups and origins may be missing. For example, if a single trip can depart from origin A at 7:15am and reach destination B within 55 minutes, but no trips departing from A at 7:30am can be completed at all, this second combination will not be included in the output. When TRUE (the default), the function identifies which combinations would be left out and fills their respective accessibility values with 0, which incurs a performance penalty.

Value

A data frame containing the accessibility estimates for each origin/destination (depending if active is TRUE or FALSE) in the travel matrix.

See Also

Other accessibility measures: balancing_cost(), cost_to_closest()

Examples

```
data_dir <- system.file("extdata", package = "accessibility")
travel_matrix <- readRDS(file.path(data_dir, "travel_matrix.rds"))
land_use_data <- readRDS(file.path(data_dir, "land_use_data.rds"))

df_linear <- gravity(
  travel_matrix,
  land_use_data,
  decay_function = decay_linear(cutoff = 50),
  opportunity = "schools",
  travel_cost = "travel_time"
)
head(df_linear)

df_exp <- gravity(
  travel_matrix,
  land_use_data,
  decay_function = decay_exponential(decay_value = 0.5),
  opportunity = "schools",
  travel_cost = "travel_time"
)
head(df_exp)
```

---

palma_ratio *Palma Ratio*

Description

Calculates the Palma Ratio of a given accessibility distribution. Originally defined as the income share of the richest 10% of a population divided by the income share of the poorest 40%, this measure has been adapted in transport planning as the average accessibility of the richest 10% divided by the average accessibility of the poorest 40%.

Usage

```
palma_ratio(
  accessibility_data,
  sociodemographic_data,
  opportunity,
  population,
  income,
  group_by = character(0)
)
```

Arguments

`accessibility_data`	A data frame. The accessibility levels whose inequality should be calculated. Must contain the columns `id` and any others specified in `opportunity`.
`sociodemographic_data`	A data frame. The distribution of sociodemographic characteristics of the population in the study area cells. Must contain the columns `id` and any others specified in `population` and `income`.
`opportunity`	A string. The name of the column in `accessibility_data` with the accessibility levels to be considered when calculating inequality levels.
`population`	A string. The name of the column in `sociodemographic_data` with the number of people in each cell. Used to weigh accessibility levels when calculating inequality.
`income`	A string. The name of the column in `sociodemographic_data` with the income variable that should be used to classify the population in socioeconomic groups. Please note that this variable should describe income per capita (e.g. mean income per capita, household income per capita, etc), instead of the total amount of income in each cell.
`group_by`	A `character` vector. When not `character(0)` (the default), indicates the `accessibility_data` columns that should be used to group the inequality estimates by. For example, if `accessibility_data` includes a `scenario` column that identifies distinct scenarios that each accessibility estimates refer to (e.g. before and after a transport policy intervention), passing `"scenario"` to this parameter results in inequality estimates grouped by scenario.

Value

A data frame containing the inequality estimates for the study area.

See Also

Other equity: concentration_index(), fgt_poverty(), gini_index(), theil_t()

Examples

```
data_dir <- system.file("extdata", package = "accessibility")
travel_matrix <- readRDS(file.path(data_dir, "travel_matrix.rds"))
land_use_data <- readRDS(file.path(data_dir, "land_use_data.rds"))

access <- cumulative_cutoff(
  travel_matrix,
  land_use_data,
  cutoff = 30,
  opportunity = "jobs",
  travel_cost = "travel_time"
)

palma <- palma_ratio(
  access,
  sociodemographic_data = land_use_data,
  opportunity = "jobs",
  population = "population",
  income = "income_per_capita"
)
palma
```

---

spatial_availability *Spatial availability*

Description

Calculates spatial availability, an accessibility measure proposed by Soukhov et al. (2023) that takes into account competition effects. The accessibility levels that result from using this measure are proportional both to the demand in each origin and to the travel cost it takes to reach the destinations. The spatial availability is a particular case of `constrained_accessibility(constraint = "singly")`.

This function is generic over any kind of numeric travel cost, such as distance, time and money.

Usage

```
spatial_availability(
  travel_matrix,
  land_use_data,
  opportunity,
  travel_cost,
  demand,
  decay_function,
```

```
  alpha = 1,
  group_by = character(0),
  fill_missing_ids = TRUE,
  detailed_results = FALSE
)
```

Arguments

travel_matrix	A data frame. The travel matrix describing the costs (i.e. travel time, distance, monetary cost, etc.) between the origins and destinations in the study area. Must contain the columns from_id, to_id and any others specified in travel_cost.
land_use_data	A data frame. The distribution of opportunities within the study area cells. Must contain the columns id and any others specified in opportunity. NA values, and ids in travel_matrix that are missing from this table, are treated as 0.
opportunity	A string. The name of the column in land_use_data with the number of opportunities/resources/services to be considered when calculating accessibility levels.
travel_cost	A string. The name of the column in travel_matrix with the travel cost between origins and destinations. The notion of cost here is generic over any kind of numeric travel cost, such as distance, time and money.
demand	A string. The name of the column in land_use_data with the number of opportunity-demanders at each origin (e.g., people) that will be considered.
decay_function	A function that converts travel cost into an impedance factor used to weight opportunities. This function should take a numeric vector and also return a numeric vector as output, with the same length as the input. For convenience, the package currently includes the following functions: decay_binary(), decay_exponential(), decay_linear(), decay_logistic(), decay_power() and decay_stepped(). See the documentation of each decay function for more details.
alpha	A numeric. A parameter used to modulate the effect of demand by population. When less than 1, opportunities are allocated more rapidly to smaller centers relative to larger ones; values higher than 1 achieve the opposite effect.
group_by	A character vector. When not character(0) (the default), indicates the travel_matrix columns that should be used to group the accessibility estimates by. For example, if travel_matrix includes a departure time column, that specifies the departure time of each entry in the data frame, passing "departure_time" to this parameter results in accessibility estimates grouped by origin and by departure time. Accessibility is calculated separately for each unique combination of grouping variables. For constrained accessibility measures, each group is treated as an independent land-use-transportation system.
fill_missing_ids	A logical. When calculating grouped accessibility estimates (i.e. when group_by is not empty), some combinations of groups and origins may be missing. For example, if a single trip can depart from origin A at 7:15am and reach destination B within 55 minutes, but no trips departing from A at 7:30am can be completed at all, this second combination will not be included in the output. When TRUE (the

default), the function identifies which combinations would be left out and fills their respective accessibility values with 0, which incurs a performance penalty.

`detailed_results`
: A `logical`. Whether to return spatial availability results aggregated by origin-destination pair (`TRUE`) or by origin (`FALSE`, the default). When `TRUE`, the output also includes the demand, impedance and combined balancing factors used to calculate spatial availability. Please note that the argument `fill_missing_ids` does not affect the output when `detailed_results` is `TRUE`.

Value

A data frame containing the spatial availability estimates for each origin in the travel matrix. When `detailed_results` is `TRUE`, the estimates are given for each origin-destination pair instead, alongside the balancing factors used to calculate them.

References

Soukhov A, Páez A, Higgins CD, Mohamed M (2023). "Introducing Spatial Availability, a Singly-Constrained Measure of Competitive Accessibility." *PLOS ONE*, **18**(1), e0278468. ISSN 1932-6203. doi:10.1371/journal.pone.0278468.

See Also

Other Constrained accessibility: `constrained_accessibility()`

Examples

```
# the example below is based on Soukhov et al. (2023) paper

travel_matrix <- data.table::data.table(
  from_id = rep(c("A", "B", "C"), each = 3),
  to_id = as.character(rep(1:3, 3)),
  travel_time = c(15, 30, 100, 30, 15, 100, 100, 100, 15)
)
land_use_data <- data.table::data.table(
  id = c("A", "B", "C", "1", "2", "3"),
  population = c(50000, 150000, 10000, 0, 0, 0),
  jobs = c(0, 0, 0, 100000, 100000, 10000)
)

df <- spatial_availability(
  travel_matrix,
  land_use_data,
  opportunity = "jobs",
  travel_cost = "travel_time",
  demand = "population",
  decay_function = decay_exponential(decay_value = 0.1)
)
df

detailed_df <- spatial_availability(
```

```
  travel_matrix,
  land_use_data,
  opportunity = "jobs",
  travel_cost = "travel_time",
  demand = "population",
  decay_function = decay_exponential(decay_value = 0.1),
  detailed_results = TRUE
)
detailed_df
```

`theil_t` *Theil T Index*

Description

Calculates the Theil T Index of a given accessibility distribution. Values range from 0 (when all individuals have exactly the same accessibility levels) to the natural log of n, in which n is the number of individuals in the accessibility dataset. If the individuals can be classified into mutually exclusive and completely exhaustive groups, the index can be decomposed into a between-groups inequality component and a within-groups component.

Usage

```
theil_t(
  accessibility_data,
  sociodemographic_data,
  opportunity,
  population,
  socioeconomic_groups = NULL,
  group_by = character(0)
)
```

Arguments

`accessibility_data`	A data frame. The accessibility levels whose inequality should be calculated. Must contain the columns `id` and any others specified in `opportunity`.
`sociodemographic_data`	A data frame. The distribution of sociodemographic characteristics of the population in the study area cells. Must contain the columns `id` and any others specified in `population` and `socioeconomic_groups`.
`opportunity`	A string. The name of the column in `accessibility_data` with the accessibility levels to be considered when calculating inequality levels.
`population`	A string. The name of the column in `sociodemographic_data` with the number of people in each cell. Used to weigh accessibility levels when calculating inequality.

socioeconomic_groups
: A string. The name of the column in sociodemographic_data whose values identify the socioeconomic groups that should be used to calculate the between- and within-groups inequality levels. If NULL (the default), between- and within-groups components are not calculated and only the total aggregate inequality is returned.

group_by
: A character vector. When not character(0) (the default), indicates the accessibility_data columns that should be used to group the inequality estimates by. For example, if accessibility_data includes a scenario column that identifies distinct scenarios that each accessibility estimates refer to (e.g. before and after a transport policy intervention), passing "scenario" to this parameter results in inequality estimates grouped by scenario.

Details

Observations with no access contribute 0 to the index, following the usual convention that `0 * log(0) = 0`. If no one has access to any opportunity (i.e. the average accessibility is 0), the index is undefined and NA is returned.

Value

If socioeconomic_groups is NULL, a data frame containing the total Theil T estimates for the study area. If not, a list containing three dataframes: one summarizing the total inequality and the between- and within-groups components, one listing the contribution of each group to the between-groups component and another listing the contribution of each group to the within-groups component.

See Also

Other equity: concentration_index(), fgt_poverty(), gini_index(), palma_ratio()

Examples

```
data_dir <- system.file("extdata", package = "accessibility")
travel_matrix <- readRDS(file.path(data_dir, "travel_matrix.rds"))
land_use_data <- readRDS(file.path(data_dir, "land_use_data.rds"))

access <- cumulative_cutoff(
  travel_matrix,
  land_use_data,
  cutoff = 30,
  opportunity = "jobs",
  travel_cost = "travel_time"
)

ti <- theil_t(
  access,
  sociodemographic_data = land_use_data,
  opportunity = "jobs",
  population = "population"
)
```

```
ti

# to calculate inequality between and within income deciles, we pass
# "income_decile" to socioeconomic_groups.
# some cells, however, are classified as in the decile NA because their
# income per capita is NaN, as they don't have any population. we filter
# these cells from our accessibility data, otherwise the output would include
# NA values (note that subsetting the data like this doesn't affect the
# assumption that groups are completely exhaustive, because cells with NA
# income decile don't have any population)

na_decile_ids <- land_use_data[is.na(land_use_data$income_decile), ]$id
access <- access[! access$id %in% na_decile_ids, ]
sociodem_data <- land_use_data[! land_use_data$id %in% na_decile_ids, ]

ti <- theil_t(
  access,
  sociodemographic_data = sociodem_data,
  opportunity = "jobs",
  population = "population",
  socioeconomic_groups = "income_decile"
)
ti
```

Index

∗ **Constrained accessibility**
  constrained_accessibility, 6
  spatial_availability, 36
∗ **Floating catchment area**
  floating_catchment_area, 28
∗ **accessibility measures**
  balancing_cost, 2
  cost_to_closest, 13
  gravity, 32
∗ **cumulative access**
  cumulative_cutoff, 15
  cumulative_interval, 17
∗ **decay functions**
  decay_binary, 20
  decay_exponential, 21
  decay_linear, 22
  decay_logistic, 23
  decay_power, 24
  decay_stepped, 25
∗ **equity**
  concentration_index, 4
  fgt_poverty, 26
  gini_index, 31
  palma_ratio, 34
  theil_t, 39

balancing_cost, 2
balancing_cost(), *14*, *34*

concentration_index, 4
concentration_index(), *28*, *32*, *36*, *40*
constrained_accessibility, 6
constrained_accessibility(), *38*
cost_to_closest, 13
cost_to_closest(), *4*, *28*, *34*
cumulative_cutoff, 15
cumulative_cutoff(), *19*
cumulative_interval, 17
cumulative_interval(), *16*

decay_binary, 20
decay_binary(), *7*, *21–25*, *29*, *33*, *37*
decay_exponential, 21
decay_exponential(), *7*, *20*, *22–25*, *29*, *33*, *37*
decay_linear, 22
decay_linear(), *7*, *20, 21*, *23–25*, *29*, *33*, *37*
decay_logistic, 23
decay_logistic(), *7*, *20–22*, *24, 25*, *29*, *33*, *37*
decay_power, 24
decay_power(), *7*, *20–23*, *25*, *29*, *33*, *37*
decay_stepped, 25
decay_stepped(), *7*, *20–24*, *29*, *33*, *37*

fgt_poverty, 26
fgt_poverty(), *6*, *32*, *36*, *40*
floating_catchment_area, 28

gini_index, 31
gini_index(), *6*, *28*, *36*, *40*
gravity, 32
gravity(), *4*, *14*

palma_ratio, 34
palma_ratio(), *6*, *28*, *32*, *40*

spatial_availability, 36
spatial_availability(), *11*
stats::median(), *18*

theil_t, 39
theil_t(), *6*, *28*, *32*, *36*