Package ‘parsec’

August 19, 2023

Version 1.2.7

Date 2023-08-19

Title Partial Orders in Socio-Economics

Author Alberto Arcagni [aut, cre],
Marco Fattore [aut]

Maintainer Alberto Arcagni <`alberto.arcagni@uniroma1.it`>

Description Implements tools for the analysis of partially ordered data, with a particular focus on the evaluation of multidimensional systems of indicators and on the analysis of poverty. References, Fattore M. (2016) <doi:10.1007/s11205-015-1059-6> Fattore M., Arcagni A. (2016) <doi:10.1007/s11205-016-1501-4> Arcagni A. (2017) <doi:10.1007/978-3-319-45421-4_19>.

Depends igraph, netrankr, methods

License GPL (>= 2)

NeedsCompilation yes

Repository CRAN

Date/Publication 2023-08-19 13:40:02 UTC

RoxygenNote 6.1.1

R topics documented:

parsec-package *PARtial orders in Socio-EConomics*

Description

The package implements tools for the analysis of partially ordered data, with a particular focus on the evaluation of multidimensional systems of indicators and on the analysis of poverty.

Its main objective is to provide socio-economic scholars with an integrated set of elementary functions for multidimensional evaluation, based on ordinal information. In particular, it provides functions for data management and basic analysis of partial orders as well as other functions for the evaluation and application of both the poset-based approach and a more classic counting method.

Author(s)

A, Arcagni M, Fattore

Maintainer: A, Arcagni <alberto.arcagni@unimib.it>

Examples

```
############################################
# a simple example of package application #
############################################

# definition of the variables by their number of grades
variables <- c(2, 2, 2)

# definition of the threshold
threshold <- c("112", "211")

# extraction of all of the possible profiles from variables; the
# function returns an object of class "wprof", weighted profiles: by default,
# weigths/frequencies are set equal to 1
profiles <- var2prof(varlen = variables)

# the following function creates matrices describing the poset, and
# provides all the results related to it
eval <- evaluation(profiles, threshold, nit = 10^5, maxint = 10^3)

# The results can then be summarized

summary(summary(eval))

# a method of the plot function returns the Hasse diagram, a frequency
# distribution of the threshold, the identification function, the rank
# distribution of each profile through a barplot, and the relative gap.
plot(eval)

#########################################################
# a second example of new functions recently introduced #
```

```
##########################################################

# definition of the variables and of the corresponding profiles
v1 <- as.ordered(c("a", "b", "c", "d"))
v2 <- 1:3
prof <- var2prof(varmod = list(v1 = as.ordered(c("a", "b", "c", "d")), v2 = 1:3))
np <- nrow(prof$profiles)

# definition of different distributions over the set of profiles
k <- 10 # number of populations
set.seed(0)
populations <- as.data.frame(lapply(1:k, function(x) round(runif(np)*100)))
rownames(populations) <- rownames(prof$profiles)
names(populations) <- paste0("P", 1:k)

prof
populations

# evaluation of the fuzzy first order dominance
res <- FFOD(profiles = prof, distributions = populations)
res

# rank stablity analysis
res <- rank_stability(res)
res

# graphical representation
plot(res)
```

`AF` *OPHI counting approach*

Description

The function implements the OPHI counting approach, in a single call. The implementation is limited to ordinal attributes.

Usage

```
AF(y, ...)
## Default S3 method:
AF(y, z, w=rep(1, ncol(y)), k=sum(w), freq=rep(1, nrow(y)), ...)
## S3 method for class 'wprof'
AF(y, ...)
```

Arguments

`y`	matrix of profiles, possibly substituted by an object of class `wprof`.
`z`	vector of attribute cutoffs.

`w`	variables' weights.
`k`	overall cutoff.
`freq`	profiles' frequencies; the argument can be omitted if `y` is an object of class `wprof.`.
`...`	any of the above.

Value

An object of S3 class `ophi` containing all the outputs related to the OPHI counting approach. The object is a `list` comprising:

`y`	matrix of profiles,
`freq`	profiles' frequencies,
`d`	number of variables
`n`	number of observations (sum of frequencies),
`z`	vector of cutoffs,
`k`	overall cutoff,
`rho`	function comparing profiles to the vector of cutoffs,
`rho_k`	function comparing profiles to the overall cutoff, by weighting variables,
`g0`	profile-variable matrix reporting the output of function `rho`,
`c`	censored vector of deprivation counts,
`Z_k`	boolean vector identifying deprived profiles, according to the specified cutoffs,
`q`	number of poor statstical units in the population,
`H`	headcount ratio, i.e. `q/n`, where `n` is the number of statistical units in the population,
`A`	average deprivation share,
`M0`	adjusted headcount ratio.

References

Alkire S., Foster J. (2011), Counting and multidimensional poverty measurement, Journal of Public Economics, 96(7-8), 476-487.

Examples

```
vl <- c(2, 3, 3, 2)
prof <- var2prof(varlen = vl)

res <- AF(prof, z = c(1, 2, 1, 1), k = 1)

res
```

`AF2threshold` *Poset threshold making the poset approach equivalent to the AF counting approach*

Description

The function computes the threshold in the profile poset, which makes the poset approach equivalent to the AF counting approach, described in argument `mpi`.

Usage

```
AF2threshold(mpi, prof, zeta = NULL)
```

Arguments

`mpi`	an object of class `ophi`, see `AF` for details.
`prof`	an object of class `wprof`.
`zeta`	an object of class `incidence`.

See Also

`AF`

Examples

```
vl <- c(2, 3, 2)
prof <- var2prof(varlen = vl, labtype = "progressive")

res <- AF(prof, z = c(1, 2, 1), k = 1)

thr <- AF2threshold(res, prof)

plot(prof, col = 1 + thr, lwd = 1 + res$c,
     main = "Comparison between OPHI and parsec",
     sub = "bold: deprived profiles identified by OPHI, red: parsec threshold")

eval <- evaluation(prof, thr, maxint = 10^4, nit = 10^7)

ord <- order(eval$idn_f, res$c)
plot(eval$idn_f[ord], col = "red", lwd=2, type = "l", xlab="",
     ylab = "", axes = FALSE, frame.plot = TRUE,
     main = "Comparison between OPHI and parsec",
     sub = "red: identification function, black: OPHI deprived profiles")
points(res$c[ord], type="l", lwd=2)
axis(2)
```

`antisymmetry` *antisymmetry*

Description

The function checks whether boolean square matrix `m` represents an antisymmetric binary relation.

Usage

```
antisymmetry(m)
```

Arguments

`m`	a square matrix.

See Also

`transitivity`, `binary`, `reflexivity`,

`is.preorder`, `is.partialorder`,

`validate.partialorder.incidence`

Examples

```
M <- c(TRUE, FALSE, FALSE, FALSE, TRUE, TRUE, FALSE, FALSE, TRUE, FALSE, TRUE,
FALSE, TRUE, TRUE, TRUE, TRUE)
M <- matrix(M, 4, 4)
rownames(M) <- colnames(M) <- LETTERS[1:4]

antisymmetry(M)
```

`average_ranks` *Average Ranks*

Description

The function evaluates the average rank, and other distribution details, for each element of the poset.

Usage

```
average_ranks(x, ...)
## S3 method for class 'cover'
average_ranks(x, level = 0.9, error = 10^(-5), ...)
## S3 method for class 'incidence'
average_ranks(x, level = 0.9, error = 10^(-5), ...)
```

Arguments

`x`	an incidence or cover matrix representing a partial order.
`level`	coverage probability of the rank intervals.
`error`	the "distance" from uniformity in the sampling distribution of linear extensions used to evaluate the average ranks. See `idn` for details.
`...`	any of above.

Details

The function computes the rank distribution for each element of the poset, through function `idn`. Next, it checks whether there are any equivalent profiles, using function `equivalences`, and makes their rank distribution equal. Finally it provides a dataframe comprising, for each element of the poset: the average rank `avrg`, the extremes `inf` and `sup` of the rank interval, the effective coverage probability of the rank interval `prob`, the estimated minimum and maximum rank values (`min` and `max`) and the rank `range`.

The output is a dataframe of class `average_ranks` /for which a method of function `plot` is available. See `plot.average_ranks` for details).

Value

A dataframe of class `average_ranks` whose columns are:

`avrg`	the average rank;
`inf`	the lower extreme of the rank interval;
`sup`	the upper extreme of the rank interval;
`prob`	the effective coverage probability of the rank interval;
`min`	the minimum rank;
`max`	the maximum rank;
`range`	the rank range.

Author(s)

Fattore M., Arcagni A.

See Also

`idn`, `equivalences`, `plot.average_ranks`

Examples

```
profiles <- var2prof(varlen = c(3, 2, 2))
Z <- getzeta(profiles)
res <- average_ranks(Z)
plot(res)
```

`binary` *binary*

Description

The function checks whether square matrix `m` represents a binary relation.

Usage

```
binary(m)
```

Arguments

`m`	a square matrix.

See Also

`transitivity`, `reflexivity`,

`antisymmetry`, `is.preorder`,

`is.partialorder`, `validate.partialorder.incidence`

Examples

```
M <- c(TRUE, FALSE, FALSE, FALSE, TRUE, TRUE, FALSE, FALSE, TRUE, FALSE, TRUE,
FALSE, TRUE, TRUE, TRUE, TRUE)
M <- matrix(M, 4, 4)
rownames(M) <- colnames(M) <- LETTERS[1:4]

binary(M)
```

`colevels` *Colevels of a poset*

Description

The function returns colevels associated to poset elements.

Usage

```
colevels(y)
```

Arguments

`y`	an object of class `cover` or `incidence`.

Examples

```
vl <- c(3, 2, 4)
prof <- var2prof(varlen = vl)
Z <- getzeta(prof)

colevels(Z)
```

`cover2incidence` *cover2incidence*

Description

The function computes the incidence matrix of a poset from its cover matrix.

Usage

```
cover2incidence(g)
```

Arguments

`g`	a cover matrix, an object of class `cover`

Value

The function returns the corresponding incidence matrix, an object of class `incidence`.

See Also

`incidence2cover`

`depths` *Depths*

Description

The function computes the depths of poset elements.

Usage

```
depths(z)
```

Arguments

`z`	an object of class `cover` or `incidence`.

Examples

```
vl <- c(3, 2, 4)
prof <- var2prof(varlen = vl)
Z <- getzeta(prof)

depths(Z)
```

`downset` *downset*

Description

The function computes a boolean vector identifying the poset elements below (or equal to) at least one element of the input subset `Q`.

Usage

```
downset(z, ...)
## S3 method for class 'cover'
downset(z, ...)
## S3 method for class 'incidence'
downset(z, Q = NULL, ...)
```

Arguments

`z`	a cover, or an incidence, matrix of S3 class `cover` or `incidence` respectively
`Q`	vector of indices identifying a subset of poset profiles
`...`	any of the above.

Examples

```
z <- getzeta(var2prof(varlen = c(2, 2, 2)))

plot(z, col = 1 + c(1, 1, 0, 0, 1, 0, 0, 0) + c(0, 0, 0, 2, 0, 0, 2, 2), lwd = 2)

Q <- c(4, 7, 8)
rownames(z)[Q]
downset(z, Q)

Q <- c("211", "112", "111")
downset(z, Q)
```

`drawedges` *drawedges*

Description

Graphical function called by `plot.cover` to draw the edges of the Hasse diagram representing the input cover matrix `C`.

Usage

```
drawedges(C, vertices, ...)
```

Arguments

`C`	cover matrix.
`vertices`	coordinates of the vertices obtained by function `vertices`.
`...`	line parameters, see `graphics{lines}`.

See Also

`plot.cover`, `vertices`, `graphics{lines}`

`equivalences` *Equivalence classes in a poset.*

Description

The function computes the set of poset elements sharing the same upset and downset.

Usage

```
equivalences(x)
```

Arguments

`x`	an object of class `incidence` or `cover`.

Value

The function computes a vector assigning an equivalence class to each profile. The vector is of class `factor`.

Author(s)

Arcagni A.

Examples

```
Lmbd <- getlambda(A > B, A > C, B > D, A > E, B > E, C > F, C > G)
res <- equivalences(Lmbd)

equivalence_classes <- levels(res)
colrs <- sapply(res, function(x) which(equivalence_classes == x)) + 1
plot(Lmbd, col = colrs, lwd = 2)
```

`evaluation` *Multidimensional evaluation on posets*

Description

Given a partial order (arguments `profiles` and/or `zeta`) and a selected `threshold`, the function returns an object of S3 class `parsec`, comprising the identification function and different severity measures, computed by uniform sampling of the linear extensions of the poset, through a C implementation of the Bubley - Dyer (1999) algorithm.

Usage

```
evaluation(
    profiles = NULL,
    threshold,
    error = 10^(-3),
    zeta = getzeta(profiles),
    weights = {
        if (!is.null(profiles))
            profiles$freq
        else rep(1, nrow(zeta))
    },
    distances = {
        n <- nrow(zeta)
        matrix(1, n, n) - diag(1, n)
    },
    linext = lingen(zeta),
    nit = floor({
        n <- nrow(zeta)
        n^5 * log(n) + n^4 * log(error^(-1))
    }),
    maxint = 2^31 - 1,
    inequality = FALSE
)
inequality(profiles = NULL, zeta = getzeta(profiles), ...)
```

Arguments

`profiles`	an object of S3 class `wprof`.
`threshold`	a vector identifying the threshold. It can be a vector of indexes (numeric), a vector of profile names (character) or a boolean vector of length equal to the number of profiles. Function `inequality` does not require its definition since its results do not depend on it.
`error`	the "distance" from uniformity in the sampling distribution of linear extensions.
`zeta`	the incidence matrix of the poset. An object of S3 class `incidence`. By default, extracted from `profiles`.
`weights`	weights assigned to profiles. If the argument `profiles` is not `NULL`, weights are by default set equal to profile frequencies, otherwise they are set equal to 1.
`distances`	matrix of distances between pairs of profiles. The matrix must be square, with dimensions equal to the number of profiles. Even if the poset is complete, the distance between two profiles is computed only if one profile covers the other.
`linext`	the linear extension initializing the sampling algorithm. By default, it is generated by `lingen(zeta)`. Alternatively, it can be provided by the user through a vector of profile positions.
`nit`	Number of ITerations in the Bubley-Dyer algorithm, by default evaluated from a formula of Karzanov and Khachiyan based on the number of profiles and the argument `error` (see Bubley and Dyer, 1999).
`maxint`	Maximum integer. By default the maximum integer obtainable in a 32bit system. This argument is used to group iterations and run the compiled C code more times, so as to avoid memory indexing problems. Users can set a lower value to `maxint` in case of low RAM availability.
`inequality`	boolean parameter (by default `FALSE`) to make the evaluation function return also a measure of inequality (which can make computations quite lengthy). It is `TRUE` in function `inequality` and can not be modified.
`...`	further optional graphical parameters. See `plot.default`.

Value

`profiles`	an object of S3 class `wprof` reporting poset profiles and their associated frequencies (number of statistical units in each profile).
`number_of_profiles`	number of profiles.
`number_of_variables`	number of variables.
`incidence`	S3 class `incidence`, incidence matrix of the poset.
`cover`	S3 class `cover`, cover matrix of the poset.
`threshold`	boolean vector specifying whether a profile belongs to the threshold.
`number_of_iterations`	number of iterations performed by the Bubley-Dyer algorithm.
`rank_dist`	matrix reporting by rows the relative frequency distributions of the ranks of each profile, over the set of sampled linear extensions.

`thr_dist`	vector reporting the relative frequency a profile is used as threshold in the sampled linear extensions.
`prof_w`	vector of weights assigned to each profile.
`edg_w`	matrix of distances between profiles, used to evaluate the gap measures.
`idn_f`	vector reporting the identification function, computed as the fraction of sampled linear extensions where a profile is in the downset of the threshold.
`svr_abs`	vector reporting, for each profile, the average graph distance from the first profile above all threshold elements, over the sampled linear extensions. In each linear extension, the distance is set equal to 0 for profiles above the threshold.
`svr_rel`	equal to svr_abs divided by its maximum, that is svr_abs of the minimal element in the linear extension.
`wea_abs`	vector reporting, for each profile, the average graph distance from the maximum threshold element, over the sampled linear extensions. In each linear extension, the distance is set equal to 0 for profiles in the downset of threshold elements.
`wea_rel`	the previous absolute distance is divided by its maximum possible value, that is the absolute distance of the threshold from the maximal element in the linear extension.
`poverty_gap`	Population mean of svr_rel
`wealth_gap`	Population mean of wea_rel
`inequality`	when the argument `inequality` is `TRUE`, the average value of the inequality index over the linear extensions (see Fattore and Arcagni, 2013). Function `inequalty` returns only this result.

References

Bubley R., Dyer M. (1999), Faster random generation of linear extensions, Discrete Math., 201, 81-88.

Fattore M., Arcagni A. (2013), Measuring multidimensional polarization with ordinal data, SIS 2013 Statistical Conference, BES-M3.1 - The BES and the challenges of constructing composite indicators dealing with equity and sustainability

Examples

```
profiles <- var2prof(varlen = c(3, 2, 2))
threshold <- c("311", "112")

res <- evaluation(profiles, threshold, maxint = 10^5)

summary(res)
plot(res)
```

FOD Fuzzy First Order Domninance analysis on partial orders

Description

The function `FOD` performs the Fuzzy First Order Dominance analysis described in Fattore and Arcagni (forthcoming).

Usage

```
FFOD(profiles, ...)
## S3 method for class 'wprof'
FFOD(profiles,
    distributions = as.data.frame(profiles$freq),
    lambda = do.call(
        getlambda, as.list(names(profiles$profiles))
    ),
    alpha = NULL, ...
)
```

Arguments

`profiles`	an object of class `wprof`.
`distributions`	a `data.frame` of frequencies/weights where the columns correspond to the different distributions and the rows to the profiles. The profiles in the rows have to be ordered as in `profiles`.
`lambda`	object of class `incidence` representing the partial order of the relative importance of the indicators. By default, the lambda poset is an antichain (i.e. all the indicators are considered equi-important).
`alpha`	vector of values to cut the `mintr.delta` matrix to generate the posets in the list `covers`. Default is `NULL` posets are generated for each different value in matrix `mintr.delta`. See section 'Value' below.
`...`	any of above.

Details

The function requires the set of profiles, through the object `profiles` of class `wprof`, and the corresponding frequencies, which can be defined by the argument `distributions` of class `data.frame`.

Notice that a warning is provided if the `rownames` of the distributions do not match the `rownamens` of the profiles.

Through poset `lambda`, it is possible to provide (ordinal) information on the relative importance of the indicators in the multi-indicator system.

Value

An object of class `FODposet` containing:

`delta`	matrix of the overall dominance degrees.
`mintr.delta`	matrix of the min-transitive closure of matrix `delta`.
`global.approx`	L1 distance between `delta` and `mintr.delta`, divided by the L1 norm of `delta`.
`global.approx.corr`	L1 distance between `delta` and `mintr.delta`, divided by the L1 norm of `delta` after removing its diagonal.
`cell.approx`	matrix of absolute differences between the elements of `delta` and the elements of `mintr.delta`.
`posets.ind`	`data.frame` with indicators describing the partial orders obtained as alpha-cuts of the min-transitive closure `mintr.delta`. For each poset, the data frame provides: its `cardinality`, the number of `comparabilities`, the number of `incomparabilities` and their ratio (`ci.ratio`).
`eqv.classes`	list of boolean matrices specifying, for each alpha-cut, the equivalence classes of the input distributions. Equivalence classes are reported by rows and the initial distributions by columns. If element ij of the matrix is `TRUE`, then distribution j belongs to the i-th equivalence class.
`covers`	list of objects of class `cover` comprising the cover matrices of the poset generated by each alpha-cut of `mintr.delta`.

Author(s)

Fattore M., Arcagni A.

References

Fattore M., Arcagni A. (forthcoming), F-FOD: Fuzzy First Order Dominance analysis and populations ranking over ordinal multi-indicator systems.

Examples

```
v1 <- as.ordered(c("a", "b", "c", "d"))
v2 <- 1:3
prof <- var2prof(varmod = list(v1 = as.ordered(c("a", "b", "c", "d")), v2 = 1:3))
np <- nrow(prof$profiles)

k <- 10 # number of populations
set.seed(0)
populations <- as.data.frame(lapply(1:k, function(x) round(runif(np)*100)))
rownames(populations) <- rownames(prof$profiles)
names(populations) <- paste0("P", 1:k)

prof
populations

res <- FFOD(profiles = prof, distributions = populations)
res
```

`gen.downset` *Antichain generating a given downset*

Description

The function returns the antichain generating the input downset `Q`, given the incidence matrix `z` of the poset.

Usage

```
gen.downset(z, Q = 1)
```

Arguments

`z`	an incidence matrix.
`Q`	a vector (`boolean`, `numeric` indexing elements, or `character` with elements' names) identifying the input downset.

Value

A `boolean` vector.

See Also

`gen.upset`

Examples

```
lv <- c(2, 3, 2)
prof <- var2prof(varlen = lv)

z <- getzeta(prof)
down <- c("111", "211", "112", "212")
gen <- gen.downset(z, down)

plot(z, lwd = 1 + (rownames(prof$profiles)%in%down), col = 1 + gen,
sub = "bold = the downset, red = the antichain generating the downset")
```

`gen.upset` *Antichain generating a given upset*

Description

The function returns the antichain generating the input upset `Q`, given the incidence matrix `z` of the poset.

Usage

```
gen.upset(z, Q = 1)
```

Arguments

`z`	an incidence matrix.
`Q`	a vector (`boolean`, `numeric` indexing elements, or `character` with elements' names) identifying the input upset.

Value

A `boolean` vector.

See Also

`gen.downset`

Examples

```
lv <- c(2, 3, 2)
prof <- var2prof(varlen = lv)

z <- getzeta(prof)
up <- c("221", "131", "231", "222", "132", "232")
gen <- gen.upset(z, up)

plot(z, lwd = 1 + (rownames(prof$profiles)%in%up), col = 1 + gen,
sub = "bold = the upset, red = the antichain generating the upset")
```

`getlambda` *Object constructor for the incidence matrix representing a partial order on variables.*

Description

The function creates an object of class `incidence` representing a partial order on the set of variables.

Usage

```
getlambda(...)
```

Arguments

`...`	Cover relations between variable pairs.

Details

Cover relations between pair of ariables are defined by the names of the two variables and the symbols < and >. For instance, if variable A is covered by variable B, write the cover relation as `A` < `B` or `B` > `A`. If a variable is not comparable to the others, write the name of the variable alone.

Value

an object of class `incidence`.

Author(s)

Alberto Arcagni

See Also

`plot.cover`

Examples

```
Lambda <- getlambda(BOTTOM < A, B > BOTTOM, INCOMP)
plot(Lambda)
```

`getzeta` *Incidence matrix generation*

Description

The function computes the incidence matrix from the set of input profiles `y`. The output is a boolean matrix of S3 class `incidence`.

Usage

```
getzeta(y)
## S3 method for class 'wprof'
getzeta(y)
```

Arguments

`y`	the set of profiles, an object of S3 class `wprof`.

Examples

```
prf <- var2prof(varlen = c(2, 3))
getzeta(prf)
```

`heights` *Heights*

Description

The function computes the vector of heights of poset elements.

Usage

```
heights(z)
```

Arguments

`z`	an object of class `cover`, `incidence` or `poset`.

Examples

```
vl <- c(3, 2, 4)
prof <- var2prof(varlen = vl)
Z <- getzeta(prof)

heights(Z)
```

`idn` *Multidimensional evaluation on posets (Identification Function only)*

Description

Given a partial order (arguments `profiles` and/or `zeta`) and a selected `threshold`, the function computes the identification function, as a S3 class object `parsec`. The identification function is computed by uniform sampling of the linear extensions of the input poset, through a C implementation of the Bubley - Dyer (1999) algorithm. idn is a simplified and faster version of evaluation, computing just the identification function.

Usage

```
idn(
    profiles = NULL,
    threshold,
    error = 10^(-3),
    zeta = getzeta(profiles),
    weights = {
        if (!is.null(profiles))
            profiles$freq
        else rep(1, nrow(zeta))
    },
    linext = lingen(zeta),
    nit = floor({
        n <- nrow(zeta)
        n^5 * log(n) + n^4 * log(error^(-1))
    }),
    maxint = 2^31 - 1
)
```

Arguments

`profiles`	an object of S3 class `wprof`.
`threshold`	a vector identifying the threshold. It can be a vector of indexes (numeric), a vector of poset element names (character) or a boolean vector of length equal to the number of elements.
`error`	the "distance" from uniformity in the sampling distribution of linear extensions.
`zeta`	the incidence matrix of the poset. An object of S3 class `incidence`. By default, extracted from `profiles`.
`weights`	weights assigned to profiles. If the argument `profiles` is not `NULL`, weights are by default set equal to profile frequencies, otherwise they are set equal to 1.
`linext`	the linear extension initializing the sampling algorithm. By default, it is generated by `lingen(zeta)`. Alternatively, it can be provided by the user through a vector of elements positions.

`nit`	Number of iterations in the Bubley-Dyer algorithm, by default evaluated using a formula of Karzanov and Khachiyan based on the number of poset elements and the argument `error` (see Bubley and Dyer, 1999).
`maxint`	Maximum integer. By default the maximum integer obtainable in a 32bit system. This argument is used to group iterations and run the compiled C code more times, so as to avoid memory indexing problems. User can set a lower value to `maxint` in case of lower RAM availability.

Value

`profiles`	an object of S3 class `wprof` reporting poset profiles and their associated frequencies (number of statistical units in each profile).
`number_of_profiles`	number of profiles.
`number_of_variables`	number of variables.
`incidence`	S3 class `incidence`, incidence matrix of the poset.
`cover`	S3 class `cover`, cover matrix of the poset.
`threshold`	boolean vector specifying whether a profile belongs to the threshold.
`number_of_iterations`	number of iterations performed by the Bubley Dyer algorithm.
`rank_dist`	matrix reporting by rows the relative frequency distribution of the poverty ranks of each profile, over the set of sampled linear extensions.
`thr_dist`	vector reporting the relative frequency a profile is used as threshold in the sampled linear extensions. This result is useful for a posteriori valuation of the poset threshold.
`prof_w`	vector of weights assigned to each profile.
`edges_weights`	matrix of distances between profiles, used to evaluate the measures of gap.
`idn_f`	vector reporting the identification function, computed as the fraction of sampled linear extensions where a profile is in the downset of the threshold.
`svr_abs`	`NA` use evaluation to obtain this result.
`svr_rel`	`NA` use evaluation to obtain this result.
`wea_abs`	`NA` use evaluation to obtain this result.
`wea_rel`	`NA` use evaluation to obtain this result.
`poverty_gap`	`NA` use evaluation to obtain this result.
`wealth_gap`	`NA` use evaluation to obtain this result.
`inequality`	`NA` use evaluation to obtain this result.

References

Bubley R., Dyer M. (1999), Faster random generation of linear extensions, Discrete Math., 201, 81-88.

Fattore M., Arcagni A. (2013), Measuring multidimensional polarization with ordinal data, SIS 2013 Statistical Conference, BES-M3.1 - The BES and the challenges of constructing composite indicators dealing with equity and sustainability

Examples

```
profiles <- var2prof(varlen = c(3, 2, 4))
threshold <- c("311", "112")

res <- idn(profiles, threshold, maxint = 10^5)

summary(res)
plot(res)
```

`incidence2cover` *incidence2cover*

Description

The function computes the cover matrix associated to the input incidence matrix (i.e. the cover matrix whose transitive closure is the input incidence matrix).

Usage

```
incidence2cover(z)
```

Arguments

`z`	an incidence matrix, an object of class `incidence`.

Value

Cover matrix, an object of class `cover`.

See Also

`cover2incidence`

`incomparability` *Incomparability between profiles*

Description

The function computes the set of pairwise incomparabilities between poset elements.

Usage

```
incomp(z)
```

Arguments

`z`	an incidence matrix.

Value

A boolean matrix whose element ij is `TRUE` when profiles i and j are incomparable.

See Also

`getzeta`

Examples

```
vl <- c(2, 2, 2)
pr <- var2prof(varlen = vl)
Z <- getzeta(pr)
incomp(Z)
```

`is.downset` *is.downset*

Description

The function checks whether the input set of poset elements `Q` is a downset of the poset represented by the incidence matrix `z`.

Usage

```
is.downset(z, Q = 1)
```

Arguments

`z`	incidence matrix
`Q`	vector identifying the input set of profiles.

Examples

```
z <- getzeta(var2prof(varlen = c(2, 2, 2)))

plot(z, col = 1 + c(1, 1, 0, 0, 1, 0, 0, 0) + c(0, 0, 0, 2, 0, 0, 2, 2), lwd = 2)

Q <- c(4, 7, 8)
rownames(z)[Q]
is.downset(z, Q)

Q <- c("211", "112", "111")
is.downset(z, Q)
```

`is.linext` *is.linext*

Description

The function checks whether the input argument `order` is a linear extension of the poset represented by the incidence matrix `z`.

Usage

```
is.linext(order, z)
```

Arguments

`order`	indexes of the poset elements (as rows and columns of `z` matrix) specifying the candidate linear order.
`z`	incidence matrix.

Examples

```
Z <- getzeta(var2prof(varlen = c(3, 3)))
ranks <- c(1, 4, 2, 3, 5, 7, 6, 8, 9)
names(ranks) <- rownames(Z)
ranks
is.linext(order = ranks, z = Z)
```

`is.partialorder` *is.partialorder*

Description

The function checks whether the input boolean square matrix `m` represents a partial order.

Usage

```
is.partialorder(m)
```

Arguments

`m`	a boolean square matrix..

See Also

`transitivity`, `binary`, `reflexivity`,

`antisymmetry`, `is.preorder`,

`validate.partialorder.incidence`

Examples

```
M <- c(TRUE, FALSE, FALSE, FALSE, TRUE, TRUE, FALSE, FALSE, TRUE, FALSE, TRUE,
FALSE, TRUE, TRUE, TRUE, TRUE)
M <- matrix(M, 4, 4)
rownames(M) <- colnames(M) <- LETTERS[1:4]

is.partialorder(M)
```

`is.preorder` *is.preorder*

Description

The function checks whether the input boolean square matrix `m` represents a preorder.

Usage

```
is.preorder(m)
```

Arguments

`m`	a boolean square matrix.

See Also

`transitivity`, `binary`, `reflexivity`,

`antisymmetry`, `is.partialorder`,

`validate.partialorder.incidence`

Examples

```
M <- c(TRUE, FALSE, FALSE, FALSE, TRUE, TRUE, FALSE, FALSE, TRUE, FALSE, TRUE,
FALSE, TRUE, TRUE, TRUE, TRUE)
M <- matrix(M, 4, 4)
rownames(M) <- colnames(M) <- LETTERS[1:4]

is.preorder(M)
```

`is.upset` *is.upset*

Description

The function checks whether the input set of elements `Q` is an upset of the poset represented by the incidence matrix `z`.

Usage

```
is.upset(z, Q = 1)
```

Arguments

`z`	an incidence matrix.
`Q`	vector specifying the input set of poset elements.

Examples

```
z <- getzeta(var2prof(varlen = c(2, 2, 2)))

plot(z, col = 1 + c(1, 1, 0, 0, 1, 0, 0, 0) + c(0, 0, 0, 2, 0, 0, 2, 2), lwd = 2)

Q <- c(4, 7, 8)
rownames(z)[Q]
is.upset(z, Q)

Q <- c("211", "112", "111")
is.upset(z, Q)
```

`latex` *latex*

Description

The function returns the LaTeX code to create a tikz figure representing the Hasse diagram drawn from a set of profiles (`prof`), an incidence matrix (`Z`) or a cover matrix (`C`). The code can be copied and pasted into a `latex` file. The `latex` source requires the `tikz` package,

Usage

```
latex(y, ...)
## S3 method for class 'wprof'
latex(y, label = "", caption = "", scale = c(1, 1), ...)
## S3 method for class 'incidence'
latex(y, label = "", caption = "", scale = c(1, 1), ...)
## S3 method for class 'cover'
latex(y, label = "", caption = "", scale = c(1, 1), ...)
```

Arguments

`y`	an object of S3 class `wprof`, an object of S3 class `incidence` or an object of S3 class `cover`.
`label`	the label of the LaTeX figure.
`caption`	the caption of the LaTeX figure.
`scale`	a vector of two elements to control the scale of the X-axis and the scale of the Y-axis in the LaTeX output.
`...`	any of above.

Examples

```
prof <- var2prof(varlen = c(2, 3))
latex(prof, label="fg:hasse", caption="Hasse diagram", scale = c(2, 2))
```

`LE` *Linear extensions*

Description

The function generates all of the linear extensions of the partial order defined by the incidence matrix `Lambda`.

Usage

```
LE(Lambda)
```

Arguments

`Lambda`	incidence matrix.

Value

a list of vectors representing all linear orders compatible with the `Lambda` incidence matrix.

Author(s)

Alberto Arcagni

See Also

`getlambda`

Examples

```
Lambda <- getlambda(A < B, A < C, D < C)
LE(Lambda)
```

`LE2incidence` *Computes the incidence matrices of lexicographic linear extensions of a profile poset.*

Description

The function generates the incidence matrices of the lexicographic linear extensions of a profile poset, given the variables (argument `varmod` or `varlen`) and a list of complete orders on them (argument `lst`).

Usage

```
LE2incidence(
    lst,
    varmod = lapply(as.list(varlen), function(lst) 1:lst),
    varlen = sapply(varmod, length)
)
## Default S3 method:
LE2incidence(
    lst,
    varmod = lapply(as.list(varlen), function(lst) 1:lst),
    varlen = sapply(varmod, length)
)
## S3 method for class 'list'
LE2incidence(
    lst,
    varmod = lapply(as.list(varlen), function(x) 1:x),
    varlen = sapply(varmod, length)
)
```

Arguments

`lst`	a vector of `characters`, or a list of specifies the names of the variables in increasing order. See details.
`varmod`	list of variables and their grades. See details.
`varlen`	a vector with the number of grades of each variable. See details.

Details

Argument `lst` is a list of chracter vectors. Each vector lists variable names in increasing order.

List `varmod` and vector `varlen` must be named so as to identify the variables they refer to. Profiles are generated as combinations of the variables' grades. The names of the profiles are the grades of the variables concatenated, after the variables order in `varmod`/`varlen`. See `var2prof` for more details about these arguments.

Value

an object of S3 class `incidence` or a list of objects of S3 class `incidence`.

Author(s)

Alberto Arcagni

See Also

`var2prof`

Examples

```
Lambda <- getlambda(A < B, C < D)
plot(Lambda)
lst <- LE(Lambda)
vl <- c(A = 2, B = 2, C = 2, D = 2)
lstZeta <- LE2incidence(lst, varlen = vl)
for (x in lstZeta)
    plot(x)
```

`levels.incidence and levels.cover`

Levels of a poset

Description

The methods return a vector associating each profile with the corresponding level. The behaviour of these methods for objects of classes `incidence` and `cover` is different from the behaviour of function `levels` for factors.

Usage

```
## S3 method for class 'incidence'
levels(x)
## S3 method for class 'cover'
levels(x)
```

Arguments

`x`	an object of class `cover` or `incidence`.

See Also

the function `levels` for objects of type factor

Examples

```
vl <- c(3, 2, 4)
prof <- var2prof(varlen = vl)
Z <- getzeta(prof)

levels(Z)
```

`lingen` *lingen*

Description

The function computes a vector of ranks, defining a linear extension of the poset represented by incidence matrix `z`.

Usage

```
lingen(z)
```

Arguments

`z`	an incidence matrix.

Examples

```
Z <- getzeta(var2prof(varlen = c(3, 3)))
lingen(Z)
```

`linzeta` *linzeta*

Description

The function computes the incidence matrix of the linear order defined by the rank vector `lin`. It returns an object of S3 class `incidence`.

Usage

```
linzeta(lin)
```

Arguments

`lin`	a vector of elements' ranks.

Examples

```
ranks <- c(5, 3, 4, 2, 1)
names(ranks) <- LETTERS[1:5]
linzeta(ranks)
plot(linzeta(ranks))
```

`maximal` *Maximal elements of a poset.*

Description

The function returns a boolean vector identifying the maximal elements of the poset.

Usage

```
maximal(z)
```

Arguments

`z`	an object of class `cover` or `incidence`.

Examples

```
vl <- c(3, 2, 4)
prof <- var2prof(varlen = vl)
Z <- getzeta(prof)

maximal(Z)
```

`merge.wprof` *Merge two sets of profiles.*

Description

Method of the function `merge` of package `base` to merge two objects of class `wprof` generated through functions `var2prof` or `pop2prof`.

Usage

```
## S3 method for class 'wprof'
merge(x, y, support = FALSE, FUN = "+", all = TRUE, ...)
```

Arguments

`x, y`	objects of class `wprof` to be coerced to one.
`support`	boolean variables specifying whether `y` is the support of `x` (`FALSE` by default).
`FUN`	function to be applied to the profiles' frequencies (by default, FUN = `sum`). It is ignored if `support` is `TRUE`.
`all`	same argument of function `merge`, by default set to `TRUE`, to get all possible profiles. If a profile is not observed in the data, its frequency is set to 0.
`...`	additional arguments to be passed to method `merge.data.frame` of the package `base`.

Details

Objects of class `wprof` are composed of a `data.frame` of profiles and a vector of frequencies. This method applies method `merge.data.frame` to the profiles and applies function `FUN` to the frequencies.

If `support` is `TRUE`, function `merge.data.frame` is not used and the output corresponds to the object `y`, but with its frequencies modified. These are set equal to the frequencies of the corresponding profiles in `x`, or to 0 for profiles not contained in `x`.

Author(s)

Arcagni A.

See Also

`merge`, `var2prof`, `pop2prof`

Examples

```
n <- 5
v1 <- as.ordered(c("a", "b", "c", "d"))
v2 <- 1:3
set.seed(0)
pop <- data.frame(
v1 = sample(v1, n, replace = TRUE),
v2 = sample(v2, n, replace = TRUE)
)

survey_weights <- round(runif(5)*10)

prof1 <- pop2prof(pop, weights = survey_weights)
prof2 <- var2prof(varmod = list(v1 = as.ordered(c("a", "b", "c", "d")), v2 = 1:3))

# prof2 is the support of prof1
merge(prof1, prof2, support = TRUE)

# union between the two sets of profiles and their frequencies are added
merge(prof1, prof2)
```

```
# intersection of the sets of profiles with the assumption
# that the minimum number of observations is shared
# between the two distributions
merge(prof1, prof2, all = FALSE, FUN = min)

prof2$freq <- prof2$freq*10
# to remove from prof2 the observations in prof1
distribution <- merge(prof2, prof1, FUN = "-"); distribution
```

`minimal` *Minimal elements of a poset*

Description

The function returns a boolean vector identifying the minimal elements of the poset.

Usage

```
minimal(z)
```

Arguments

`z`	an object of class `cover` or `incidence`.

Examples

```
vl <- c(3, 2, 4)
prof <- var2prof(varlen = vl)
Z <- getzeta(prof)

minimal(Z)
```

`mrg` *Merge posets*

Description

The function merges posets defined through a list of incidence matrices or a list of complete orders between the variables (argument `lst`). In the second case the variables must be defined (argument `varmod` or `varlen`).

Usage

```
mrg(
    lst,
    varmod = lapply(as.list(varlen), function(x) 1:x),
    varlen = sapply(varmod, length)
)
## S3 method for class 'incidence'
mrg(lst, varmod = NULL, varlen = NULL)
## S3 method for class 'character'
mrg(
lst,
    varmod = lapply(as.list(varlen), function(x) 1:x),
    varlen = sapply(varmod, length)
)
```

Arguments

`lst`	a list of incidence matrices (class `incidence`) or list of vectors of `characters`. See details.
`varmod`	list of variables and their grades. See details.
`varlen`	a vector of number of grades of each variable. See details.

Details

For efficiency reasons, the argument `lst` can be also a list of vectors of characters. In this case, each vector lists the names of the variables in increasing order.

The list `varmod` and the vector `varlen` must be named, so as to identify the variables they refer to. The profiles are generated by the combinations of the variables grades. The names of the profiles are the grades of the variables concatenated, according to variables order in `varmod`/`varlen`. See `var2prof` for more details about these arguments.

Value

an object of S3 class `incidence`.

Author(s)

Alberto Arcagni

See Also

`var2prof`, `LE2incidence`

Examples

```
# Example with lst as list of incidence matrices
Lambda <- getlambda(A < B, C < D)
plot(Lambda)
lst <- LE(Lambda)
```

```
vl <- c(A = 2, B = 2, C = 2, D = 2)
lstZeta <- LE2incidence(lst, varlen = vl)
for (x in lstZeta)
    plot(x)
mrg(lstZeta)

# Example with lst as list of characters
Lambda <- getlambda(A < B, C < D)
lst <- LE(Lambda)
vl <- c(A = 2, B = 2, C = 2, D = 2)
Zeta <- mrg(lst, varlen = vl)
plot(Zeta)
```

`MRP` *Mutial ranking probability matrix*

Description

Function to evaluate Mutial Ranking Probability (MRP) matrix based on `netrankr` package.

Usage

```
MRP(Z, method = c("exact", "mcmc", "approx"), error = 10^(-3), nit = NULL)
## S3 method for class 'incidence'
MRP(Z, method = c("exact", "mcmc", "approx"), error = 10^(-3), nit = NULL)
```

Arguments

`Z`	an incidence matrix, an object of class `incidence`.
`method`	a string to choose the method applied to evaluate the MRP matrix. The default value is `"exact"`. See section 'Details' below.
`error`	considered only if `mcmc` method is selected. The "distance" from uniformity in the sampling distribution of linear extensions.
`nit`	considered only if `mcmc` method is selected. Number of ITerations in the Bubley-Dyer algorithm, by default evaluated indicated in Bubley and Dyer (1999) depending on the value of `error`.

Details

Package `netrankr` provides three functions to evaluate MRP matrix. Note that MRP matrix definition in `netrankr` is a little different from the one used in Fattore and Arcagni (2018), therefore this function unifies the results to the second definition.

Parameter `method` allows the selection of which function of package `netrankr` to use: `"exact"` runs the function `exact_rank_prob` that provides the exact results, `"mcmc"` the function `mcmc_rank_prob` that provide the estimated results through the Bubley Dyer algorithm and `"approx"` runs the function `approx_rank_relative` that provide the Bruggemann and Carlsen (2011) approximated results. For small posets it is possible to evaluate the exact MRP matrix, for larger posets it is necessary to use the appoximated results.

Value

An object of class `matrix` representing the MRP matrix. Dimensions names are equal to incidence matrix ones.

References

Bruggemann R., Carlsen L., (2011). An improved estimation of averaged ranks of partial orders. MATCH Commun. Math. Comput. Chem., 65(2):383-414.

Bubley R., Dyer M. (1999), Faster random generation of linear extensions, Discrete Math., 201, 81-88.

Fattore M., Arcagni A. (2018). Using mutual ranking probabilities for dimensionality reduction and ranking extraction in multidimensional systems of ordinal variables. Advances in Statistical Modelling of Ordinal Data, 117.

See Also

`exact_rank_prob`, `mcmc_rank_prob`, `approx_rank_relative`

Examples

```
L <- getlambda(A < B, C < B, B < D)
MRP(L)
```

`MRPlex` *Mutual ranking probabilities on the lexicographic linear extensions set*

Description

The function returns the mutual ranking probabilities matrix evaluated considering only the lexicographic linear extensions. Results are obtained by exact formula.

Usage

```
MRPlex(profiles, selection = NULL)
```

Arguments

`profiles`	an object of S3 class `wprof`.
`selection`	a vector of `string` indicating a subset of profiles to evaluate the mutual ranking probabilities. If `NULL` the mutual ranking probabilities are evaluated for all profiles.

Value

The MRP matrix of the selected profiles.

Examples

```
prf <- var2prof(varlen = c(2, 2, 2))
MRPlex(prf)
```

`obsprof` *Remove unobserved profiles.*

Description

The function removes, from the set of possible profiles `prof` derived from the multi-indicator system, those unobserved in the input dataset (i.e. profiles with associated frequency equal to zero). It returns an object of class S3 `wprof` comprising the observed profiles and their frequencies.

Usage

```
obsprof(prof)
## S3 method for class 'wprof'
obsprof(prof)
```

Arguments

`prof`	object of S3 class `wprof`.

Examples

```
prf <- var2prof(varlen = c(3, 3, 3))
prf$freq <- sample(c(0, 1), 3*3*3, replace = TRUE)
prf <- obsprof(prf)
plot(prf, shape = "equispaced")
```

`parsec2igraph` *Converting a partial order to an object of the package* `igraph`.

Description

The function turns a cover matrix to an `igraph` object, so as to allow using the graphical power of `igraph` to plot Hasse diagrams. Objects of class `cover` are boolean matrices where element ij is equal to 1 if element i *is covered by* element j. This makes the cover matrix the transpose of the adjacency matrix of a graph, describing the cover relation in `igraph`.

Usage

```
parsec2igraph(p, ...)
## S3 method for class 'cover'
parsec2igraph(p, ...)
## S3 method for class 'incidence'
parsec2igraph(p, ...)
```

Arguments

`p`	an object of class `cover` or `incidence`.
`...`	additional arguments of the function `vertices`.

Value

The function returns an object of class `igraph`, representing the directed graph defined by the cover relation.

The function adds to the graph a layout generated through function `vertices`, so as to plot the graph according to the conventions used for Hasse diagrams.

Author(s)

Arcagni, A.

References

Csardi G, Nepusz T: The igraph software package for complex network research, InterJournal, Complex Systems 1695. 2006. http://igraph.org

See Also

`igraph`, `vertices`

Examples

```
example(merge.wprof)
poset <- getzeta(distribution)
incidence2cover(poset)

G <- parsec2igraph(poset, noise = TRUE)
get.adjacency(G)
# tkplot(G, vertex.size = distribution$freq, vertex.color = "white")

G <- parsec2igraph(poset, noise = 10)
# tkplot(G, vertex.size = distribution$freq, vertex.color = "white")
```

`plot.average_ranks` *Method of function* `plot` *for objects of class* `average_ranks`

Description

From the output of the function `average_ranks`, the function plots the average rank and the associated rank interval, for each element of the poset.

Usage

```
## S3 method for class 'average_ranks'
plot(x,
    range.first = TRUE, range.col = "black", range.lty = 1,
    range.lwd = 1, type = "p", ylim = c(nrow(x), 1),
    xlab = "", ylab = "Average rank", pch = c(16, 3, 3),
    col = "black", cex = c(1, 1, 1), ...
)
```

Arguments

`x`	An object of class `average_ranks`.
`range.first`	A boolean attribute to specify whether the interval is plotted in background (`TRUE`) or in foreground (`FALSE`).
`range.col`	Color of the interval.
`range.lty`	The line type to represent the range; the values are the same of the attribute `lty` in the `plot.default` function.
`range.lwd`	Width of the lines representing the range.
`type`	Attribute of the function `plot.default`, here `"p"` by default.
`ylim`	Attribute of function `plot.default`, here `c(max(x$sup), 1)` by default (this way, the Y-axis is reversed, so that rank 1 corresponds to "best").
`xlab`	Attribute of the function `plot.default`, here `""` by default.
`ylab`	Attribute of the function `plot.default`, here `"Average rank"` by default.
`pch`	Attribute of the function `plot.default`, here `c(16, 3, 3)` by default. This method uses the `matplot` function to plot the average ranks and their range. The first value refers to the point character of the average rank, the other two to the point characters of the range.
`col`	Attribute of the function `plot.default`, here `"black"` by default. The average ranks and their ranges are of the same color, but similarly to pch, users can provide a vector of different colors.
`cex`	Attribute of the function `plot.default`, here `c(1, 1, 1)` by default.
`...`	Other arguments of the function `plot.default`.

See Also

`average_ranks`, `plot.default`, `matplot`

Examples

```
profiles <- var2prof(varlen = c(3, 2, 4))
Z <- getzeta(profiles)
res <- average_ranks(Z)
plot(res)
```

`plot.cover` *Hasse diagram*

Description

`plot` methods to draw Hasse diagrams, for objects of S3 classes `wprof`, `incidence`, `cover`,

Usage

```
## S3 method for class 'wprof'
plot(x,  shape = c("square", "circle", "equispaced"), noise = FALSE, ...)
## S3 method for class 'incidence'
plot(x, shape = c("square", "circle", "equispaced"), noise = FALSE, ...)
## S3 method for class 'cover'
plot(x, shape = c("square", "circle", "equispaced"), noise = FALSE,
    pch = 21, cex = max(nchar(rownames(x))) + 2, bg = "white", ...)
```

Arguments

`x`	an object of S3 class `wprof`, an object of S3 class `incidence` or an object of S3 class `cover`.
`shape`	shape of the Hasse diagram. See `vertices`.
,	
`noise`	jittering in the shape of the Hasse diagram. See `vertices`.
`pch`	graphical parameter. See `plot.default`.
`cex`	graphical parameter. See `plot.default`.
`bg`	graphical parameter. See `plot.default`.
`...`	further optional graphical parameters. See `plot.default`.

Examples

```
prf <- var2prof(varlen = c(5, 5, 5))
prf$freq <- sample(c(rep(0, 20), 1, 2, 3), 5*5*5, replace = TRUE)
prf <- obsprof(prf)

z <- getzeta(prf)

plot(z, shape = "equispaced", col = prf$freq, lwd = 2)
```

---

plot.parsec *Plot the outputs of the PARSEC function* `evaluation`*.*

Description

Several representations of the results provided by the evaluation function.

Usage

```
## S3 method for class 'parsec'
plot(
    x,
    which = c("Hasse", "threshold", "identification", "rank", "gap"),
    ask = dev.interactive(),
    shape = c("square", "circle", "equispaced"),
    noise = FALSE,
    ...
)
```

Arguments

`x`	an object of S3 class `parsec`, output of the `evaluation` function.
`which`	the names of the graphs to be plotted (`all`, by default); the user can choose among • `Hasse`, the Hasse diagram of the poset, see `plot.cover` for details, • `threshold`, the relative frequencies of the times a profile is used as threshold in the sampled linear extensions. • `rank`, barplot providing the rank distribution of each profile (X-axis). The heights of the blocks represent relative frequencies (the sum of the heights over profiles is equal to 1) and the color represents the rank: white for rank one, black for the highest rank and a gray scale for intermediate ranks. • `gap`, a unified representation of the relative (e.g. poverty) gap and of the relative (e.g. wealth) gap. The horizontal lines represent the average (e.g. poverty) gap and the average (e.g. wealth gap). The darker vertical dashed lines represent the threshold profiles.
`ask`	boolean value indicating whether the system has to ask users before changing the plot.
`shape`	the shape of the Hasse diagram, see `plot.cover` for details.
`noise`	jittering in the shape of the Hasse diagram. See `vertices`.
`...`	further arguments for the `plot.cover` function.

See Also

`evaluation`, `plot.cover`

Examples

```
profiles <- var2prof(varlen = c(3, 2, 4))
threshold <- c("311", "112")

res <- evaluation(profiles, threshold, nit = 10^3)

plot(res)
```

`plot.rank_stability` *Plot outputs of PARSEC function* `rank_stability`.

Description

The function generates four plots, to reproduce the sequence of the average ranks and of the positions of the elements, in the rankings associated to the alpha-cut posets.

Rankings and average ranks have to be evaluted with the function `rank_stability`.

First and third plots show the sequence of average ranks, second and fourth show the sequence of rankings. Sequences in first and second plots are shown against the sequence of alpha-cuts, in third and fourth plots as a function of alpha values.

Usage

```
## S3 method for class 'rank_stability'
plot(x,
    which = 1:4, legend = TRUE, legend.x = "bottomleft",
    legend.y = NULL, legend.bg = "white", grid = TRUE,
    grid.lty = 2, grid.col = rgb(0, 0, 0, 1/7),
    grid.lwd = 1, y_axis = "reversed", ask = dev.interactive(),
    type = "l", col = gray(1:ncol(x$ranking)/ncol(x$ranking)/1.3),
    lwd = 3, lty = 1, ...
)
```

Arguments

`x`	object of class `rank_stability` generated by function `rank_stability`.
`which`	select a subset of the numbers 1:4, to specify the desired plots. See caption below (and the 'Details').
`legend`	boolean argument to choose whether to show the legend in the plots.
`legend.x, legend.y, legend.bg`	arguments `x`, `y` and `bg` of the function `legend` defining the coordinates and the backgroud color of the legend.
`grid`	boolean argument to choose whether to show the grid in the plots.
`grid.lty, grid.col, grid.lwd`	arguments defining the line type, color and width of the grid.
`y_axis`	if it is set equal to "reversed" plots show the y axis reversed.

`ask`	boolean argument indicating whether the system has to ask users before changing plots.
`type`	1-character string giving the desired type of plot. See `plot.default` for details.
`col`	vector of colors. See `matplot` for details.
`lwd`	vector of line widths. See `matplot` for details.
`lty`	vector of line types. See `matplot` for details.
`...`	other arguments of function `matplot`.

See Also

`rank_stability`, `legend`, `plot.default`, `matplot`

Examples

```
v1 <- as.ordered(c("a", "b", "c", "d"))
v2 <- 1:3
prof <- var2prof(varmod = list(v1 = as.ordered(c("a", "b", "c", "d")), v2 = 1:3))
np <- nrow(prof$profiles)

k <- 10 # number of populations
set.seed(0)
populations <- as.data.frame(lapply(1:k, function(x) round(runif(np)*100)))
rownames(populations) <- rownames(prof$profiles)
names(populations) <- paste0("P", 1:k)

x <- FFOD(profiles = prof, distributions = populations)

res <- rank_stability(x)
plot(res)
```

---

pop2prof *Population to profiles*

Description

Extract the observed profiles and the corresponding frequencies, out of the statistical population.

Usage

```
pop2prof(
  y,
  labtype = c("profiles", "progressive", "rownames"),
  sep = "",
  weights = rep(1, nrow(y))
)
```

Arguments

`y`	a dataset, used to count profile frequencies. See details.
`labtype`	users can choose the type of labels to assign to profiles. See details.
`sep`	variables separator in the profiles labels.
`weights`	a vector of length equal to the number of observations in `y`, representing the survey weights of each observation.

Details

`y` is a data.frame of observations on the ordinal or numeric variables. The partial order must be defined within the object type, so as to build the incidence matrix of the order relation (see `getzeta`).

The function extracts variables and their observed modalities from the population; it builds all possible profiles and assigns to them the corresponding frequency. If some modalities are not observed in the population, they will not be used to build the profiles. If one is interested in the set of all possible profiles from a given set of variables, function `var2prof` is to be used.

Users can choose the label type to assign to profiles. Accepetd types are: `profiles` the variabiles modalities, `progressive` a progressive numeration, `rownames` the rownames in the dataset.

Value

The function returns a S3 class object `wprof`, "weighted profiles", containing the `data.frame` named `profiles` and the frequency vector `freq`.

See Also

`var2prof`, `getzeta`

Examples

```
n <- 5
v1 <- as.ordered(c("a", "b", "c", "d"))
v2 <- 1:3
pop <- data.frame(
    v1 = sample(v1, n, replace = TRUE),
    v2 = sample(v2, n, replace = TRUE)
)
pop2prof(pop)
```

`popelem` *popelem*

Description

The function identifies in a matrix `y`, profiles in `prof`. For each row of matrix `y`, the function returns the location of the corresponding profile in object `prof`.

Usage

```
popelem(prof, ...)
## S3 method for class 'wprof'
popelem(prof, y, ...)
```

Arguments

`prof`	an object of S3 class `wprof`.
`y`	a `matrix` or `data.frame` representing a set of observations with variables (the same contained in `prof`) by columns.
`...`	any of the above.

Examples

```
vl <- c(2, 3, 2)
prf <- var2prof(varlen = vl)
pop <- matrix(c(2, 1, 1, 1, 2, 1, 2, 3, 1), 3, 3)
rownames(pop) <- LETTERS[1:3]

v <- popelem(prof = prf, y = pop)
v
prf$profiles[v,]
```

`proFreq` *Observed profile frequencies*

Description

The function computes profile frequencies, by counting the number of times a profile appears in the population.

Usage

```
proFreq(profiles, population)
```

Arguments

`profiles`	an object of S3 class `wprof`.
`population`	a `matrix` or `data.frame` representing a set of observations with variables (the same contained in `prof`) by columns.

Value

An object of class `wprof` with the same profiles of the argument but with different frequencies.

Author(s)

Alberto Arcagni

See Also

popelem

Examples

```
vl <- c(2, 3, 2)
prf <- var2prof(varlen = vl)
pop <- matrix(c(2, 1, 1, 1, 2, 1, 2, 3, 1), 3, 3)
rownames(pop) <- LETTERS[1:3]

proFreq(profiles = prf, population = pop)
```

`rank_stability` *Rank stability analysis in posetic FOD*

Description

The function computes the average ranks and the positions in the ranking of the elements of the alpha-cuts.

Usage

```
rank_stability(x, ...)
## S3 method for class 'FODposet'
rank_stability(x,
    selection = 1:length(x$covers),
    coverage_probability = 0.9,
    error = 10^(-5), ...
)
```

Arguments

`x`	object of class `FODposet` generated by function `FOD`.
`selection`	numeric vector or a vector of names to select the cover matrices in argument `x`.
`coverage_probability`	least coverage probability of the rank intervals with extremes `lower_ranks` and `upper_ranks`.
`error`	the "distance" from uniformity in the sampling distribution of linear extensions used to evaluate the average ranks. See `idn` for details.
`...`	any of above.

Value

`alpha`	vector of the alpha values defining the alpha-cuts.
`average_ranks`	data frame of average ranks of the poset elements (by columns) at different alpha values (by rows).
`lower_ranks`	data frame of the lower bounds of the rank interval, of each poset element (by columns) at different alpha values (by rows).
`upper_ranks`	data frame of the upper bounds of the rank interval, of each poset element (by columns) at different alpha values (by rows).
`ranking`	data frame of the positions of poset elements (by columns), in the ranking extracted from the posets associated to alpha-cuts (by rows).
`resolution`	number of elements of the posets associated to the alpha-cuts.

Author(s)

Fattore M., Arcagni A.

See Also

`FFOD`, `idn`

Examples

```
v1 <- as.ordered(c("a", "b", "c", "d"))
v2 <- 1:3
prof <- var2prof(varmod = list(v1 = as.ordered(c("a", "b", "c", "d")), v2 = 1:3))
np <- nrow(prof$profiles)

k <- 10 # number of populations
set.seed(0)
populations <- as.data.frame(lapply(1:k, function(x) round(runif(np)*100)))
rownames(populations) <- rownames(prof$profiles)
names(populations) <- paste0("P", 1:k)

x <- FFOD(profiles = prof, distributions = populations)

res <- rank_stability(x)
res
```

`reflexivity` *reflexivity*

Description

The function checks whether the input boolean square matrix `m` represents a reflexive binary relation.

Usage

```
reflexivity(m)
```

Arguments

`m`	a boolean square matrix.

See Also

`transitivity`, `binary`, `antisymmetry`,

`is.preorder`, `is.partialorder`,

`validate.partialorder.incidence`

Examples

```
M <- c(TRUE, FALSE, FALSE, FALSE, TRUE, TRUE, FALSE, FALSE, TRUE, FALSE, TRUE,
FALSE, TRUE, TRUE, TRUE, TRUE)
M <- matrix(M, 4, 4)
rownames(M) <- colnames(M) <- LETTERS[1:4]

reflexivity(M)
```

`rmProfiles` *Removing profiles*

Description

Function to remove profiles from an object of class `wprof`.

Usage

```
rmProfiles(y, ...)
## S3 method for class 'wprof'
rmProfiles(y, v, ...)
```

Arguments

`y`	object of class `wprof`.
`v`	a vector pointing to the profiles to be removed. The vector can be of type: • `numeric` whose components refer to the positions of profiles in `y`; • `logical` of the same length as the number of profiles in `y`; • `character`, referring to profile names in `y`.
`...`	any of the above.

Value

The function returns an `wprof` object equal to `y` but without the profiles in `v`.

Examples

```
vl <- c(3, 3, 3)
prof <- var2prof(varlen = vl)
rownames(prof$profiles)

prof <- rmProfiles(prof, c("123", "321"))

plot(prof)
```

`summary.cover` *Summary method for cover and incidence objects.*

Description

The function computes a summary of cover and incidence S3 objects. Currently, the function returns just the number of profiles and the number of comparabilities.

Usage

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

Arguments

`object`	a cover matrix or an incidence matrix.
`...`	added for consistency with the generic method.

Examples

```
vl <- c(2, 3, 3)
prf <- var2prof(varlen = vl)
Z <- getzeta(prf)
summary(Z)
C <- incidence2cover(Z)
summary(C)
```

`summary.parsec` *Summary of outputs of the evaluation function.*

Description

S3 method of function `summary` reporting main information for an object of class `parsec`, obtained from function `evaluation`. In particular, the function computes a table showing, for each profile:

- the variables' grades identifying the profile (if these are returned by `evaluation`.
- the assigned weight.
- whether or not it belongs to the threshold.
- the corresponding value of the identification function.
- the average poverty rank.
- the different gap measures (see `evaluation` for details).

If the number of profiles is higher than ten, the shown table gets cut, but the method returns a `data.frame` providing the complete output.

Usage

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

Arguments

`object`	an object of S3 class `parsec`, output of the `evaluation` function.
`...`	added for consistency with the generic method.

See Also

`evaluation`

Examples

```
profiles <- var2prof(varlen = c(3, 2, 4))
threshold <- c("311", "112")

res <- evaluation(profiles, threshold, nit = 10^3)

sm <- summary(res)
summary(sm)
```

`transitiveClosure` *Transitive Closure*

Description

The function computes the transitive closure of a reflexive and antisymmetric binary relation.

Usage

```
transitiveClosure(m)
```

Arguments

`m`	a generic square boolean matrix representing a reflexive and antisymmetric binary relation, an object of class `cover` or an object of class `incidence`.

Value

Incidence matrix of the transitive closure of the input matrix `m`.

See Also

`is.partialorder`

Examples

```
m <- c(1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1)
m <- matrix(m, 4, 4)

transitiveClosure(m)
```

`transitivity` *transitivity*

Description

The function checks whether the boolean square matrix `m` representes a transitive binary relation.

Usage

```
transitivity(m)
```

Arguments

`m`	a boolean square matrix.

See Also

binary, reflexivity, antisymmetry,
is.preorder, is.partialorder,
validate.partialorder.incidence

Examples

```
M <- c(TRUE, FALSE, FALSE, FALSE, TRUE, TRUE, FALSE, FALSE, TRUE, FALSE, TRUE,
FALSE, TRUE, TRUE, TRUE, TRUE)
M <- matrix(M, 4, 4)
rownames(M) <- colnames(M) <- LETTERS[1:4]

transitivity(M)
```

`upset` *upset*

Description

The function computes a boolean vector specifying which poset elements belong to the downset generated by subposet `Q`.

Usage

```
upset(z, ...)
## S3 method for class 'cover'
upset(z, ...)
## S3 method for class 'incidence'
upset(z, Q = NULL, ...)
```

Arguments

`z`	a cover or an incidence matrix, of S3 classes `cover` or `incidence`, respectively.
`Q`	vector specifying a subposet of the poset represented by `z`.
`...`	any of the above.

Examples

```
z <- getzeta(var2prof(varlen = c(2, 2, 2)))

plot(z, col = 1 + c(1, 1, 0, 0, 1, 0, 0, 0) + c(0, 0, 0, 2, 0, 0, 2, 2), lwd = 2)

Q <- c(4, 7, 8)
rownames(z)[Q]
upset(z, Q)

Q <- c("211", "112", "111")
upset(z, Q)
```

`validate.partialorder.incidence` *validate.partialorder.incidence*

Description

The function checks whether the boolean square matrix `m` represents a partial order. If yes, the function returns the same input matrix as a S3 class object `incidence`. Otherwise, the unfulfilled partial order properties of matrix `m` are returned.

Usage

```
validate.partialorder.incidence(m)
```

Arguments

`m`	a boolean square matrix.

See Also

`transitivity`, `binary`, `reflexivity`,

`antisymmetry`, `is.preorder`, `is.partialorder`

Examples

```
M <- c(TRUE, FALSE, FALSE, FALSE, TRUE, TRUE, FALSE, FALSE, TRUE, FALSE, TRUE,
FALSE, TRUE, TRUE, TRUE, TRUE)
M <- matrix(M, 4, 4)
rownames(M) <- colnames(M) <- LETTERS[1:4]

M <- validate.partialorder.incidence(M)

plot(M)
```

`var2prof` *Variables to profiles*

Description

The function computes the list of all of the profiles from a list of input ordinal variables. See details for how to define variables.

Usage

```
var2prof(varmod = lapply(as.list(varlen), function(x) 1:x),
    varlen = sapply(varmod, length), freq = NULL,
    labtype = c("profiles", "progressive"), y=NULL)
```

Arguments

`varmod`	list of variables and their grades. See details.
`varlen`	a vector of number of grades of each variable. See details.
`freq`	profiles frequency distribution. By default, the frequencies are set equal to 1.
`labtype`	type of labels to assign to profiles. See details.
`y`	a matrix of observations, used to count profiles frequencies. See details.

Details

Variables can be defined through their names and grades, using a list as argument `varmod`. The names of the objects in the list are taken as variable names. The objects in the list must be ordered vectors or numeric vectors.

A faster way to define variables is through a vector with the number of grades of each variable, as argument `varlen`. This way, variables and their grades are assigned arbitrary names. In particular, grades are identified by their ranks in the variable definition.

The user can choose the type of label to assign to profiles. `profiles` is the combination of grades identifying the profiles. When the names of the grades are too long, it is suggested to choose `progressive`.

`y` is a matrix of observations on the ordinal variables (observations by rows and variables by columns). Variables must be ordered as defined in the previous arguments. The names of variable grades must match their definition. By this argument, the function counts the number of times a profile is observed in the population, assigning the result to the `freq` output. This method should be used when the variables and their grades are known, otherwise the function `pop2prof` is available.

Value

The function returns a S3 class object `wprof`, "weighted profiles", comprising the `data.frame` `profiles` and the vector of frequencies `freq`.

See Also

`pop2prof`, `getzeta`

Examples

```
# 2 variables with 2 modalities, frequencies detected from population
pop <- matrix(sample(1:2, 100, replace=TRUE), 50, 2)
var2prof(varlen=c(2, 2))

# 2 variables:
# - mood: 2 modalities
# - weather: 3 modalities
# 2*3 profiles and frequencies sampled from a Binomial distribution n = 10, p = 0.5
var <- list(
    mood = ordered(c("bad", "good"), levels = c("bad", "good")),
   weather = ordered(c("rainy", "cloudy", "sunny"), levels = c("rainy", "cloudy", "sunny"))
)
var2prof(var, freq = rbinom(2*3, 10, 0.5), labtype = "progressive")
```

`vertices` *Coordinates of the vertices of the Hasse diagram, representing the input cover relation.*

Description

The function computes the coordinates of the vertices of the Hasse diagram.

Usage

```
vertices(C, shape = c("square", "circle", "equispaced"), noise = FALSE)
```

Arguments

`C`	cover matrix, an object of class S3 `cover`.
`shape`	shape of the diagram. See details.
`noise`	some jittering on the x axis, so as to improve readability. Values can be boolean or positive values, to get different jittering intensities.

Details

Possible Hasse diagram shapes: `square`; `circle`; `equispaced`. The last option is suggested when the poset has more than one maximal or minimal elements. The function is used by the plot methods defined in the package (see `plot.cover`).

See Also

`plot.cover`

Index