Package 'deducorrect'

July 15, 2015

Maintainer Mark van der Loo <mark.vanderloo@gmail.com>

License GPL-3

Title Deductive Correction, Deductive Imputation, and Deterministic Correction

Type Package

LazyLoad yes

Author Mark van der Loo, Edwin de Jonge, and Sander Scholtus

Description A collection of methods for automated data cleaning where all actions are logged.

Version 1.3.7

URL https://github.com/data-cleaning/deducorrect

Date 2015-07-15

Depends editrules (>= 2.9.0)

Imports stats, utils, methods

NeedsCompilation no

Repository CRAN

Date/Publication 2015-07-15 13:23:47

R topics documented:

deducorrect-package 2
correctionRules 3
correctRounding 4
correctSigns 7
correctTypos 10
correctWithRules 12
damerauLevenshteinDistance 14
deducorrect-object 14
deductiveLevels 15
deductiveZeros 17
deduImpute 19

getSignCorrection 22
hellerTompkins 23
imputess 23
isTotallyUnimodular 26
newdeducorrect 27
raghavachari 28
reduceMatrix 29
solSpace 29
status 32

Index **34**

`deducorrect-package` *Deductive correction methods for sign, rounding and typing errors*

Description

Deductive correction methods for sign, rounding and typing errors It also contains functionality to check if a matrix of linear restrictions is totally unimodular. See also

Details

- `correctRounding`
- `correctTypos`
- `deducorrect-object` and `status` for output specification
- `isTotallyUnimodular`
- `deduImpute`
- `solSpace`, `imputess`
- `deductiveZeros`
- `deductiveLevels`

Examples

```
require(editrules)

# some data
dat <- data.frame(
    x = c( 3, 14, 15),
    y = c(13, -4,  5),
    z = c(10, 10,-10))

dat
# ... which has to obey
E <- editmatrix("z == x-y")
```

```
# All signs may be flipped, no swaps.
 correctSigns(E, dat)

correctTypos(E, dat)

correctRounding(E, dat)
```

---

correctionRules *Rules for deterministic correction*

Description

Rules for deterministic correction

Usage

```
correctionRules(x, strict = TRUE, allowed = getOption("allowedSymbols"),
  ...)

## S3 method for class 'character'
correctionRules(x, strict = TRUE,
  allowed = getOption("allowedSymbols"), file = TRUE, ...)

## S3 method for class 'expression'
correctionRules(x, strict = TRUE,
  allowed = getOption("allowedSymbols"), ...)

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

## S3 method for class 'correctionRules'
as.character(x, oneliner = FALSE, ...)

## S3 method for class 'correctionRules'
getVars(E, ...)
```

Arguments

x	character or expression vector.
strict	If TRUE an error is thrown if any forbidden symbol is used (see details).
allowed	A character vector of allowed symbols
...	Currently unused.
file	If file=TRUE, x is treated as a filename from which the rules are read.
oneliner	Coerce to oneliner
E	object of class correctionRules

Value

`correctionRules` returns an object of class `correctionRules`

`getVars` returns a character vector of variable names.

Details

Data editing processes are rarely completely governed by in-record consistency rules. Many *ad-hoc* rules are commonly used to impute empty or erroneous values. Such rules are often applied manually or hidden in source code. This function, together with `correctWithRules` allows for easy definition and execution of simle deterministic replacement rules.

These functions are ment to support very simple rules, such as *if variable x is missing, then set it to zero*. Such actions usually basically model-free corrections stemming from subject-matter knowledge. Given the nature of such rules, the type of rules are by default limited to R-statements containing conditionals (`if-else`), arithmetic and logical operators, and brackets and assignment operators. see `getOption('allowedSymbols')` for a complete list.

If you cannot execute your 'simple' corrections with just these functions, we strongly recommend to write a separate imputation or correction routine. However, it's a free world, so you may alter the list of allowed symbols as you wish.

Note

`getVars` is overloaded from the `editrules` package.

See Also

`correctWithRules`

`correctRounding` *Correct records under linear restrictions for rounding errors*

Description

This algorithm tries to detect and repair records that violate linear (in)equality constraints by correcting possible rounding errors as described by Scholtus(2008). Typically data is constrainted by $Rx = a$ and $Qx \geq b$.

Usage

```
correctRounding(E, dat, ...)

## S3 method for class 'editset'
correctRounding(E, dat, ...)

## S3 method for class 'editmatrix'
correctRounding(E, dat, fixate = NULL, delta = 2,
  K = 10, round = TRUE, assumeUnimodularity = FALSE, ...)
```

Arguments

E	editmatrix or editset as generated by the editrules package.
dat	data.frame with the data to be corrected
...	arguments to be passed to other methods.
fixate	character with variable names that should not be changed.
delta	tolerance on checking for rounding error
K	number of trials per record. See details
round	should the solution be rounded, default TRUE
assumeUnimodularity	If FALSE, a test is performed before corrections are computed (expensive).

Details

The algorithm first finds violated constraints $|r_i'x - a_i| > 0$, and selects edits that may be due to a rounding error $0 < |r_i'x - a_i| \leq \delta$. The algorithm then makes a correction suggestion where the errors are attributed to randomly selected variables under the lineair equality constraints. It checks if the suggested correction does not violate the inequality matrix Q. If it does, it will try to generate a different solution up till K times.

Value

A deducorrrect object.

References

Scholtus S (2008). Algorithms for correcting some obvious inconsistencies and rounding errors in business survey data. Technical Report 08015, Statistics Netherlands.

See Also

deducorrect-object status

Examples

```
E <- editmatrix(expression(
    x1 + x2 == x3,
    x2 == x4,
    x5 + x6  + x7 == x8,
    x3 + x8 == x9,
    x9 - x10 == x11
    )
)

dat <- data.frame( x1=12
                 , x2=4
                 , x3=15
                 , x4=4
```

```
                  , x5=3
                  , x6=1
                  , x7=8
                  , x8=11
                  , x9=27
                  , x10=41
                  , x11=-13
                  )

sol <- correctRounding(E, dat)


# example with editset
for ( d in dir("../pkg/R/",full.names=TRUE) ) dmp <- source(d)
E <- editmatrix(expression(
    x + y == z,
    x >= 0,
    y >= 0,
    z >= 0,
    if ( x > 0 ) y > 0
    ))
dat <- data.frame(
    x = 1,
    y = 0,
    z = 1)
# solutions causing new violations of conditional rules are rejected
sol <- correctRounding(E,dat)

# An example with editset
E <- editset(expression(
    x + y == z,
    x >= 0,
    y > 0,
    y < 2,
    z > 1,
    z < 3,
    A %in% c('a','b'),
    B %in% c('c','d'),
    if ( A == 'a' ) B == 'b',
    if ( B == 'b' ) x < 1
))
dat <- data.frame(
    x = 0,
    y = 1,
    z = 2,
    A = 'a',
    B = 'b'
)

correctRounding(E,dat)
```

correctSigns *Correct sign errors and value interchanges in data records*

Description

Correct sign errors and value interchanges in data records.

Usage

```
correctSigns(E, dat, ...)

## S3 method for class 'editset'
correctSigns(E, dat, ...)

## S3 method for class 'editmatrix'
correctSigns(E, dat, flip = getVars(E), swap = list(),
  maxActions = length(flip) + length(swap), maxCombinations = 1e+05,
  eps = sqrt(.Machine$double.eps), weight = rep(1, length(flip) +
  length(swap)), fixate = NA, ...)
```

Arguments

E	An object of class editmatrix
dat	data.frame, the records to correct.
...	arguments to be passed to other methods.
flip	A character vector of variable names who's values may be sign-flipped
swap	A list of character 2-vectors of variable combinations who's values may be swapped
maxActions	The maximum number of flips and swaps that may be performed
maxCombinations	The number of possible flip/swap combinations in each step of the algorithm is choose(n,k), with n the number of flips+swaps, and k the number of actions taken in that step. If choose(n,k) exceeds maxCombinations, the algorithm returns a record uncorrected.
eps	Tolerance to check equalities against. Use this to account for sign errors masked by rounding errors.
weight	weight vector. Weights can be assigned either to actions (flips and swap) or to variables. If length(weight)==length(flip)+length(swap), weights are assiged to actions, if length(weight)==ncol(E), weights are assigned to variables. In the first case, the first length{flip} weights correspond to flips, the rest to swaps. A warning is issued in the second case when the weight vector is not named. See the examples for more details.
fixate	a character vector with names of variables whos values may not be changed

Details

This algorithm tries to correct records violating linear equalities by sign flipping and/or value interchanges. Linear inequalities are taken into account when judging possible solutions. If one or more inequality restriction is violated, the solution is rejected. It is important to note that the `status` of a record has the following meaning:

`valid`	The record obeys all equality constraints on entry. No error correction is performed. It may therefore still contain inequality errors.
`corrected`	Equality errors were found, and all of them are solved without violating inequalities.
`partial`	Does not occur
`invalid`	The record contains equality violations which could not be solved with this algorithm
`NA`	record could not be checked. It contained missings.

The algorithm applies all combinations of (user-allowed) flip- and swap combinations to find a solution, and minimizes the number of actions (flips+swaps) that have to be taken to correct a record. When multiple solutions are found, the solution of minimal weight is chosen. The user may provide a weight vector with weights for every flip and every swap, or a named weight vector with a weight for every variable. If the weights do not single out a solution, the first one found is chosen.

If arguments `flip` or `swap` contain a variable not in `E`, these variables will be ignored by the algorithm.

Value

a `deducorrect-object`. The `status` slot has the following columns for every records in `dat`.

`status`	a `status` factor, showing the status of the treated record.
`degeneracy`	the number of solutions found, *after* applying the weight
`weight`	the weight of the chosen solution
`nflip`	the number of applied sign flips
`nswap`	the number of applied value interchanges

References

Scholtus S (2008). Algorithms for correcting some obvious inconsistencies and rounding errors in business survey data. Technical Report 08015, Netherlands.

See Also

`deducorrect-object`

Examples

```
# some data
dat <- data.frame(
    x = c( 3,14,15,  1, 17,12.3),
    y = c(13,-4, 5,  2,  7, -2.1),
    z = c(10,10,-10, NA,10,10 ))
```

```
# ... which has to obey
E <- editmatrix(c("z == x-y"))

# All signs may be flipped, no swaps.

correctSigns(E, dat)

# Allow for rounding errors
correctSigns(E, dat, eps=2)

# Limit the number of combinations that may be tested
correctSigns(E, dat, maxCombinations=2)

# fix z, flip everything else
correctSigns(E, dat,fixate="z")

# the same result is achieved with
correctSigns(E, dat, flip=c("x","y"))

# make x and y swappable, allow no flips
correctSigns(E, dat, flip=c(), swap=list(c("x","y")))

# make x and y swappable, swap a counts as one flip
correctSigns(E, dat, flip="z", swap=list(c("x","y")))

# same, but now, swapping is preferred (has lower weight)
correctSigns(E, dat, flip="z", swap=list(c("x","y")), weight=c(2,1))

# same, but now becayse x any y carry lower weight. Also allow for rounding errors
correctSigns(E, dat, flip="z", swap=list(c("x","y")), eps=2, weight=c(x=1, y=1, z=3))

# demand that solution has y>0
E <- editmatrix(c("z==x-y", "y>0"))
correctSigns(E,dat)

# demand that solution has y>0, taking acount of roundings in equalities
correctSigns(E,dat,eps=2)

# example with editset
E <- editset(expression(
    x + y == z,
    x >= 0,
    y > 0,
    y < 2,
    z > 1,
    z < 3,
    A %in% c('a','b'),
    B %in% c('c','d'),
    if ( A == 'a' ) B == 'b',
    if ( B == 'b' ) x < 1
))

x <- data.frame(
```

```
    x = -1,
    y = 1,
    z = 2,
    A = 'a',
    B = 'b'
)

correctSigns(E,x)
```

`correctTypos` *Correct records under linear restrictions using typographical error suggestions*

Description

This algorithm tries to detect and repair records that violate linear equality constraints by correcting simple typo's as described in Scholtus (2009). The implemention of the detection of typing errors differs in that it uses the restricted Damerau-Levensthein distance. Furthermore it solves a broader class of problems: the original paper describes the class of equalities: $Ex = 0$ (balance edits) and this implementation allows for $Ex = a$.

Usage

```
correctTypos(E, dat, ...)

## S3 method for class 'editset'
correctTypos(E, dat, ...)

## S3 method for class 'editmatrix'
correctTypos(E, dat, fixate = NULL, cost = c(1, 1, 1,
  1), eps = sqrt(.Machine$double.eps), maxdist = 1, ...)
```

Arguments

`E`	`editmatrix` or `editset`
`dat`	`data.frame` with data to be corrected.
`...`	arguments to be passed to other methods.
`fixate`	`character` with variable names that should not be changed.
`cost`	for a deletion, insertion, substition or transposition.
`eps`	`numeric`, tolerance on edit check. Default value is `sqrt(.Machine$double.eps)`. Set to 2 to allow for rounding errors. Set this parameter to 0 for exact checking.
`maxdist`	`numeric`, tolerance used in finding typographical corrections. Default value 1 allows for one error. Used in combination with `cost`.

Details

For each row in `dat` the correction algorithm first detects if row `x` violates the equality constraints of `E` taking possible rounding errors into account. Mathematically: $|\sum_{i=1}^{n} E_{ji}x_i - a_j| \leq \varepsilon, \quad \forall j$

It then generates correction suggestions by deriving alternative values for variables only involved in the violated edits. The correction suggestions must be within a typographical edit distance (default = 1) to be selected. If there are more then 1 solutions possible the algorithm tries to derive a partial solution, otherwise the solution is applied to the data.

`correctTypos` returns an object of class `deducorrect` object describing the status of the record and the corrections that have been applied.

Inequalities in editmatrix `E` will be ignored in this algorithm, so if this is the case, the corrected records are valid according to the equality restrictions, but may be incorrect for the given inequalities.

Please note that if the returned status of a record is "partial" the corrected record still is not valid. The partially corrected record will contain less errors and will violate less constraints. Also note that the status "valid" and "corrected" have to be interpreted in combination with `eps`. A common scenario is first to correct for typo's and then correct for rounding errors. This means that in the first step the algorithm should allow for typo's (e.g. `eps==2`). The returned "valid" record therefore may still contain rounding errors.

Value

`deducorrect` object with corrected data.frame, applied corrections and status of the records.

References

Scholtus S (2009). Automatic correction of simple typing errors in numerical data with balance edits. Discussion paper 09046, Statistics Netherlands, The Hague/Heerlen.

Damerau F (1964). A technique for computer detection and correction of spelling errors. Communications of the ACM, 7,issue 3

Levenshtein VI (1966). Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10: 707-10

A good description of the restricted DL-distance can be found on wikipedia: http://en.wikipedia.org/wiki/Damerau

See Also

`damerauLevenshteinDistance`

Examples

```
library(editrules)

# example from section 4 in Scholtus (2009)

E <- editmatrix( c("x1 + x2 == x3"
                  ,"x2 == x4"
                  ,"x5 + x6 + x7 == x8"
                  ,"x3 + x8 == x9"
```

```
                  ,"x9 - x10 == x11"
                  )
               )

dat <- read.csv(txt<-textConnection(
"    , x1, x2 , x3  , x4 , x5 , x6, x7, x8 , x9   , x10 , x11
4  , 1452, 116, 1568, 116, 323, 76, 12, 411,  1979, 1842, 137
4.1, 1452, 116, 1568, 161, 323, 76, 12, 411,  1979, 1842, 137
4.2, 1452, 116, 1568, 161, 323, 76, 12, 411, 19979, 1842, 137
4.3, 1452, 116, 1568, 161,   0,  0,  0, 411, 19979, 1842, 137
4.4, 1452, 116, 1568, 161, 323, 76, 12,   0, 19979, 1842, 137"
))
close(txt)
(cor <- correctTypos(E,dat))



# example with editset
E <- editset(expression(
    x + y == z,
    x >= 0,
    y > 0,
    y < 2,
    z > 1,
    z < 3,
    A %in% c('a','b'),
    B %in% c('c','d'),
    if ( A == 'a' ) B == 'b',
    if ( B == 'b' ) x > 3
))

x <- data.frame(
    x = 10,
    y = 1,
    z = 2,
    A = 'a',
    B = 'b'
)

correctTypos(E,x)
```

`correctWithRules` *Deterministic correction*

Description

Apply simple replacement rules to a `data.frame`.

Usage

```
correctWithRules(rules, dat, strict = TRUE)
```

Arguments

rules	object of class correctionRules
dat	data.frame
strict	If TRUE, an error is produced when the rules use variables other than in the data.frame.

Value

list with altered data ($corrected) and a list of alterations ($corrections).

Details

This function applies the the rules one by one to dat and logs their actions. Rules are excuted in order of occurrence in the correctionRules so order may matter for the final result. Rules are applied to one record at the time, so the use of statistical funtions such as mean is useless, and forbidden by default. See correctionRules for details on the type of rules that are possible.

See Also

correctionRules

Examples

```
## Some example data
dat <- data.frame(
   x = c(NA, 2, 0,-10),
   y = c(1, NA,NA, 6)
)

## a few rules
u <- correctionRules(expression(
   if ( is.na(x) ) x <- 0,
   if ( x == 0 && is.na(y) ) y <- 0,
   if ( is.na(y) ) y <- 1,
   if ( x < 0 ) y <- 0
))

correctWithRules(u,dat)
```

`damerauLevenshteinDistance` *Calculate the Damerau Levenshtein Distance between two strings*

Description

The restricted Damerau Levenshtein Distance between two strings is commonly used for checking typographical errors in strings. It takes the deletion and insertion of a character, a wrong character (substition) or the swapping (transposition) of two characters into account. By default these operations each account for distance 1.

Usage

```
damerauLevenshteinDistance(sa, sb, w = c(1, 1, 1, 1))
```

Arguments

`sa`	character vector
`sb`	character vector of equal `length(sa)`
`w`	integer vector for cost of deletion, insertion, substitution and transposition.

Value

integer vector with pairwise edit distances

References

Damerau F (1964). A technique for computer detection and correction of spelling errors. Communications of the ACM, 7,issue 3

Levenshtein VI (1966). Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10: 707-10 Damerau Levenshtein Distance calculates the difference between two strings used for typographical errors (typo's)

`deducorrect-object` *deducorrect object*

Description

deducorrect object

Details

All `correct*` functions of the deducorrect package return an S3 object of class `deducorrect`. The "public slots" are

- `corrected` A copy of the input `data.frame`, with records corrected where possible.
- `corrections` A data.frame with the following colums:
  - `row` Row number where correction was applied
  - `variable` Variable name where correction was applied
  - `old` Old value of adapted variable
  - `new` New value of adapted variable
- `status` A `data.frame` with the same number of rows as `corrected`. It has at least a column called `status`. Further columns might be present, depending on the used correction function.
- `generatedby` The name of the function that called `newdeducorrect` to construct the object.
- `timestamp` The time and date when the object was generated, as returned by `date`.
- `user` The system's username of the user running R. Note that this may yield unexpected results when R accessed on a remote (web)server.

`deductiveLevels` *Derive imputation values for categorical data*

Description

Deduce imputation values for categorical data. By substituting all known values and interatively eliminating the unknowns from the set of edits, unique imputation values are derived where possible.

Usage

```
deductiveLevels(E, x, adapt = rep(FALSE, length(x)),
  checkFeasibility = TRUE)
```

Arguments

`E`	editarray
`x`	a named `character` vector
`adapt`	boolean vector indicating which variables may be adapted.
`checkFeasibility`	Test wether the assumed-correct values (observed and not designated by adapt) can lead to a consistent record.

Details

Imputation values are derived for missing variables (NA) and for variables indicated by 'adapt'.

Value

A named vector with imputation values for x

References

T. De Waal, J. Pannekoek and S. Scholtus (2011) Handbook of statistical data editing Chpt 9.2.1 - 9.2.2

Examples

```
# a simple example. We know the subject is pregnant. What is the gender?
E <- editarray(c(
    "gender \%in\% c('male','female')",
    "pregnant \%in\% c(TRUE,FALSE)",
    "if ( gender=='male') !pregnant"))
# a record with unknown gender
x <- c(gender=NA,pregnant=TRUE)

# find imputations
(s <- deductiveLevels(E,x))

# imputation can be done as follows:
x[names(s)] <- s



# Here's an example from De Waal et al (2011) (ex. 9.3)
E <- editarray(c(
    "x1 \%in\% letters[1:4]",
    "x2 \%in\% letters[1:3]",
    "x3 \%in\% letters[1:3]",
    "x4 \%in\% letters[1:2]",
    "if (x2 == 'c'  & x3 != 'c' & x4 == 'a' ) FALSE",
    "if (x2 != 'a'  & x4 == 'b') FALSE",
    "if (x1 != 'c'  & x2 != 'b' & x3 != 'a') FALSE",
    "if (x1 == 'c'  & x3 != 'a' & x4 == 'a' ) FALSE"
))

x <- c(x1='c',x2='b',x3=NA,x4=NA)
(s <- deductiveLevels(E,x))

x[names(s)] <- s


# another example, partial imputation
y <- c(x1=NA,x2=NA,x3=NA,x4='b')
(s <- deductiveLevels(E,y))

y[names(s)] <- s
```

deductiveZeros *Find out which variables can deductively be imputed with 0*

Description

Find out which variables can deductively be imputed with 0

Interface for deductiveZeros for objects of class editmatrix. This interface is robust for variables in x not occuring in E.

Suppose x is a record under linear constraints $Ax = b$ and $x \geq 0$. In certain cases some missing values can be imputed uniquely with zeros. For example, in the case that $x_1 + x_2 = x_3$, if x_2 is missing and $x_1 = x_3 \geq 0$, then x_2 is uniquely determined to be 0. This function returns a boolean vector indicating which of the missing values are uniquely determined to be zero.

Usage

```
deductiveZeros(E, x, ...)

## S3 method for class 'editmatrix'
deductiveZeros(E, x, ...)

## S3 method for class 'matrix'
deductiveZeros(E, x, b, adapt = logical(length(x)),
  nonneg = rep(TRUE, length(x)), roundNearZeros = TRUE,
  tol = sqrt(.Machine$double.eps), ...)
```

Arguments

E	editmatrix or Equality constraint matrix.
x	named numeric vector. Naming is optional if E is an equality constraint matrix.
...	extra parameters to pass to deductiveZeros,matrix
b	Equality constraint constant vector
adapt	logical vector. Extra values to adapt, order must be the same as in x
nonneg	logical vector of length(x). Determines which x-values have to obey nonnegativity constraints.
roundNearZeros	Round near zero values of A before determining the sign?
tol	tolerance used for zero-rounding.

Details

There is some added flexibility. Users my define 'extra missings' by specifying the adapt vector.

By default it is assumed that all values must obey the nonnegativity constraint. However this can be determined by specifying nonneg.

See Also

deduImpute, solSpace

Examples

```
# a simple example"
E <- editmatrix(c(
    "x1 + x2 + x3 == xt",
    "x1 >= 0", "x2>=0","x3>=0", "xt>=0"))
x <- c(x1=10,x2=NA,x3=5,xt=15)

# with deductiveZeros we get:
( I <- deductiveZeros(E,x) )
x[I] <- 0

any(violatedEdits(E,x))

# This example is taken from De Waal et al (2011) (Examples 9.1-9.2)
E <- editmatrix(c(
    "x1 + x2      == x3",
    "x2           == x4",
    "x5 + x6 + x7 == x8",
    "x3 + x8      == x9",
    "x9 - x10     == x11",
    "x6 >= 0",
    "x7 >= 0"
))

x <- c(
  x1 = 145,
  x2 = NA,
  x3 = 155,
  x4 = NA,
  x5 = 86,
  x6 = NA,
  x7 = NA,
  x8 = 86,
  x9 = NA,
  x10 = 217,
  x11 = NA)

# determine zeros:
I <- deductiveZeros(E,x)
# impute:
x[I] <- 0
```

deduImpute *Deductive imputation of numerical or categorical values*

Description

Based on observed values and edit rules, impute as many variables deductively as possible.

If `E` is an `editset`, imputation based on numerical rules (if any) is performed, and imputations violating extra edits are reverted. Next, this procedure is repeated for pure categorical rules. The results are combined and returned in a `deducorrect` object.

For categorical data: The funcion `deductiveLevels` is used to derive deductive imputations for as many fields as possible

For numerical data: Given (equality) rules and a number of values to impute or adapt, in some cases unique solutions can be derived. This function uses `solSpace` and `deductiveZeros` (iteratively) to determine which values can be imputed deductively. Solutions causing new violations of (in)equality rules are rejected by default by testing if the observed values can lead to a feasible record. This may be switched off by passing `checkFeasibility=FALSE`. This may be desirable for performance reasons. If `adapt` was computed with an error localization algorithm, such as `editrules::localizeErrors`, the feasibility check is also not nessecary.

Usage

```
deduImpute(E, dat, adapt = NULL, ...)

## S3 method for class 'editset'
deduImpute(E, dat, adapt = NULL, ...)

## S3 method for class 'editarray'
deduImpute(E, dat, adapt = NULL, ...)

## S3 method for class 'editmatrix'
deduImpute(E, dat, adapt = NULL,
  tol = sqrt(.Machine$double.eps), round = TRUE, ...)
```

Arguments

`E`	An `editmatrix` or `editarray`
`dat`	A `data.frame`
`adapt`	(optional) A boolean array of dim(dat), e.g. the result editrules::localizeErrors(E,dat). Column names must match those of `dat`.
`...`	arguments to be passed to `solSpace` (numerical data) or `deductiveLevels` (categorical data)
`tol`	tolerance to use in `solSpace` and in `deductiveZeros`
`round`	should the result be rounded?

Value

A deducorrect-object

Note

When adapt is not NULL, values in dat where adapt==TRUE are replaced with NA. The output may therefore contain missings at positions that were previously filled (with wrong values, according to adapt).

References

T. De Waal, J. Pannekoek and S. Scholtus (2011) Handbook of statistical data editing Chpt 9.2.1 - 9.2.2

See Also

deductiveZeros, solSpace, deductiveLevels

Examples

```
############################################
# IMPUTATION OF NUMERIC DATA
############################################

# These examples are taken from De Waal et al (2011) (Examples 9.1-9.2)
E <- editmatrix(c(
    "x1 + x2      == x3",
    "x2           == x4",
    "x5 + x6 + x7 == x8",
    "x3 + x8      == x9",
    "x9 - x10     == x11",
    "x6 >= 0",
    "x7 >= 0"
))


dat <- data.frame(
    x1=c(145,145),
    x2=c(NA,NA),
    x3=c(155,155),
    x4=c(NA,NA),
    x5=c(NA, 86),
    x6=c(NA,NA),
    x7=c(NA,NA),
    x8=c(86,86),
    x9=c(NA,NA),
    x10=c(217,217),
    x11=c(NA,NA)
)
```

```
dat

d <- deduImpute(E,dat)
d$corrected
d$status
d$corrections



##############################################
# IMPUTATION OF CATEGORICAL DATA
##############################################


# Here's an example from Katrika (2001) [but see De Waal et al (2011), ex. 9.3)]
E <- editarray(c(
    "x1 \%in\% letters[1:4]",
    "x2 \%in\% letters[1:3]",
    "x3 \%in\% letters[1:3]",
    "x4 \%in\% letters[1:2]",
    "if (x2 == 'c'  & x3 != 'c' & x4 == 'a' ) FALSE",
    "if (x2 != 'a'  & x4 == 'b') FALSE",
    "if (x1 != 'c'  & x2 != 'b' & x3 != 'a') FALSE",
    "if (x1 == 'c'  & x3 != 'a' & x4 == 'a' ) FALSE"
))


dat <- data.frame(
    x1 = c('c', NA ),
    x2 = c('b', NA ),
    x3 = c(NA , NA ),
    x4 = c(NA , 'b'),
    stringsAsFactors=FALSE)


s <- deduImpute(E,dat)
s$corrected
s$status
s$corrections


E <- editset(expression(
    x + y == z,
    x >= 0,
    A %in% c('a','b'),
    B %in% c('c','d'),
    if ( A == 'a' ) B == 'b',
    if ( B == 'b' ) x > 0
))

x <- data.frame(
    x = NA,
```

```
    y = 1,
    z = 1,
    A = 'a',
    B = NA
)
# deduImpute will impute x=0 and B='b',which violates the
# last edit. Hence, imputation will be reverted.
deduImpute(E,x)
```

`getSignCorrection` *workhorse for correctSigns*

Description

Workhorse for correctSigns

Usage

```
getSignCorrection(r, A1, C1, eps, A2, C2, epsvec, flip, swap, w, swapIsOneFlip,
  maxActions, maxCombinations)
```

Arguments

`r`	The numerical record to correct
`A1`	Equality matrix
`C1`	Constant vector for equalities
`eps`	Tolerance for equality-checking
`A2`	Inequality matrix
`C2`	Constant vector for inequalities
`epsvec`	Vector to check against. (`.Machine.double.eps` for > inequalities, otherwise 0.)
`flip`	indices in `r`, where r may be sign-flipped
`swap`	$n \times 2$ matrix with indices in `r`, each row indicating a possible value swap.
`w`	weight vector of `length(flips)+nrow(swaps)` if `swapIsOneFlip==TRUE`, otherwise of `length(r)`
`swapIsOneFlip`	logical. If `TRUE`, weights are assigned to each flip or swap action. If `FALSE` weights are assigned to every changed variable.
`maxActions`	Maximum number of `flips+swaps` to try.
`maxCombinations`	the maximum number of flip/swap combinations to try. See the description in `correctSigns`

Value

a list containing

S	n x length(r) array with corrected versions of r
weight	vector of length n with total weight for each solution
nFlip	number of sign flips for every solution
nSwap	number of value swaps for every solution

See Also

correctSigns

`hellerTompkins` *Determine if a matrix is totally unimodular using Heller and Tompkins criterium.*

Description

This function is deducorrect internal

Usage

```
hellerTompkins(A)
```

Arguments

A	An object of class matrix in $\{-1, 0, 1\}^{m \times n}$. Each column must have exactly 2 nonzero elements. (This is tested by isTotallyUnimodular).

Value

TRUE if matrix is unimodular, otherwise FALSE

See Also

isTotallyUnimodular

`imputess` *Impute values from solution space*

Description

Given a record x with observerd x_{obs} and missing values x_{miss} under linear equality constraints $Ax = b$. The function solSpace returns the solution space which can be written as $x_{miss} = x_0 + Cz$, where x_0 is are a constant vector (of dimension d=length(x_{miss})) and C a constant matrix of dimension $d \times d$.

Usage

```
imputess(x, x0, C, z = NULL, tol = sqrt(.Machine$double.eps))
```

Arguments

x	(named) numerical vector to be imputed
x0	x0 outcome of solSpace
C	C outcome of solSpace
z	real vector of dimension ncol(C).
tol	tolerance used to check which rows of C equal zero.

Details

If C has rows equal to zero, then those missing values may be imputed deductively. For the other missing values, some z must be chosen or another imputation method used.

The function imputess imputes missing values in a vector x, based on the solution space and some chosen vector z. If no z is passed as argument, only deductive imputations are performend (i.e. some missings may be left).

If C is a named matrix (as returned by solSpace), rows of $x0$ and C are matched by name to x. Otherwise it is assumed that the missings in x occur in the order of the rows in C (which is also the case when x0 and C are computed by solSpace).

Examples

```
############################################
# IMPUTATION OF NUMERIC DATA
############################################

# These examples are taken from De Waal et al (2011) (Examples 9.1-9.2)
E <- editmatrix(c(
    "x1 + x2      == x3",
    "x2           == x4",
    "x5 + x6 + x7 == x8",
    "x3 + x8      == x9",
    "x9 - x10     == x11",
    "x6 >= 0",
    "x7 >= 0"
))


dat <- data.frame(
    x1=c(145,145),
    x2=c(NA,NA),
    x3=c(155,155),
    x4=c(NA,NA),
    x5=c(NA, 86),
    x6=c(NA,NA),
    x7=c(NA,NA),
```

```
    x8=c(86,86),
    x9=c(NA,NA),
    x10=c(217,217),
    x11=c(NA,NA)
)

dat

d <- deduImpute(E,dat)
d$corrected
d$status
d$corrections




##############################################
# IMPUTATION OF CATEGORICAL DATA
##############################################


# Here's an example from Katrika (2001) [but see De Waal et al (2011), ex. 9.3)]
E <- editarray(c(
    "x1 \%in\% letters[1:4]",
    "x2 \%in\% letters[1:3]",
    "x3 \%in\% letters[1:3]",
    "x4 \%in\% letters[1:2]",
    "if (x2 == 'c'  & x3 != 'c' & x4 == 'a' ) FALSE",
    "if (x2 != 'a'  & x4 == 'b') FALSE",
    "if (x1 != 'c'  & x2 != 'b' & x3 != 'a') FALSE",
    "if (x1 == 'c'  & x3 != 'a' & x4 == 'a' ) FALSE"
))


dat <- data.frame(
    x1 = c('c', NA ),
    x2 = c('b', NA ),
    x3 = c(NA , NA ),
    x4 = c(NA , 'b'),
    stringsAsFactors=FALSE)


s <- deduImpute(E,dat)
s$corrected
s$status
s$corrections


E <- editset(expression(
    x + y == z,
    x >= 0,
    A %in% c('a','b'),
    B %in% c('c','d'),
```

```
    if ( A == 'a' ) B == 'b',
    if ( B == 'b' ) x > 0
))

x <- data.frame(
    x = NA,
    y = 1,
    z = 1,
    A = 'a',
    B = NA
)
# deduImpute will impute x=0 and B='b',which violates the
# last edit. Hence, imputation will be reverted.
deduImpute(E,x)
```

`isTotallyUnimodular` *Test for total unimodularity of a matrix.*

Description

Check wether a matrix is totally unimodular.

Usage

```
isTotallyUnimodular(A)
```

Arguments

`A`	An object of class `matrix`.

Details

A matrix for which the determinant of every square submatrix equals -1, 0 or 1 is called *totally unimodular*. This function tests if a matrix with coefficients in $\{-1, 0, 1\}$ is totally unimodular. It tries to reduce the matrix using the reduction method described in Scholtus (2008). Next, a test based on Heller and Tompkins (1956) or Raghavachari is performed.

Value

logical

References

Heller I and Tompkins CB (1956). An extension of a theorem of Danttzig's In kuhn HW and Tucker AW (eds.), pp. 247-254. Princeton University Press.

Raghavachari M (1976). A constructive method to recognize the total unimodularity of a matrix. _Zeitschrift fur operations research_, *20*, pp. 59-61.

Scholtus S (2008). Algorithms for correcting some obvious inconsistencies and rounding errors in business survey data. Technical Report 08015, Netherlands.

Examples

```
# Totally unimodular matrix, reduces to nothing
A <- matrix(c(
 1,1,0,0,0,
 -1,0,0,1,0,
 0,0,01,1,0,
 0,0,0,-1,1),nrow=5)
isTotallyUnimodular(A)

# Totally unimodular matrix, by Heller-Tompson criterium
A <- matrix(c(
 1,1,0,0,
 0,0,1,1,
 1,0,1,0,
 0,1,0,1),nrow=4)
isTotallyUnimodular(A)

# Totally unimodular matrix, by Raghavachani recursive criterium
A <- matrix(c(
    1,1,1,
    1,1,0,
    1,0,1))
isTotallyUnimodular(A)
```

`newdeducorrect` *Generate an S3* `deducorrect` *object*

Description

Generate an S3 `deducorrect` object

Usage

```
newdeducorrect(corrected, corrections, status, Call = sys.call(-1))
```

Arguments

corrected	The corrected data.frame
corrections	A data.frame listing old and new values for every row and variable where corrections were applied
status	A data.frame with at least one status column.
Call	Optionally, a call object.

Value

an S3 object of class deducorrect

See Also

deducorrect-object

---

raghavachari *Determine if a matrix is unimodular using recursive Raghavachari criterium*

Description

This function is deducorrect internal

Usage

```
raghavachari(A)
```

Arguments

A	An object of class Matrix in $\{-1, 0, 1\}^{m \times n}$.

Value

TRUE or FALSE

See Also

isTotallyUnimodular

reduceMatrix *Apply reduction method from Scholtus (2008)*

Description

Apply the reduction method in the appendix of Scholtus (2008) to a matrix. Let A with coefficients in $\{-1, 0, 1\}$. If, after a possible permutation of columns it can be written in the form $A = [B, C]$ where each column in B has at most 1 nonzero element, then A is totally unimodular if and only if C is totally unimodular. By transposition, a similar theorem holds for the rows of A. This function iteratively removes rows and columns with only 1 nonzero element from A and returns the reduced result.

Usage

```
reduceMatrix(A)
```

Arguments

A	An object of class matrix in $\{-1, 0, 1\}^{m \times n}$.

Value

The reduction of A.

References

Scholtus S (2008). Algorithms for correcting some obvious inconsistencies and rounding errors in business survey data. Technical Report 08015, Netherlands.

See Also

isTotallyUnimodular

solSpace *Solution space for missing values under equality constraints*

Description

Solution space for missing values under equality constraints

solSpace method for editmatrix

This function finds the space of solutions for a numerical record x with missing values under linear constraints $Ax = b$. Write $x = (x_{obs}, x_{miss})$. Then the solution space for x_{miss} is given by $x_0 + Cz$, where x_0 is a constant vector, C a constant matrix and z is any real vector of dimension ncol(C). This function computes x_0 and C.

Usage

```
solSpace(E, x, ...)

## S3 method for class 'editmatrix'
solSpace(E, x, adapt = logical(length(x)),
  checkFeasibility = TRUE, ...)

## S3 method for class 'matrix'
solSpace(E, x, b, adapt = logical(length(x)),
  tol = sqrt(.Machine$double.eps), ...)
```

Arguments

E	and `editmatrix` or equality constraint matrix
x	a named numeric vector.
...	Extra parameters to pass to `solSpace.matrix`
adapt	A named logical vector with variables in the same order as in x
checkFeasibility	Check if the observed values can lead to a consistent record
b	Equality constraint constant vector
tol	tolerance used to determine 0-singular values when determining generalized inverse and to round coefficients of C to zero. See `MASS::ginv`.

Details

The user can specify extra fields to include in x_{miss} by specifying `adapt`. Also note that the method rests on the assumtion that all nonmissng values of x are correct.

The most timeconsuming step involves computing the generalized inverse of A_{miss} using `MASS::ginv` (code copied from MASS to avoid dependency). See the package vignette and De Waal et al. (2011) for more details.

Value

A `list` with elements $x0$ and C or `NULL` if the solution space is empty

References

T. De Waal, J. Pannekoek and S. Scholtus (2011) Handbook of statistical data editing Chpt 9.2.1

Venables, W. N. & Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth Edition. Springer, New York. ISBN 0-387-95457-0

See Also

`deduImpute`, `deductiveZeros`

Examples

```
# This example is taken from De Waal et al (2011) (Examples 9.1-9.2)
E <- editmatrix(c(
    "x1 + x2      == x3",
    "x2           == x4",
    "x5 + x6 + x7 == x8",
    "x3 + x8      == x9",
    "x9 - x10     == x11",
    "x6 >= 0",
    "x7 >= 0"
))


dat <- data.frame(
    x1=c(145,145),
    x2=c(NA,NA),
    x3=c(155,155),
    x4=c(NA,NA),
    x5=c(NA, 86),
    x6=c(NA,NA),
    x7=c(NA,NA),
    x8=c(86,86),
    x9=c(NA,NA),
    x10=c(217,217),
    x11=c(NA,NA)
)

# example with solSpace method for editmatrix
# example 9.1 of De Waal et al (2011).
x <-t(dat)[,1]
s <- solSpace(E,x)
s

# some values are uniquely determined and may be imputed directly:
imputess(x,s$x0,s$C)


# To impute everything, we choose z=1 (arbitrary)
z <- rep(1,sum(is.na(x)))
(y <- imputess(x,s$x0,s$C,z))

# did it work? (use a tolerance in checking to account for machine rounding)
# (FALSE means an edit is not violated)
any(violatedEdits(E,y,tol=1e-8))


# here's an example showing that solSpace only looks at missing values unless
# told otherwise.
Ey <- editmatrix(c(
    "yt == y1 + y2 + y3",
    "y4 == 0"))
```

```
y <- c(yt=10, y1=NA, y2=3, y3=7,y4=12)
# since solSpace by default checks the feasibility, we get no solution (since
# y4 violates the second edit)"
solSpace(Ey,y)


# If we ask solSpace not to check for feasibility, y4 is left alone (although
# the imputed answer is clearly wrong).
(s <- solSpace(Ey,y,checkFeasibility=FALSE))
imputess(y, s$x0, s$C)

# by setting 'adapt' we can include y4 in the imputation Since we know that
# with this adapt vector, imputation can be done consistently, we save some
# time by switching the feasibility check off.
(s <- solSpace(Ey,y,adapt=c(FALSE,FALSE,FALSE,FALSE,TRUE),
  checkFeasibility=FALSE))
imputess(y,s$x0,s$C)
```

`status` *Create empty status vector*

Description

Create empty status vector

Usage

```
status(n, ini = NA)
```

Arguments

`n`	length of status vector
`ini`	initial value, defaults to `NA`

Details

Every function in `deducorrect` returns the status of every row after treatment. The status vector is an `ordered` factor with levels

`invalid`	record is invalid but could not be corrected
`partial`	record violates less edits then before entering the function
`corrected`	record satisfies all edit restrictions after correction
`valid`	record violates no edit restrictions

where `invalid < partial < corrected < valid`

This function is deducorrect internal.

Value

an ordered factor with levels mentioned under details

Examples

```
# create statusvector

status <- deducorrect:::status(5)
status[1:5] <- c("invalid",NA,"corrected","valid","partial")

#
which(status < "valid")
```

Index

as.character.correctionRules
(correctionRules), 3

correctionRules, *3*, 3, *13*
correctRounding, *2*, 4
correctSigns, 7, *22, 23*
correctTypos, *2*, 10
correctWithRules, *4*, 12

damerauLevenshteinDistance, *11*, 14
date, *15*
deducorrect, *11*, *32*
deducorrect-object, 14
deducorrect-package, 2
deducorrrect, *5*
deductiveLevels, *2*, 15, *19, 20*
deductiveZeros, *2*, 17, *19, 20*, *30*
deduImpute, *2*, *18*, 19, *30*

editmatrix, *7*

getSignCorrection, 22
getVars.correctionRules
(correctionRules), 3

hellerTompkins, 23

imputess, *2*, 23
isTotallyUnimodular, *2*, *23*, 26, *28, 29*

matrix, *26*

newdeducorrect, *15*, 27

print.correctionRules
(correctionRules), 3
print.deducorrect (newdeducorrect), 27

raghavachari, 28
reduceMatrix, 29

solSpace, *2*, *18–20*, *23, 24*, 29
status, *2*, *5*, *8*, *15*, *28*, 32