Package 'strum'

June 12, 2015

Type Package

Title STRUctural Modeling of Latent Variables for General Pedigree

Version 0.6.2

Date 2015-06-11

Description Implements a broad class of latent variable and structural equation models for general pedigree data.

License GPL (>= 2)

Depends R (>= 3.0.0), methods, pedigree, Rgraphviz(>= 2.6.0)

Imports Matrix, MASS, Rcpp, graph

LinkingTo Rcpp, RcppArmadillo

NeedsCompilation yes

Author Nathan Morris [aut, cre],
Yeunjoo Song [aut],
Stephen Cahn [ctb]

Maintainer Nathan Morris <nathan.morris@cwru.edu>

Repository CRAN

Date/Publication 2015-06-12 06:43:35

R topics documented:

`strum-package` *A Package for STRUctural Modeling of Latent Variables for General Pedigree Data*

Description

This package implements the simulation and fitting of a broad range of latent variable and structural equation models. It can handle multilevel models, polygenic random effects and linkage random effects. Traditional structural equation models and confirmatory factor analysis may also be performed. The framework implemented now can only handle quantitative variables. Ordinal and nominal variables will be included in a future release.

Details

Package:	strum
Type:	Package
Version:	0.6.2
Date:	2015-06-11
License:	GPL (>= 2)

The following illustrates the typical steps for a strum analysis.

1. Construct a strumModel by `createStrumModel` function.
2. Prepare data using `createStrumData` function.
3. Run analysis by the function call `strum`.

This package can also be used for simulation studies as following.

1. Construct a strumSimModel by `createSimModel` function.
2. Simulate data using `simulateStrumData` function.
3. Run analysis on the simulated data by the function call `strum`.

Please check "Examples" section for the full illustration of strum use.

Author(s)

Nathan Morris, Yeunjoo Song, Stephen Cahn

Maintainer: Nathan Morris <nathan.morris@cwru.edu>, Yeunjoo Song<yeunjoo.song@cwru.edu>

References

Morris, N.J., Elston, R.C., & Stein, C.M. (2010). A framework for structural equation models in general pedigrees. Human Heredity 70:278-286.

Song, Y.E., Stein, C.M., & Morris, N.J. (2015). strum: an R package for *stru*ctural *m*odeling of latent variables for general pedigrees. BMC Genetics 16:35.

Examples

```
## Not run:
#================
# strum analysis
#================

# 1. Construct strumModel
#-------------------------

## 1.1 Model formulas
#---------------------
testForm1 = 'bp     =~ SBP + DBP
             anger  =~ A1 + A2
             stress =~ S1 + S2
             bp ~ anger + stress
             stress ~ anger + rs6040343
             var(stress)=.1
            '

testForm2 = 'L1 =~ SBP + DBP
             L1 ~ sex + <a,p,e>
            '

## 1.2 Create a strumModel
#-------------------------
myStrumModel1 = createStrumModel(formulas = testForm1)
myStrumModel2 = createStrumModel(formulas = testForm2)

# 2. Prepare data
#-----------------

## 2.1 Read a data file
#----------------------
dF = read.table("simped.dat", header=T)

## 2.2 Create a strumData object
#-------------------------------

### 2.2.1 No IBD file
#--------------------
myStrumData1 = createStrumData(dF, "Pedigree")

### 2.2.2 With IBD file
#----------------------
```

```
myStrumData2 = createStrumData(dF, "Pedigree", ibdFileName="GENIBD.chr1.ibd")

# 3. Run strum analysis
#-----------------------

## 3.1 Model with no ibd markers
#-------------------------------
myResult1 = strum(myStrumModel1, myStrumData1)

## 3.2 When an ibd marker is specified
#-------------------------------------
myResult2 = strum(myStrumModel2, myStrumData2, iMarkers=c("chr1marker1"))

#==================
# simulation study
#==================

# 1. Construct simModel
#-----------------------

## 1.1 Get some hapmap data & selct 10 snps
#------------------------------------------
hap20 = importHapmapData(20) # 'load(file="hap20.rdata")' with saved hapmap data
hap20 = hap20[(1:10)*10,]

## 1.2 Create strumMarker object with hapmap data
#------------------------------------------------
snpStrumMarker = createStrumMarker(hap20)

## 1.3 Ascertainment function
#----------------------------
aFunction = function(data) return(any(data$disease == 1))

## 1.4 Model formula
#--------------------
simForm = 'bp     =~ SBP + DBP
           anger  =~ A1 + 0.5*A2
           stress =~ S1 + 0.9*S2
           bp ~ anger + stress + <p,e>
           stress ~ anger + rs6040343
           var(stress)=.1
          '

## 1.5 Create a strumModel
#-------------------------
mySimModel = createSimModel(
                 formulas = simForm,
                 markerInfo = snpStrumMarker,
                 ascertainment = aFunction
               )

# 2. Simulate data based on the given data structure
#----------------------------------------------------
```

```
## 2.1 Read a data file
#----------------------
dF = read.csv("chr1.csv")[,1:8]
names(dF) = c("family","id", "father","mother",names(dF)[5:8])
aStrumData = createStrumData(dF, "Pedigree")

## 2.2 Simulate data
#-------------------
mySimulatedStrumData = simulateStrumData(mySimModel, aStrumData)

# 3. Run strum analysis using simulated data
#-------------------------------------------
mySimResult = strum(myStrumModel1, mySimulatedStrumData)

## End(Not run)
```

`createSimModel` *Create strumSimModel Object*

Description

A function to create an object of class strumSimModel which contains the information used for simulation.

Usage

```
createSimModel(formulas,
               tMissingRate = c(0),
               markerInfo = NULL,
               ascertainment = NULL,
               defaultError = '<p,e>',
               assumeExogCovariate = TRUE)
```

Arguments

`formulas`	Vector of expressions that defines the relationship among the variables in the model with the fixed coefficient (see Details).
`markerInfo`	Object containing a strumMarker or NULL.
`tMissingRate`	Numeric vector to hold the missing rate(s) of the simulated trait(s) in the model. The length of vector is equal to the number of observed traits or 1 if one missing rate is applied to all traits.
`ascertainment`	Function stating the ascertainment criteria of the data (see Details).
`defaultError`	Vector of chractor listing the default variance components in the model (see Details).
`assumeExogCovariate`	Logical. If TRUE, then it is assumed that any observed, quantitative exogenous traits are covariates. Otherwise, they are treated as latent and dependent variables.

Details

This function is used to create a `strumSimModel` object which will be an input parameter for `simulateStrumData` function.

The `formulas` argument is given as a charactor string of several expressions that defines the relationship among the variables. Blank lines and comments (line start with #) can be included between formulas.

Note that the coefficients of the right hand side variables of the measurement equations are fixed with the value given in the formulas.

Three different equations are allowed in the formulas syntax:

1. Measurement equations

   The "=~" operator may be read as "measured by" and it specifies the measurement equations in the model. The left hand side of the "=~" must contain one unobserved or latent factor variable name. The right hand side of the "=~" are observed variables, observed covariates, and/or any measurement errors enclosed with "<>".

2. Structural equations

   The "~" operator specifies the structural equations in the model. The left hand side of the "~" must contain one unobserved or latent factor variable name. The right hand side of the "~" are unobserved or latent factors, observed covariates, and/or any random effects enclosed with "<>".

3. Constraints

   The "=" operator specifies the constraints in the model, i.e., fixing a model parameter - fixed variance, fixed covariance. The left hand side of the "=" must contain a reserved word for constraint - "var" or "cov" - with one or two variable names depending on the word. The right hand side of the "=" is a fixed value. Here are some examples:

   - Fixing the variance of a latent variable:

     `var(stress) = .1`
   - Fixing the covariance between two latent variables:

     `cov(z1,z2) = 4`

If a random effect such as `p`, `e`, `a` or `c` is not included in the model formulas, then, by default, the value of `defaultError` is included in the model. For quantitative traits, the `e` term should always be present. Therefore, the program automatically include `e` term even if no error terms are specified either in `formulas` nor in `defaultError`.

Note that "a" (additive), "p" (polygenic), "c" (common environmental) and "e"(independent environmental) are reserved variable names to specify a type of variance component, so that may not be used as input arguments.

The return value of `ascertainment` function is composed of two components for each pedigree, allowing either one of components or both components to be returned as a list. The first component is a TRUE/FALSE value indicating the ascertainment status of the pedigree. The second component is a vector of TRUE/FALSE stating the proband status of each member of the pedigree. Here are some examples:

1. Ascertainment status only

```
aFunction = function(ped) return(TRUE)
aFunction = function(ped) return(any(ped$SBP > 2))
```

The pedigree is ascertained when a "TRUE" is returned for that pedigree.

2. Proband status only

```
aFunction = function(ped) return(ped$SBP > 2)
aFunction = function(ped) return(ped$SBP > 2 && ped$DBP < 1)
aFunction = function(ped) return(ped$disease == 1)
```

In this case, the ascertainment status of the pedigree is determined according to the existance of a proband in each pedigree. If no probands who meet the criteria exist in a simulated pedigree, then that pedigree is not selected.

3. Both

```
aFunction = function(ped)
{
  asc = any(ped$disease==1)
  pro = ped$SBP > 2
  return(list(aStatus=asc, pStatus=pro))
}
```

When a list with both information is returned, it is required that the ascertainment status (a logical value) is the first component, and the proband status (a vector of logicals of pedigree size) is the second component. The pedigree is selected when the returned first component is a "TRUE" for that pedigree.

Value

Returns an object of type `strumSimModel`.

See Also

`strumSimModel`, `strumMarker`

Examples

```
## Not run:
# Create a strumSimModel.
# - simForm is a formulas string.
# - snpStrumMarker is an object of strumMarker class.
# - aFunction is an ascertainment function definition.
#-------------------------------------------------------
mySimModel = createSimModel(
                 formulas = simForm,
                 markerInfo = snpStrumMarker,
                 ascertainment = aFunction
               )

## End(Not run)
```

createStrumData *Create strumData Object*

Description

A function to create strumData object.

Usage

```
createStrumData(inData, dType, ibdFileName=NULL, fileType="SAGE")
```

Arguments

inData	Object of class data.frame containing input data.
dType	Character stating the type of input data, "Pedigree" or "RawData"
ibdFileName	Character stating the name of ibd file to import.
fileType	Character stating the type of ibd file, the deafult value is "SAGE".

Details

This function is used to create a strumData class object for strum analysis. The value of inData has to be a data.frame. The allowed values for dType is either "Pedigree" or "RawData". Note that, if dType="Pedigree", the data must be a data.frame with 4 required fields - family, id, father, mother.

The ibd information for the family data can be imported by specifying the name of ibd file into ibdFileName. Currently, the ibd file generated by the program GENIBD in S.A.G.E. package is supported.

Value

Returns an object of class strumData.

See Also

strumData, simulateStrumData

Examples

```
## Not run:
# Create a strumData object with different type of input data.
# - dF is a data.frame containing input data.
#-------------------------------------------------------------
rawStrumData = createStrumData(dF, "RawData")
pedStrumData = createStrumData(dF, "Pedigree")
pedStrumDataIBD = createStrumData(dF, "Pedigree", ibdFileName="ch20.ibd")

## End(Not run)
```

createStrumMarker *Create strumMarker Object*

Description

A function to create strumMarker object.

Usage

```
createStrumMarker(hapMapData,
                  populationRecombRate = 50,
                  errorRate = 0,
                  mutationRate = 0,
                  missingRate = 0,
                  coding = c(0,1,2),
                  returnIBD = FALSE,
                  intervalIBD = 10)
```

Arguments

hapMapData	Object of class data.frame containing input data (see Details).
populationRecombRate	Numeric value stating the average number of crossovers per cM when simulating founders.
errorRate	Numeric value stating the probability of a genotype error. If a genotype error occurs, a genotype (other than the true genotype) is selected with uniform probability.
mutationRate	Numeric value stating the probability that an allele will be mutated before it is passed on. If a mutation occurs, a new allele (not the original allele) is selected with uniform probability.
missingRate	Numeric value stating the probability of a missing genotype.
coding	Numeric vector of length three stating the coding of inheritance model. The first element corresponds to the code assigned to the homozygous minor allele. The second element corresponds to the heterozygous genotype. The third element corresponds to the homozygous major allele. The default coding is additive (i.e. coding = c(0, 1, 2)).
returnIBD	Logical. If TRUE, then the ibd slot in the strumData object, simulated using the returned strumMarker object from this function, will be populated. If FALSE, the ibd slot will be left empty.
intervalIBD	Numeric value stating the interval that ibd values should be calculated.

Details

The value of `hapMapData` must be a data.frame containing hapmap data. The three columns that state the SNP information must present:

1. `rsID` contains the names of the marker.
2. `chr` is the chromosome or linkage group.
3. `phys_position` is the physical base pair position. It is converted to Haldane (genetic distance) map position in centiMorgans as phys_position/1000000.

Value

Returns an object of class `strumMarker`.

See Also

`strumMarker`, `importHapmapData`

Examples

```
## Not run:
# Get some hapmap data & selct 10 snps.
#-------------------------------------
hap20 = importHapmapData(20)
hap20 = hap20[(1:10)*10,]

# Create strumMarker object with hapmap data.
#--------------------------------------------
snpStrumMarker = createStrumMarker(hap20)

## End(Not run)
```

`createStrumModel` *Create strumModel Object*

Description

A function to create an object of class strumModel.

Usage

```
createStrumModel(formulas,
                 ascertainment = NULL,
                 defaultError = '<p,e>',
                 assumeExogCovariate = TRUE,
                 fixLoadingToOne = TRUE)
```

Arguments

formulas	Vector of expressions that defines the relationship among the variables in the model (see Details).
ascertainment	Charactor stating the name of the column in data file that contains the indicator variable (1, 0) designating the probands of the pedigrees.
defaultError	Vector of chractor listing the default variance components in the model (see Details).
assumeExogCovariate	Logical. If TRUE, then it is assumed that any observed, quantitative exogenous traits are covariates. Otherwise, they are treated as latent and dependent variables.
fixLoadingToOne	Logical. If TRUE, then it sets the first indicator variable for each factor to have a coefficient of 1. Otherwise, all coefficients are estimated.

Details

This function is used to create a trait model which will be an input parameter for `strum` function.

The formulas argument is given as a charactor string of several expressions that defines the relationship among the variables. Blank lines and comments (line start with #) can be included between formulas.

Three different equations are allowed in the formulas syntax:

1. Measurement equations
   The "=~" operator may be read as "measured by" and it specifies the measurement model equations in the model. The left hand side of the "=~" must contain one unobserved or latent factors. The right hand side of the "=~" are observed variables, observed covariates, and/or any measurement errors enclosed with "<>".
2. Structural equations
   The "~" operator specifies the structural equations in the model. The left hand side of the "~" must contain one unobserved or latent factor variable name. The right hand side of the "~" are unobserved or latent factors, observed covariates, and/or any random effects enclosed with "<>".
3. Adding a covariance
   By default, the endogenous error terms are uncorrelated, but the exogenous variables are correlated. To add a correlation between endogenous terms, use "cov" operator. For example,

   ```
   cov(x,y) = NA
   ```

   .
4. Removing a constraint on a coefficient
   By default, the value of `fixLoadingToOne` equals TRUE, setting the first indicator variable for each factor to have a coefficient of 1. To disable this option selectively for a coefficient to be estimated, use "coef" operator. For example,

   ```
   coef(y1,z1) = NA
   ```

   .

5. Constraints

   The "=" operator specifies the constraints in the model, i.e., fixing a model parameter - fixed variance, fixed covariance, or fixed coefficient. The left hand side of the "=" must contain a reserved word for constraint - "var", "cov", or "coef" - with one or two variable names depending on the word. The right hand side of the "=" is a fixed value. Here are some examples:

   - Fixing the variance of a latent variable:
     `var(stress) = .1`
   - Fixing the covariance between two latent variables:
     `cov(z1,z2) = 4`
   - Fixing the coefficient between two variables:
     `coef(y1,z1) = 2`

If a random effect such as `p`, `e`, `a` or `c` is not included in the model formulas, then, by default, the value of `defaultError` is included in the model. For quantitative traits, the `e` term should always be present. Therefore, the program automatically include `e` term even if no error terms are specified either in `formulas` nor in `defaultError`.

Note again that "a" (additive), "p" (polygenic), "c" (common environmental) and "e"(independent environmental) are reserved variable names to specify a type of variance component, so that may not be used as input arguments.

Value

Returns an object of type `strumModel`.

See Also

`strumModel`

Examples

```
# Model formulas.
#----------------
strumForm = 'bp     =~ SBP + DBP
             anger  =~ A1 + A2 + A3
             stress =~ S1 + S2 + S3
             bp ~ anger + stress + <p,e>
             stress ~ anger + rs6040343
             var(stress)=.1
            '

# Create a strumModel.
#---------------------
myStrumModel = createStrumModel(
                 formulas = strumForm,
                 ascertainment = "disease"
               )
```

importHapmapData *Import Hapmap Data*

Description

A function to import hapmap data from the website stated.

Usage

```
importHapmapData(chr, pop="CEU", ...)
```

Arguments

chr	Numeric value stating a chromosome number.
pop	Character value stating a population
...	Other arguments

Source

http://hapmap.ncbi.nlm.nih.gov/downloads/phasing/2009-02_phaseIII/HapMap3_r2/CEU/TRIOS/hapmap3_r2_b36_fwd.consensus.qc.poly.chr_ceu.phased.gz

Examples

```
## Not run:
hap20 = importHapmapData(20)

## End(Not run)
```

simulateStrumData *Simulate strumData Object*

Description

A function to create a strumData object containing the simulated data according to the simulation model.

Usage

```
simulateStrumData(simModel, inData=NULL, N=NULL)
```

Arguments

simModel	Object of class strumSimModel.
inData	Object of class strumData, data.frame, or NULL.
N	A positive interger number to specify the particular size of simulated data or NULL.

Details

This function is used to simulate a data according to the specified simulation model. Note that either inData or N, or both must be specified.

Value

Returns an object of class `strumData`.

See Also

`strumSimModel`, `strumData`

Examples

```
## Not run:
# Simulate a strumData object with different type of input data.
# - mySimModel is an object of strumSimModel class.
# - dF is a data.frame containing input data.
# - rawStrumData is an object of "RawData" type strumData class.
# - pedStrumData is an object of "Pedigree" type strumData class.
#------------------------------------------------------------------
myStrumData = simulateStrumData(mySimModel, N=1000)
myStrumData = simulateStrumData(mySimModel, dF)
myStrumData = simulateStrumData(mySimModel, rawStrumData)
myStrumData = simulateStrumData(mySimModel, pedStrumData)

## End(Not run)
```

`strum` *STRUctural Modeling of Latent Variables for Family Data*

Description

Main function to run strum analysis and return the result as a (list of) strumFittedModel.

Usage

```
strum(myStrumModel,
      myStrumData,
      step1OptimControl = list(maxit=5500, fnscale=-10),
      startValueControl = list(initPopulation=NULL,
                               nChildren=NULL,
                               nGenerations=NULL,
                               selection1=NULL,
                               selection2=NULL),
      step2OptimControl = list(maxit=5000, reltol=.Machine$double.eps),
      ibdMarkers=NULL)
```

Arguments

myStrumModel	Object of class strumModel.
myStrumData	Object of class strumData.
step1OptimControl	List of control parameters for optim function in step1.
startValueControl	List of control parameters for strating value generation function in step2.
step2OptimControl	List of control parameters for optim function in step2.
ibdMarkers	Character vector contaning the name of marker(s) for additive genetic variance component a (see Details).

Details

Three parameters of this function provide more flexibility to users to control the strum analysis in different stage.

The step1OptimControl argument is given as a list of several control parameters that is passed directly to the function optim in the step 1 of model fitting.

The step2OptimControl argument is given as a list of several control parameters that is passed directly to the function optim in the step 2 of model fitting.

The startValueControl argument is given as a list of 5 parameters for the function to generate the starting values (using genetic algorithm) of model fitting in step 2. Five different parameters in the list are:

- initPopulation
  The size of initial population. The default value is the number of model parameters * 120.
- nChildren
  The number of children. The default value is the number of model parameters * 2.
- nGenerations
  The number of generation. The default value is 20.
- selection1
  The size of the initial selection. The default value is 15.
- selection2
  The size of the final selection. The default value is 15.

Note that if the analysis model includes an additive genetic variance component a for the random effects, the value of iMarkers is used to determine the name of marker(s) for a from the imported ibd file. If no values are specified for ibdMarkers, then, by default, all imported ibd markers that exist in the myStrumData object are used, analyzing one by one. Therefore, the analysis result includes a list of strumFittedModel.

Value

Returns a (list of) strumFittedModel object.

See Also

createStrumModel, createStrumData

Examples

```
## Not run:
# Run strum analysis.
# - myStrumModel is a strumModel object.
# - myStrumData is a strumData object.
#---------------------------------------
fitResult = strum(myStrumModel, myStrumData, ibdMarkers=c("marker1","marker2"))

## End(Not run)
```

---

strumData-class *Class* "strumData"

Description

strumData is an S4 class that contains the input data for a STRUctural Modeling of latent variables for family data.

Objects from the Class

Objects should not be created by calls of the form new("strumData", ...) but by the calls to the createStrumData function.

Slots

dataType: Character stating the type of input data, must be "Pedigrees" or "RawData"

dataVals: Object of class data.frame containing input data.

phi: Object of class "list" containing the kinship matrices for each pedigree. This is empty if the dataType is not "Pedigree".

ibd: Object of class "strumIBD" containing the IBD information for input pedigrees. This is empty if the dataType is not "Pedigree".

Methods

importIBD signature(object = "strumData", fileName, fileType="SAGE"): Import the ibd information of the input data.

show signature(object = "strumData"): Print a summary of the class.

See Also

strumIBD, createStrumData, simulateStrumData

Examples

```
showClass("strumData")
```

strumFittedModel-class

Class "strumFittedModel"

Description

strumSimModel is an S4 class that contains the result from a strum analysis, i.e., information of a trait model with fitted parameter values.

Objects from the Class

Objects should be created by the calls to the strum function.

Slots

myStrumModel: Object of class "strumModel" stating the analysis model.

modelValidity: Indicator stating the validity of the analysis model.

fittedParameters: Object of class "data.frame" containing the fitted parameter values of the model with the standard errors, confidence intervals, and p-values.

fittedParametersCovMatrix: The variance-covariance matrix of the fitted parameters.

deltaParameters: For internal use

deltaParametersCovMatrix: For internal use

parDiff: For internal use

parDiffCovMatrix: For internal use

chiTestOut: Object of class "data.frame" containing the un-adjusted(1), mean scaled(2), and mean/variance scaled(3) chi-square index of fits with the degrees of freedom and p-values along with the theoretically corrected p-value(4) from simulation.

fitIndices: Object of class "data.frame" containing the model fit indices.

Methods

fittedParameters signature(object = "strumFittedModel"): Accessor function, returns the fittedParameters of the object.

fittedParametersCovMatrix signature(object = "strumFittedModel"): Accessor function, returns the fittedParametersCovMatrix of the object.

chiTestOut signature(object = "strumFittedModel"): Accessor function, returns the chiTestOut of the object.

show signature(object = "strumFittedModel"): Print a summary of the fitted model.

See Also

strumModel

Examples

```
showClass("strumFittedModel")
```

strumIBD-class *Class* "strumIBD"

Description

strumIBD is an S4 class that represents IBD information of the input data for a STRUctural Modeling of latent Variables for family data.

Objects from the Class

Objects can be created by calls of the form new("strumIBD", ...) or by the importIBD method in strumData class.

Slots

ibdMarker: Object of class "data.frame" containing the information about the markers. Typically contains two columns, "Marker" and "Position".

ibdMatrix: Object of class "list" containing the lists of identical by descent (ibd) matrices, for each family and marker.

Methods

ibdMarker signature(object = "strumIBD"): Accessor function, returns the marker information.

ibdMatrix signature(object = "strumIBD"): Accessor function, returns the list of ibd matrices of the input data.

show signature(object = "strumIBD"): Print a summary of the class.

See Also

strumSimModel, strumData

Examples

```
showClass("strumIBD")
```

strumMarker-class *Class* "strumMarker"

Description

strumMarker is an S4 class that represents a marker data for a STRUctural Modeling of latent variables for family data.

Objects from the Class

Objects should not be created by calls of the form new("strumMarker", ...) but by the calls to the createStrumMarker function.

Slots

markerFacts: Object of class "data.frame" containing the SNP information - markerName, chrom, mapPos, minorAllele, majorAllele.

haplotypes: Object of class "matrix" containing hapmap data.

populationRecombRate: Numeric value stating the average number of crossovers per cM when simulating founders.

errorRate: Numeric value stating the probability of a genotype error. If a genotype error occurs, a genotype (other than the true genotype) is selected with uniform probability.

mutationRate: Numeric value stating the probability that an allele will be mutated before it is passed on. If a mutation occurs, a new allele (not the original allele) is selected with uniform probability.

missingRate: Numeric value stating the probability of a missing genotype.

coding: Numeric vector of length three stating the coding of inheritance model. The first element corresponds to the code assigned to the homozygous minor allele. The second element corresponds to the heterozygous genotype. The third element corresponds to the homozygous major allele. The default coding is additive (i.e. coding = c(0, 1, 2)).

returnIBD: Logical. If TRUE, then the ibd slot in the strumData object, simulated using the returned strumMarker object from this function, will be populated. If FALSE, the ibd slot will be left empty.

intervalIBD: Numeric value stating the interval that ibd values should be calculated.

Methods

markerFacts signature(object = "strumMarker"): Accessor function, returns the value of the slot markerFacts.

haplotypes signature(object = "strumMarker"): Accessor function, returns the value of the slot haplotypes.

populationRecombRate signature(object = "strumMarker"): Accessor function, returns the value of the slot populationRecombRate.

errorRate `signature(object = "strumMarker")`: Accessor function, returns the value of the slot `errorRate`.

mutationRate `signature(object = "strumMarker")`: Accessor function, returns the value of the slot `mutationRate`.

missingRate `signature(object = "strumMarker")`: Accessor function, returns the value of the slot `missingRate`.

coding `signature(object = "strumMarker")`: Accessor function, returns the value of the slot `coding`.

returnIBD `signature(object = "strumMarker")`: Accessor function, returns the value of the slot `returnIBD`.

intervalIBD `signature(object = "strumMarker")`: Accessor function, returns the value of the slot `intervalIBD`.

show `signature(object = "strumMarker")`: Print a summary of the class.

See Also

`createStrumMarker`, `simulateStrumData`

Examples

```
showClass("strumMarker")
```

`strumModel-class` *Class* `"strumModel"`

Description

`strumModel` is an S4 class that represents a STRUctural Modeling of latent Variables for family data.

Objects from the Class

Objects should not be created by calls of the form `new("strumModel", ...)` but by the calls to the `createStrumModel` function.

Slots

`varList`**:** Object of class `"data.frame"` containing a list of variables and their properties.

`formulas`**:** Object of class `"character"` containing the information about the formulas and dependencies.

`allRandomEffects`**:** Object of class `"character"` listing all variance components in the model.

`paramNames`**:** Object of class `"character"` describing the names of the model parameters.

`ascertainment`**:** Object of class `"ANY"` stating the ascertainment criteria (function) of the data.

`E`**:** Object of class `"list"` containing the variance matrices of measurement errors of the model.

`Z`: Object of class `"list"` containing the variance matrices of random effects for the structural equation of the model.

`L`: Object of class `"function"` to generate a matrix to relate the unobserved latent factors to the observed traits in the model.

`B`: Object of class `"function"` to generate a matrix denoting the causal relationship among the latent factors in the model.

`Gs`: Object of class `"function"` to generate a matrix to relate the observed covariates to the unobserved latent factors in the model.

`Gm`: Object of class `"function"` to generate a matrix to relate the observed covariates to the observed traits in the model.

`thToThB`: Object of class `"numeric"` for future use.

Extends

Class `"strumVirtualModel"`, directly.

Methods

show `signature(object = "strumModel")`: Print a summary of the model.

plot `signature(object = "strumModel", layoutType="dot", name="strumModel", toFile=TRUE, fileType="dot")`
: Plot the model.

See Also

`strumSimModel`, `strumVirtualModel`, `createStrumModel`

Examples

```
showClass("strumModel")
```

---

strumSimModel-class *Class* `"strumSimModel"`

Description

`strumSimModel` is an S4 class that contains the information of a trait model for simulation.

Objects from the Class

Objects should not be created by calls of the form `new("strumSimModel", ...)` but by the calls to the `createSimModel` function.

Slots

`markerInfo`: Object of class `"ANY"` containing either a strumMarker object or NULL.

`traitMissingRate`: Object of class `"numeric"` containing the missing rate(s) of the simulated traits in the model.

`varList`: Object of class `"data.frame"` containing a list of variables and their properties.

`formulas`: Object of class `"character"` containing the information about the formulas and dependencies.

`allRandomEffects`: Object of class `"character"` listing all variance components in the model.

`paramNames`: Object of class `"character"` describing the names of the model parameters.

`ascertainment`: Object of class `"ANY"` stating the ascertainment criteria (function) of the data.

`E`: Object of class `"list"` containing the variance matrices of measurement errors of the model.

`Z`: Object of class `"list"` containing the variance matrices of random effects for the structural equation of the model.

`L`: Object of class `"function"` to generate a matrix to relate the unobserved latent factors to the observed traits in the model.

`B`: Object of class `"function"` to generate a matrix denoting the causal relationship among the latent factors in the model.

`Gs`: Object of class `"function"` to generate a matrix to relate the observed covariates to the unobserved latent factors in the model.

`Gm`: Object of class `"function"` to generate a matrix to relate the observed covariates to the observed traits in the model.

`thToThB`: Object of class `"numeric"` for future use.

Extends

Class `"strumVirtualModel"`, directly.

Methods

markerInfo `signature(object = "strumSimModel")`: Accessor function, returns the markerInfo of the model.

traitMissingRate `signature(object = "strumSimModel")`: Accessor function, returns the traitMissingRate of the model.

show `signature(object = "strumSimModel")`: Print a summary of the model.

plot
```
signature(object = "strumSimModel", layoutType="dot",
    name="strumSimModel", toFile=TRUE, fileType="dot")
```
: Plot the model.

See Also

`strumModel`, `strumVirtualModel`, `createSimModel`

Examples

```
showClass("strumSimModel")
```

strumVirtualModel-class
Class "strumVirtualModel"

Description

The strumVirtualModel class is a virtual class to represent a STRUctural Modeling of latent Variables for family data. It contains a description of the model as specified by the user through the model formulas and an set of functions to construct the internal matrix representation of the model.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

varList: Object of class "data.frame" containing a list of variables and their properties.

formulas: Object of class "character" containing the information about the formulas and dependencies.

allRandomEffects: Object of class "character" listing all variance components in the model.

paramNames: Object of class "character" describing the names of the model parameters.

ascertainment: Object of class "ANY" stating the ascertainment criteria (function) of the data.

E: Object of class "list" containing the variance matrices of measurement errors of the model.

Z: Object of class "list" containing the variance matrices of random effects for the structural equation of the model.

L: Object of class "function" to generate a matrix to relate the unobserved latent factors to the observed traits in the model.

B: Object of class "function" to generate a matrix denoting the causal relationship among the latent factors in the model.

Gs: Object of class "function" to generate a matrix to relate the observed covariates to the unobserved latent factors in the model.

Gm: Object of class "function" to generate a matrix to relate the observed covariates to the observed traits in the model.

thToThB: Object of class "numeric" for future use.

Methods

formulas signature(object = "strumVirtualModel"): Accessor function, returns the formulas of the model.

formulas<- signature(object = "strumVirtualModel", value): Modifier function, sets the formulas of the model with a given value.

covariatePhenotypes signature(object = "strumVirtualModel"): Accessor function, returns the names of covariate phenotypes of the model.

varList `signature(object = "strumVirtualModel")`: Accessor function, returns the variable list of the model as a data.frame.

varList<- `signature(object = "strumVirtualModel", value)`: Modifier function, sets the variable list of the model with a given value.

allRandomEffects `signature(object = "strumVirtualModel")`: Accessor function, returns the names of all random effects of the model.

allRandomEffects<- `signature(object = "strumVirtualModel", value)`: Modifier function, sets the allRandomEffects of the model with a given value.

paramNames `signature(object = "strumVirtualModel")`: Accessor function, returns the names of all parameters of the model.

paramNames<- `signature(object = "strumVirtualModel", value)`: Modifier function, sets the paramNames of the model with a given value.

ascertainment `signature(object = "strumVirtualModel")`: Accessor function, returns the ascertainment definition of the model.

ascertainment<- `signature(object = "strumVirtualModel", value)`: Modifier function, sets the ascertainment of the model with a given value.

show `signature(object = "strumVirtualModel")`: Print a summary of the model.

plot `signature(object = "strumVirtualModel", layoutType="dot", name="strumVirtualModel", toFile=TRUE, fileType="dot")`

: Plot the model.

See Also

`strumModel`, `strumSimModel`

Examples

```
showClass("strumVirtualModel")
```

Index