Package 'PhenotypeSimulator'

June 12, 2017

Title Flexible Phenotype Simulation from Different Genetic and Noise Models

Version 0.1.2

URL https://github.com/HannahVMeyer/PhenotypeSimulator

BugReports https://github.com/HannahVMeyer/PhenotypeSimulator/issues

Description Phenotype simulator allows for the flexible simulation of phenotypes under different models, including fixed and background genetic effects as well as correlated, fixed and background noise effects. Different phenotypic effects can be combined into a final phenotype while controling for the proportion of variance explained by each of the components. For each component, the number of variables, their distribution and the design of their effect across traits can be customised. The final simulated phenotypes and its components can be automatically saved into .rds or .csv files. In addition, for simulated genotypes, export into plink format is possible.

Depends R (>= 3.3.2)

LinkingTo Rcpp

Imports methods, optparse, R.utils, plyr, mvtnorm, snpStats, Rcpp (>= 0.12.11)

Suggests testthat, knitr, rmarkdown

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

VignetteBuilder knitr

NeedsCompilation yes

Author Hannah Meyer [aut, cre], Konrad Rudolph [ctb]

Maintainer Hannah Meyer <hannah@ebi.ac.uk>

Repository CRAN

Date/Publication 2017-06-12 11:31:26 UTC

R topics documented:

`addNonNulls` *Add all non-NULL elements of list.*

Description

Add all non-NULL elements of list.

Usage

```
addNonNulls(compList)
```

Arguments

`compList`	list of numeric matrices or data.frames of the equal dimensions

Value

matrix or data.frame containing sum of all list elements where `is.null` is FALSE

commaList2vector *Comma-separated string to numeric vector.*

Description

Split input of comma-separated string into vector of numeric values.

Usage

```
commaList2vector(commastring)
```

Arguments

commastring	input character vector containing numbers separated by commas

Value

numeric vector of values extracted from commastring

correlatedBgEffects *Simulate correlated background effects.*

Description

correlatedBgEffects computes a background effect that simulates structured correlation between the phenotypes.

Usage

```
correlatedBgEffects(N, P, pcorr)
```

Arguments

N	number [integer] of samples to simulate
P	number [integer] of phenotypes to simulate
pcorr	initial strength of correlation [double] between neighbouring traits

Details

correlatedBgEffects can be used to simulate phenotypes with a defined level of correlation between traits. The level of correlation depends on the distance of the traits. Traits of distance d=1 (adjacent columns) will have correlation cor=$pcorr^1$, traits with d=2 have cor=$pcorr^2$ up to traits with d=(P-1) cor=$pcorr^{(P-1)}$. The correlated background effect correlated is simulated based on this correlation structure C: $correlated\ N_{NP}(0, C)$.

Value

N x P matrix of correlated background effects

See Also

rmvnorm which is used to simulate the multivariate normal distribution

Examples

```
correlatedBg <- correlatedBgEffects(N=100, P=20, pcorr=0.4)
```

---

createPheno *Combine different phenotype components.*

Description

createPheno takes precomputed phenotype components and rescales them according to the specified proportion of variance they should explain in the final phenotype.

Usage

```
createPheno(P, N, sampleID = "ID_", phenoID = "trait_",
  correlatedBg = NULL, genFixed = NULL, genBg = NULL, noiseFixed = NULL,
  noiseBg = NULL, genVar = NULL, h2s = NULL, h2bg = NULL,
  noiseVar = NULL, rho = NULL, delta = NULL, phi = NULL, gamma = 0.8,
  theta = 0.8, eta = 0.8, alpha = 0.8, pcorr = 0.6,
  modelNoise = "noNoise", modelGenetic = "noGenetic", verbose = TRUE)
```

Arguments

P	number [integer] of phenotypes to simulate
N	number [integer] of samples to simulate
sampleID	prefix [string] for naming samples (followed by sample number from 1 to N)
phenoID	prefix [string] for naming traits (followed by trait number from 1 to P)
correlatedBg	list of correlated background effects as obtained by correlatedBgEffects
genFixed	list of independent and shared genetic fixed effects as obtained by geneticFixedEffects
genBg	list of independent and shared genetic background effects as obtained by geneticBgEffects
noiseFixed	list of independent and shared noise fixed effects as obtained by noiseFixedEffects
noiseBg	list of independent and shared noise background effects as obtained by noiseBgEffects
genVar	Proportion [double] of total genetic variance
h2s	Proportion [double] of variance of fixed genetic effects
h2bg	Proportion [double] of variance of background genetic effects; either h2s or h2b have to be specified and h2s + h2b = genVar

noiseVar	Proportion [double] of total noise variance
rho	Proportion [double] of variance of correlated noise effects
delta	Proportion [double] of fixed noise variance
phi	Proportion [double] of variance of background noise effects
gamma	Proportion [double] of variance of shared fixed noise effects
theta	Proportion [double] of variance of shared fixed genetic effects
eta	Proportion [double] of variance of shared bg genetic effects
alpha	Proportion [double] of variance of shared bg noise effect
pcorr	Correlation [double] between phenotypes
modelNoise	name [string] of noise model for the phenotype simulation; based on model independentation, phenotype components will be added to the final phenotype. Possible models: "noNoise", "noiseFixedOnly", "noiseBgOnly", "noiseCorrelatedOnly", "noiseFixedAndBg","noiseCorrelatedAndBg", "noiseFixedAndCorrelated", "noiseFixedAndBgAndCorrelated"
modelGenetic	name [string] of genetic model for the phenotype simulation; based on model independentation, phenotype components will be added to the final phenotype. Possible models: "noGenetic","geneticBgOnly", "geneticFixedOnly","geneticFixedAndBg"
verbose	[boolean]; if TRUE, progress info is printed to standard out

Value

named list of absolute levels of variance explained by each phenotype component (varComponents), the scaled phenotype components (phenoComponents) and and parameters used for phenotype set-up and labeling (setup)

See Also

rescaleVariance for the function used to rescale phenotype components

Examples

```
noiseBg <- noiseBgEffects(N=100, P=50)
phenotypeNoiseBgOnly <- createPheno(P=50, N=100, noiseBg=noiseBg,
modelNoise="noiseBgOnly", modelGenetic="noGenetic")

genotypes <- simulateGenotypes(N=100, NrSNP=50)
causalSNPs <- getCausalSNPs(genotypes=genotypes, standardise=FALSE)
genFixed <- geneticFixedEffects(N=100, P=50, X_causal=causalSNPs)
phenotypeNoiseBgOnlyGenFixed <- createPheno(P=50, N=100, noiseBg=noiseBg,
genFixed=genFixed, modelNoise="noiseBgOnly",
modelGenetic="geneticFixedOnly", genVar=0.05)
```

`geneticBgEffects` *Simulate genetic background effects.*

Description

geneticBgEffects simulates two random genetic effects (shared and independent) based on the kinship estimates of the (simulated) samples.

Usage

```
geneticBgEffects(P, kinship)
```

Arguments

`P`	number [integer] of phenotypes to simulate
`kinship`	[N x N] matrix of kinship estimates [double]

Details

For the simulation of the genetic background effects, three matrix components are used: i) the kinship matrix K [N x N] which is treated as the sample-design matrix (the genetic profile of the samples), ii) an effect-size matrix B [N x P] with vec(B) drawn from a normal distribution and iii) the trait design matrix A [P x P]. For the independent effect, A is a diagonal matrix with normally distributed values. A for the shared effect is a matrix of rowrank one, with normally distributed entries in row 1 and zeros elsewhere. The final effect E, the three matrices are multiplied as: E = KBA

Value

named list of shared background genetic effects (shared: [N x P] matrix) and independent background genetic effects (independent: [N x P] matrix)

Examples

```
genotypes <- simulateGenotypes(N=100, NrSNP=400, verbose=FALSE)
kinship <- getKinship(genotypes$X_sd, norm=TRUE, verbose=FALSE)
geneticBg <- geneticBgEffects(P=10, kinship=kinship)
```

`geneticFixedEffects` *Simulate genetic fixed effects.*

Description

geneticFixedEffects takes a matrix of SNPs which should be added as fixed effect to the phenotype. SNPs can have effects across all traits (shared) or to a number of traits only (independent); the proportion of independent SNPs from the total input SNPs can be chosen via pIndependentGenetic. The number of traits that are associated with independent genetic effects can be chosen via pTraitIndependentGenetic.

Usage

```
geneticFixedEffects(X_causal, P, N = NULL, pIndependentGenetic = 0.4,
  pTraitIndependentGenetic = 0.2, verbose = TRUE)
```

Arguments

`X_causal`	[N x NrCausalSNPs] matrix of standardised (depending on standardise option) SNPs
`P`	number [integer] of phenotypes to simulate
`N`	number [integer] of samples to simulate; has to be less than or equal to the number of samples in X_causal (rows); if less than number of samples in X_causal, N random rows of X_causal will be drawn; if not specified, N assumed to be equal to samples in X_causal
`pIndependentGenetic`	Proportion [double] of genetic effects (SNPs) to have an independent fixed effect
`pTraitIndependentGenetic`	Proportion [double] of traits influenced by independent fixed genetic effects
`verbose`	[boolean]; if TRUE, progress info is printed to standard out

Value

named list of shared fixed genetic effects (shared: [N x P] matrix), independent fixed genetic effects (independent: [N x P] matrix), the causal SNPs named by having a shared or independent effect (cov: [NrCausalSNPs x N] matrix) and the simulated effect sizes of the causal SNPs (cov_effect: [P x NrCausalSNPs] dataframe)

Examples

```
genotypes <- simulateGenotypes(N=100, NrSNP=20, verbose=FALSE)
causalSNPs <- getCausalSNPs(genotypes=genotypes)
geneticFixed <- geneticFixedEffects(X_causal=causalSNPs, P=10, N=100)
```

getAlleleFrequencies *Compute allele frequencies from genotype data.*

Description

Compute allele frequencies from genotype data.

Usage

```
getAlleleFrequencies(snp)
```

Arguments

snp	vector of length N samples with genotypes encoded as 0,1 and 2

Value

vector with minor and major allele frequency

Examples

```
# create snp vector with minor allele frequency 0.3
snp <- rbinom(200, 2, 0.3)
allelefreq <- getAlleleFrequencies(snp)
```

getCausalSNPs *Draw random SNPs from genotypes.*

Description

Draw random SNPs from either simulated genotypes or external genotype files.

Usage

```
getCausalSNPs(NrCausalSNPs = 20, genotypes = NULL, chr = NULL,
  chr_string = NULL, NrChrCausal = NULL, genoFilePrefix = NULL,
  genoFileSuffix = NULL, genoFileDelimiter = ",", sampleID = "ID_",
  standardise = FALSE, verbose = TRUE)
```

Arguments

NrCausalSNPs	number [integer] of SNPs to chose at random
genotypes	[NrSamples x totalNrSNPs] matrix of genotypes
chr	numeric vector of chromosomes to chose NrCausalSNPs from; only used when external genotype data is provided i.e. is.null(genoFilePrefix) == FALSE
chr_string	[string] alternative to chr, a string with chromosomes separated by comma; most often used when run as a command line application
NrChrCausal	Number [integer] of causal chromosomes to chose NrCausalSNPs from (as opposed to the actual chromosomes to chose from via chr 'chr_string); only used when external genotype data is provided i.e. is.null(genoFilePrefix) == FALSE.
genoFilePrefix	full path/to/chromosome-wise-genotype-file-ending- before-"chrChromosomeNumber" (no '~' expansion!) [string]
genoFileSuffix	[string] following chromosome number including .fileformat (e.g. ".csv"); has to be a text format i.e. comma/tab/space separated
genoFileDelimiter	field separator [string] of genotype file
sampleID	prefix [string] for naming samples (followed by sample number from 1 to NrSamples)
standardise	[boolean]; if TRUE standardised genotypes will be returned
verbose	[boolean]; if TRUE, progress info is printed to standard out

Details

In order to chose SNPs from external genotype files without reading them into memory, genotypes for each chromosome need to be accesible as [SNPs x samples] in a separate file, containing "chrChromosomenumber" (e.g chr22) in the file name (e.g. /path/to/dir/related_nopopstructure_chr22.csv). All genotype files need to be saved in the same directory. genoFilePrefix (/path/to/dir/related_nopopstructure_) and genoFileSuffix (.csv) specify the strings leading and following the "chrChromosomenumber". The first column in each file needs to be the SNP_ID and files cannot contain a header. getCausalSNPs generates a vector of chromosomes from which to sample the SNPs . For each of the chromosomes, it counts the number of SNPs in the chromosome file and creates vectors of random numbers ranging from 1:NrSNPSinFile. Only the lines corresponding to these numbers are then read into R. The example data provided for chromosome 22 contains genotypes (50 samples) of the first 500 SNPs on chromosome 22 with a minor allele frequency of greater than 2 from the European populations of the the 1000 Genomes project.

Value

N x NrCausalSNPs matrix of randomly drawn, standardised (depending on standardise option) SNPs

See Also

standardiseGenotypes

Examples

```
geno <- simulateGenotypes(N=10, NrSNP=10)
causalSNPsFromSimulatedGenoStandardised <- getCausalSNPs(NrCausalSNPs=10,
genotypes=geno,
standardise=TRUE)

genotypeFile <- system.file("extdata/genotypes/",
"genotypes_chr22.csv",
package = "PhenotypeSimulator")
genoFilePrefix <- gsub("chr.*", "", genotypeFile)
genoFileSuffix <- ".csv"
causalSNPsFromFile <- getCausalSNPs(NrCausalSNPs=10, chr=22,
genoFilePrefix=genoFilePrefix,
genoFileSuffix=genoFileSuffix)
```

`getKinship` *Get genetic kinship.*

Description

Estimate kinship from standardised genotypes or read pre-computed kinship file. Standardised genotypes can be obtained via `standardiseGenotypes` or directly from the `simulateGenotypes` return object. Kinship matrices can optionally be normalised.

Usage

```
getKinship(X = NULL, kinshipfile = NULL, sampleID = "ID_", norm = TRUE,
  standardise = TRUE, sep = ",", header = TRUE, verbose = TRUE)
```

Arguments

`X`	[NrSamples x totalNrSNPs] matrix of standardised genotypes
`kinshipfile`	path/to/kinshipfile [string] to be read; either X or kinshipfile must be provided
`sampleID`	prefix [string] for naming samples (followed by sample number from 1 to NrSamples)
`norm`	[boolean], if TRUE kinship matrix will be normalised by the mean of its diagonal elements and 1e-4 added to the diagonal for numerical stability
`standardise`	[boolean], if TRUE genotypes will be standardised before kinship estimation
`sep`	field separator [string] of kinship file
`header`	[boolean], if TRUE kinship file has header information
`verbose`	[boolean]; if TRUE, progress info is printed to standard out

Details

The kinship is estimated as $K = XX_T$, with X the standardised genotypes of the samples. When estimating the kinship from the provided genotypes, the kinship should be normalised by the mean of its diagonal elements and 1e-4 added to the diagonal for numerical stability via norm=TRUE. If a kinship file is provided, normalising can optionally be chosen. For the provided kinship file, normalisation has already been done a priori and norm should be set to FALSE. The provided kinship contains estimates for 50 samples across the entire genome.

Value

NrSamples x NrSamples matrix of kinship estimate

Examples

```
geno <- simulateGenotypes(N=10, NrSNP=50)
K_fromGenotypesNormalised <- getKinship(geno$X_sd, norm=TRUE)

kinshipfile <- system.file("extdata/kinship",
"kinship.csv",
package = "PhenotypeSimulator")
K_fromFile <- getKinship(kinshipfile=kinshipfile, norm=FALSE)
```

---

noiseBgEffects *Simulate noise background effects.*

Description

noiseBgEffects simulates two random noise effects (shared and independent).

Usage

```
noiseBgEffects(N, P, mean = 0, sd = 1)
```

Arguments

N	number [integer] of samples to simulate
P	number [integer] of phenotypes to simulate
mean	mean [double] of the normal distribution
sd	standard deviation [double] of the normal distribution

Details

The independent background effect is simulated as vec(shared) ~ N(mean,sd). The shared background effect is simulated as the matrix product of two normal distributions A [N x 1] and B [P x 1]: AB^t

Value

named list of shared background noise effects (shared: [N x P] matrix) and independent background noise effects (independent: [N x P] matrix)

Examples

```
noiseBG <- noiseBgEffects(N=100, P=20, mean=2)
```

`noiseFixedEffects` *Simulate noise fixed effects.*

Description

noiseFixedEffects simulates a number of fixed noise effects (confounders). Confounders can have effects across all traits (shared) or to a number of traits only (independent); the proportion of independent confounders from the total of simulated confounders can be chosen via pIndependentConfounders. The number of traits that are associated with independent noise effects can be chosen via pTraitIndependentConfounders. Confounders can be simulated as categorical variables or following a binomial , uniform or normal distribution. Effect sizes for the noise effects can be simulated from a uniform or normal distribution. Multiple confounder sets drawn from different distributions/different parameters of the same distribution can be simulated by specifying NrFixedEffects and supplying the respective distribution parameters (*Confounders and *Beta) explained below. If a model parameter and its "String' version are provided, i.e. NrConfounders and NrConfoundersStrings, the "String' specification will be used!

Usage

```
noiseFixedEffects(N, P, NrFixedEffects = 1, NrConfounders = 10,
  pIndependentConfounders = 0.4, pTraitIndependentConfounders = 0.2,
  distConfounders = "norm", mConfounders = 0, sdConfounders = 1,
  catConfounders = NULL, probConfounders = NULL, distBeta = "norm",
  mBeta = 0, sdBeta = 1, NrConfoundersString = NULL,
  pIndependentConfoundersString = NULL,
  pTraitIndependentConfoundersString = NULL, distConfoundersString = NULL,
  mConfoundersString = NULL, sdConfoundersString = NULL,
  catConfoundersString = NULL, probConfoundersString = NULL,
  distBetaString = NULL, mBetaString = NULL, sdBetaString = NULL)
```

Arguments

`N`	number [integer] of samples to simulate
`P`	number [integer] of phenotypes to simulate
`NrFixedEffects`	number [integer] of different fixed effects to simulate ; allows to simulate fixed effects from different distributions or with differen parameters
`NrConfounders`	vector of number(s) [integer] of confounders to simulate

`pIndependentConfounders`
: vector of proportion(s) [double] of noise effects (confounders) to have a trait-independent effect

`pTraitIndependentConfounders`
: vector of proportion(s) [double] of traits influenced by independent fixed noise effects

`distConfounders`
: vector of name(s) [string] of distribution to use to simulate confounders; one of "unif", "norm", "bin", "cat_norm", "cat_unif"

`mConfounders`
: vector of mean/midpoint(s) [double] of normal/uniform distribution for confounders

`sdConfounders`
: vector of standard deviation(s)/distance from midpoint(s) [double] of normal/uniform distribution for confounders

`catConfounders`
: vector of number(s) of confounder categories [integer]; required if distConfounders "cat_norm" or "cat_unif"

`probConfounders`
: vector of probability(s) [double] of binomial confounders (0/1); required if distConfounders "bin"

`distBeta`
: vector of name(s) [string] of distribution to use to simulate effect sizes of confounders; one of "unif" or "norm"

`mBeta`
: vector of mean/midpoint [double] of normal/uniform distribution for effect sizes of confounders

`sdBeta`
: vector of standard deviation/distance from midpoint [double] of normal/uniform distribution for effect sizes of confounders

`NrConfoundersString`
: alternative to NrConfounder, a comma-separated [string] with number(s) [integer] of confounders to simulate; typically used when run as command line application

`pIndependentConfoundersString`
: alternative to pIndependentConfounders, a comma-separated [string] with proportion(s) [double] of noise effects (confounders) to have a trait-independent effect; typically used when run as command line application

`pTraitIndependentConfoundersString`
: alternative to pTraitIndependentConfounders, a comma-separated [string] with proportion(s) [double] of traits influenced by independent fixed noise effects; typically used when run as command line application

`distConfoundersString`
: alternative to distConfounders, a comma- separated [string] with name(s) [string] of distributions to use to simulate confounders; one of "unif", "norm", "bin", "cat_norm", "cat_unif"; typically used when run as command line application

`mConfoundersString`
: alternative to mConfounders, a comma- separated [string] with of mean/midpoint(s) [double] of normal/uniform distribution for confounders; typically used when run as command line application

`sdConfoundersString`
: alternative to sdConfounders, a comma- separated [string] with standard deviation(s)/distance from midpoint(s) [double] of normal/uniform distribution for confounders; typically used when run as command line application

`catConfoundersString`
: alternative to catConfounders, a comma- separated [string] with the number of confounder categories [integer]; required if distConfounders "cat_norm" or "cat_unif"; typically used when run as command line application

`probConfoundersString`
: alternative to probConfounders, a comma- separated [string] with probability(s) [double] of binomial confounders (0/1); required if distConfounders "bin"; typically used when run as command line application

`distBetaString`
: alternative to distBeta, a comma-separated [string] with name(s) [string] of distribution to use to simulate effect sizes of confounders; one of "unif" or "norm"; typically used when run as command line application

`mBetaString`
: alternative to mBeta, a comma- separated [string] with means/midpoints [double] of normal/uniform distribution for effect sizes of confounders; typically used when run as command line application

`sdBetaString`
: alternative to sdBeta, a comma- separated [string] with standard deviation/distance from midpoint [double] of normal/uniform distribution for effect sizes of confounders; typically used when run as command line application

Value

named list of shared fixed noise effects (shared: [N x P] matrix), independent fixed noise effects (independent: [N x P] matrix), the causal SNPs named by having a shared or independent effect (cov: [NrConfounders x N] matrix) and the simulated effect sizes of the confounders (cov_effect: [P x NrConfounders] dataframe)

See Also

`simulateDist`

Examples

```
# fixed noise effect with default setting
noiseFE <- noiseFixedEffects(P=5, N=20)

# 1 categorical fixed noise effect with uniform distribution of the
# categories
noiseFE_catUnif <- noiseFixedEffects(P=10, N=20, NrConfounders=1,
distConfounders="cat_unif", catConfounders=3)

# 10 fixed noise effect with uniform distribution between 1 and 5 (3 +/- 2)
# categories
noiseFE_uniformConfounders_normBetas <- noiseFixedEffects(P=10, N=20,
NrConfounders=10, distConfounders="unif", mConfounders=3, sdConfounders=2,
distBeta="norm",  sdBeta=2)
```

```
 # 4 fixed noise effect with binomial distribution with p=0.2
noiseFE_binomialConfounders_uniformBetas <- noiseFixedEffects(P=10, N=20,
NrConfounders=4, distConfounders="bin", probConfounders=0.2, distBeta="norm",
sdBeta=2)

 # 4 fixed noise effect with 2 binomial confounders and 2 normally
 # distributed confounders
 noiseFE_binomialandNormalConfounders <- noiseFixedEffects(P=10, N=20,
 NrFixedEffects=2, NrConfounders=c(2,2), distConfounders=c("bin", "norm"),
 probConfounders=0.2)
```

`PhenotypeSimulator` *PhenotypeSimulator: A package for simulating phenotypes from different genetic and noise components*

Description

PhenotypeSimulator: A package for simulating phenotypes from different genetic and noise components

---

`read_lines` *Scan file for specific line numbers*

Description

Scan file for specific line numbers

Usage

```
read_lines(filename, lines, sep = "\n")
```

Arguments

`filename`	/path/to/chromosomefile [string]
`lines`	vector of line numbers [integer] to be read
`sep`	[string] end-of-line delimiter

rescaleVariance *Scale phenotype component.*

Description

The function scales the specified phenotype component such that the average column variance is equal to the user-specified proportion of variance.

Usage

```
rescaleVariance(component, propvar)
```

Arguments

component	numeric [N x P] phenotype matrix where N are the number of observations and P numer of phenotypes
propvar	numeric specifying the proportion of variance that should be explained by this phenotype component

Value

component_scaled numeric [N x P] phenotype matrix if propvar != 0 or NULL

Examples

```
x <- matrix(rnorm(100), nc=10)
x_scaled <- rescaleVariance(x, propvar=0.4)
```

runSimulation *Run phenotype simulation.*

Description

runSimulation wraps around the phenotype component functions (genFixedEffects , genBgEffects, noiseBgEffects, noiseFixedEffects and correlatedBgEffects) and combines the simulated phenotype components via create phenotype.

Usage

```
runSimulation(N = 1000, P = 10, tNrSNP = 5000, cNrSNP = 20,
  NrConfounders = 10, seed = 219453, chr_string = NULL, chr = NULL,
  NrChrCausal = NULL, genVar = NULL, h2s = NULL, theta = 0.8,
  h2bg = NULL, eta = 0.8, noiseVar = NULL, rho = NULL, delta = NULL,
  gamma = 0.8, phi = NULL, alpha = 0.8, sampleID = "ID_",
  phenoID = "Trait_", genoFilePrefix = NULL, genoFileSuffix = NULL,
  genoFileDelimiter = ",", kinshipfile = NULL, kinshipHeader = TRUE,
```

```
kinshipDelimiter = ",", normalise = TRUE, standardise = TRUE,
NrFixedEffects = 1, distConfounders = "norm", mConfounders = 0,
sdConfounders = 1, catConfounders = NULL, probConfounders = NULL,
distBeta = "norm", mBeta = 0, sdBeta = 1,
pIndependentConfounders = 0.4, pTraitIndependentConfounders = 0.2,
pcorr = 0.8, meanNoiseBg = 0, sdNoiseBg = 1, SNPfrequencies = c(0.1,
0.2, 0.4), SNPfrequencyString = NULL, pIndependentGenetic = 0.4,
pTraitIndependentGenetic = 0.2, NrConfoundersString = NULL,
pIndependentConfoundersString = NULL,
pTraitIndependentConfoundersString = NULL, distConfoundersString = NULL,
mConfoundersString = NULL, sdConfoundersString = NULL,
catConfoundersString = NULL, probConfoundersString = NULL,
distBetaString = NULL, mBetaString = NULL, sdBetaString = NULL,
verbose = TRUE)
```

Arguments

N	number [integer] of samples to simulate
P	number [integer] of phenotypes to simulate
tNrSNP	total number [integer] of SNPs to simulate; these SNPs are used for kinship estimation
cNrSNP	number [integer] of causal SNPs; used as genetic fixed effects
NrConfounders	number [integer] of confounders; used as noise fixed effects
seed	seed [integer] to initiate random number generation
chr_string	alternative to chr, a [string] with chromosomes separated by comma; most often used when run as a command line application.
chr	numeric vector of chromosomes to chose NrCausalSNPs from; only used when external genotype data is provided i.e. is.null(genoFilePrefix) == FALSE
NrChrCausal	Number [integer] of causal chromosomes to chose NrCausalSNPs from (as opposed to the actual chromosomes to chose from via chr 'chr_string); only used when external genotype data is provided i.e. is.null(genoFilePrefix) == FALSE.
genVar	Proportion [double] of total genetic variance
h2s	Proportion [double] of gentic variance of fixed effects
theta	Proportion [double] of variance of shared fixed genetic effects
h2bg	Proportion [double] of genetic variance of background effects; either h2s or h2bg have to be specified and h2s + h2bg = 1
eta	Proportion [double] of variance of shared bg genetic effects
noiseVar	Proportion [double] of total noise variance
rho	Proportion [double] of noise variance of correlated effects; sum of rho, delta and phi cannot be greater than 1
delta	Proportion [double] of noise variance of fixed effects; sum of rho, delta and phi cannot be greater than 1
gamma	Proportion [double] of variance of shared fixed noise effects

phi	Proportion [double] of noise variance of background effects; sum of rho, delta and phi cannot be greater than 1
alpha	Variance [double] of shared bg noise effect
sampleID	prefix [string] for naming samples (followed by sample number from 1 to N)
phenoID	prefix [string] for naming traits (followed by trait number from 1 to P)
genoFilePrefix	path/to/chromosome-wise-genotype-file-ending-before- "chrChromosomeNumber" [string]
genoFileSuffix	[string] following chromosome number including .fileformat (e.g. ".csv")
genoFileDelimiter	field separator [string] of genotype file
kinshipfile	path/to/kinshipfile [string] to be read; either X or kinshipfile must be provided
kinshipHeader	[boolean], if TRUE kinship file has header information
kinshipDelimiter	field separator [string] of kinship file
normalise	[boolean], if TRUE kinship matrix will be normalised by the mean of its diagonal elements and 1e-4 added to the diagonal for numerical stability
standardise	[boolean], if TRUE standardised genotypes will be returned
NrFixedEffects	number [integer] of different fixed effects to simulate ; allows to simulate fixed effects from different distributions or with different parameters
distConfounders	name [string] of distribution to use to simulate confounders; one of "unif", "norm", "bin", "cat_norm", "cat_unif"
mConfounders	mean/midpoint [double] of normal/uniform distribution for confounders
sdConfounders	standard deviation/extension from midpoint [double] of normal/uniform distribution for confounders
catConfounders	confounder categories [factor]; required if distConfounders "cat_norm" or "cat_unif"
probConfounders	probability [double] of binomial confounders (0/1); required if distConfounders "bin"
distBeta	name [string] of distribution to use to simulate effect sizes of confounders; one of "unif" or "norm"
mBeta	mean/midpoint [double] of normal/uniform distribution for effect sizes of confounders
sdBeta	standard deviation/extension from midpoint [double] of normal/ uniform distribution for effect sizes of confounders
pIndependentConfounders	Proportion [double] of noise effects (confounders) to have a trait-independent effect
pTraitIndependentConfounders	Proportion [double] of traits influenced by independent fixed noise effects
pcorr	Correlation [double] between phenotypes
meanNoiseBg	mean [double] of the normal distribution for noise bg effects

`sdNoiseBg` standard deviation [double] of the normal distribution for noise bg effects

`SNPfrequencies` vector of allele frequencies [double] from which to sample

`SNPfrequencyString`
alternative to a frequencies vector, a [string] with frequencies separated by comma

`pIndependentGenetic`
Proportion [double] of genetic effects (SNPs) to have a trait-independent fixed effect

`pTraitIndependentGenetic`
Proportion [double] of traits influenced by independent fixed genetic effects

`NrConfoundersString`
alternative to NrConfounder, a comma-separated [string] with number(s) [integer] of confounders to simulate; typically used when run as command line application

`pIndependentConfoundersString`
alternative to pIndependentConfounders, a comma-separated [string] with proportion(s) [double] of noise effects (confounders) to have a trait-independent effect; typically used when run as command line application

`pTraitIndependentConfoundersString`
alternative to pTraitIndependentConfounders, a comma-separated [string] with proportion(s) [double] of traits influenced by independent fixed noise effects; typically used when run as command line application

`distConfoundersString`
alternative to distConfounders, a comma- separated [string] with name(s) [string] of distributions to use to simulate confounders; one of "unif", "norm", "bin", "cat_norm", "cat_unif"; typically used when run as command line application

`mConfoundersString`
alternative to mConfounders, a comma- separated [string] with of mean/midpoint(s) [double] of normal/uniform distribution for confounders; typically used when run as command line application

`sdConfoundersString`
alternative to sdConfounders, a comma- separated [string] with standard deviation(s)/distance from midpoint(s) [double] of normal/uniform distribution for confounders; typically used when run as command line application

`catConfoundersString`
alternative to catConfounders, a comma- separated [string] with the number of confounder categories [integer]; required if distConfounders "cat_norm" or "cat_unif"; typically used when run as command line application

`probConfoundersString`
alternative to probConfounders, a comma- separated [string] with probability(s) [double] of binomial confounders (0/1); required if distConfounders "bin"; typically used when run as command line application

`distBetaString` alternative to distBeta, a comma-separated [string] with name(s) [string] of distribution to use to simulate effect sizes of confounders; one of "unif" or "norm"; typically used when run as command line application

`mBetaString` alternative to mBeta, a comma- separated [string] with means/midpoints [double] of normal/uniform distribution for effect sizes of confounders; typically used when run as command line application

sdBetaString	alternative to sdBeta, a comma- separated [string] with standard deviation/distance from midpoint [double] of normal/uniform distribution for effect sizes of confounders; typically used when run as command line application
verbose	[boolean]; if TRUE, progress info is printed to standard out

Value

named list of absolute levels of variance explained by each phenotype component (varComponents), the scaled phenotype components (phenoComponents) and and parameters used for phenotype setup and labeling (setup)

See Also

createPheno, which this function wraps

Examples

```
# simulate phenotype of 100 samples, 10 traits from genetic and noise
# background effects, with variance explained of 0.2 and 0.8 respectively
genVar = 0.2
simulatedPhenotype <- runSimulation(N=100, P=5, cNrSNP=10,
genVar=genVar, h2s=1, phi=1)
```

---

savePheno *Save final phenotype and phenotype components.*

Description

savePheno saves model setup parameters and simulated genotypes to the specified directories. Requires a simulatedData list which is the output of either runSimulation or createPheno

Usage

```
savePheno(simulatedData, directoryGeno, directoryPheno,
  sample_subset_vec = NULL, pheno_subset_vec = NULL,
  sample_subset_string = NULL, pheno_subset_string = NULL,
  outstring = NULL, saveAsTable = TRUE, saveAsRDS = FALSE,
  saveAsPlink = FALSE, verbose = TRUE)
```

Arguments

simulatedData	named list of i) dataframe of proportion of variance explained for each component (varComponents), ii) a named list with the simulated phenotype components (phenoComponents) and iii) a named list of parameters describing the model setup (setup); obtained from either runSimulation or createPheno
directoryGeno	absolute path (no tilde expansion) to parent directory [string] where genotypes from simulations should be saved [needs user writing permission]

directoryPheno absolute path (no tilde expansion) to parent directory [string] where final phenotype and phenotype components from simulations should be saved [needs user writing permission]

sample_subset_vec optional vector of sample subset sizes [integer]; if provided, draws subsets of samples out of the total simulated dataset and saves them separately

pheno_subset_vec optional vector of phenotype subset sizes [integer] if provided, draws subsets of traits out of the total simulated dataset and saves them separately

sample_subset_string optional [string] of comma-separated numbers e.g. "50,100,500"; alternative to sample_subset_vec, typically used when run as command line application

pheno_subset_string optional [string] of comma-separated numbers e.g. "10,50,80"; alternative to pheno_subset_vec, typically used when run as command line application

outstring optional name [string] of subdirectory (in relation to directoryPheno/directoryGeno) to save set-up independent simulation results

saveAsTable [boolean] if TRUE, data will be saved as .csv

saveAsRDS [boolean] if TRUE, data will be saved as .RDS; at least one of 'saveAsTable' or 'saveAsRDS' has to be TRUE, both can be TRUE

saveAsPlink [boolean] if TRUE, simulated genotype data will be additionally be saved in binary PLINK format: .bed, .bim and .fam

verbose [boolean]; if TRUE, progress info is printed to standard out

Value

list of paths [strings] to final output phenotype (directoryPheno) and genotype (directoryGeno) directories. If outstring or subset settings not NULL, these directories will be subdirectories of the input phenotype and genotype directories.

Examples

```
simulatedPhenotype <- runSimulation(N=100, P=5, cNrSNP=10,
genVar=0.2, h2s=1, phi=1)
#not run
#outputdir <- savePheno(simulatedPhenotype, directoryGeno="/path/to/dir/",
#directoryPheno="/path/to/dir/", outstring="Date_simulation",
#saveAsPlink=TRUE)
```

---

setModel *Set simulation model.*

Description

Based on parameters provided, this function sets the name for the phenotype simulation. The model name is needed downstream for phenotype component simulations, but can also be set manually.

Usage

```
setModel(genVar = NULL, h2s = NULL, theta = 0.8, h2bg = NULL,
  eta = 0.8, noiseVar = NULL, delta = NULL, gamma = 0.8, rho = NULL,
  phi = NULL, alpha = 0.8, pcorr = 0.6, pIndependentConfounders = 0.4,
  pTraitIndependentConfounders = 0.2, pIndependentGenetic = 0.4,
  pTraitIndependentGenetic = 0.2, v = TRUE)
```

Arguments

genVar	Total genetic variance [double]
h2s	Proportion [double] of variance of fixed genetic effects
theta	Proportion [double] of variance of shared fixed genetic effects
h2bg	Proportion [double] of variance of random genetic effects
eta	Proportion [double] of variance of shared bg genetic effects
noiseVar	Total genetic variance [double]
delta	Proportion [double] of fixed noise variance
gamma	Proportion [double] of variance of shared fixed noise effects
rho	Proportion [double] of variance of correlated noise effects
phi	Proportion [double] of variance of background noise effects
alpha	Proportion [double] of Variance of shared bg noise effect
pcorr	Correlation [double] between phenotypes
pIndependentConfounders	Proportion [double] of noise effects (confounders) to have a trait-independent effect
pTraitIndependentConfounders	Proportion [double] of traits influenced by independent fixed noise effects
pIndependentGenetic	Proportion [double] of genetic effects (SNPs) to have a trait-independent fixed effect
pTraitIndependentGenetic	Proportion [double] of traits influenced by independent fixed genetic effects
v	[boolean]; if TRUE, progress info is printed to standard out

Value

named list containing the genetic model (modelGenetic) and the noise model (modelNoise). Options are: modelNoise: "nNoise", "noiseFixedOnly", "noiseBgOnly", "noiseCorrelatedOnly", "noiseFixedAndBg","noiseCorrelatedAndBg", "noiseFixedAndCorrelated", "noiseFixedAndBgAndCorrelated" modelGenetic: "noGenetic","geneticBgOnly", "geneticFixedOnly", "geneticFixedAndBg"

Examples

```
#genetic fixed effects only
model <- setModel(genVar=1, h2s=1)

#genetic fixed and bg effects
model <- setModel(genVar=1, h2s=0.01)

#genetic and noise fixed effects only
model <- setModel(genVar=0.4, h2s=1, delta=1)
```

`simulateDist` *Data simulation for different distributions.*

Description

Wrapper function to simulate data from different distribution with different parameter settings.

Usage

```
simulateDist(x, dist = c("unif", "norm", "bin", "cat_norm", "cat_unif"),
  m = NULL, std = 1, categories = NULL, prob = NULL)
```

Arguments

`x`	the number [integer] of observations to simulate
`dist`	name of distribution [string] from which the observations are drawn. 'norm' is the normal distribution, 'unif' the uniform distribution 'bin' the binomial distribution, "cat_norm" samples categorical variables according to a normal distribution and "cat_unif" according to a uniform distribution. For "cat_norm", length(category)/2 is used mean for the normal distribution unless specified otherwise.
`m`	the mean of the normal distribution [double]/the mean between min and max for the uniform distribution [double]/ the rank of the category to be used as mean for "cat_norm" [integer]
`std`	the standard deviation of the normal distribution or the distance of min/max from the mean for the uniform distribution [double]
`categories`	number of categories [integer] for simulating categorical variables (for distr="cat_norm" or "cat_unif")
`prob`	the probability [double] of success for each trial (for distr="bin")

Value

numeric vector of length [x] with the sampled values

See Also

`runif`, `rnorm`, `rbinom` for documentation of the underlying distributions

Examples

```
normal <- simulateDist(x=10, dist="norm", m=2, std=4)
cat_normal <- simulateDist(x=10, dist="cat_norm", categories=5)
cat_uniform <- simulateDist(x=10, dist="cat_unif", categories=5)
uniform <- simulateDist(x=10, dist="unif", m=4, std=1)
binomial <- simulateDist(x=10, dist="bin", prob=0.4)
```

`simulateGenotypes` *Simulate bi-allelic genotypes.*

Description

Simulate bi-allelic genotypes.

Usage

```
simulateGenotypes(N, NrSNP = 5000, frequencies = c(0.1, 0.2, 0.4),
  sampleID = "ID_", verbose = TRUE, frequencyString = NULL)
```

Arguments

`N`	number of samples for which to simulate bi-allelelic genotypes
`NrSNP`	number of SNPs to simulate
`frequencies`	vector of allele frequencies [double] from which to sample
`sampleID`	prefix for naming samples (followed by sample number from 1 to N)
`verbose`	boolean; if TRUE, progress info is printed to standard out
`frequencyString`	alternative to a frequencies vector, a [string] with frequencies separated by comma can be supplied; most often used when run as a command line application

Value

list of a [N x NrSNP] matrix of simulated genotypes, [N x NrSNP] matrix of standardised simulated genotypes and vector of sample IDs

See Also

`standardiseGenotypes`

Examples

```
N10NrSNP10 <- simulateGenotypes(N=10, NrSNP=10)
N10NrSNP10 <- simulateGenotypes(N=10, NrSNP=10,
frequencyString="0.2,0.3,0.4")
```

simulatePhenotypes *Command line execution for PhenotypeSimulator*

Description

simulatePhenotypes runs without arguments. Upon call, it reads command-line parameters and supplies these to runSimulation and savePheno. For details on input to runSimulation and savePheno, please refer to their help pages. For help on the command line arguments that can be passed, see examples below. From the command line, the help function can be called via 'Rscript -e "PhenotypeSimulator::simulatePhenotypes()" –args –help

Usage

```
simulatePhenotypes()
```

Examples

```
# (not run)
# Simulate simple phenotype of genetic and noise background effects only:
# (not run)
# Rscript -e "PhenotypeSimulator::simulatePhenotypes()" \
#--args \
#--NrSamples=100 --NrPhenotypes=15 \
#--tNrSNP=10000 --cNrSNP=30 \
#--SNPfrequencies=0.05,0.1,0.3,0.4 \
#--genVar=0.4 --h2s=0.025 --phi=0.6 --delta=0.3 --gamma=1 \
#--pcorr=0.8 \
#--NrFixedEffects=4 --NrConfounders=1,2,1,2 \
#--pIndependentConfounders=0,1,1,0.5 \
#--distConfounders=bin,cat_norm,cat_unif,norm \
#--probConfounders=0.2 \
#--catConfounders=0,3,4,0 \
#--directoryGeno=~/tmp/genotypes \
#--directoryPheno=~/tmp/phenotypes \
#--sampleSubset=50,70 \
#--phenoSubset=5,10 \
#--normalise \
#--showProgress \
#--saveTable \
#--savePlink
```

standardiseGenotypes *Standardise genotypes*

Description

Genotypes are standardised as described in Yang et al: snp_standardised = (snp - 2 * minor_allele_freq)/ sqrt(2 * minor_allele_freq * major_allele_freq)

Usage

```
standardiseGenotypes(geno)
```

Arguments

geno	[N x NrSNP] matrix/dataframe of genotypes

Value

N x NrSNP matrix of standardised genotypes

References

Yang, J., Lee, S.H., Goddard, M.E., Visscher, P.M. (2011) GCTA: a tool for genome-wide complex trait analysis, AJHG: 88

See Also

getAlleleFrequencies

Examples

```
geno <- cbind(rbinom(100, 2, 0.3), rbinom(100, 2, 0.4))
geno_sd <- standardiseGenotypes(geno)
```

---

vmessage *Print userinfo.*

Description

Wrapper function around message that allows to turn the printing of messages to standard out. on or off

Usage

```
vmessage(userinfo, verbose = TRUE, sep = " ")
```

Arguments

userinfo	string or vector of string elements and variables
verbose	[boolean], if TRUE message is displayed on standard out, if FALSE, message is suppressed
sep	delimiter [string] to separate message elements when userinfo given as vector

See Also

message which this function wraps

Index