Package 'PhenotypeSimulator'

January 20, 2025

Title Flexible Phenotype Simulation from Different Genetic and Noise Models

Version 0.3.4

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

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

Description Simulation is a critical part of method development and assessment in quantitative genetics. 'PhenotypeSimulator' allows for the flexible simulation of phenotypes under different models, including genetic variant and infinitesimal genetic effects (reflecting population structure) as well as non-genetic covariate effects, observational noise and additional correlation effects. The different phenotype components are combined into a final phenotype while controlling for the proportion of variance explained by each of the components. For each effect component, the number of variables, their distribution and the design of their effect across traits can be customised. For the simulation of the genetic effects, external genotype data from a number of standard software ('plink', 'hapgen2'/ 'impute2', 'genome', 'bimbam', simple text files) can be imported. The final simulated phenotypes and its components can be automatically saved into .rds or .csv files. In addition, they can be saved in formats compatible with commonly used genetic association software ('gemma', 'bimbam', 'plink', 'snptest', 'LiMMBo').

Depends R (>= 3.5.0)

LinkingTo Rcpp

Imports methods, optparse, Hmisc, R.utils, mvtnorm, snpStats, zoo, data.table (>= 1.11.0), Rcpp (>= 0.12.11), cowplot, ggplot2, reshape2, dplyr

Suggests testthat, knitr, formatR, rmarkdown

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.1.1

VignetteBuilder knitr

NeedsCompilation yes

Author Hannah Meyer [aut, cre] (<https://orcid.org/0000-0003-4564-0899>),
Konrad Rudolph [ctb] (<https://orcid.org/0000-0002-9866-7051>)

Maintainer Hannah Meyer <hannah.v.meyer@gmail.com>

Repository CRAN

Date/Publication 2021-07-16 13:30:02 UTC

Contents

`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, logical or character values.

Usage

```
commaList2vector(commastring = NULL, type = "numeric")
```

Arguments

`commastring`	Input [character] vector containing numbers separated by commas.
`type`	Name [string] of the type of input variables; one of numeric, logical or character

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 = NULL,
  corr_mat = NULL,
  sampleID = "ID_",
  phenoID = "Trait_",
  id_samples = NULL,
  id_phenos = NULL,
  verbose = FALSE
)
```

Arguments

`N`	Number [integer] of samples to simulate.
`P`	Number [integer] of phenotypes to simulate.
`pcorr`	Initial strength of correlation [double] between neighbouring traits. Decreases by pcorr^(distance); distance from 0 to P-1. See details.
`corr_mat`	[P x P] correlation matrix [double] as covariance component for the multivariate normal distribution. If not provided, pcorr is used to construct the correlation matrix.
`sampleID`	Prefix [string] for naming samples.
`phenoID`	Prefix [string] for naming traits.
`id_samples`	Vector of [NrSamples] sample IDs [string]; if not provided constructed by paste(sampleID, 1:N, sep="").
`id_phenos`	Vector of [NrTraits] phenotype IDs [string]; if not provided constructed by paste(phenoID, 1:P, sep="").
`verbose`	[boolean] If TRUE, progress info is printed to standard out.

Details

correlatedBgEffects can be used to simulate phenotypes with a defined level of correlation between traits. If the corr_mat is not provided, a simple correlation structure based on the distance of the traits will be constructed. 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=$pcorr^{(P-1)}$ and 0 < pcorr < 1. The correlated background effect correlated is simulated based on this correlation structure C: $correlated\ N_{NP}(0, C)$.

Value

Named list with [N x P] matrix of correlated background effects (correlatedBg) and the correlation matrix (cov_correlated). If corr_mat provided corr_mat == cov_correlated.

See Also

`rmvnorm` which is used to simulate the multivariate normal distribution

Examples

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

expGen2probGen *Rewrite expected genotypes into genotype probabilities.*

Description

Convert genotype frequencies to genotypes encoded as triplets of probablities (p(AA), p(Aa), p(aa)).

Usage

```
expGen2probGen(geno)
```

Arguments

geno	Vector [numeric] with genotypes

Value

Numeric vector of length [length(geno)*3] with the genotype encoded as probabbilities (p(AA), p(Aa), p(aa)).

Examples

```
nrSamples <- 10
# Simulate binomial SNP with 0.2 allele frequency
geno <- rbinom(nrSamples, 2, p=0.2)
geno_prob<- expGen2probGen(geno)
```

geneticBgEffects *Simulate infinitesimal genetic effects (reflecting sample kinship).*

Description

geneticBgEffects simulates an infinitesimal genetic effects with a proportion of the effect shared across samples and a proportion independent across samples; they are based on the kinship estimates of the (simulated) samples.

Usage

```
geneticBgEffects(
  P,
  N,
  kinship,
  phenoID = "Trait_",
  id_samples = colnames(kinship),
  shared = TRUE,
  independent = TRUE,
  id_phenos = NULL
)
```

Arguments

`P`	Number [integer] of phenotypes to simulate .
`N`	Number [integer] of samples to simulate; has to be provided as a dimnesionality check for kinship and downstream analyses; nrow(kinship) has to be equal to N.
`kinship`	[N x N] Matrix of kinship estimates [double].
`phenoID`	Prefix [string] for naming traits.
`id_samples`	Vector of [NrSamples] sample IDs [string]; if not provided colnames(kinship) are used.
`shared`	[bool] shared effect simulated if set to TRUE; at least one of shared or independent has to be set to TRUE.
`independent`	[bool] independent effect simulated if set to TRUE.
`id_phenos`	Vector of [NrTraits] phenotype IDs [string]; if not provided constructed by paste(phenoID, 1:P, sep="").

Details

For the simulation of the infinitesimal genetic effects, three matrix components are used: i) the kinship matrix K [N x N] which is treated as the sample design matrix, ii) 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. To construct the final effects, the three matrices are multiplied as: E = cholesky(K)BA^T.

Value

Named list of shared infinitesimal genetic effects (shared: [N x P] matrix) and independent infinitesimal genetic effects (independent: [N x P] matrix), the covariance term of the shared effect (cov_shared: [P x P] matrix), the covariance term of the independent effect (cov_independent: [P x P] matrix), the eigenvectors (eigenvec_kinship: [N x N]) and eigenvalues (eigenval_kinship: [N]) of the kinship matrix.

Examples

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

`geneticFixedEffects` *Simulate genetic variant effects.*

Description

geneticFixedEffects takes genetic variants which should be added as genetic variant effects to the phenotype. These variants can have the same effects across all traits (shared) or can be independent across traits (independent); in addition, only a certain proportion of traits can be affected by the genetic variants.

Usage

```
geneticFixedEffects(
  X_causal,
  P,
  N,
  phenoID = "Trait_",
  id_samples = rownames(X_causal),
  id_phenos = NULL,
  pTraitsAffected = 1,
  pIndependentGenetic = 0.4,
  pTraitIndependentGenetic = 0.2,
  keepSameIndependent = FALSE,
  distBeta = "norm",
  mBeta = 0,
  sdBeta = 1,
  verbose = FALSE
)
```

Arguments

`X_causal`	[N x NrCausalSNPs] Matrix of [NrCausalSNPs] SNPs from [N] samples.
`P`	Number [integer] of phenotypes to simulate.
`N`	Number [integer] of samples to simulate; has to be provided as a dimnesionality check for X_causal and downstream analyses; nrow(X_causal) has to be equal to N.
`phenoID`	Prefix [string] for naming traits.
`id_samples`	Vector of [NrSamples] sample IDs [string]; if not provided colnames(X_causal) used.
`id_phenos`	Vector of [NrTraits] phenotype IDs [string]; if not provided constructed by paste(phenoID, 1:P, sep="").
`pTraitsAffected`	Proportion [double] of traits affected by the genetic effect. For non-integer results of pTraitsAffected*P, the ceiling of the result is used. Allows to simulate for instance different levels of pleiotropy.
`pIndependentGenetic`	Proportion [double] of genetic effects (SNPs) to have an independent fixed effect.
`pTraitIndependentGenetic`	Proportion [double] of traits influenced by independent fixed genetic effects.
`keepSameIndependent`	[boolean] If set to TRUE, the independent genetic effects always influence the same subset of traits.
`distBeta`	Vector of name(s) [string] of distribution to use to simulate effect sizes of SNPs; one of "unif" or "norm".
`mBeta`	Vector of mean/midpoint(s) [double] of normal/uniform distribution for effect sizes of SNPs.

`sdBeta`	Vector of standard deviation/distance from midpoint [double] of normal/uniform distribution for effect sizes of SNPs.
`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 labeled as 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(N=100, genotypes=genotypes$genotypes)
geneticFixed <- geneticFixedEffects(N=100, X_causal=causalSNPs,
P=10)
```

`getAlleleFrequencies` *Compute allele frequencies from genotype data.*

Description

Compute allele frequencies from genotype data.

Usage

```
getAlleleFrequencies(snp)
```

Arguments

`snp`	[N x 1] Vector of length [N] samples with genotypes of a single bi-allelic genetic variant/SNP encoded as 0,1 and 2.

Value

Vector with ref (0-encoded) and alt (1-encoded) allele frequencies.

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 genotypes provided or external genotype files. When drawing from external genotype files, only lines of randomly chosen SNPs are read, which is recommended for large genotype files. See details for more information. The latter option currently supports file in simple delim-formats (with specified delimiter and optional number of fields to skip) and the bimbam and the oxgen format.

Usage

```
getCausalSNPs(
  N,
  NrCausalSNPs = 20,
  genotypes = NULL,
  chr = NULL,
  NrSNPsOnChromosome = NULL,
  NrChrCausal = NULL,
  genoFilePrefix = NULL,
  genoFileSuffix = NULL,
  format = "delim",
  delimiter = ",",
  header = FALSE,
  skipFields = NULL,
  probabilities = FALSE,
  sampleID = "ID_",
  verbose = TRUE
)
```

Arguments

`N`	Number [integer] of samples to simulate.
`NrCausalSNPs`	Number [integer] of SNPs to chose at random.
`genotypes`	[NrSamples x totalNrSNPs] Matrix of genotypes [integer]/ [double].
`chr`	Vector of chromosome(s) [integer] to chose NrCausalSNPs from; only used when external genotype data is provided i.e. !is.null(genoFilePrefix).
`NrSNPsOnChromosome`	Vector of number(s) of SNPs [integer] per entry in chr (see above); has to be the same length as chr. If not provided, number of SNPS in file will be determined from line count (which can be slow for large files); (optional) header lines will be ignored, so accurate number of SNPs not lines in file should be specified.
`NrChrCausal`	Number [integer] of causal chromosomes to sample NrCausalSNPs from (as opposed to the actual chromosomes to chose from via chr); only used when external genotype data is provided i.e. !is.null(genoFilePrefix).

`genoFilePrefix`	full path/to/chromosome-wise-genotype-file-ending- before-"chrChromosomeNumber" (no '~' expansion!) [string].
`genoFileSuffix`	[string] Following chromosome number including .fileformat (e.g. ".csv"); File described by genoFilePrefix-genoFileSuffix has to be a text format i.e. comma/tab/space separated.
`format`	Name [string] of genotype file format. Options are: "oxgen", "bimbam" or "delim". See readStandardGenotypes for details.
`delimiter`	Field separator [string] of genotypefile or genoFilePrefix-genoFileSuffix file if format == 'delim'.
`header`	[logical] Can be set to indicate if genoFilePrefix-genoFileSuffix file has a header for format == 'delim'. See details.
`skipFields`	Number [integer] of fields (columns) to skip in genoFilePrefix-genoFileSuffix file if format == 'delim'. See details.
`probabilities`	[boolean]. If set to TRUE, the genotypes in the files described by genoFilePrefix-genoFileSuffix are provided as triplets of probabilities (p(AA), p(Aa), p(aa)) and are converted into their expected genotype frequencies by 0*p(AA) + p(Aa) + 2p(aa) via probGen2expGen.
`sampleID`	Prefix [string] for naming samples (will be followed by sample number from 1 to N when constructing id_samples)
`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". If format== delim, the first column in each file needs to be the SNP_ID, the first row can either contain sample IDs or the first row of genotypes (specified with header). Subsequent columns containing additional SNP information can be skipped by setting skipFields. If format==oxgen or bimbam, files need to be in the oxgen or bimbam format (see readStandardGenotypes for details) and no additional information about delim, header or skipFields will be considered. 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 Genomes project.

Value

[N x NrCausalSNPs] Matrix of randomly drawn genotypes [integer]/ [double]

See Also

`standardiseGenotypes`

Examples

```
# get causal SNPs from genotypes simulated within PhenotypeSimulator
geno <- simulateGenotypes(N=10, NrSNP=10)
causalSNPsFromSimulatedGenoStandardised <- getCausalSNPs(N=10,
NrCausalSNPs=10, genotypes=geno$genotypes)

# Get causal SNPs by sampling lines from large SNP files
genotypeFile <- system.file("extdata/genotypes/",
"genotypes_chr22.csv",
package = "PhenotypeSimulator")
genoFilePrefix <- gsub("chr.*", "", genotypeFile)
genoFileSuffix <- ".csv"
causalSNPsFromLines <- getCausalSNPs(N=50, 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.

Usage

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

Arguments

N	Number [integer] of samples to simulate.
sampleID	Prefix [string] for naming samples (will be followed by sample number from 1 to N when constructing id_samples).
X	[NrSamples x totalNrSNPs] Matrix of (standardised) genotypes.
kinshipfile	path/to/kinshipfile [string] to be read; either X or kinshipfile must be provided.
id_samples	Vector of [NrSamples] sample IDs [string]; if not provided constructed by paste(sampleID, 1:N, sep="").

`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 is normalised by the mean of its diagonal elements and 1e-4 added to the diagonal for numerical stability.

Value

[NrSamples x NrSamples] Matrix of kinship estimate.

Examples

```
geno <- simulateGenotypes(N=10, NrSNP=50)
K_fromGenotypesNormalised <- getKinship(N=10, X=geno$genotypes,
standardise=TRUE)

kinshipfile <- system.file("extdata/kinship",
"kinship.csv",
package = "PhenotypeSimulator")
K_fromFile <- getKinship(N=50, kinshipfile=kinshipfile)
```

`noiseBgEffects` *Simulate observational noise effects.*

Description

noiseBgEffects simulates observational noise with a proportion of the effect shared across samples and a proportion independent across samples.

Usage

```
noiseBgEffects(
  N,
  P,
  mean = 0,
  sd = 1,
  sampleID = "ID_",
  phenoID = "Trait_",
  shared = TRUE,
  independent = TRUE,
  id_samples = NULL,
  id_phenos = NULL
)
```

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.
`sampleID`	Prefix [string] for naming samples.
`phenoID`	Prefix [string] for naming traits.
`shared`	[bool] shared effect simulated if set to TRUE; at least one of shared or independent has to be set to TRUE.
`independent`	[bool] independent effect simulated if set to TRUE.
`id_samples`	Vector of [NrSamples] sample IDs [string]; if not provided constructed by paste(sampleID, 1:N, sep="").
`id_phenos`	Vector of [NrTraits] phenotype IDs [string]; if not provided constructed by paste(phenoID, 1:P, sep="").

Details

For the simulation of the observational noise effects, two components are used: i) matrix B [N x P] with vec(B) drawn from a normal distribution with mean=mean and sd=sd and ii) 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. To construct the final effects, the two matrices are multiplied as: E = BA^T.

Value

Named list of shared noise effects (shared: [N x P] matrix) and independent noise effects (independent: [N x P] matrix), the covariance term of the shared effect (cov_shared: [P x P] matrix) and the covariance term of the independent effect (cov_independent: [P x P] matrix).

Examples

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

`noiseFixedEffects` *Simulate noise fixed effects.*

Description

noiseFixedEffects simulates a number of non-genetic covariate effects (confounders). Confounders can have effects across all traits (shared) or to a number of traits only (independent); in addition, only a certain proportion of traits can be affected by the confounders. 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.

Usage

```
noiseFixedEffects(
  N,
  P,
  NrConfounders = 10,
  sampleID = "ID_",
  phenoID = "Trait_",
  id_samples = NULL,
  id_phenos = NULL,
  pTraitsAffected = 1,
  NrFixedEffects = 1,
  pIndependentConfounders = 0.4,
  pTraitIndependentConfounders = 0.2,
  keepSameIndependent = FALSE,
  distConfounders = "norm",
  mConfounders = 0,
  sdConfounders = 1,
  catConfounders = NULL,
  probConfounders = NULL,
  distBeta = "norm",
  mBeta = 0,
  sdBeta = 1,
  verbose = FALSE
)
```

Arguments

`N`	Number [integer] of samples to simulate.
`P`	Number [integer] of phenotypes to simulate.
`NrConfounders`	Vector of number(s) [integer] of confounders from a specified distribution to simulate.
`sampleID`	Prefix [string] for naming samples.
`phenoID`	Prefix [string] for naming traits.
`id_samples`	Vector of [NrSamples] sample IDs [string]; if not provided constructed by paste(sampleID, 1:N, sep="").
`id_phenos`	Vector of [NrTraits] phenotype IDs [string]; if not provided constructed by paste(phenoID, 1:P, sep="").
`pTraitsAffected`	Vector of proportion(s) [double] of traits affected by the confounders. For non-integer results of pTraitsAffected*P, the ceiling of the result is used.
`NrFixedEffects`	Number [integer] of different confounder effects to simulate; allows to simulate fixed effects from different distributions or with different parameters; if only one type of confounder distribution is wanted, set NrFixedEffects=1 and choose the number of confounders with eg NrConfounders=10.
`pIndependentConfounders`	Vector of proportion(s) [double] of confounders to have a trait-independent effect.

`pTraitIndependentConfounders`	Vector of proportion(s) [double] of traits influenced by independent confounder effects.
`keepSameIndependent`	[boolean] If set to TRUE, the independent genetic effects always influence the same subset of traits.
`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.
`verbose`	[boolean] If TRUE, progress info is printed to standard out

Value

Named list of shared confounder effects (shared: [N x P] matrix), independent confoudner effects (independent: [N x P] matrix), the confounders labeled as 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)

 # 2 fixed noise effect with 1 binomial confounders and 1 normally
 # distributed confounder; the latter only affects 2 traits
 noiseFE_binomialandNormalConfounders <- noiseFixedEffects(P=10, N=20,
 NrFixedEffects=2, pTraitsAffected =c (1,0.2), NrConfounders=c(2,2),
 distConfounders=c("bin", "norm"),  probConfounders=0.2)
```

probGen2expGen — *Compute expected genotypes from genotype probabilities.*

Description

Convert genotypes encoded as triplets of probablities (p(AA), p(Aa), p(aa)) into their expected genotype frequencies by 0*p(AA) + p(Aa) + 2p(aa).

Usage

```
probGen2expGen(probGeno)
```

Arguments

`probGeno`	Vector [numeric] with genotype probabilites; has to be a multiple of 3.

Value

Numeric vector of length [length(probGeno)/3] with the expected genotype value per individual.

Examples

```
nrSamples <- 10
# Construct genotype probability vector (usually from external input)
# First, assign zero probabilty of AA, Aa and aa for all samples
genotype_prob <- rep(0, 3*nrSamples)
# Second, for each sample draw one of 0,1,2 (corresponding to AA, Aa and aa)
genotype_prob[seq(1, nrSamples*3, 3) + sample(0:2, 10, replace=TRUE)] <- 1
genotype_exp <- probGen2expGen(genotype_prob)
```

`readStandardGenotypes` *Read genotypes from file.*

Description

readStandardGenotypes can read genotypes from a number of input formats for standard GWAS (binary plink, snptest, bimbam, gemma) or simulation software (binary plink, hapgen2, genome). Alternatively, simple text files (with specified delimiter) can be read. For more information on the different file formats see *External genotype software and formats*.

Usage

```
readStandardGenotypes(
  N,
  filename,
  format = NULL,
  verbose = TRUE,
  sampleID = "ID_",
  snpID = "SNP_",
  delimiter = ","
)
```

Arguments

`N`	Number [integer] of samples to simulate.
`filename`	path/to/genotypefile [string] in plink, oxgen (impute2/snptest/hapgen2), genome, bimbam or [delimiter]-delimited format (for format information see *External genotype software and formats*).
`format`	Name [string] of genotype file format. Options are: "plink", "oxgen", "genome", "bimbam" or "delim".
`verbose`	[boolean] If TRUE, progress info is printed to standard out.
`sampleID`	Prefix [string] for naming samples (will be followed by sample number from 1 to N when constructing id_samples).
`snpID`	Prefix [string] for naming SNPs (will be followed by SNP number from 1 to NrSNP when constructing id_snps).
`delimiter`	Field separator [string] of genotype file when format == "delim".

Details

The file formats and software formates supported are described below. For large external genotypes, consider the option to only read randomly selected, causal SNPs into memory via getCausalSNPs.

Value

Named list of [NrSamples X NrSNPs] genotypes (genotypes), their [NrSNPs] SNP IDs (id_snps), their [NrSamples] sample IDs (id_samples) and format-specific additional files (such as format-specific genotypes encoding or sample information; might be used for output writing).

External genotype software and formats

Formats::

- PLINK: consists of three files: .bed, .bim and .fam. When specifying the filepath, only the core of the name without the ending should be specified (i.e. for geno.bed, geno.bim and geno.fam, geno should be specified). When reading data from plink files, the absolute path to the plink-format file has to be provided, tilde expansion not provided. From https://www.cog-genomics.org/plink/1.9/formats: The .bed files contain the primary representation of genotype calls at biallelic variants in a binary format. The .bim is a text file with no header line, and one line per variant with the following six fields: i) Chromosome code (either an integer, or 'X'/'Y'/'XY'/'MT'; '0' indicates unknown) or name, ii) Variant identifier, iii) Position in morgans or centimorgans (safe to use dummy value of '0'), iv) Base-pair coordinate (normally 1-based, but 0 ok; limited to 231-2), v) Allele 1 (corresponding to clear bits in .bed; usually minor), vi) Allele 2 (corresponding to set bits in .bed; usually major). The .fam file is a text file with no header line, and one line per sample with the following six fields: i) Family ID ('FID'), ii), Within- family ID ('IID'; cannot be '0'), iii) Within-family ID of father ('0' if father isn't in dataset, iv) within-family ID of mother ('0' if mother isn't in dataset), v) sex code ('1' = male, '2' = female, '0' = unknown), vi) Phenotype value ('1' = control, '2' = case, '-9'/'0'/non-numeric = missing data if case/control)
- oxgen: consists of two files: the space-separated genotype file ending in .gen and the space-separated sample file ending in .sample. When specifying the filepath, only the core of the name without the ending should be specified (i.e. for geno.gen and geno.sample, geno should be specified). From https://www.well.ox.ac.uk/~gav/snptest/#input_file_formats: The genotype file stores data on a one-line-per-SNP format. The first five entries of each line should be the SNP ID, RS ID of the SNP, base-pair position of the SNP, the allele coded A and the allele coded B. The SNP ID can be used to denote the chromosome number of each SNP. The next three numbers on the line should be the probabilities of the three genotypes AA, AB and BB at the SNP for the first individual in the cohort. The next three numbers should be the genotype probabilities for the second individual in the cohort. The next three numbers are for the third individual and so on. The order of individuals in the genotype file should match the order of the individuals in the sample file. The sample file has three parts (a) a header line detailing the names of the columns in the file, (b) a line detailing the types of variables stored in each column, and (c) a line for each individual detailing the information for that individual. For more information on the sample file visit the above url or see writeStandardOutput.
- genome: The entire output of genome can be saved via 'genome -options > outputfile'. The /path/to/outputfile should be provided and this function extracts the relevant genotype information from this output file. http://csg.sph.umich.edu/liang/genome/
- bimbam: Mean genotype file format of bimbam which is a single, comma- separated file, without information on individuals. From the documentation for bimbam at http://www.haplotype.org/software.html: the first column of the mean genotype files is the SNP ID, the second and third columns are allele types with minor allele first. The remaining columns are the mean genotypes of different individuals – numbers between 0 and 2 that represents the (posterior) mean genotype, or dosage of the minor allele.
- delim: a [delimter]-delimited file of [(NrSNPs+1) x (NrSamples+1)] genotypes with the snpIDs in the first column and the sampleIDs in the first row and genotypes encoded as numbers between 0 and 2 representing the (posterior) mean genotype, or dosage of the minor allele. Can be user-genotypes or genotypes simulated with foward-time algorithms such as

simupop (http://simupop.sourceforge.net/Main/HomePage) that allow for user-specified output formats.

Genotype simulation characteristics::

- PLINK: simple, bi-allelelic genotypes without LD structure, details can be found at https://www.cog-genomics.org/plink/1.9/input#simulate.
- Hapgen2: resampling-based genotype simulation, details can be found at http://mathgen.stats.ox.ac.uk/genetics_software/hapgen/hapgen2.html.
- Genome: coalescent-based genotype simulation, details can be found at http://csg.sph.umich.edu/liang/genome/GENOME-manual.pdf.

Examples on how to call these genotype simulation tools can be found in the vignette *sample-scripts-external-genotype-simulation*.

Examples

```
# Genome format
filename_genome  <- system.file("extdata/genotypes/genome/",
"genotypes_genome.txt",
package = "PhenotypeSimulator")
data_genome <- readStandardGenotypes(N=100, filename_genome, format ="genome")

filename_hapgen  <- system.file("extdata/genotypes/hapgen/",
"genotypes_hapgen.controls.gen",
package = "PhenotypeSimulator")
filename_hapgen <- gsub("\\.gen", "", filename_hapgen)
data_hapgen <- readStandardGenotypes(N=100, filename_hapgen, format='oxgen')

filename_plink  <- system.file("extdata/genotypes/plink/",
"genotypes_plink.bed",
package = "PhenotypeSimulator")
filename_plink <- gsub("\\.bed", "", filename_plink)
data_plink <- readStandardGenotypes(N=100, filename=filename_plink,
format="plink")

filename_delim  <- system.file("extdata/genotypes/",
"genotypes_chr22.csv",
package = "PhenotypeSimulator")
data_delim <- readStandardGenotypes(N=50, filename=filename_delim,
format="delim")
```

`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 component such that the average column variance is equal to the user-specified proportion of variance.

Usage

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

Arguments

`component`	[N x P] Phenotype matrix [double] where [N] are the number of samples and P the number of phenotypes
`propvar`	Number [double] specifying the proportion of variance that should be explained by this phenotype component

Value

If propvar != 0, a named list with the [N x P] matrix of the scaled component (component) and its scale factor [double] (scale_factor) else returns NULL

Examples

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

runSimulation *Run phenotype simulation.*

Description

runSimulation wraps around setModel, the phenotype component functions (genFixedEffects, genBgEffects, noiseBgEffects, noiseFixedEffects and correlatedBgEffects), rescales each component and combines them into the final phenotype. For details to all parameters, see the respective functions.

Usage

```
runSimulation(
  N,
  P,
  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,
  tNrSNP = 5000,
  cNrSNP = 20,
  SNPfrequencies = c(0.1, 0.2, 0.4),
  genotypefile = NULL,
  format = "delim",
  genoFilePrefix = NULL,
  genoFileSuffix = NULL,
  genoDelimiter = ",",
  skipFields = NULL,
  header = FALSE,
  probabilities = FALSE,
  chr = NULL,
  NrSNPsOnChromosome = NULL,
  NrChrCausal = NULL,
  kinshipfile = NULL,
  kinshipHeader = FALSE,
  kinshipDelimiter = ",",
  standardise = TRUE,
  distBetaGenetic = "norm",
  mBetaGenetic = 0,
  sdBetaGenetic = 1,
  pTraitsAffectedGenetics = 1,
  pIndependentGenetic = 0.4,
  pTraitIndependentGenetic = 0.2,
  keepSameIndependentSNPs = FALSE,
  NrFixedEffects = 1,
  NrConfounders = 10,
  distConfounders = "norm",
  mConfounders = 0,
  sdConfounders = 1,
  catConfounders = NULL,
  probConfounders = NULL,
  distBetaConfounders = "norm",
  mBetaConfounders = 0,
  sdBetaConfounders = 1,
  pTraitsAffectedConfounders = 1,
  pIndependentConfounders = 0.4,
  pTraitIndependentConfounders = 0.2,
  keepSameIndependentConfounders = FALSE,
  pcorr = 0.8,
  corrmatfile = NULL,
  meanNoiseBg = 0,
  sdNoiseBg = 1,
  nonlinear = NULL,
  logbase = 10,
  expbase = NULL,
  power = NULL,
  customTransform = NULL,
  transformNeg = "abs",
  proportionNonlinear = 0,
  sampleID = "ID_",
  phenoID = "Trait_",
  snpID = "SNP_",
  seed = 219453,
  verbose = FALSE
)
```

Arguments

`N`	Number [integer] of samples to simulate.
`P`	Number [integer] of phenotypes to simulate.
`genVar`	Proportion [double] of total genetic variance.
`h2s`	Proportion [double] of genetic variance of genetic variant effects.
`theta`	Proportion [double] of variance of shared genetic variant effects.
`h2bg`	Proportion [double] of genetic variance of infinitesimal genetic effects; either h2s or h2bg have to be specified and h2s + h2bg = 1.
`eta`	Proportion [double] of variance of shared infinitesimal genetic effects.
`noiseVar`	Proportion [double] of total noise variance.

`rho`	Proportion [double] of noise variance of correlated effects; sum of rho, delta and phi has to be equal 1.
`delta`	Proportion [double] of noise variance of non-genetic covariate effects; sum of rho, delta and phi has to be equal 1.
`gamma`	Proportion [double] of variance of shared non-genetic covariate effects.
`phi`	Proportion [double] of noise variance of observational noise effects; sum of rho, delta and phi has to be equal 1.
`alpha`	Variance [double] of shared observational noise effect.
`tNrSNP`	Total number [integer] of SNPs to simulate; these SNPs are used for kinship estimation.
`cNrSNP`	Number [integer] of causal SNPs; used as genetic variant effects.
`SNPfrequencies`	Vector of allele frequencies [double] from which to sample.
`genotypefile`	Needed when reading external genotypes (into memory), path/to/genotype file [string] in format specified by format.
`format`	Needed when reading external genotypes, specifies the format of the genotype data; has to be one of plink, oxgen, genome, bimbam and delim when reading files into memory, or one of oxgen, bimbam or delim if sampling genetic variants from file; for details see readStandardGenotypes and getCausalSNPs.
`genoFilePrefix`	Needed when sampling cuasal SNPs from file, full path/to/chromosome-wise-genotype-file-ending-before-"chrChromosomeNumber" (no ’~’ expansion!) [string]
`genoFileSuffix`	Needed when sampling causal SNPs from file, following chromosome number including fileformat (e.g. ".csv") [string]
`genoDelimiter`	Field separator [string] of genotypefile or genoFile if format == delim.
`skipFields`	Number [integer] of fields (columns) in to skip in genoFilePrefix-genoFileSuffix-file. See details in getCausalSNPs if format == delim.
`header`	[logical] Can be set to indicate if genoFilePrefix-genoFileSuffix file has a header for format == ’delim’. See details in getCausalSNPs.
`probabilities`	[bool]. If set to TRUE, the genotypes in the files described by genoFilePrefix and genoFileSuffix are provided as triplets of probablities (p(AA), p(Aa), p(aa)) and are converted into their expected genotype frequencies by 0*p(AA) + p(Aa) + 2p(aa) via probGen2expGen.
`chr`	Numeric vector of chromosomes [integer] to chose NrCausalSNPs from; only used when external genotype data is sampled i.e. !is.null(genoFilePrefix)
`NrSNPsOnChromosome`	Specifies the number of SNPs [integer] per entry in chr (see above); has to be the same length as chr. If not provided, lines in genoFilePrefix-genoFileSuffix file will be counted (which can be slow for large files).
`NrChrCausal`	Number [integer] of causal chromosomes to chose NrCausalSNPs from (as opposed to the actual chromosomes to chose from via chr); only used when external genotype data is sampled i.e. !is.null(genoFilePrefix).
`kinshipfile`	path/to/kinshipfile [string]; if provided, kinship for simulation of genetic backgound effect will be read from file.
`kinshipHeader`	[boolean] If TRUE kinship file has header information.

`kinshipDelimiter`	Field separator [string] of kinship file.
`standardise`	[boolean] If TRUE genotypes will be standardised for kinship estimation (recommended).
`distBetaGenetic`	Name [string] of distribution to use to simulate effect sizes of genetic variants; one of "unif" or "norm".
`mBetaGenetic`	Mean/midpoint [double] of normal/uniform distribution for effect sizes of genetic variants.
`sdBetaGenetic`	Standard deviation/extension from midpoint [double] of normal/uniform distribution for effect sizes of genetic variants.
`pTraitsAffectedGenetics`	Proportion [double] of traits affected by the genetic variant effect. For non-integer results of pTraitsAffected*P, the ceiling of the result is used. Allows to simulate for instance different levels of pleiotropy.
`pIndependentGenetic`	Proportion [double] of genetic variant effects to have a trait-independent fixed effect.
`pTraitIndependentGenetic`	Proportion [double] of traits influenced by independent genetic variant effects.
`keepSameIndependentSNPs`	[boolean] If set to TRUE, the independent SNPs effects always influence the same subset of traits.
`NrFixedEffects`	Number [integer] of different non-genetic covariate effects to simulate; allows to simulate non-genetic covariate effects from different distributions or with different parameters.
`NrConfounders`	Number [integer] of non-genetic covariates; used as non-genetic covariate effects.
`distConfounders`	Vector of name(s) [string] of distributions to use to simulate confounders; one of "unif", "norm", "bin", "cat_norm", "cat_unif".
`mConfounders`	Vector of mean(s)/midpoint(s) [double] of normal/uniform distribution for confounders.
`sdConfounders`	Vector of standard deviation(s)/extension from midpoint(s) [double] of normal/uniform distribution for confounders.
`catConfounders`	Vector of confounder categories [factor]; required if distConfounders "cat_norm" or "cat_unif".
`probConfounders`	Vector of probability(ies) [double] of binomial confounders (0/1); required if distConfounders "bin".
`distBetaConfounders`	Vector of name(s) [string] of distribution to use to simulate effect sizes of confounders; one of "unif" or "norm".
`mBetaConfounders`	Vector of mean(s)/midpoint(s) [double] of normal/uniform distribution for effect sizes of confounders.

`sdBetaConfounders`	Vector of standard deviation(s)/extension from midpoint(s) [double] of normal/uniform distribution for effect sizes of confounders.
`pTraitsAffectedConfounders`	Proportion(s) [double] of traits affected by the non-genetic covariates. For non-integer results of pTraitsAffected*P, the ceiling of the result is used.
`pIndependentConfounders`	Vector of proportion(s) [double] of non-genetic covariate effects to have a trait-independent effect.
`pTraitIndependentConfounders`	Vector of proportion(s) [double] of traits influenced by independent non-genetic covariate effects.
`keepSameIndependentConfounders`	[boolean] If set to TRUE, the independent confounder effects always influence the same subset of traits.
`pcorr`	Correlation [double] between phenotypes.
`corrmatfile`	path/to/corrmatfile.csv [string] with comma-separated [P x P] numeric [double] correlation matrix; if provided, correlation matrix for simulation of correlated backgound effect will be read from file; file should NOT contain an index or header column.
`meanNoiseBg`	Mean [double] of the normal distributions for the simulation observational noise effects.
`sdNoiseBg`	Standard deviation [double] of the normal distributions for the simulations of the observational noise effects.
`nonlinear`	nonlinear transformation method [string]; one exp (exponential), log (logarithm), poly (polynomial), sqrt (squareroot) or custom (user-supplied function); if log or exp, base can be specified; if poly, power can be specified; if custom, a custom function (see for details). Non-linear transformation is optional, default is NULL ie no transformation (see details).
`logbase`	[int] base of logarithm for non-linear phenotype transformation (see details).
`expbase`	[int] base of exponential function for non-linear phenotype transformation (see details).
`power`	[double] power of polynomial function for non-linear phenotype transformation.
`customTransform`	[function] custom transformation function accepting a single argument.
`transformNeg`	[string] transformation method for negative values in non linear phenotype transformation. One of abs (absolute value) or set0 (set all negative values to zero). If nonlinear==log and transformNeg==set0, negative values set to 1e-5
`proportionNonlinear`	[double] proportion of the phenotype to be non- linear (see details)
`sampleID`	Prefix [string] for naming samples (will be followed by sample number from 1 to N when constructing sample IDs); only used if genotypes/kinship are simulated/do not have sample IDs.
`phenoID`	Prefix [string] for naming traits (will be followed by phenotypes number from 1 to P when constructing phenotype IDs).

snpID	Prefix [string] for naming SNPs (will be followed by SNP number from 1 to NrSNP when constructing SNP IDs).
seed	Seed [integer] to initiate random number generation.
verbose	[boolean]; If TRUE, progress info is printed to standard out

Details

Phenotypes are modeled under a linear additive model where Y = WA + BX + G + C + Phi, with WA the non-genetic covariates, BX the genetic variant effects, G the infinitesimal genetic effects, C the correlated background effects and the Phi the observational noise. For more information on these components look at the respective function descriptions (see also) Optionally the phenotypes can be non-linearly transformed via: Y_trans = (1-alpha) x Y + alpha x f(Y). Alpha is the proportion of non- linearity of the phenotype and f is a non-linear transformation, and one of exp, log or sqrt.

Value

Named list of i) dataframe of proportion of variance explained for each component (varComponents), ii) a named list with the final simulated phenotype components (phenoComponentsFinal), iii) a named list with the intermediate simulated phenotype components (phenoComponentsIntermediate), iv) a named list of parameters describing the model setup (setup) and v) a named list of raw components (rawComponents) used for genetic effect simulation (genotypes and/or kinship, eigenvalues and eigenvectors of kinship)

See Also

setModel, geneticFixedEffects, geneticBgEffects, noiseBgEffects, noiseFixedEffects, correlatedBgEffects and rescaleVariance.

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 simulated phenotypes and their components, model setup parameters and variance components to the specified directories. Requires a simulatedData list which is the output of runSimulation.

Usage

```
savePheno(
  simulatedData,
  directory,
  format = ".csv",
  outstring = "",
  saveIntermediate = TRUE,
  intercept_gemma = TRUE,
  verbose = TRUE
)
```

Arguments

`simulatedData`	Named list of i) dataframe of proportion of variance explained for each component (varComponents), ii) a named list with the final simulated phenotype components (phenoComponentsFinal), iii) a named list with the intermediate simulated phenotype components (phenoComponentsIntermediate), iv) a named list of parameters describing the model setup (setup) and v) a named list of raw components (rawComponents) used for genetic effect simulation (genotypes and/or kinship); obtained from runSimulation
`directory`	Absolute path (no tilde expansion) to parent directory [string] where simulated data should be saved [needs user writing permission]
`format`	Vector of format name(s) [string] specifying the output format; multiple output formats can be requested. Options are: plink, bimbam, snptest, gemma, limmbo, csv or rds. For information on format see details. In orde to save intermediate phenotype components, at least one of csv or rds need to be specified. plink/bimbam/snptest will only save final phenotype/genotype, kinship and covariate data.
`outstring`	Optional name [string] of subdirectory (in relation to directory) to save set-up dependent simulation results; if set to NULL, subdirectory named by NrSamples, NrSNPs, genetic Model and noise Model and genVar is created.
`saveIntermediate`	[bool] If TRUE, intermediate phenotype components such as shared and independent effect components are saved.
`intercept_gemma`	[boolean] When modeling an intercept term in gemma, a column of 1's have to be appended to the covariate files. Set intercept_gemma to TRUE to include a column of 1's in the output; only used when "gemma" %in% format
`verbose`	[boolean]; If TRUE, progress info is printed to standard out

Value

Path [string] to final output directory. If outstring is NULL, this directory will be a subdirectory of the input directory.

Examples

```
simulatedPhenotype <- runSimulation(N=100, P=5, cNrSNP=10,
genVar=0.2, h2s=0.2, phi=1)
## Not run:
outputdir <- savePheno(simulatedPhenotype, directory=tempdir(),
outstring="Data_simulation", format=c("csv", "plink"))
## End(Not run)
```

`setModel` *Set simulation model.*

Description

Based on parameters provided, this function sets the name for the phenotype simulation. It carries out compatibiltiy checks of the specifie parameters and checks for any missing information.

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,
  proportionNonlinear = 0,
  cNrSNP = NULL,
  NrConfounders = 10,
  verbose = TRUE
)
```

Arguments

`genVar`	Total genetic variance [double].
`h2s`	Proportion [double] of variance of genetic variant effects.
`theta`	Proportion [double] of variance of shared genetic variant effects.

`h2bg`	Proportion [double] of variance of infinitesimal genetic effects i.e. correlation introduced by sample kinship).
`eta`	Proportion [double] of variance of shared infinitesimal genetic effects.
`noiseVar`	Total noise variance [double].
`delta`	Proportion [double] of variance of non-genetic covariate effect.
`gamma`	Proportion [double] of variance of shared non-genetic covariate effects.
`rho`	Proportion [double] of variance of correlated noise effects.
`phi`	Proportion [double] of variance of observational noise effects.
`alpha`	Proportion [double] of variance of shared observational noise effect.
`pcorr`	Correlation [double] between phenotypes.
`pIndependentConfounders`	Proportion [double] of non-genetic covariate to have a trait-independent effect.
`pTraitIndependentConfounders`	Proportion [double] of traits influenced by independent non-genetic covariate effects.
`pIndependentGenetic`	Proportion [double] of genetic variant effects to have a trait-independent fixed effect.
`pTraitIndependentGenetic`	Proportion [double] of traits influenced by independent genetic variant effects.
`proportionNonlinear`	[double] proportion of the phenotype to be non- linear
`cNrSNP`	Number [integer] of causal SNPs; used as genetic variant effects.
`NrConfounders`	Number [integer] of non-genetic covariates; used as non-genetic covariate effects.
`verbose`	[boolean]; If TRUE, progress info is printed to standard out.

Value

Named list containing the genetic model (modelGenetic), the noise model (modelNoise) and the input parameters (h2s, h2bg, noiseVar, rho, delta, phi, gamma, theta, eta, alpha, pcorr, proportionNonlinear). Model options are: modelNoise: "noNoise", "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`	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`	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`	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_",
  snpID = "SNP_",
  verbose = TRUE
)
```

Arguments

`N`	Number of samples for which to simulate bi-allelic genotypes.
`NrSNP`	Number of SNPs to simulate.
`frequencies`	Vector of allele frequencies [double] from which to sample.
`sampleID`	Prefix [string] for naming samples (will be followed by sample number from 1 to N when constructing id_samples).
`snpID`	Prefix [string] for naming SNPs (will be followed by SNP number from 1 to NrSNP when constructing id_snps).
`verbose`	[boolean] If TRUE, progress info is printed to standard out.

Value

Named list with [N x NrSNP] matrix of simulated genotypes (genotypes), their SNP frequencies (freq), a vector of sample IDs (id_samples) and a vector of SNP IDs (id_snps).

See Also

`standardiseGenotypes`

Examples

```
N10NrSNP10 <- simulateGenotypes(N=10, NrSNP=10)
N10NrSNP10 <- simulateGenotypes(N=10, NrSNP=10,
frequencies=c(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 \
#--tNrSNPs=10000 --cNrSNPs=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 \
#--directory=/tmp \
#--showProgress \
```

`standardiseGenotypes` *Standardise genotypes.*

Description

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

Usage

```
standardiseGenotypes(geno, impute = FALSE)
```

Arguments

`geno`	[N x NrSNP] Matrix/dataframe of genotypes [integer]/[double].
`impute`	[logical] Indicating if missing genotypes should be imputed; if set FALSE and data contains missing values, `standardiseGenotypes` will return an error.

Details

Missing genotypes can be mean-imputed and rounded to nearest integer before standardisation. If genotypes contain missing values and impute is set to FALSE, `standardiseGenotypes` will return an error.

Value

[N x NrSNP] Matrix of standardised genotypes [double].

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(2000, 2, 0.3), rbinom(2000, 2, 0.4),rbinom(2000, 2, 0.5))
geno_sd <- standardiseGenotypes(geno)
```

testNumerics *Test lists for different properties of numerics.*

Description

Test all elements of a list if they are numeric, positive numbers, integers or proportions (range 0-1)

Usage

```
testNumerics(numbers, positives = NULL, integers = NULL, proportions = NULL)
```

Arguments

`numbers`	List whose elements are tested for being numeric
`positives`	List whose elements are tested for being positive numbers
`integers`	List whose elements are tested for being integers
`proportions`	List whose elements are tested for being proportions between 0 and 1

transformNonlinear *Phenotype transformation.*

Description

Transformation of phenotype component by applying a user-specified non-linear transformation to the phenotype component.

Usage

```
transformNonlinear(
  component,
  alpha,
  method,
  logbase = 10,
  power = 2,
  expbase = NULL,
  transformNeg = "abs",
  f = NULL,
  verbose = TRUE
)
```

Arguments

`component`	[N x P] Phenotype matrix [double] where [N] are the number of samples and P the number of phenotypes
`alpha`	[double] weighting scalar for non-linearity: alpha==0 fully linear phenotype, alpha==1 fully non-linear phenotype. See @details.
`method`	[string] one of exp (exponential), log (logarithm), poly (polynomial), sqrt (square-root) or custom (user-supplied function)
`logbase`	[int] when method==log, sets the log base for transformation
`power`	[double] when method==poly, sets the power to raise to.
`expbase`	[double] when method==exp, sets the exp base for transformation.
`transformNeg`	[string] one of abs (absolute value) or set0 (set all negative values to zero). If method==log and transformNeg==set0, negative values set to 1e-5
`f`	[function] function accepting component as a single argument.
`verbose`	[boolean]; If TRUE, progress info is printed to standard out.

Details

transformNonlinear takes a phenotype component as input and transforms it according to the specified transformation method. The user can choose how strongly non-linear the resulting phenotype component should be, by specifying the weighting parameter alpha: component_transformed = (1 - alpha) * component + alpha * transformfunction(component)

Value

[N x P] transformed phenotype matrix [double]

Examples

```
# Simulate non-genetic covariate effects
cov_effects <- noiseFixedEffects(N=100, P=5)
# Transform logarithmically
covs_log <- transformNonlinear(cov_effects$shared, alpha=0.5, method="log",
transformNeg="abs")
# Transform custom
f_custom <- function(x) {x^2 + 3*x}
covs_custom <- transformNonlinear(cov_effects$shared, alpha=0.5,
method="custom", f=f_custom)
```

`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`	Vector of [string] element(s) 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

`writeStandardOutput` *Write simulated data into formats used by standard GWAS software.*

Description

writeStandardOutput can write genotypes and phenotypes as well as possible covariates and kinship matrices into a number of formats for standard GWAS software: plink, snptest, bimbam, gemma, limmbo. For more information on the different file formats see *External formats*.

Usage

```
writeStandardOutput(
  directory,
  phenotypes = NULL,
  genotypes = NULL,
  additionalPhenotypes = NULL,
  covariates = NULL,
  kinship = NULL,
  eval_kinship = NULL,
  evec_kinship = NULL,
  id_samples,
  id_snps,
  id_phenos,
  outstring = NULL,
  standardInput_samples = NULL,
  standardInput_genotypes = NULL,
  format = NULL,
  intercept_gemma = FALSE,
  nameAdditional = "_nonLinear",
  verbose = TRUE
)
```

Arguments

`directory`	Absolute path (no tilde expansion) to parent directory [string] where the data should be saved [needs user writing permission]
`phenotypes`	[NrSamples x NrTrait] Data.frame/matrix of phenotypes [doubles].
`genotypes`	[NrSamples x NrSNP] Data.frame/matrix of genotypes [integers]/[doubles].
`additionalPhenotypes`	[NrSamples x NrTrait] Data.frame/matrix of additional phenotypes (for instance non-linearly tranformed orginal
`covariates`	[NrSamples x NrCovariates] Data.frame/matrix of covariates [integers]/[doubles].
`kinship`	[NrSamples x NrSamples] Data.frame/matrix of kinship estimates [doubles].
`eval_kinship`	[NrSamples] vector with eigenvalues of kinship matrix [doubles].
`evec_kinship`	[NrSamples x NrSamples] Data.frame/matrix with eigenvectors of kinship matrix [doubles].
`id_samples`	Vector of [NrSamples] sample IDs [string] of simulated phenotypes, genotypes and covariates.
`id_snps`	Vector of [NrSNPs] SNP IDs [string] of (simulated) genotypes.
`id_phenos`	Vector of [NrTraits] phenotype IDs [string] of simulated phenotypes.
`outstring`	(optional) Name [string] of subdirectory (in relation to directory) to save set-up independent simulation results.
`standardInput_samples`	(optional) Data.frame of sample information obtained when genotypes were read from plink, oxgen or genome file.
`standardInput_genotypes`	(optional) Data.frame of genotypes obtained when reading genotypes from plink, oxgen, or genome file.
`format`	Vector of name(s) [string] of file formats, options are: "plink", "snptest", "gemma", "bimbam", "delim". For details on the file formats see *External formats*.
`intercept_gemma`	[boolean] When modeling an intercept term in gemma, a column of 1's have to be appended to the covariate files. Set intercept_gemma to TRUE to include a column of 1's in the output.
`nameAdditional`	name [string] of additonal phenotypes to be appended to filename.
`verbose`	[boolean]; If TRUE, progress info is printed to standard out

External formats

- plink format: consists of three files, .bed, .bim and .fam. From `https://www.cog-genomics.org/plink/1.9/formats`: The .bed files contain the primary representation of genotype calls at biallelic variants in a binary format. The .bim is a text file with no header line, and one line per variant with the following six fields: i) Chromosome code (either an integer, or 'X'/'Y'/'XY'/'MT'; '0' indicates unknown) or name, ii) Variant identifier, iii) Position in morgans or centimorgans (safe to use dummy value of '0'), iv) Base-pair coordinate (normally 1-based, but 0 ok; limited to 231-2), v) Allele 1 (corresponding to clear bits in .bed;

usually minor), vi) Allele 2 (corresponding to set bits in .bed; usually major). The .fam file is a text file with no header line, and one line per sample with the following six fields: i) Family ID ('FID'), ii), Within- family ID ('IID'; cannot be '0'), iii) Within-family ID of father ('0' if father isn't in dataset, iv) within-family ID of mother ('0' if mother isn't in dataset), v) sex code ('1' = male, '2' = female, '0' = unknown), vi) Phenotype value ('1' = control, '2' = case, '-9'/'0'/non-numeric = missing data if case/control)

- snptest format: consists of two files, the genotype file ending in .gen (genotypes_snptest.gen) and the sample file ending in .sample (Ysim_snptest.sample). From `https://www.well.ox.ac.uk/~gav/snptest/#input_file_formats`: The genotype file stores data on a one-line-per-SNP format. The first 5 entries of each line should be the SNP ID, RS ID of the SNP, base-pair position of the SNP, the allele coded A and the allele coded B. The SNP ID can be used to denote the chromosome number of each SNP. The next three numbers on the line should be the probabilities of the three genotypes AA, AB and BB at the SNP for the first individual in the cohort. The next three numbers should be the genotype probabilities for the second individual in the cohort. The next three numbers are for the third individual and so on. The order of individuals in the genotype file should match the order of the individuals in the sample file. The sample file has three parts (a) a header line detailing the names of the columns in the file, (b) a line detailing the types of variables stored in each column, and (c) a line for each individual detailing the information for that individual. a) The header line needs a minimum of three entries. The first three entries should always be ID_1, ID_2 and missing. They denote that the first three columns contain the first ID, second ID and missing data proportion of each individual. Additional entries on this line should be the names of covariates or phenotypes that are included in the file. In the above example, there are 4 covariates named cov_1, cov_2, cov_3, cov_4, a continuous phenotype named pheno1 and a binary phenotype named bin1. All phenotypes should appear after the covariates in this file. b) The second line (the variable type line) details the type of variables included in each column. The first three entries of this line should be set to 0. Subsequent entries in this line for covariates and phenotypes should be specified by the following rules: D for Discrete covariates (coded using positive integers), C for Continuous covariates, P for Continuous Phenotype, B for Binary Phenotype (0 = Controls, 1 = Cases). c) Individual information: one line for each individual containing the information specified by the entries of the header line. Entries of the sample file are separated by spaces.
- bimbam format: consists of a) a simple, tab-separated phenotype file without sample or phenotype header/index (Ysim_bimbam.txt) and b) the mean genotype file format which is a single file, without information on individuals: (genotypes.bimbam). From the bimbam documentation at `http://www.haplotype.org/software.html`: The first column of the mean genotype files is the SNP ID, the second and third columns are allele types with minor allele first. The rest columns are the mean genotypes of different individuals – numbers between 0 and 2 that represents the (posterior) mean genotype, or dosage of the minor allele.
- gemma format: consists of a) a simple, tab-separated phenotype file without sample or phenotype header/index (Ysim_gemma.txt) and b) the mean genotype file format which is a single file, without information on individuals(genotypes.gemma); a) and b) both the same as above for bimbam format). In addition and if applicable, c) a kinship file (kinship_gemma.txt) and d) covariate file (Covs_gemma.txt). From `http://www.xzlab.org/software/GEMMAmanual.pdf`: The kinship file contains a NrSample × NrSample matrix, where each row and each column corresponds to individuals in the same order as in the mean genotype file, and ith row and jth column is a number indicating the relatedness value between ith and jth individuals. The covariates file has the same format as the phenotype file dsecribed above and must contain a column of 1's if one wants to include an intercept term (set parameter inter-

cept_gemma=TRUE).

- limmbo format: consists of a) a comma-separated phenotype file without sample IDs as index and phenotype IDs as header (Ysim_limmbo.csv), b) the mean genotype file format with one genetic variant per line. The first column contains the variant ID, column 2-N+1 contain the genotype code (numbers between 0 and 2 that represent the (posterior) mean genotype/dosage of the minor allele) for N samples, c) a kinship file (kinship_limmbo.csv) and d) covariate file (covs_limmbo.csv). From

See Also

readStandardGenotypes

Examples

```
simulation <- runSimulation(N=10, P=2, genVar=0.4, h2s=0.2, phi=1)
genotypes <- simulation$rawComponents$genotypes
kinship <-  simulation$rawComponents$kinship
phenotypes <- simulation$phenoComponents$Y

## Not run:
# Save in plink format (.bed, .bim, .fam, Y_sim_plink.txt)
writeStandardOutput(directory=tempdir(),
genotypes=genotypes$genotypes, phenotypes=phenotypes,
id_samples = genotypes$id_samples, id_snps = genotypes$id_snps,
id_phenos = colnames(phenotypes), format="plink")

# Save in gemma and snptest format
writeStandardOutput(directory=tempdir(),
genotypes=genotypes$genotypes, phenotypes=phenotypes,
id_samples = genotypes$id_samples, id_snps = genotypes$id_snps,
id_phenos = colnames(phenotypes), kinship=kinship,
format=c("snptest", "gemma"))

## End(Not run)
```

Index