Package 'RFGLS'

February 19, 2015

Version 1.1

Date 2013-8-29

Title Rapid Feasible Generalized Least Squares

Author Robert M. Kirkpatrick <rkirkpatrick2@vcu.edu>, Xiang Li <lixxx554@umn.edu>, and Saonli Basu <saonli@umn.edu>.

Maintainer Saonli Basu <saonli@umn.edu>

Description
RFGLS uses a generalized least-squares method to perform single-marker association analysis, in datasets of nuclear families containing parents, twins, and/or adoptees

Depends stats, bdsmatrix, Matrix, R (>= 2.15.0)

License GPL (>= 2)

NeedsCompilation no

Repository CRAN

Date/Publication 2013-09-03 19:29:45

R topics documented:

RFGLS-package *Rapid Feasible Generalized Least Squares*

Description

RFGLS uses a generalized least-squares method to perform single-marker association analysis, in datasets of nuclear families containing parents, twins, and/or adoptees. It is designed for families of no greater than four members. When conducting association analysis with a large number of markers, as in GWAS, *RFGLS* uses *rapid* feasible generalized least-squares, an approximation to feasible generalized least-squares (FGLS) that considerably reduces computation time with minimal bias in *p*-values, and with negligible loss in power.

The package includes four functions. Function `gls.batch()` actually conducts GWAS using the rapid feasible generalized least-squares approximation, under which the residual variance-covariance matrix is estimated once from a regression of the phenotype onto covariates only, and is subsequently "plugged in" for use in all subsequent single-SNP analyses. Function `fgls()` is called by `gls.batch()`, and conducts a single FGLS regression. It can be used to simultaneously estimate fixed-effects regression coefficients and the residual covariance matrix. Function `gls.batch.get()` is useful to restructure data, for use with `fgls()`. Function `FSV.frompedi()` creates family-structure variables based upon available information in a pedigree file. Functions `gls.batch()` and `gls.batch.get()` use these family-structure variables, which represent the type of family to which each participant belongs and how s/he fits into that family.

Details

Package:	RFGLS
Version:	1.1
Date:	2013/8/29
Depends:	R (>= 2.15.0), stats, bdsmatrix, Matrix
License:	GPL (>= 2)

Author(s)

Robert M. Kirkpatrick <rkirkpatrick2@vcu.edu>, Xiang Li <lixxx554@umn.edu>, and Saonli Basu <saonli@umn.edu> .

Maintainer: Saonli Basu <saonli@umn.edu>

References

Li X, Basu S, Miller MB, Iacono WG, McGue M: A Rapid Generalized Least Squares Model for a Genome-Wide Quantitative Trait Association Analysis in Families. *Human Heredity* 2011;71:67-82 (DOI: 10.1159/000324839)

fgls *Feasible Generalized Least Squares regression with family GWAS data.*

Description

Jointly estimates the fixed-effects coefficients and residual variance-covariance matrix in a generalized least squares model by minimizing the (multivariate-normal) negative loglikelihood function, via `optim()` in the R base distribution. The residual variance-covariance matrix is block-diagonal sparse, constructed with `bdsmatrix()` from the `bdsmatrix` package.

Usage

```
fgls(fixed, data=parent.frame(), tlist=tlist, sizelist=sizelist,
  med=c("UN","VC"), vmat=NULL, start=NULL, theta=NULL, drop=NULL,
  get.hessian=FALSE, optim.method="BFGS", control=list(), weights=NULL,
  sizeLab=NULL,Mz=NULL,Bo=NULL,Ad=NULL,Mix=NULL,indobs=NULL)
```

Arguments

`fixed`	An object of class 'formula' (or one that can be coerced to that class): a symbolic description of the regression model to be fitted. The RHS of the formula contains the fixed effects of the model.
`data`	An optional data frame, list or environment (or object coercible by `as.data.frame()` to a data frame) containing the variables in the model, as specified in argument '`fixed`'. If not found in '`data`' the variables are taken from `environment(formula)`, typically the environment from which `fgls()` is called. If it contains a column named `"ID"`, then that column's values will be the row and column names of output element `sigma` (see below, under "Value").
`tlist`	The character vector of the family labels ("famlab") in the data. The length of the vector equals the number of family units. It should be ordered in the same order as the families appear in the data. Object '`tlist`' is created by the `gls.batch()` and `gls.batch.get()` functions.
`sizelist`	The integer vector of the family sizes in the data. The length of the vector equals the number of family units. It should be ordered in the same order as the families appear in the data. Object '`sizelist`' is created by the `gls.batch()` and `gls.batch.get()` functions.
`med`	A character string, either `"UN"` or `"VC"`, which are the two RFGLS methods described by Li et al. (2011). If `"UN"` (default), which stands for "unstructured," the residual covariance matrix will be constructed from, at most, 12 parameters (8 correlations and 4 variances). If `"VC"`, which stands for "variance components," the residual covariance matrix will be constructed from, at most, 3 variance components (additive-genetic, shared-environmental, and unshared-environmental).

vmat — The previously estimated (or known) residual covariance matrix (for conducting *Rapid* FGLS). If it is NULL (default), the matrix will either be jointly estimated with the fixed-effects coefficients (FGLS), or will be constructed values supplied to 'theta'. If not NULL, must be either (1) an object of class 'bdsmatrix' (from the bdsmatrix package), or (2) a character string specifying the filename and path for a single-column text file, with header, containing the "blocks" of a bdsmatrix. Note that at least one of 'vmat' and 'theta' must be NULL.

start — A numeric vector of initial values for the residual-covariance parameters. If NULL (default), generic start values are used. Otherwise, it must be a numerical vector of either length 12 if med="UN", or of length 3 if med="VC". Each vector element provides the initial value for the parameter corresponding to its index (serial position). Values of NA are accepted, and will be replaced with the generic start value for that parameter. See below under "Details" for which parameters correspond to which indices. Users should bear in mind that especially poor start values can cause optimization to fail. Ignored if no residual-covariance parameters are estimated.

theta — A numeric vector of previously estimated (or known) residual-covariance parameters. Defaults to NULL, in which case it is ignored. Otherwise, it must be a numerical vector of of either length 12 if med="UN", or of length 3 if med="VC". Each vector element provides the value for the parameter corresponding to its index (serial position). Values of NA are accepted for extraneous parameters. See below under "Details" for which parameters correspond to which indices. The residual covariance matrix is constructed from the elements of 'theta', exactly as-is. Note that at least one of 'vmat' and 'theta' must be NULL.

drop — An integer vector of indices (serial positions) specifying which residual-covariance parameters to drop. Dropped parameters are not estimated. In addition to those specified by 'drop', fgls() automatically identifies which parameters are *completely* unidentified from the data (i.e., zero observations in the data are informative about them), and drops them as well. If 'drop' is NULL (default), no user-specified parameters are dropped. Otherwise, it must be a vector of integers, either between 1 and 12 (inclusive) if med="UN", or between 1 and 2 (inclusive) if med="VC". Note that if a user-specified-dropped parameter ends up being needed to construct the residual covariance matrix, its value is taken to be that of its OLS equivalent: zero for correlations (med="UN") and for the familial variance components (med="VC"), and the OLS residual variance for variances (med="UN"). It may be prudent to drop parameters when very few observations in the data are informative about them, which can at least save computation time. See below under "Details" for which parameters correspond to which indices. Ignored if no residual-covariance parameters are estimated.

get.hessian — Logical; default is FALSE. If TRUE, fgls() will include the Hessian matrix from optim() in its output list. Otherwise, the entry 'hessian' in the list will be NULL. Ignored if no residual-covariance parameters are estimated.

optim.method — Character string, passed as 'method' to optim(). Ignored if no residual-covariance parameters are estimated. The default, "BFGS", is usually fast and is recommended for general use. If method "L-BFGS-B" is used, fgls() will supply optim() with reasonable box constraints on the parameters, intended for use with optim()'s default control parameters (see argument 'control' below).

Method "BFGS" (the default) may fail when any of the residual-covariance parameters are poorly identified from the data. In these cases, it may be wise simply to drop the offending parameters. Other optimization methods, including "L-BFGS-B", can succeed where "BFGS" fails. Method "SANN" should *not* generally be relied upon to find the global optimum, but it can sometimes produce reasonable, approximate solutions in instances where no other method works. As a last-resort diagnostic, one can combine optim.method="SANN" with hessian=TRUE, since the resulting Hessian matrix may provide clues as to which parameters are causing problems.

control
: A list of control parameters passed to optim(), intended for advanced users. The default is also optim()'s default, which should be adequate for general use.

weights
: A numeric vector of weights, with length equal to the number of observations in the data. Defaults to NULL.

sizeLab
: This is an optional argument, and may be eliminated in future versions of this package. Defaults to NULL; otherwise, must be a character string. If the number of characters in the string is not equal to the size of the largest family in the data, fgls() will produce a warning.

Mz, Bo, Ad, Mix, indobs
: These arguments are deprecated, and their values are ignored. They are retained in this package version for legacy reasons, but will be eliminated in future versions.

Details

Function fgls() was originally intended to be called automatically, from within gls.batch(). However, calling it directly is likely to be useful to advanced users. The difficulty when directly invoking fgls() is supplying the function with arguments 'tlist' and 'sizelist'. But, these can be obtained easily via gls.batch.get().

When residual-covariance parameters are to be estimated, fgls() will attempt optimization, at most, two times. If the initial attempt fails, fgls() prints a message saying so to the console, and tries a second time. On the second attempt, before each evaluation of the objective function, the blocks composing the block-diagonal residual covariance matrix are forced to be positive definite. This uses nearPD() from the *Matrix* package, which turns each block matrix into its nearest positive-definite approximation (where "nearest" is meant in a least-squares sense). Forcing positive-definiteness in this way is only used for the second attempt, and not for the initial attempt (which has its own way of ensuring a positive-definite solution), since it slows down optimization and is unnecessary when the parameters are well-identified. Furthermore, it can have consequences the user might not expect. For instance, in fgls()'s output (see below, under "Value"), the elements of the residual covariance matrix sigma might not correspond to the parameter estimates in estimates, or covariances that are supposed to be the same across families might not be so in the actual matrix sigma. Nevertheless, the second attempt may succeed when the initial attempt fails.

When med="UN", the residual covariance matrix is constructed from, at most, 12 parameters–8 correlations and 4 variances. Below is an enumerated list of those 12 parameters, in which the number of each list entry is the index (serial position) of that parameter, and the quoted text is the element name of each estimated parameter as it appears in fgls() output:

1. "cor(m,f)", correlation between mothers and fathers.

2. "cor(c/b,m)", correlation between biological offspring and mothers.
3. "cor(c/b,f)", correlation between biological offspring and fathers.
4. "cor(c,c)", MZ-twin correlation.
5. "cor(b,b)", full-sibling (DZ-twin) correlation.
6. "cor(a,m)", correlation between adoptees and mothers.
7. "cor(a,f)", correlation between adoptees and fathers.
8. "cor(a,a)", adoptive-sibling correlation.
9. "var(O)", offspring variance.
10. "var(m)", mother variance.
11. "var(f)", father variance.
12. "var(ind)", variance for "independent observations."

When `med="VC"`, the residual covariance matrix is constructed from, at most, 3 variance components. Below is an enumerated list of those 3 parameters, in which the number of each list entry is the index (serial position) of that parameter, and the quoted text is the label of each estimated parameter as it appears in `fgls()` output:

1. "A", additive-genetic variance.
2. "C", shared-environmental variance (compound-symmetric within families).
3. "E", unshared-environmental variance (which cannot be dropped).

Additive-genetic variance contributes to covariance between family members commensurately to the expected proportion of segregating alleles they share: 1.0 for MZ twins, 0.5 for first-degree relatives, 0 for spouses and adoptive relatives. Shared-environmental variance, as defined here, represents covariance between biologically unrelated family members (including spouses).

In package version 1.0, arguments '`subset`' and '`na.action`' were accepted, and passed to `lm()`. Neither are accepted any longer. Subsetting should be done before directly calling `fgls()`; the function handles `NA`'s in the data by what is (in effect) `na.action=na.omit`.

Value

An object of class 'fgls'. It includes the following components:

`ctable` Table of coefficients reminiscent of output from `summary.lm()`. Each fixed-effect term (including the intercept) has one row of the table, which are ordered as the terms appear in argument '`fixed`'. Each row contains a point estimate, an estimated standard error, a *t*-statistic, and a two-tailed *p*-value.

`Rsqd` The generalized-least-squares coefficient of determination, *a la* Buse (1973).

`estimates` The vector of MLEs of the parameters used to construct the residual covariance matrix, ordered as in the lists above, under "Details." Dropped parameters are given value `NA`. If no residual-covariance parameters are estimated, will instead be a single `NA`.

`drop` A vector of parameter indices, representing which residual-covariance parameters were dropped (not estimated). See above, under "Details," for which parameters correspond to which indices. `NULL` if no parameters were dropped or estimated.

`iter` NULL if no residual-covariance parameters were estimated. Otherwise, a single-row data frame, containing miscellaneous output pertaining to the optimization, specifically, the following named columns:

1. `iterations` (integer): the number of function iterations, as returned from `optim()`.
2. `convergence` (integer): convergence code, as returned from `optim()`; value 0 means that convergence was successful.
3. `message` (character): additional information from the optimizer; a single whitespace means that `optim()` returned a message of NULL.
4. `first_try` (logical): Did `fgls()`'s first attempt at optimization succeed? If FALSE, then during the second attempt, `fgls()` had to force the block matrices of the residual covariance matrix to be positive-definite, as described above, under "Details."

`loglik` The *negative* loglikelihood, at the solution. If the residual-covariance parameters were estimated, then it equals -1 times the maximized joint loglikelihood of those parameters and the regression coefficients. If the residual-covariance parameter values were provided with argument '`vmat`' or '`theta`', then it equals -1 times the maximized joint loglikelihood of the regression coefficients, *conditional* on the values supplied for the residual-covariance parameters.

`sigma` The residual covariance matrix. It is of class 'bdsmatrix'. Its row and column names are taken from the column named "ID", if any, in argument `data`, otherwise its row and column names are sequential numbers. One of its slots, `sigma@blocks`, can be written to a single-column text file and subsequently read in by `gls.batch()`. Due to its potential size, it is not advised to return `sigma` to R's standard output or print it to the console.

`hessian` If `get.hessian=TRUE` and residual-covariance parameters were estimated, the Hessian matrix from `optim()` for those parameters; NULL otherwise.

`n` Sample size (i.e., number of individual participants), after excluding those with missing data (NA's).

`df.residual` Residual degrees of freedom in the feasible generalized-least-squares regression, as returned by `lm()`. Note that it only reflects the number of regression coefficients, and not the number of residual-covariance parameters that were estimated.

`residuals` Residuals from the feasible generalized-least-squares regression. It is a vector of length `n`, i.e. it is not padded with NA's for participants with missing data.

`fitted.values` Predicted phenotype scores from the feasible generalized-least-squares regression. It is a vector of length `n`, i.e. it is not padded with NA's for participants with missing data.

`variance` The estimated covariance matrix for (the sampling distribution of) the fixed-effects regression coefficients.

`call` Echo of `fgls()` function call.

Function `fgls()` also prints to console the estimates of non-dropped residual-covariance parameters (if any).

Author(s)

Xiang Li <lixxx554@umn.edu>, Robert M. Kirkpatrick <kirk0191@umn.edu>, and Saonli Basu <saonli@umn.edu> .

References

Li X, Basu S, Miller MB, Iacono WG, McGue M: A Rapid Generalized Least Squares Model for a Genome-Wide Quantitative Trait Association Analysis in Families. *Human Heredity* 2011;71:67-82 (DOI: 10.1159/000324839)

Buse, A: Goodness of Fit in Generalized Least Squares Estimation *The American Statistician* 1973;27:106-108

See Also

gls.batch

Examples

```
data(pheno)
data(geno)
data(map)
data(pedigree)
data(rescovmtx)
foo <- gls.batch.get(
  phenfile=pheno,genfile=data.frame(t(geno)),pedifile=pedigree,
  covmtxfile.in=NULL,theta=NULL,snp.names=map[,2],input.mode=c(1,2,3),
  pediheader=FALSE,pedicolname=c("FAMID","ID","PID","MID","SEX"),
  sep.phe=" ",sep.gen=" ",sep.ped=" ",
  phen="Zscore",covars="IsFemale",med=c("UN","VC"),
  outfile,col.names=TRUE,return.value=FALSE,
  covmtxfile.out=NULL,
  covmtxparams.out=NULL,
  sizeLab=NULL,Mz=NULL,Bo=NULL,Ad=NULL,Mix=NULL,indobs=NULL)

bar <- fgls(
  Zscore ~ rs3934834 + IsFemale, data=foo$test.dat, tlist=foo$tlist,
  sizelist=foo$sizelist,med=c("UN","VC"),
  vmat=rescovmtx, #<--Resid. cov. matrix from fgls onto IsFemale only.
  start=NULL, theta=NULL, drop=NULL, get.hessian=FALSE,
  optim.method="BFGS", control=list(), weights=NULL,
  sizeLab=NULL,Mz=NULL,Bo=NULL,Ad=NULL,Mix=NULL,indobs=NULL)

bar$ctable
##  To simultaneously estimate residual covariance matrix
##  and regression coefficients for rs3934834 & IsFemale,
##  use the same syntax, except with vmat = NULL .
```

FSV.frompedi *Family-Structure Variables from Pedigree File*

Description

This function creates the family-structure variables "FTYPE" (family-type) and "INDIV" (individual code) from available information in a pedigree file. Note that FSV.frompedi() is called internally by gls.batch() and gls.batch.get() when their argument 'input.mode' is set to 3.

Usage

```
FSV.frompedi(pedi.dat,phen.dat)
```

Arguments

pedi.dat	A pedigree file, as a data frame, with named columns. Typically, it will contain at least the following five named columns (which correspond to the default for argument 'pedicolname' to gls.batch()): "FAMID", (family IDs), "ID" (unique individual IDs), "PID" (paternal ID), "MID" (maternal ID), and "SEX" (coded 1 for male, 2 for female). The paternal and maternal IDs of founders must either be 0 or NA. Argument 'pedi.dat' may also contain any/all of the following three named columns, the effects of which are described below under "Details": "ZYGOSITY", "ADOPTED", and "INDEP". The "ZYGOSITY" column must contain a value of 1 for each MZ twin, and a value of 2 for each DZ twin. The "ADOPTED" column must be a dummy variable for adoptive status, i.e. with value 1 for adoptees and value 0 otherwise (NA's are treated as 0). The "INDEP" column must be a dummy variable for whether the individual should be treated as an "independent observation" (family-type 6), with 1 for "yes" and 0 for "no" (NA's are treated as 0).
phen.dat	A phenotype file, as a data frame with named columns. At the bare minimum, it must contain a column of unique individual IDs, named "ID". The value returned by FSV.frompedi is this same data frame, with columns named "FTYPE" and "INDIV" appended thereto, unless columns with those names were already present, in which case their contents will be overwritten with new values. Any other named columns in 'phen.dat' are ignored.

Details

RFGLS recognizes six recognized family types, which are distinguished primarily by how the offspring in the family are related to one another:

- FTYPE=1, containing MZ twins;
- FTYPE=2, containing DZ twins;
- FTYPE=3, containing adoptees;
- FTYPE=4, containing non-twin full siblings;

- FTYPE=5, "mixed" families containing one biological offspring and one adoptee;
- FTYPE=6, containing "independent observations" who do not fit into a four-person nuclear family.

It is assumed that all offspring except adoptees are biological children of the parents in the family. The four individual codes are:

- INDIV=1 is for "Offspring #1;"
- INDIV=2 is for "Offspring #2;"
- INDIV=3 is for mothers;
- INDIV=4 is for fathers.

The distinction between "Offspring #1" and "#2" is mostly arbitrary, except that in "mixed" families(FTYPE=5), the biological offspring MUST have INDIV=1, and the adopted offspring, INDIV=2.

The way that FSV.frompedi() assigns family-types and individual codes to participants depends upon the presence/absence of eight named columns in 'pedi.dat': "ID", "FAMID", "PID", "MID", "SEX", "ZYGOSITY", "ADOPTED", "INDEP". If any of the first five of these are absent, all participants are assigned FTYPE=6 and INDIV=1, with a warning. Assuming that those first five columns are present, what FSV.frompedi() does depends upon the presence/absence of the other three columns, as follows.

If "INDEP" is present, then FSV.frompedi() assigns FTYPE=6, INDIV=1 to participants with INDEP=1. These participants are then disregarded for the rest of the job. Like the other functions in this package, FSV.frompedi() treats participants with FTYPE=6 as the sole members of their own family units, and not as part of the family corresponding to their family ID.

If "ZYGOSITY" and "ADOPTED" are both absent, then (after first checking for "INDEP", as above), all participants are assigned FTYPE=4. Non-founders are identified as offspring, and participants whose IDs appear in "MID" or "PID" are assigned INDIV=3 or INDIV=4, respectively. Offspring individual codes are adjusted so that each family has only one instance each of INDIV=1 and INDIV=2. If more than two offspring are identified in a family, or if more than one mother or more than one father are identified in family, these participants are forced to FTYPE=6, INDIV=1. Also, any participant not otherwise assigned an individual code is given FTYPE=6, INDIV=1.

If "ZYGOSITY" is present but "ADOPTED" is absent, then FSV.frompedi() behaves similarly, except that (after first checking for "INDEP", as above) known twins are identified as offspring, and participants belonging to a family containing at least one twin are assigned FTYPE=1 (for MZ) or FTYPE=2 (for DZ), as the case may be. Member of families with no twins are assigned FTYPE=4. The program then proceeds as described in the immediately preceding paragraph.

If "ADOPTED" is present, FSV.frompedi() first makes some simple family-type assignments: if "ZYGOSITY" is present, to FTYPE=1 and FTYPE=2 as appropriate (see above), and then if "INDEP" is present, to FTYPE=6, INDIV=1 as appropriate (see above). Then, within each family, the program resolves each member in order of ID, from least to greatest. The first non-founder is assigned INDIV=1, the second, INDIV=2, and any thereafter, FTYPE=6, INDIV=1. The first adoptee is assigned INDIV=2, the second, INDIV=1, and any thereafter, FTYPE=6, INDIV=1. The first female non-adoptee non-founder is assigned INDIV=3, and any others are assigned FTYPE=6, INDIV=1. The first male non-adoptee non-founder is assigned INDIV=4, and any others are assigned FTYPE=6, INDIV=1. If family-type has not yet been assigned, then it is resolved as FTYPE=3 if there are two adoptees, FTYPE=5 if there is one adoptee and one biological offspring, and as FTYPE=4 otherwise.

Function FSV.frompedi() produces a warning whenever it forces a non-founder to FTYPE=6, INDIV=1.

Note that there is definitely a degree of arbitrariness in how ambiguous cases are resolved, in that FSV.frompedi() scans through the pedigree file from top to bottom *after it has sorted the file* by family ID, and by ID within the same family. So for example, if two participants in the same family are both provisionally assigned INDIV=3, then the apparent mother with the smaller ID retains INDIV=3, and the other is forced to FTYPE=6, INDIV=1.

Value

A data frame, containing the same columns as 'phen.dat', with the addition of "FTYPE" and "INDIV". Usually, this data frame will simply be 'phen.dat' with "FTYPE" and "INDIV" appended thereto. However, if 'phen.dat' contained columns named "FTYPE" or "INDIV", the values in these columns will be overwritten with the new values produced by FSV.frompedi().

Author(s)

Robert M. Kirkpatrick <kirk0191@umn.edu>.

See Also

gls.batch, gls.batch.get

Examples

```
data(pheno)
data(pedigree)
table(pheno$FTYPE) ##<--Frequencies of correct family types.


fsvtest1 <- FSV.frompedi(pedi.dat=pedigree,
  phen.dat=data.frame(ID=pheno[,2])) ##<--Bare minimum phenotype file.
table(fsvtest1$FTYPE) ##<--Not correct, because pedigree file
                      ##doesn't have enough additional info
                      ##to recover the actual family-types
                      ##and individual codes.

#Create "ZYGOSITY" column:
pedigree$ZYGOSITY <- NA
pedigree$ZYGOSITY[pheno$FTYPE==1 & pheno$INDIV<3] <- 1
pedigree$ZYGOSITY[pheno$FTYPE==2 & pheno$INDIV<3] <- 2

fsvtest2 <- FSV.frompedi(pedi.dat=pedigree,phen.dat=data.frame(ID=pheno[,2]))
table(fsvtest2$FTYPE) ##<--Still not right, because pedigree file
                      ##lacks info about adoptees.

#Create "ADOPTED" column:
pedigree$ADOPTED <- 0
pedigree$ADOPTED[pheno$FTYPE==3 & pheno$INDIV<3] <- 1
pedigree$ADOPTED[pheno$FTYPE==5 & pheno$INDIV==2] <- 1
fsvtest3 <- FSV.frompedi(pedi.dat=pedigree,phen.dat=data.frame(ID=pheno[,2]))
table(fsvtest3$FTYPE) ##<--Almost there.

#Create "INDEP" column:
```

```
pedigree$INDEP <- 0
pedigree$INDEP[pheno$FTYPE==6] <- 1
fsvtest4 <- FSV.frompedi(pedi.dat=pedigree,phen.dat=data.frame(ID=pheno[,2]))
table(fsvtest4$FTYPE) ##<--Correct family types have been recovered.
table(pheno$FTYPE) ##<--Compare.
all(pheno$FTYPE==fsvtest4$FTYPE) ##<--TRUE.
```

---

geno *Simulated genotypic dataset*

Description

A dataset of simulated genotypes on 10 arbitrary SNPs, for the same simulees in datasets pheno and pedigree.

Usage

```
data(geno)
```

Format

A data frame containing only integers, with 1 row per SNP, and 1 column per subject. The row and column names are rs numbers and individual IDs, respectively.

Details

The genotypes are coded as counts of each SNPs reference allele on the HapMap (http://hapmap.ncbi.nlm.nih.gov/) positive strand. First, 10 SNPs, one each from the first 10 human chromosomes, were selected arbitrarily. Then, genotypes were generated for founders (parents, adoptees, and "independent observations"), under Hardy-Weinberg equilibrium, using the allele frequencies from HapMap's CEU reference data (representing Caucasians of European Ancestry). After that, genes were "dropped" from parents to offspring. Subjects' genotypes on the arbitrarily chosen effect locus, rs7681769, were conditioned upon to simulate quantitative phenotype scores (Zscore in dataset pheno). The true effect size in the data-generating distribution is approximately 0.5% of phenotypic variance.

Dataset geno has both row and column names, which is acceptable for a data frame to be provided as argument 'genfile' to gls.batch(). However, a genotype file to be read from disk should have NEITHER an extra column of row labels nor an extra row of column headers.

Details about each SNP may be found in dataset map.

Examples

```
data(geno)
str(data.frame(t(geno)))
round(cor(t(geno)),3)  ##<--SNPs are on different chromosomes, so no LD.
##Also see examples for functions fgls(), gls.batch(), and gls.batch.get().
```

gls.batch *Generalized least-squares batch analysis.*

Description

Fits a generalized least-squares regression model to test association between a quantitative phenotype and all SNPs in a genotype file, one at a time, via Rapid Feasible Generalized Least Squares. For each SNP, genotype is treated as a fixed effect, and the residual variance-covariance matrix is also estimated. In each trait-SNP association test, the `fgls()` function is used for parameter estimation.

The arguments to `gls.batch()` may be regarded as belonging to four groups:

1. those concerning how to load the **input** (‘`phenfile`’, ‘`genfile`’, ‘`pedifile`’, ‘`covmtxfile.in`’, ‘`theta`’, ‘`snp.names`’, ‘`input.mode`’, ‘`pediheader`’, ‘`pedicolname`’, ‘`sep.phe`’, ‘`sep.gen`’, ‘`sep.ped`’);
2. those concerning **what to do** with the input, that is, the actual analysis (‘`phen`’, ‘`covars`’, ‘`med`’);
3. those concerning how to provide the **output** (‘`outfile`’, ‘`col.names`’, ‘`return.value`’, ‘`covmtxfile.out`’, ‘`covmtxparams.out`’);
4. and those that merely trigger **optional checks** on the input (‘`sizeLab`’, ‘`Mz`’, ‘`Bo`’, ‘`Ad`’, ‘`Mix`’, ‘`indobs`’).

Usage

```
gls.batch(phenfile,genfile,pedifile,covmtxfile.in=NULL,theta=NULL,
  snp.names=NULL,input.mode=c(1,2,3),pediheader=FALSE,
  pedicolname=c("FAMID","ID","PID","MID","SEX"),
  sep.phe=" ",sep.gen=" ",sep.ped=" ",
  phen,covars=NULL,med=c("UN","VC"),
  outfile,col.names=TRUE,return.value=FALSE,
  covmtxfile.out=NULL,
  covmtxparams.out=NULL,
  sizeLab=NULL,Mz=NULL,Bo=NULL,Ad=NULL,Mix=NULL,indobs=NULL)
```

Arguments

`phenfile`	This can be either (1) a character string specifying a phenotype file on disk which includes the phenotypes and other covariates, or (2) a data frame object containing the same data. In either case, the data must be appropriately structured. See below under "Details."
`genfile`	This can be `NULL`, in which case no SNPs are analyzed, and `gls.batch()` conducts a single `fgls()` regression of the phenotype onto an intercept and covariates (if any). Otherwise, this argument can be either (1) a character string specifying a genotype file of genotype scores (such as 0,1,2, for the additive genetic model) to be read from disk, or (2) a data frame object containing them. In such a file, each row must represent a SNP, each column must represent a

subject, and there should NOT be column headers or row numbers. In such a data frame, the reverse holds: each row must represent a subject, and each column, a SNP (e.g. geno). If the data frame–say, geno–need be transposed, then use genfile=data.frame(t(geno)). Using a matrix instead of a data frame is not recommended, because it makes the process of merging data very memory-intensive, and will likely overflow R's workspace unless the sample size or number of SNPs is quite small.

Note that genotype scores need not be integers; they can also be numeric. So, gls.batch() can be used to analyze imputed dosages, etc.

pedifile — This can be either (1) a character string specifying the pedigree file corresponding to 'genfile', to be read from disk, or (2) a data frame object containing this pedigree information. At minimum, 'pedifile' must have a column of subject IDs, named "ID", ordered in the same order as subjects' genotypic data in 'genfile'. Every row in 'pedifile' is matched to a participant in 'genfile'. That is, if reading files from disk (which is recommended), each row *i* of the pedigree file, which has *n* rows, matches column *i* of the genotype file, which has *n* columns. This is how the program matches subjects in the phenotype file to their genotypic data.

The pedigree file or data frame can also include other columns of pedigree information, like father's ID, mother's ID, etc. Argument 'pediheader' (see below) is an indicator of whether the pedigree file on disk has a header or not, with default as FALSE. Argument 'pedicolnames' (see below) gives the names that gls.batch() will assign to the columns of 'pedifile', and the default, c("FAMID","ID","PID","MID","SEX"), is the familiar "pedigree table" format. In any event, the user's input *must* somehow provide the program with a column of IDs, labeled as "ID".

covmtxfile.in — Optional; can be either (1) a character string specifying a file on disk from which the residual variance-covariance matrix is to be read, or (2) the matrix itself. If NULL, then gls.batch() will estimate this matrix. The file to be read in must be a single column, with a header, containing the contents of the 'blocks' of an object of class bdsmatrix; no other file structures are presently compatible. If 'covmtxfile.in' is an actual matrix object, then using one of class bdsmatrix is a virtual requirement. See below under "Details" for more information.

theta — An optional vector of previously estimated (or known) residual-covariance parameters. Defaults to NULL, in which case it is ignored. Otherwise, it must be a numerical vector of of either length 12 if med="UN", or of length 3 if med="VC". Each vector element provides the value for the parameter corresponding to its index (serial position). Values of NA are accepted for extraneous parameters. See fgls(), under "Details," for which parameters correspond to which indices. Note that at least one of 'covmtxfile.in' and 'theta' must be NULL.

snp.names — An optional character vector with length equal to the number of markers in 'genfile', providing names for those markers. Defaults to NULL, in which case generic SNP names are used. Ignored if 'genfile' is NULL.

input.mode — Either 1 (default), 2, or 3, which tells gls.batch() where to look for the family-structure variables "FTYPE" and "INDIV" (see below, under "Details"). By default, gls.batch() first looks in the phenotype file, and if the variables are not found there, then looks in the pedigree file, and if the variables are not

there, attempts to create them from information available in the pedigree file, via FSV.frompedi(). If input.mode=2, then gls.batch() skips looking in the phenotype file, and begins by looking in the pedigree file. If input.mode=3, then gls.batch() skips looking in the phenotype file and pedigree file, and goes straight to FSV.frompedi().

pediheader — A logical indicator specifying whether the pedigree file to be read from disk has a header row, to ensure it is read in correctly. Even if TRUE, gls.batch() assigns the values in 'pedicolname' to the column names after the pedigree file has been read in. Defaults to FALSE. Also see 'pedifile' above, and under "Details" below.

pedicolname — A vector of character strings giving the column names that gls.batch() will assign to the columns of the pedigree file (starting with the first column and moving left to right). The default, c("FAMID","ID","PID","MID","SEX"), is the familiar "pedigree table" format. The two criteria this vector must have are that it must (1) assign the name "ID" to the column of subject IDs in the pedigree file, and (2) its length must not exceed the number of columns of the pedigree file. If its length is less than the number of columns, columns to which it does not assign a name are discarded. Also see 'pedifile' above, and under "Details" below.

sep.phe — Separator character of the phenotype file to be read from disk. Defaults to a single space.

sep.gen — Separator character of the genotype file to be read from disk. Defaults to a single space.

sep.ped — Separator character of the pedigree file. Defaults to a single space.

phen — A character string specifying the phenotype (column name) in the phenotype file to be analyzed.

covars — A character string or character vector that holds the (column) names of the covariates, in the phenotype file, to be used in the regression model. Defaults to NULL, in which case no covariates are included.

med — A character string, either "UN" or "VC", which are the two RFGLS methods described by Li et al. (2011). If "UN" (default), which stands for "unstructured," the residual covariance matrix will be constructed from, at most, 12 parameters (8 correlations and 4 variances). If "VC", which stands for "variance components," the residual covariance matrix will be constructed from, at most, 3 variance components (additive-genetic, shared-environmental, and unshared-environmental). For more information, see fgls().

outfile — Either a character string specifying the path and filename for the output file to be written, or NULL, in which case no output file is written. The output file contains the SNP analysis results, so argument 'outfile' is ignored if 'genfile' is NULL. Note that gls.batch() will not simultaneously accept outfile=NULL and return.value=FALSE.

Users are warned that if a file with the same path and filename already exists, gls.batch() will overwrite it!

col.names — A logical indicator specifying whether to write column names in the output file to be written to disk. Defaults to TRUE.

`return.value` A logical indicator specifying whether function `gls.batch()` should actually return a value. Defaults to `FALSE`, in which case the function merely returns `NULL`. If `TRUE` and non-`NULL` value was supplied to '`genfile`', the function returns a data frame containing the results of the SNP analyses(i.e., the output file as a data frame). If `TRUE` and `genfile=NULL`, the function returns the `fgls()` output from a regression of the phenotype onto an intercept and covariates (if any). Note that `gls.batch()` will not simultaneously accept `outfile=NULL` and `return.value=FALSE`.

`covmtxfile.out` An optional character string specifying the filename and path to which the residual covariance matrix, if it is to be constructed (i.e., if `covmtxfile.in=NULL`), will be written. The default is `NULL`, in which case no such file is written to disk. See below under "Details" for more information.

Users are warned that if a file with the same path and filename already exists, `gls.batch()` will overwrite it!

`covmtxparams.out`

An optional character string specifying the filename and path to which the vector of residual-covariance parameters, if they are to be estimated (i.e., if '`covmtxfile.in`' and '`theta`' are both `NULL`), will be written. The default is `NULL`, in which case no such file is written to disk. See below under "Details" for more information.

Users are warned that if a file with the same path and filename already exists, `gls.batch()` will overwrite it!

`sizeLab` This is an optional argument, and may be eliminated in future versions of this package. Defaults to `NULL`; otherwise, must be a character string, and if the number of characters in the string is not equal to the size of the largest family in the data, `gls.batch()` will produce a warning.

`Mz, Bo, Ad, Mix`

These are optional logical indicators that specify whether families containing MZ twins ('`MZ`'; family-type 1), DZ twins or full siblings ('`Bo`'; family-types 2 and 4), two adoptees ('`Ad`'; family-type 3), or 1 biological offspring and 1 adoptee ('`Mix`'; family-type 5) are present in the data. The values of each are checked against the actual family types present, after loading and merging the data and trimming out incomplete cases, and a warning is generated for each mismatch. If any of these four arguments is `NULL` (which is the default), the check corresponding to that family type is skipped.

`indobs` An optional logical indicator of whether there are "independent observations" who do not fit into a four-person nuclear family present in the data. After loading and merging the data and trimming out incomplete cases, the value of '`indobs`' is checked against whether such individuals are actually present, and a warning is generated in case of a mismatch. If `indobs=NULL`, which is the default, this check is skipped.

Details

Reference is frequently made throughout this documentation to the "phenotype file," the "genotype file," and so forth, because `gls.batch()` was intended to be used with potentially large datafiles to be read from disk. This should be evident from the presence of the word "file" in the names of many of this function's arguments, and the fact that all of those arguments may be character strings

providing a filename and path. However, it can also accept the data if the file has already been loaded into R's workspace as a data frame object, in which case "the [whatever] file" should be taken to refer to such a data frame. For details specific to each argument, see above.

The function `gls.batch()` first reads in the files and merges them into a data frame with columns of family-structure information, phenotype, covariates, and genotypes. Then, it creates a '`tlist`' vector and a '`sizelist`' vector, which comprise the family labels and family sizes in the data. Finally, it carries out single-SNP association analyses for all the SNPs in the genotype file.

At the *bare minimum*, the phenotype file must contain columns named `"ID"`, `"FAMID"`, and whatever character string is supplied to '`phen`'. These columns respectively contain individual IDs, family IDs, and phenotype scores; individual IDs must be unique.

At the *bare minimum*, the pedigree file need only contain a column consisting of unique individual IDs, corresponding to the label `"ID"` in '`pedicolname`'. The number of participants in the pedigree file must equal the number of participants in the genotype file, with participants ordered the same way in both files. However, the default value for argument '`pedicolname`' (see above) assumes five columns, in the familiar "pedigree table" format.

The phenotype file or pedigree file may also contain the two key family-structure variables, `"FTYPE"` (family-type) and `"INDIV"` (individual code). If both contain these variables, then by default, they are read from the phenotype file (but see argument '`input.mode`' above). There are six recognized family types, which are distinguished primarily by how the offspring in the family are related to one another:

- `FTYPE=1`, containing MZ twins;
- `FTYPE=2`, containing DZ twins;
- `FTYPE=3`, containing adoptees;
- `FTYPE=4`, containing non-twin full siblings;
- `FTYPE=5`, "mixed" families containing one biological offspring and one adoptee;
- `FTYPE=6`, containing "independent observations" who do not fit into a four-person nuclear family.

It is assumed that all offspring except adoptees are biological children of the parents in the family. The four individual codes are:

- `INDIV=1` is for "Offspring #1;"
- `INDIV=2` is for "Offspring #2;"
- `INDIV=3` is for mothers;
- `INDIV=4` is for fathers.

The distinction between "Offspring #1" and "#2" is mostly arbitrary, except that in "mixed" families(`FTYPE=5`), the biological offspring MUST have `INDIV=1`, and the adopted offspring, `INDIV=2`. If the phenotype file contains variables `"FTYPE"` and `"INDIV"`, it should be ordered by family ID (`"FAMID"`), and by individual code `"INDIV"` within family ID. Note that `gls.batch()` treats participants with `FTYPE=6` as the sole members of their own family units, and not as part of the family corresponding to their family ID.

If neither the phenotype nor pedigree file contain `"FTYPE"` and `"INDIV"`, `gls.batch()` will construct them via `FSV.frompedi()`.

When one is conducting parallel analyses on a computing array, judicious use of arguments 'covmtxfile.in', 'theta', 'covmtxparams.out', and 'covmtxfile.out' can save time. For example, suppose one is analyzing different SNP sets in parallel but using a common phenotype file for all. In this case, one could calculate the residual covariance matrix ahead of time and write it to a file. Then, use the same filename and path for argument 'covmtxfile.in', for all jobs running in parallel. The matrix can be calculated by using gls.batch.get() and then fgls(). One could similarly obtain the residual-covariance parameters ahead of time, and supply them as a vector to 'theta' in all jobs running in parallel.

Value

If return.value=FALSE, then gls.batch() simply returns NULL. If return.value=TRUE and genfile=NULL, then gls.batch() returns the fgls() output from a regression of the phenotype onto an intercept and covariates (if any). If return.value=TRUE and 'genfile' is non-NULL, then gls.batch() returns a data frame containing the results of the single-SNP analyses, 1 row per SNP. Specifically, this data frame includes the following named columns:

- snp (character): the names of the SNPs; equal to 'snp.names' if any were supplied.
- coef (numeric): the regression coefficients of the SNPs.
- se (numeric): estimated standard errors of SNPs' regression coefficients.
- t.stat (numeric): *t*-statistics, i.e. regression coefficients divided by their estimated standard errors.
- df (integer): degrees-of-freedom (see df.residual, from fgls() output).
- pval (numeric): two-tailed *p*-values, from corresponding *t*-statistics and degrees-of-freedom.

Function gls.batch() also has optional side effects of writing as many as three files to disk, depending on arguments 'outfile', 'covmtxfile.out', and 'covmtxparams.out'. Note that if a file is written for 'outfile', that file will contain the single-SNP analysis results described above.

Author(s)

Xiang Li <lixxx554@umn.edu>, Robert M. Kirkpatrick <kirk0191@umn.edu>, and Saonli Basu <saonli@umn.edu> .

References

Li X, Basu S, Miller MB, Iacono WG, McGue M: A Rapid Generalized Least Squares Model for a Genome-Wide Quantitative Trait Association Analysis in Families. *Human Heredity* 2011;71:67-82 (DOI: 10.1159/000324839)

See Also

fgls, pheno

Examples

```
data(pheno)
data(geno)
data(map)
data(pedigree)
data(rescovmtx)
minigwas <- gls.batch(
  phenfile=pheno,genfile=data.frame(t(geno)),pedifile=pedigree,
  covmtxfile.in=rescovmtx, #<--Precomputed, to save time.
  theta=NULL,snp.names=map[,2],input.mode=c(1,2,3),pediheader=FALSE,
  pedicolname=c("FAMID","ID","PID","MID","SEX"),
  sep.phe=" ",sep.gen=" ",sep.ped=" ",
  phen="Zscore",covars="IsFemale",med=c("UN","VC"),
  outfile=NULL,col.names=TRUE,return.value=TRUE,
  covmtxfile.out=NULL,covmtxparams.out=NULL,
  sizeLab=NULL,Mz=NULL,Bo=NULL,Ad=NULL,Mix=NULL,indobs=NULL)
minigwas
```

---

gls.batch.get *Data restructuring for* fgls().

Description

Carries out the data restructuring performed by gls.batch(). Useful if calling fgls() directly.

Several arguments to gls.batch.get() are accepted only for the sake of parallelism with gls.batch(), and are ignored: 'covmtxfile.in', 'theta', 'outfile', 'col.names', 'return.value', 'covmtxfile.out', and 'covmtxparams.out'.

Usage

```
gls.batch.get(phenfile,genfile,pedifile,covmtxfile.in=NULL,theta=NULL,
  snp.names=NULL,input.mode=c(1,2,3),pediheader=FALSE,
  pedicolname=c("FAMID","ID","PID","MID","SEX"),
  sep.phe=" ",sep.gen=" ",sep.ped=" ",
  phen,covars=NULL,med=c("UN","VC"),
  outfile,col.names=TRUE,return.value=FALSE,
  covmtxfile.out=NULL,
  covmtxparams.out=NULL,
  sizeLab=NULL,Mz=NULL,Bo=NULL,Ad=NULL,Mix=NULL,indobs=NULL)
```

Arguments

phenfile	This can be either (1) a character string specifying a phenotype file on disk which includes the phenotypes and other covariates, or (2) a data frame object containing the same data. In either case, the data must be appropriately structured. See below under "Details."

genfile — This can be NULL, in which case no SNP data are loaded. Otherwise, this argument can be either (1) a character string specifying a genotype file of genotype scores (such as 0,1,2, for the additive genetic model) to be read from disk, or (2) a data frame object containing them. In such a file, each row must represent a SNP, each column must represent a subject, and there should NOT be column headers or row numbers. In such a data frame, the reverse holds: each row must represent a subject, and each column, a SNP (e.g. geno). If the data frame–say, geno–need be transposed, then use genfile=data.frame(t(geno)). Using a matrix instead of a data frame is not recommended, because it makes the process of merging data very memory-intensive, and will likely overflow R's workspace unless the sample size or number of SNPs is quite small.

Note that genotype scores need not be integers; they can also be numeric. So, gls.batch() can be used to analyze imputed dosages, etc.

pedifile — This can be either (1) a character string specifying the pedigree file corresponding to 'genfile', to be read from disk, or (2) a data frame object containing this pedigree information. At minimum, 'pedifile' must have a column of subject IDs, named 'ID', ordered in the same order as subjects' genotypic data in 'genfile'. Every row in 'pedifile' is matched to a participant in 'genfile'. That is, if reading files from disk (which is recommended), each row *i* of the pedigree file, which has *n* rows, matches column *i* of the genotype file, which has *n* columns. This is how the program matches subjects in the phenotype file to their genotypic data.

The pedigree file or data frame can also include other columns of pedigree information, like father's ID, mother's ID, etc. Argument 'pediheader' (see below) is an indicator of whether the pedigree file on disk has a header or not, with default as FALSE. Argument 'pedicolnames' (see below) gives the names that gls.batch.get() will assign to the columns of 'pedifile', and the default, c("FAMID","ID","PID","MID","SEX"), is the familiar "pedigree table" format. In any event, the user's input *must* somehow provide the program with a column of IDs, labeled as "ID".

covmtxfile.in — Accepted but not used.

theta — Accepted but not used.

snp.names — An optional character vector with length equal to the number of markers in 'genfile', providing names for those markers. Defaults to NULL, in which case generic SNP names are used. Ignored if 'genfile' is NULL.

input.mode — Either 1 (default), 2, or 3, which tells gls.batch.get() where to look for the family-structure variables "FTYPE" and "INDIV" (see below, under "Details"). By default, gls.batch.get() first looks in the phenotype file, and if the variables are not found there, then looks in the pedigree file, and if the variables are not there, attempts to create them from information available in the pedigree file, via FSV.frompedi(). If input.mode=2, then gls.batch.get() skips looking in the phenotype file, and begins by looking in the pedigree file. If input.mode=3, then gls.batch.get() skips looking in the phenotype file and pedigree file, and goes straight to FSV.frompedi().

pediheader — A logical indicator specifying whether the pedigree file to be read from disk has a header row, to ensure it is read in correctly. Even if TRUE, gls.batch()

assigns the values in 'pedicolname' to the columns after it has been read in. Defaults to FALSE. Also see 'pedifile' above and under "Details" below.

pedicolname — A vector of character strings giving the column names that gls.batch.get() will assign to the columns of the pedigree file (starting with the first column and moving left to right). The default, c("FAMID","ID","PID","MID","SEX"), is the familiar "pedigree table" format. The two criteria this vector must have are that it must (1) assign the name "ID" to the column of subject IDs in the pedigree file, and (2) its length must not exceed the number of columns of the pedigree file. If its length is less than the number of columns, columns to which it does not assign a name are discarded. Also see 'pedifile' above, and under "Details" below.

sep.phe — Separator character of the phenotype file to be read from disk. Defaults to a single space.

sep.gen — Separator character of the genotype file to be read from disk. Defaults to a single space.

sep.ped — Separator character of the pedigree file. Defaults to a single space.

phen — A character string specifying the phenotype (column name) in the phenotype file to be analyzed.

covars — A character string or character vector that holds the (column) names of the covariates, in the phenotype file, to be used in the regression model. Defaults to NULL, in which case no covariates are included.

med — A character string, either "UN" or "VC", which are the two RFGLS methods described by Li et al. (2011). If "UN" (default), which stands for "unstructured," the residual covariance matrix will be constructed from, at most, 12 parameters (8 correlations and 4 variances). If "VC", which stands for "variance components," the residual covariance matrix will be constructed from, at most, 3 variance components (additive-genetic, shared-environmental, and unshared-environmental).

outfile — Accepted but not used.

col.names — Accepted but not used.

return.value — Accepted but not used.

covmtxfile.out — Accepted but not used.

covmtxparams.out — Accepted but not used.

sizeLab — This is an optional argument, and may be eliminated in future versions of this package. Defaults to NULL; otherwise, must be a character string. If the number of characters in the string is not equal to the size of the largest family in the data, gls.batch.get() will produce a warning.

Mz, Bo, Ad, Mix — These are optional logical indicators that specify whether families containing MZ twins ('MZ'; family-type 1), DZ twins or full siblings ('Bo'; family-types 2 and 4), two adoptees ('Ad'; family-type 3), or 1 biological offspring and 1 adoptee ('Mix'; family-type 5) are present in the data. The values of each are checked against the actual family types present, after loading and merging the data and trimming out incomplete cases, and a warning is generated for each

mismatch. If any of these four arguments is NULL (which is the default), the check corresponding to that family type is skipped.

indobs — An optional logical indicator of whether there are "independent observations" who do not fit into a four-person nuclear family present in the data. After loading and merging the data and trimming out incomplete cases, the value of 'indobs' is checked against whether such individuals are actually present, and a warning is generated in case of a mismatch. If indobs=NULL, which is the default, this check is skipped.

Details

Though originally used for debugging purposes, gls.batch.get() was included because it facilitates directly invoking fgls() when the need arises. This function first reads in the files and merges the files into a data frame with columns of family-structure information, phenotype, covariates, and genotypes. It then creates a 'tlist' vector and a 'sizelist' vector, which comprise the family labels and family sizes in the data. It returns a list containing the merged data frame, and the 'tlist' and 'sizelist' vectors.

At the *bare minimum*, the phenotype file must contain columns named "ID", "FAMID", and whatever character string is supplied to 'phen'. These columns respectively contain individual IDs, family IDs, and phenotype scores; individual IDs must be unique.

At the *bare minimum*, the pedigree file need only contain a column consisting of unique individual IDs, corresponding to the label "ID" in 'pedicolname'. The number of participants in the pedigree file must equal the number of participants in the genotype file, with participants ordered the same way in both files. However, the default value for argument 'pedicolname' (see above) assumes five columns, in the familiar "pedigree table" format.

The phenotype file or pedigree file may also contain the two key family-structure variables, "FTYPE" (family-type) and "INDIV" (individual code). If both contain these variables, then by default, they are read from the phenotype file (but see argument 'input.mode' above). There are six recognized family types, which are distinguished primarily by how the offspring in the family are related to one another:

- FTYPE=1, containing MZ twins;
- FTYPE=2, containing DZ twins;
- FTYPE=3, containing adoptees;
- FTYPE=4, containing non-twin full siblings;
- FTYPE=5, "mixed" families containing one biological offspring and one adoptee;
- FTYPE=6, containing "independent observations" who do not fit into a four-person nuclear family.

It is assumed that all offspring except adoptees are biological children of the parents in the family. The four individual codes are:

- INDIV=1 is for "Offspring #1;"
- INDIV=2 is for "Offspring #2;"
- INDIV=3 is for mothers;
- INDIV=4 is for fathers.

The distinction between "Offspring #1" and "#2" is mostly arbitrary, except that in "mixed" families(FTYPE=5), the biological offspring MUST have INDIV=1, and the adopted offspring, INDIV=2. If the phenotype file contains variables "FTYPE" and "INDIV", it should be ordered by family ID ("FAMID"), and by individual code "INDIV" within family ID. Note that gls.batch.get() treats participants with FTYPE=6 as the sole members of their own family units, and not as part of the family corresponding to their family ID.

If neither the phenotype nor pedigree file contain "FTYPE" and "INDIV", gls.batch() will construct them via FSV.frompedi().

Value

A list with these three components:

test.dat — The merged data frame of family-structure variables, phenotype, covariates, and genotypes. Participants of family-type 6 will be moved to the end of the data frame. There will also be three additional columns:

- famsize (integer): The size of the family to which each participant belongs.
- unisid (character): Single-character representation of each participants' "FTYPE" and "INDIV". Adoptees have "a", MZ twins have "c", non-MZ-twin biological offspring have "b", mothers have "m", fathers have "f", and members of family-type 6 have NA.
- famlab (character): "Family labels;" the unisid's of the members of each participant's family, pasted together in order of "INDIV".

tlist — A character vector of family labels, with length equal to the number of families in the data (each participant of family-type 6 is treated as a separate family). The names of its components are the family IDs.

sizelist — A vector of family sizes, with length equal to the number of families in the data (each participant of family-type 6 is treated as a separate family). The names of its components are the family IDs.

Author(s)

Xiang Li <lixxx554@umn.edu>, Robert M. Kirkpatrick <kirk0191@umn.edu>, and Saonli Basu <saonli@umn.edu> .

See Also

fgls, gls.batch

Examples

```
data(pheno)
data(geno)
data(map)
data(pedigree)
foo <- gls.batch.get(
  phenfile=pheno,genfile=data.frame(t(geno)),pedifile=pedigree,
  covmtxfile.in=NULL,theta=NULL,snp.names=map[,2],input.mode=c(1,2,3),
  pediheader=FALSE,pedicolname=c("FAMID","ID","PID","MID","SEX"),
```

```
  sep.phe=" ",sep.gen=" ",sep.ped=" ",
  phen="Zscore",covars="IsFemale",med=c("UN","VC"),
  outfile,col.names=TRUE,return.value=FALSE,
  covmtxfile.out=NULL,
  covmtxparams.out=NULL,
  sizeLab=NULL,Mz=NULL,Bo=NULL,Ad=NULL,Mix=NULL,indobs=NULL)
olsmod <- lm(   ##<--OLS regression could be applied to the merged dataset...
    Zscore ~ rs3934834 + IsFemale, data=foo$test.dat)
summary(olsmod)  ##<--...but the standard errors and t-statistics will not be valid.

##The 'tlist' vector can be useful for figuring out if any residual-covariance
##parameters are poorly identified in the data:
pheno2 <- subset(pheno, (pheno$INDIV<3 & pheno$FAMID>20) |
                        (pheno$ID %in% c(11,12,13,21,22,23)))
foo2 <- gls.batch.get(
  phenfile=pheno2,
  genfile=data.frame(t(geno)),pedifile=pedigree,
  covmtxfile.in=NULL,theta=NULL,snp.names=map[,2],input.mode=c(1,2,3),
  pediheader=FALSE,pedicolname=c("FAMID","ID","PID","MID","SEX"),
  sep.phe=" ",sep.gen=" ",sep.ped=" ",
  phen="Zscore",covars="IsFemale",med=c("UN","VC"),
  outfile,col.names=TRUE,return.value=FALSE,
  covmtxfile.out=NULL,
  covmtxparams.out=NULL,
  sizeLab=NULL,Mz=NULL,Bo=NULL,Ad=NULL,Mix=NULL,indobs=NULL)
table(foo2$tlist)
##Only two families have the label 'ccm', that is, only two have
##a mother.  So, if calling fgls()
##with med="UN", it would probably be a good idea to drop the
##mother variance [drop=10], or the biological mother-offspring
##correlation [drop=2], or both [drop=c(2,10)].
```

`map` *SNP "map" file*

Description

A table of information about the 10 SNPs in dataset geno, in a format reminiscent of a PLINK binary map (.bim) file.

Usage

```
data(map)
```

Format

A data frame with 10 rows, 1 per SNP. There are 6 labeled columns, for the following variables:

`chr` Chromosome numbers

`SNP` A vector of SNP rs numbers, as character strings.

`position` SNP position, in base-pairs, from HapMap Build 36.

`Ref_Allele` Vector of reference alleles, as character strings, from the HapMap positive strand. Genotypes on each SNP in dataset geno are counts of its reference allele.

`Other_Allele` The "other allele." Also a vector of character strings.

`Freq_Ref_Allele` The relative frequency of the reference allele, in HapMap CEU reference data.

Details

See documentation for dataset geno for details on how the genotypic data on these SNPs was simulated. See documentation for dataset pheno for details on how the effect locus, rs7681769, was used to simulate phenotypic data.

Examples

```
data(map)
str(map)
map
```

`pedigree` *Pedigree table*

Description

A pedigree table for the same simulees as in dataset pheno.

Usage

```
data(pedigree)
```

Format

A data frame with 4050 observations on the following 6 integer-valued variables:

`FAMID` "Family ID." Each family in the dataset is uniquely identified by a value of `FAMID`, which are all multiples of 10.

`ID` *Individual* ID. Each subject in the dataset is uniquely identified by a value of `ID`.

`PID` "Paternal ID." Coded 0 for founders (parents, adoptees, and "independent observations.")

`MID` "Maternal ID." Coded 0 for founders (parents, adoptees, and "independent observations.")

`SEX` Coded 1 for male and 2 for female.

Details

Merely a pedigree table in a commonly used format. Note that its column names are the default names that `gls.batch()` or `gls.batch.get()` assign to the pedigree file. However, the only column that those two functions *strictly* require is `ID`; see examples below.

Examples

```
data(pedigree)
data(pheno)
data(geno)
data(map)

foo <- gls.batch.get(
  phenfile=pheno,genfile=data.frame(t(geno)),pedifile=pedigree,
  covmtxfile.in=NULL,theta=NULL,snp.names=map[,2],
  input.mode=1,
  pediheader=FALSE,pedicolname=c("FAMID","ID","PID","MID","SEX"),
  sep.phe=" ",sep.gen=" ",sep.ped=" ",
  phen="Zscore",covars="IsFemale",med=c("UN","VC"),
  outfile,col.names=TRUE,return.value=FALSE,
  covmtxfile.out=NULL,
  covmtxparams.out=NULL,
  sizeLab=NULL,Mz=NULL,Bo=NULL,Ad=NULL,Mix=NULL,indobs=NULL)
str(foo)

##Also works, since phenfile provides 'FTYPE' and 'INDIV',
##and input.mode=1:
pedigree2 <- pedigree
pedigree2[,-2] <- NA    ##<--Change all but column 'ID' to NA.
foo2 <- gls.batch.get(
  phenfile=pheno,genfile=data.frame(t(geno)),
  pedifile=pedigree2,                       ##<--Note change.
  covmtxfile.in=NULL,theta=NULL,snp.names=map[,2],
  input.mode=1, ##<-- =2 or =3 would need more pedifile columns
  pediheader=FALSE,pedicolname=c("FAMID","ID","PID","MID","SEX"),
  sep.phe=" ",sep.gen=" ",sep.ped=" ",
  phen="Zscore",covars="IsFemale",med=c("UN","VC"),
  outfile,col.names=TRUE,return.value=FALSE,
  covmtxfile.out=NULL,
  covmtxparams.out=NULL,
  sizeLab=NULL,Mz=NULL,Bo=NULL,Ad=NULL,Mix=NULL,indobs=NULL)
str(foo2)
```

---

pheno *Simulated quantitative-trait dataset*

Description

A dataset of observations on a normally distributed phenotype, generated with means conditional on genotype at the effect locus. All six family-types recognized in package *RFGLS* are represented.

Usage

```
data(pheno)
```

Format

A data frame with 4050 observations on the following 6 variables:

`FAMID` "Family ID." Each family in the dataset is uniquely identified by a value of `FAMID`, which are all multiples of 10.

`ID` *Individual* ID. Each subject in the dataset is uniquely identified by a value of `ID`, which is equal to his/her `INDIV` plus his/her `FAMID`.

`FTYPE` "Family-type." *RFGLS* recognizes six different family-types, five of which are, at largest, four-person nuclear families (two parents, two offspring), distinguished by how the two offspring are related to one another:

1. MZ-twin families,
2. DZ-twin families,
3. Adoptive-offspring families,
4. Non-twin bio-offspring families,
5. "Mixed" families with one bio and one adopted offspring, and
6. "Independent observations," who do not fit into a four-person nuclear family.

`INDIV` "Individual code," which represents how a subject fits into his/her family: `INDIV=1` is for "Offspring #1," `INDIV=2` is for "Offspring 2," `INDIV=3` is for the mother, and `INDIV=4` is for the father. In families of `FTYPE=5`, the biological offspring has `INDIV=1`, and the adopted offspring, `INDIV=2`. All "independent observations" (i.e., `FTYPE=6`) have `INDIV=1`. Note that individuals in a given family are ordered by their `INDIV`.

`Zscore` The phenotype score.

`IsFemale` Binary indicator; a value of 1 indicates female sex. All offspring in families of type #2 (DZ twins) happen to be same-sex, but this is not a requirement.

Details

Each family's phenotype scores were generated from a multivariate normal distribution (`mvrnorm` in package *mvtnorm*) with a centroid defined conditional upon the family members' genotypes on the effect locus (rs7681769 in dataset geno), and a variance matrix with 1s on its diagonal and covariances (really, correlations) consistent with an additive heritability of 0.5 and a shared-environmentality of 0.2, but zero assortative mating.

Examples

```
data(pheno)
str(pheno)
qqnorm(pheno$Zscore[pheno$INDIV==1]) ##<--Normally distributed phenotype.
qqline(pheno$Zscore[pheno$INDIV==1])
##Also see examples for functions fgls(), gls.batch(), and gls.batch.get().
```

rescovmtx *Residual variance-covariance matrix.*

Description

The residual covariance matrix used in package examples. In the previous package version, its name was "resVCmtx".

Usage

```
data(rescovmtx)
```

Format

An object of class bdsmatrix (from package *bdsmatrix*). It is a 4050-by-4050 block-diagonal sparse matrix, with off-diagonal elements of 0. It has six slots, the descriptions of which may be found in the documentation for bdsmatrix-class.

Details

It is perhaps most instructive to see the syntax by which rescovmtx can be reproduced from datasets pheno, geno, and pedigree:

```
data(pheno)
data(geno)
data(pedigree)
foo <- gls.batch.get(
  phenfile=pheno,genfile=data.frame(t(geno)),pedifile=pedigree,
  covmtxfile.in=NULL,theta=NULL,snp.names=NULL,input.mode=c(1,2,3),
  pediheader=FALSE,pedicolname=c("FAMID","ID","PID","MID","SEX"),
  sep.phe=" ",sep.gen=" ",sep.ped=" ",
  phen="Zscore",covars="IsFemale",med=c("UN","VC"),
  outfile,col.names=TRUE,return.value=FALSE,
  covmtxfile.out=NULL,
  covmtxparams.out=NULL,
  sizeLab=NULL,Mz=NULL,Bo=NULL,Ad=NULL,Mix=NULL,indobs=NULL)
bar <- fgls(
  Zscore ~ IsFemale, data=foo$test.dat, tlist=foo$tlist,
  sizelist=foo$sizelist,med=c("UN","VC"),
  vmat=NULL,
  start=NULL, theta=NULL, drop=NULL, get.hessian=FALSE,
  optim.method="BFGS", control=list(), weights=NULL,
  sizeLab=NULL,Mz=NULL,Bo=NULL,Ad=NULL,Mix=NULL,indobs=NULL)
```

Then, bar$sigma is identical to rescovmtx.

Examples

```
data(rescovmtx)
str(rescovmtx)
##Also see examples for functions fgls() and gls.batch().
```

Index