Package 'SFSI'

August 24, 2024

Title Sparse Family and Selection Index

Version 1.4.1

Date 2024-08-23

Description Here we provide tools for the estimation of coefficients in penalized regressions when the (co)variance matrix of predictors and the covariance vector between predictors and response, are provided. These methods are extended to the context of a Selection Index (commonly used for breeding value prediction). The approaches offer opportunities such as the integration of high-throughput traits in genetic evaluations ('Lopez-Cruz et al., 2020') <doi:10.1038/s41598-020-65011-2> and solutions for training set optimization in Genomic Prediction ('Lopez-Cruz & de los Campos, 2021') <doi:10.1093/genetics/iyab030>.

URL https://github.com/MarcooLopez/SFSI

LazyLoad true

Depends R (>= 3.6.0)

Imports stats, scales, tensorEVD (>= 0.1.3), parallel, reshape2, viridis, igraph, stringr, ggplot2

Suggests BGLR, knitr, rmarkdown

VignetteBuilder knitr

Encoding UTF-8

License GPL-3

NeedsCompilation yes

Author Marco Lopez-Cruz [aut, cre], Gustavo de los Campos [aut], Paulino Perez-Rodriguez [ctb]

Maintainer Marco Lopez-Cruz <maraloc@gmail.com>

Repository CRAN

Date/Publication 2024-08-23 22:00:06 UTC

Contents

`1. Save/read binary files`

Save/read binary files

Description

Save/read a numeric data as a fortran-formatted binary file at a defined precision (single or double).

Usage

```
saveBinary(X, file = paste0(tempdir(), "/file.bin"),
           precision.format = c("double","single"),
           verbose = TRUE)

readBinary(file = paste0(tempdir(), "/file.bin"),
           rows = NULL, cols = NULL,
           drop = TRUE, verbose = TRUE)
```

Arguments

`X`	(numeric matrix) Data to save
`file`	(character) Name of the binary file to save/read
`precision.format`	(character) Either 'single' or 'double' for numeric precision and memory occupancy (4 bytes/32-bit or 8 bytes/64-bit, respectively) of the matrix to save

rows	(integer vector) Which rows are to be read from the file. Default rows=NULL will read all the rows
cols	(integer vector) Which columns are to be read from the file. Default cols=NULL will read all the columns
drop	Either TRUE or FALSE to whether return a uni-dimensional vector when data is a matrix with either 1 row or 1 column
verbose	TRUE or FALSE to whether printing file information

Value

Function 'saveBinary' does not return any value but print a description of the file saved.

Function 'readBinary' returns the data that was read.

Examples

```
require(SFSI)

# A numeric matrix
X = matrix(rnorm(500*100), ncol=100)

# Save matrix as double-precision
filename1 = paste0(tempdir(),"/Matrix1.bin")
saveBinary(X, filename1, precision.format="double")

# Save matrix as single-precision
filename2 = paste0(tempdir(),"/Matrix2.bin")
saveBinary(X, filename2, precision.format="single")

# Read the double-precision matrix
X2 = readBinary(filename1)
max(abs(X-X2))             # No loss of precision
file.info(filename1)$size  # Size of the file

# Read the single-precision matrix
X3 = readBinary(filename2)
max(abs(X-X3))             # Loss of precision
file.info(filename2)$size  # But smaller-size file

# Read specific rows and columns
rows = c(2,4,5,8,10)
cols = c(1,2,5)
(X2 = readBinary(filename1, rows=rows, cols=cols))
# Equal to:
X[rows,cols]
```

2. Covariance matrix transformations

Conversion of a covariance matrix to a distance/correlation matrix

Description

Transformation into correlation matrix or distance matrix from a covariance matrix

Usage

```
cov2dist(A, a = 1, inplace = FALSE)

cov2cor2(A, a = 1, inplace = FALSE)
```

Arguments

A	(numeric matrix) Variance-covariance matrix
inplace	TRUE or FALSE to whether operate directly on the input matrix. When TRUE no result is produced but the input A is modified. Default inplace=FALSE
a	(numeric) A number to multiply the whole resulting matrix by. Default a = 1

Details

For any variables X_i and X_j with mean zero and with sample vectors $\mathbf{x}_i = (x_{i1}, ..., x_{in})'$ and $\mathbf{x}_j = (x_{j1}, ..., x_{jn})'$, their (sample) variances are equal (up-to a constant) to their cross-products, this is, $var(X_i) = \mathbf{x}_i'\mathbf{x}_i$ and $var(X_j) = \mathbf{x}_j'\mathbf{x}_j$. Likewise, the covariance is $cov(X_i, X_j) = \mathbf{x}_i'\mathbf{x}_j$.

Distance. The Euclidean distance $d(X_i, X_j)$ between the variables expressed in terms of cross-products is

$$d(X_i, X_j) = \sqrt{\mathbf{x}_i'\mathbf{x}_i + \mathbf{x}_j'\mathbf{x}_j - 2\mathbf{x}_i'\mathbf{x}_j}$$

Therefore, the output distance matrix will contain as off-diagonal entries

$$d(X_i, X_j) = \sqrt{var(X_i) + var(X_j) - 2cov(X_i, X_j)}$$

while in the diagonal, the distance between one variable with itself is $d(X_i, X_i) = 0$

Correlation. The correlation between the variables is obtained from variances and covariances as

$$cor(X_i, X_j) = cov(X_i, X_j)/(sd(X_i)sd(X_j))$$

where $sd(X_i) = \sqrt{var(X_i)}$; while in the diagonal, the correlation between one variable with itself is $cor(X_i, X_i) = 1$

Variances are obtained from the diagonal values while covariances are obtained from the out-diagonal.

Value

Function 'cov2dist' returns a matrix containing the Euclidean distances. Function 'cov2cor2' returns a correlation matrix

Examples

```
require(SFSI)
data(wheatHTP)

X = scale(M)[1:100,]/sqrt(ncol(M))
A = tcrossprod(X)        # A 100x100 covariance matrix

# Covariance matrix to distance matrix
D = cov2dist(A)
# (it must equal (but faster) to:)
D0 = as.matrix(dist(X))
max(D-D0)

# Covariance to a correlation matrix
R = cov2cor2(A)
# (it must equal (but faster) to:)
R0 = cov2cor(A)
max(R-R0)

# Inplace calculation
A[1:5,1:5]
cov2dist(A, inplace=TRUE)
A[1:5,1:5]  # notice that A was modified
```

---

3. Solve an Elastic-Net problem

Coordinate Descent algorithm to solve Elastic-Net-type problems

---

Description

Computes the entire Elastic-Net solution for the regression coefficients for all values of the penalization parameter, via the Coordinate Descent (CD) algorithm (Friedman, 2007). It uses as inputs a variance matrix among predictors and a covariance vector between response and predictors

Usage

```
solveEN(Sigma, Gamma, alpha = 1, lambda = NULL, nlambda = 100,
        lambda.min = .Machine$double.eps^0.5, lambda.max = NULL,
        common.lambda = TRUE, beta0 = NULL, nsup.max = NULL,
        scale = TRUE, sdx = NULL, tol = 1E-5, maxiter = 1000,
        mc.cores = 1L, save.at = NULL, fileID = NULL,
        precision.format = c("double","single"), sparse = FALSE,
        eps = .Machine$double.eps*100, verbose = FALSE)
```

Arguments

`Sigma`	(numeric matrix) Variance-covariance matrix of predictors
`Gamma`	(numeric matrix) Covariance between response variable and predictors. If it contains more than one column, the algorithm is applied to each column separately as different response variables
`lambda`	(numeric vector) Penalization parameter sequence. Default is `lambda = NULL`, in this case a decreasing grid of `'nlambda'` lambdas will be generated starting from a maximum equal to `max(abs(Gamma)/alpha)` to a minimum equal to zero. If `alpha = 0` the grid is generated starting from a maximum equal to 5
`nlambda`	(integer) Number of lambdas generated when `lambda = NULL`
`lambda.min, lambda.max`	(numeric) Minimum and maximum value of lambda that are generated when `lambda = NULL`
`common.lambda`	`TRUE` or `FALSE` to whether computing the coefficients for a grid of lambdas common to all columns of `Gamma` or for a grid of lambdas specific to each column of `Gamma`. Default is `common.lambda = TRUE`
`beta0`	(numeric vector) Initial value for the regression coefficients that will be updated. If `beta0 = NULL` a vector of zeros will be considered. These values will be used as starting values for the first lambda value
`alpha`	(numeric) Value between 0 and 1 for the weights given to the L1 and L2-penalties
`scale`	`TRUE` or `FALSE` to scale matrix `Sigma` for variables with unit variance and scale `Gamma` by the standard deviation (`sdx`) of the corresponding predictor taken from the diagonal of `Sigma`
`sdx`	(numeric vector) Scaling factor that will be used to scale the regression coefficients. When `scale = TRUE` this scaling factor vector is set to the squared root of the diagonal of `Sigma`, otherwise a provided value is used assuming that `Sigma` and `Gamma` are scaled
`tol`	(numeric) Maximum error between two consecutive solutions of the CD algorithm to declare convergence
`maxiter`	(integer) Maximum number of iterations to run the CD algorithm at each lambda step before convergence is reached
`nsup.max`	(integer) Maximum number of non-zero coefficients in the last solution. Default `nsup.max = NULL` will calculate solutions for the entire lambda grid
`mc.cores`	(integer) Number of cores used. When `mc.cores > 1`, the analysis is run in parallel for each column of `Gamma`. Default is `mc.cores = 1`
`save.at`	(character) Path where regression coefficients are to be saved (this may include a prefix added to the files). Default `save.at = NULL` will no save the regression coefficients and they are returned in the output object

`fileID`	(character) Suffix added to the file name where regression coefficients are to be saved. Default `fileID = NULL` will automatically add sequential integers from 1 to the number of columns of `Gamma`
`precision.format`	(character) Either 'single' or 'double' for numeric precision and memory occupancy (4 or 8 bytes, respectively) of the regression coefficients. This is only used when `save.at` is not `NULL`
`sparse`	`TRUE` or `FALSE` to whether matrix `Sigma` is sparse with entries being zero or near-zero
`eps`	(numeric) A numerical zero to determine if entries are near-zero. Default is the machine precision
`verbose`	`TRUE` or `FALSE` to whether printing progress

Details

Finds solutions for the regression coefficients in a linear model

$$y_i = \mathbf{x}_i'\boldsymbol{\beta} + e_i$$

where y_i is the response for the i^{th} observation, $\mathbf{x}_i = (x_{i1}, ..., x_{ip})'$ is a vector of p predictors assumed to have unit variance, $\boldsymbol{\beta} = (\beta_1, ..., \beta_p)'$ is a vector of regression coefficients, and e_i is a residual.

The regression coefficients $\boldsymbol{\beta}$ are estimated as function of the variance matrix among predictors ($\boldsymbol{\Sigma}$) and the covariance vector between response and predictors ($\boldsymbol{\Gamma}$) by minimizing the penalized mean squared error function

$$-\boldsymbol{\Gamma}'\boldsymbol{\beta} + 1/2\boldsymbol{\beta}'\boldsymbol{\Sigma}\boldsymbol{\beta} + \lambda J(\boldsymbol{\beta})$$

where λ is the penalization parameter and $J(\boldsymbol{\beta})$ is a penalty function given by

$$1/2(1-\alpha)||\boldsymbol{\beta}||_2^2 + \alpha||\boldsymbol{\beta}||_1$$

where $0 \leq \alpha \leq 1$, and $||\boldsymbol{\beta}||_1 = \sum_{j=1}^{p} |\beta_j|$ and $||\boldsymbol{\beta}||_2^2 = \sum_{j=1}^{p} \beta_j^2$ are the L1 and (squared) L2-norms, respectively.

The "partial residual" excluding the contribution of the predictor x_{ij} is

$$e_i^{(j)} = y_i - \mathbf{x}_i'\boldsymbol{\beta} + x_{ij}\beta_j$$

then the ordinary least-squares (OLS) coefficient of x_{ij} on this residual is (up-to a constant)

$$\beta_j^{(ols)} = \Gamma_j - \boldsymbol{\Sigma}_j'\boldsymbol{\beta} + \beta_j$$

where Γ_j is the j^{th} element of $\boldsymbol{\Gamma}$ and $\boldsymbol{\Sigma}_j$ is the j^{th} column of the matrix $\boldsymbol{\Sigma}$.

Coefficients are updated for each $j = 1, ..., p$ from their current value β_j to a new value $\beta_j(\alpha, \lambda)$, given α and λ, by "soft-thresholding" their OLS estimate until convergence as fully described in Friedman (2007).

Value

Returns a list object containing the elements:

- `lambda`: (vector) all the sequence of values of the penalty.
- `beta`: (matrix) regression coefficients for each predictor (in rows) associated to each value of the penalization parameter lambda (in columns).
- `nsup`: (vector) number of non-zero predictors associated to each value of lambda.

The returned object is of the class 'LASSO' for which methods `coef` and `fitted` exist. Function 'path.plot' can be also used

References

Friedman J, Hastie T, Höfling H, Tibshirani R (2007). Pathwise coordinate optimization. *The Annals of Applied Statistics*, **1**(2), 302–332.

Examples

```
require(SFSI)
data(wheatHTP)

y = as.vector(Y[,"E1"])  # Response variable
X = scale(X_E1)          # Predictors

# Training and testing sets
tst = which(Y$trial %in% 1:10)
trn = seq_along(y)[-tst]

# Calculate covariances in training set
XtX = var(X[trn,])
Xty = cov(X[trn,],y[trn])

# Run the penalized regression
fm = solveEN(XtX,Xty,alpha=0.5,nlambda=100)

# Regression coefficients
dim(coef(fm))
dim(coef(fm, ilambda=50)) # Coefficients associated to the 50th lambda
dim(coef(fm, nsup=25))    # Coefficients with around nsup=25 are non-zero

# Predicted values
yHat1 = predict(fm, X=X[trn,])  # training data
yHat2 = predict(fm, X=X[tst,])  # testing data

# Penalization vs correlation
plot(-log(fm$lambda[-1]),cor(y[trn],yHat1[,-1]), main="training", type="l")
plot(-log(fm$lambda[-1]),cor(y[tst],yHat2[,-1]), main="testing", type="l")
```

4. Least Angle Regression (LARS)

Least Angle Regression to solve LASSO-type problems

Description

Computes the entire LASSO solution for the regression coefficients, starting from zero, to the least-squares estimates, via the Least Angle Regression (LARS) algorithm (Efron, 2004). It uses as inputs a variance matrix among predictors and a covariance vector between response and predictors.

Usage

```
LARS(Sigma, Gamma, method = c("LAR","LASSO"),
     nsup.max = NULL, steps.max = NULL,
     eps = .Machine$double.eps*100, scale = TRUE,
     sdx = NULL, mc.cores = 1L, save.at = NULL,
     precision.format = c("double","single"),
     fileID = NULL, verbose = 1)
```

Arguments

`Sigma`	(numeric matrix) Variance-covariance matrix of predictors
`Gamma`	(numeric matrix) Covariance between response variable and predictors. If it contains more than one column, the algorithm is applied to each column separately as different response variables
`method`	(character) Either: • `'LAR'`: Computes the entire sequence of all coefficients. Values of lambdas are calculated at each step. • `'LASSO'`: Similar to `'LAR'` but solutions when a predictor leaves the solution are also returned. Default is `method = 'LAR'`
`nsup.max`	(integer) Maximum number of non-zero coefficients in the last LARS solution. Default `nsup.max = NULL` will calculate solutions for the entire lambda sequence
`steps.max`	(integer) Maximum number of steps (i.e., solutions) to be computed. Default `steps.max = NULL` will calculate solutions for the entire lambda sequence
`eps`	(numeric) A numerical zero. Default is the machine precision
`scale`	`TRUE` or `FALSE` to scale matrix `Sigma` for variables with unit variance and scale `Gamma` by the standard deviation (`sdx`) of the corresponding predictor taken from the diagonal of `Sigma`
`sdx`	(numeric vector) Scaling factor that will be used to scale the regression coefficients. When `scale = TRUE` this scaling factor vector is set to the squared root of the diagonal of `Sigma`, otherwise a provided value is used assuming that `Sigma` and `Gamma` are scaled

`mc.cores` (integer) Number of cores used. When `mc.cores` > 1, the analysis is run in parallel for each column of `Gamma`. Default is `mc.cores` = 1

`save.at` (character) Path where regression coefficients are to be saved (this may include a prefix added to the files). Default `save.at` = `NULL` will no save the regression coefficients and they are returned in the output object

`fileID` (character) Suffix added to the file name where regression coefficients are to be saved. Default `fileID` = `NULL` will automatically add sequential integers from 1 to the number of columns of `Gamma`

`precision.format` (character) Either 'single' or 'double' for numeric precision and memory occupancy (4 or 8 bytes, respectively) of the regression coefficients. This is only used when `save.at` is not `NULL`

`verbose` If numeric greater than zero details on each LARS step will be printed

Details

Finds solutions for the regression coefficients in a linear model

$$y_i = \mathbf{x}_i'\boldsymbol{\beta} + e_i$$

where y_i is the response for the i^{th} observation, $\mathbf{x}_i = (x_{i1}, ..., x_{ip})'$ is a vector of p predictors assumed to have unit variance, $\boldsymbol{\beta} = (\beta_1, ..., \beta_p)'$ is a vector of regression coefficients, and e_i is a residual.

The regression coefficients $\boldsymbol{\beta}$ are estimated as function of the variance matrix among predictors ($\boldsymbol{\Sigma}$) and the covariance vector between response and predictors ($\boldsymbol{\Gamma}$) by minimizing the penalized mean squared error function

$$-\boldsymbol{\Gamma}'\boldsymbol{\beta} + 1/2\boldsymbol{\beta}'\boldsymbol{\Sigma}\boldsymbol{\beta} + 1/2\lambda||\boldsymbol{\beta}||_1$$

where λ is the penalization parameter and $||\boldsymbol{\beta}||_1 = \sum_{j=1}^{p} |\beta_j|$ is the L1-norm.

The algorithm to find solutions for each β_j is fully described in Efron (2004) in which the "current correlation" between the predictor x_{ij} and the residual $e_i = y_i - \mathbf{x}_i'\boldsymbol{\beta}$ is expressed (up-to a constant) as

$$r_j = \Gamma_j - \boldsymbol{\Sigma}_j'\boldsymbol{\beta}$$

where Γ_j is the j^{th} element of $\boldsymbol{\Gamma}$ and $\boldsymbol{\Sigma}_j$ is the j^{th} column of the matrix $\boldsymbol{\Sigma}$

Value

Returns a list object with the following elements:

- `lambda`: (vector) all the sequence of values of the LASSO penalty.
- `beta`: (matrix) regression coefficients for each predictor (in rows) associated to each value of the penalization parameter lambda (in columns).
- `nsup`: (vector) number of non-zero predictors associated to each value of lambda.

The returned object is of the class 'LASSO' for which methods `coef` and `predict` exist. Function 'path.plot' can be also used

Author(s)

Adapted from the 'lars' function in package 'lars' (Hastie & Efron, 2013)

References

Efron B, Hastie T, Johnstone I, Tibshirani R (2004). Least angle regression. *The Annals of Statistics*, **32**(2), 407–499.

Hastie T, Efron B (2013). lars: least angle regression, Lasso and forward stagewise. https://cran.r-project.org/package=lars.

Examples

```
require(SFSI)
data(wheatHTP)

y = as.vector(Y[,"E1"])   # Response variable
X = scale(X_E1)           # Predictors

# Training and testing sets
tst = which(Y$trial %in% 1:10)
trn = seq_along(y)[-tst]

# Calculate covariances in training set
XtX = var(X[trn,])
Xty = cov(X[trn,],y[trn])

# Run the penalized regression
fm = LARS(XtX, Xty, method="LASSO")

# Regression coefficients
dim(coef(fm))
dim(coef(fm, ilambda=50)) # Coefficients associated to the 50th lambda
dim(coef(fm, nsup=25))    # Coefficients with around nsup=25 are non-zero

# Predicted values
yHat1 = predict(fm, X=X[trn,])  # training data
yHat2 = predict(fm, X=X[tst,])  # testing data

# Penalization vs correlation
plot(-log(fm$lambda[-1]),cor(y[trn],yHat1[,-1]), main="Training", type="l")
plot(-log(fm$lambda[-1]),cor(y[tst],yHat2[,-1]), main="Testing", type="l")
```

Description

Computes the entire Elastic-Net solution for the regression coefficients of a Selection Index for a grid of values of the penalization parameter.

An optimal penalization can be chosen using cross-validation (CV) within a specific training set.

Usage

```
SGP(y = NULL, X = NULL, b = NULL, Z = NULL, K = NULL,
    trn = NULL, tst = NULL, varU = NULL, varE = NULL,
    ID_geno = NULL, ID_trait = NULL, intercept = TRUE,
    alpha = 1, lambda = NULL, nlambda = 100,
    lambda.min = .Machine$double.eps^0.5,
    common.lambda = TRUE, subset = NULL, tol = 1E-4,
    maxiter = 500, method = c("REML","ML"), tag = NULL,
    save.at = NULL, precision.format = c("single","double"),
    mc.cores = 1L, verbose = 2)

SGP.CV(y, X = NULL, b = NULL, Z = NULL, K,
       trn = NULL, varU = NULL, varE = NULL,
       ID_geno = NULL, ID_trait = NULL,
       intercept = TRUE, alpha = 1, lambda = NULL,
       nlambda = 100, lambda.min = .Machine$double.eps^0.5,
       common.lambda = TRUE, nfolds = 5, nCV = 1L,
       folds = NULL, seed = NULL, subset = NULL, tol = 1E-4,
       maxiter = 500, method = c("REML","ML"), tag = NULL,
       save.at = NULL, mc.cores = 1L, verbose = TRUE)
```

Arguments

y	(numeric vector) Response variable. It can be a matrix with each column representing a different response variable. If it is passed to the 'SGP' function, predicted values for testing data are computed using phenotypes from training data
X	(numeric matrix) Design matrix for fixed effects. When X = NULL, a vector of ones is constructed only for the intercept (default)
b	(numeric vector) Fixed effects. When b = NULL, only the intercept is estimated from training data using generalized least squares (default)
K	(numeric matrix) Kinship relationship matrix
Z	(numeric matrix) Design matrix for random effects. When Z = NULL an identity matrix is considered (default) thus **G = K**; otherwise **G** $=$ **Z K Z'** is used
varU, varE	(numeric) Genetic and residual variances. When either varU = NULL or varE = NULL (default), they are calculated from training data using the function 'fitBLUP' (see help(fitBLUP)) for single-trait analysis. For multi-trait analysis, unstructured matrices are calculated using the function 'getGenCov' in a pairwise fashion
ID_geno	(character/integer) For multi-trait analysis only, vector with either names or indices mapping entries of the vector y to rows/columns of matrix G

`ID_trait` (character/integer) For multi-trait analysis only, vector with either names or indices mapping entries of the vector `y` to rows/columns of matrices `varU` and `varE`

`intercept` `TRUE` or `FALSE` to whether fit an intercept. When `FALSE`, the model assumes a null intercept

`trn, tst` (integer vector) Which elements from vector `y` are in training and testing sets, respectively. When both `trn = NULL` and `tst = NULL`, non-NA entries in vector `y` will be used as training set

`subset` (integer vector) $c(m, M)$ to fit the model only for the m^{th} subset out of M subsets that the testing set will be divided into. Results can be automatically saved as per the `save.at` argument and can be retrieved later using function 'read_SGP' (see `help(read_SGP)`). In cross-validation, it should has format $c(fold, CV)$ to fit the model for a given fold within partition

`alpha` (numeric) Value between 0 and 1 for the weights given to the L1 and L2-penalties

`lambda` (numeric vector) Penalization parameter sequence. Default is `lambda = NULL`, in this case a decreasing grid of `nlambda` lambdas will be generated starting from a maximum equal to

```
max(abs(G[trn,tst])/alpha)
```

to a minimum equal to zero. If `alpha = 0` the grid is generated starting from a maximum equal to 5

`nlambda` (integer) Number of lambdas generated when `lambda = NULL`

`lambda.min` (numeric) Minimum value of lambda in the generated grid when `lambda = NULL`

`nfolds` (integer/character) Either 2,3,5,10 or 'n' indicating the number of non-overlapping folds in which the data is split into to do cross-validation. When `nfolds = 'n'` a leave-one-out CV is performed

`seed` (numeric vector) Seed to fix randomization when creating folds for cross-validation. If it is a vector, a number equal to its length of CV repetitions are performed

`nCV` (integer) Number of CV repetitions to be performed. Default is `nCV = 1`

`folds` (integer matrix) A matrix with `nTRN` rows and `nCV` columns where each column represents a different partition with `nfolds` folds. It can be created using the function 'get_folds'

`common.lambda` `TRUE` or `FALSE` to whether computing the coefficients for a grid of lambdas common to all individuals in testing set or for a grid of lambdas specific to each individual in testing set. Default is `common.lambda = TRUE`

`mc.cores` (integer) Number of cores used. When `mc.cores > 1`, the analysis is run in parallel for each testing set individual. Default is `mc.cores = 1`

`tol` (numeric) Maximum error between two consecutive solutions of the CD algorithm to declare convergence

`maxiter` (integer) Maximum number of iterations to run the CD algorithm at each lambda step before convergence is reached

`save.at` (character) Path where files (regression coefficients and output object) are to be saved (this may include a prefix added to the files). Default `save.at = NULL` will no save any results and they are returned in the output object. No regression coefficients are saved for function 'SGP.CV'

`precision.format` (character) Either 'single' or 'double' for numeric precision and memory occupancy (4 or 8 bytes, respectively) of the regression coefficients. This is only used when `save.at` is not `NULL`

`method` (character) Either 'REML' (Restricted Maximum Likelihood) or 'ML' (Maximum Likelihood) to calculate variance components as per the function 'fitBLUP'

`tag` (character) Name given to the output for tagging purposes. Default `tag = NULL` will give the name of the method used

`verbose` (integer) If greater than zero analysis details will be printed

Details

The basic linear mixed model that relates phenotypes with genetic values is of the form

$$\mathbf{y} = \mathbf{Xb} + \mathbf{Zg} + \mathbf{e}$$

where **y** is a vector with the response, **b** is the vector of fixed effects, **g** is the vector of the genetic values of the genotypes, **e** is the vector of environmental residuals, and **X** and **Z** are design matrices for the fixed and genetic effects, respectively. Genetic effects are assumed to follow a Normal distribution as $\mathbf{g} \sim N(\mathbf{0}, \sigma_u^2\mathbf{K})$, and environmental terms are assumed $\mathbf{e} \sim N(\mathbf{0}, \sigma_e^2\mathbf{I})$.

The resulting vector of genetic values $\mathbf{u} = \mathbf{Zg}$ will therefore follow $\mathbf{u} \sim N(\mathbf{0}, \sigma_u^2\mathbf{G})$ where $\mathbf{G} = \mathbf{ZKZ}'$. In the un-replicated case, $\mathbf{Z} = \mathbf{I}$ is an identity matrix, and hence $\mathbf{u} = \mathbf{g}$ and $\mathbf{G} = \mathbf{K}$.

The values $\mathbf{u}_{tst} = \{u_i\}$, $i = 1, 2, ..., n_{tst}$, for a testing set are estimated individual-wise using (as predictors) all available observations in a training set as

$$u_i = \boldsymbol{\beta}_i'(\mathbf{y}_{trn} - \mathbf{X}_{trn}\mathbf{b})$$

where $\boldsymbol{\beta}_i$ is a vector of weights that are found separately for each individual in the testing set, by minimizing the penalized mean squared error function

$$-\sigma_u^2\mathbf{G}'_{trn,tst(i)}\boldsymbol{\beta}_i + 1/2\boldsymbol{\beta}_i'(\sigma_u^2\mathbf{G}_{trn} + \sigma_e^2\mathbf{I})\boldsymbol{\beta}_i + \lambda J(\boldsymbol{\beta}_i)$$

where $\mathbf{G}_{trn,tst(i)}$ is the i^{th} column of the sub-matrix of **G** whose rows correspond to the training set and columns to the testing set; $\mathbf{G}_{trn}$ is the sub-matrix corresponding to the training set; λ is the penalization parameter; and $J(\boldsymbol{\beta}_i)$ is a penalty function given by

$$1/2(1-\alpha)||\boldsymbol{\beta}_i||_2^2 + \alpha||\boldsymbol{\beta}_i||_1$$

where $0 \leq \alpha \leq 1$, and $||\boldsymbol{\beta}_i||_1 = \sum_{j=1}^{n_{trn}} |\beta_{ij}|$ and $||\boldsymbol{\beta}_i||_2^2 = \sum_{j=1}^{n_{trn}} \beta_{ij}^2$ are the L1 and (squared) L2-norms, respectively.

Function 'SGP' calculates each individual solution using the function 'solveEN' (via the Coordinate Descent algorithm, see `help(solveEN)`) by setting the argument `Sigma` equal to $\sigma^2_u\mathbf{G}_{trn} + \sigma^2_e\mathbf{I}$ and `Gamma` equal to $\sigma^2_u\mathbf{G}_{trn,tst(i)}$.

Function 'SGP.CV' performs cross-validation within the training data specified in argument `trn`. Training data is divided into k folds and the SGP is sequentially derived for (all individuals in) one fold (as testing set) using information from the remaining folds (as training set).

Value

Function 'SGP' returns a list object of the class 'SGP' for which methods `coef`, `predict`, `plot`, and `summary` exist. Functions 'net.plot' and 'path.plot' can be also used. It contains the elements:

- `b`: (vector) fixed effects solutions (including the intercept).
- `Xb`: (vector) total fixed values (**Xb**).
- `u`: (matrix) total genetic values ($\mathbf{u} = \mathbf{Zg}$) for testing individuals (in rows) associated to each value of lambda (in columns).
- `varU`, `varE`, `h2`: variance components solutions.
- `alpha`: value for the elastic-net weights used.
- `lambda`: (matrix) sequence of values of lambda used (in columns) for each testing individual (in rows).
- `nsup`: (matrix) number of non-zero predictors at each solution given by lambda for each testing individual (in rows).
- `file_beta`: path where regression coefficients are saved.

Function 'SGP.CV' returns a list object of length `nCV` of the class 'SGP'. Optimal cross-validated penalization values can be obtained using the`summary` method. Method `plot` is also available.

Examples

```
require(SFSI)
data(wheatHTP)

index = which(Y$trial %in% 1:12)    # Use only a subset of data
Y = Y[index,]
M = scale(M[index,])/sqrt(ncol(M))  # Subset and scale markers
G = tcrossprod(M)                   # Genomic relationship matrix
Y0 = scale(as.matrix(Y[,4:6]))      # Response variable

#-----------------------------------------------------
# Single-trait model
#-----------------------------------------------------
y = Y0[,1]

# Training and testing sets
tst = which(Y$trial %in% 2:3)
trn = seq_along(y)[-tst]

# Sparse genomic prediction
fm1 = SGP(y, K=G, trn=trn, tst=tst)
```

```
uHat = predict(fm1)        # Predicted values for each testing element
out = summary(fm1)         # Useful function to get results
corTST = out$accuracy      # Testing set accuracy (correlation cor(y,yHat))
out$optCOR                 # SGP with maximum accuracy
B = coef(fm1)              # Regression coefficients for all tst
B = coef(fm1, iy=1)        # Coefficients for first tst (tst[1])
B = coef(fm1, ilambda=10)  # Coefficients associated to the 10th lambda
B = coef(fm1, nsup=10)     # Coefficients for which nsup=10

plot(fm1)                  # Penalization vs accuracy plot
plot(fm1, y.stat="MSE", ylab='Mean Square Error', xlab='Sparsity')

varU = fm1$varU
varE = fm1$varE
b = fm1$b

#-----------------------------------------------------
# Predicting a testing set using a value of lambda
# obtained from cross-validation in a traning set
#-----------------------------------------------------
# Run a cross validation in training set
fm2 = SGP.CV(y, K=G, varU=varU, varE=varE, b=b, trn=trn, nfolds=5, tag="1 5CV")
lambda = summary(fm2)$optCOR["lambda"]

# Fit the index with the obtained lambda
fm3 = SGP(y, K=G, varU=varU, varE=varE, b=b, trn=trn, tst=tst, lambda=lambda)
summary(fm3)$accuracy        # Testing set accuracy

# Compare the accuracy with that of the non-sparse index (G-BLUP)
summary(fm1)$accuracy[fm1$nlambda,1] # we take the last one

#-----------------------------------------------------
# Multi-trait SGP
#-----------------------------------------------------
ID_geno = as.vector(row(Y0))
ID_trait = as.vector(col(Y0))
y = as.vector(Y0)

# Training and testing sets
tst = c(which(ID_trait==1)[Y$trial %in% 2:3],
        which(ID_trait==2)[Y$trial %in% 2],
        which(ID_trait==3)[Y$trial %in% 3])
trn = seq_along(y)[-tst]

fmMT = SGP(y=y, K=G, ID_geno=ID_geno, ID_trait=ID_trait,
           trn=trn, tst=tst)

multitrait.plot(fmMT)
```

6. BLUP estimation from Linear Mixed Model

Fitting a Linear Mixed model to calculate BLUP

Description

Solves the Linear Mixed Model and calculates the Best Linear Unbiased Predictor (BLUP)

Usage

```
fitBLUP(y, X = NULL, Z = NULL, K = NULL, trn = NULL,
        EVD = NULL, varU = NULL, varE = NULL,
        ID_geno = NULL, ID_trait = NULL, intercept = TRUE,
        BLUP = TRUE, method = c("REML","ML"),
        interval = c(1E-9,1E9), tol = 1E-8, maxiter = 1000,
        n.regions = 10, verbose = TRUE)
```

Arguments

y	(numeric vector) Response variable. It can be a matrix with each column representing a different response variable
X	(numeric matrix) Design matrix for fixed effects. When X=NULL a vector of ones is constructed only for the intercept (default)
Z	(numeric matrix) Design matrix for random effects. When Z=NULL an identity matrix is considered (default) thus **G = K**; otherwise **G** = **Z K Z'** is used
K	(numeric matrix) Kinship relationship matrix
trn	(integer vector) Which elements from vector y are to be used for training. When trn = NULL, non-NA entries in vector y will be used as training set
EVD	(list) Eigenvectors and eigenvalues from eigenvalue decomposition (EVD) of **G** corresponding to training data
ID_geno	(character/integer) For within-trait analysis only, vector with either names or indices mapping entries of the vector y to rows/columns of matrix G
ID_trait	(character/integer) For within-trait analysis only, vector with either names or indices mapping entries of the vector y to different traits
varU, varE	(numeric) Genetic and residual variances. When both varU and varE are not NULL they are not calculated; otherwise, the likelihood function (REML or ML) is optimized to search for the genetic/residual variances ratio
intercept	TRUE or FALSE to whether fit an intercept. When FALSE, the model assumes a null intercept
BLUP	TRUE or FALSE to whether return the random effects estimates
method	(character) Either 'REML' (Restricted Maximum Likelihood) or 'ML' (Maximum Likelihood)

`tol`	(numeric) Maximum error between two consecutive solutions (convergence tolerance) when finding the root of the log-likelihood's first derivative
`maxiter`	(integer) Maximum number of iterations to run before convergence is reached
`interval`	(numeric vector) Range of values in which the root is searched
`n.regions`	(numeric) Number of regions in which the searched 'interval' is divided for local optimization
`verbose`	`TRUE` or `FALSE` to whether show progress

Details

The basic linear mixed model that relates phenotypes with genetic values is of the form

$$\mathbf{y} = \mathbf{Xb} + \mathbf{Zg} + \mathbf{e}$$

where **y** is a vector with the response, **b** is the vector of fixed effects, **u** is the vector of the (random) genetic effects of the genotypes, **e** is the vector of environmental residuals (random error), and **X** and **Z** are design matrices for the fixed and genetic effects, respectively. Genetic effects are assumed to follow a Normal distribution as $\mathbf{g} \sim N(\mathbf{0}, \sigma_u^2\mathbf{K})$, and the error terms are assumed $\mathbf{e} \sim N(\mathbf{0}, \sigma_e^2\mathbf{I})$.

The vector of total genetic values $\mathbf{u} = \mathbf{Zg}$ will therefore follow $\mathbf{u} \sim N(\mathbf{0}, \sigma_u^2\mathbf{G})$ where $\mathbf{G} = \mathbf{ZKZ}'$. In the un-replicated case, $\mathbf{Z} = \mathbf{I}$ is an identity matrix, and hence $\mathbf{u} = \mathbf{g}$ and $\mathbf{G} = \mathbf{K}$.

The predicted values $\mathbf{u}_{trn} = \{u_i\}$, $i = 1, 2, ..., n_{trn}$, corresponding to observed data (training set) are derived as

$$\mathbf{u}_{trn} = \mathbf{B}(\mathbf{y}_{trn} - \mathbf{X}_{trn}\mathbf{b})$$

where **B** is a matrix of weights given by

$$\mathbf{B} = \sigma_u^2\mathbf{G}_{trn}(\sigma_u^2\mathbf{G}_{trn} + \sigma_e^2\mathbf{I})^{-1}$$

where $\mathbf{G}_{trn}$ is the sub-matrix corresponding to the training set. This matrix can be rewritten as

$$\mathbf{B} = \mathbf{G}_{trn}(\mathbf{G}_{trn} + \theta\mathbf{I})^{-1}$$

where $\theta = \sigma_e^2/\sigma_u^2$ is the residual/genetic variances ratio representing a ridge-like shrinkage parameter.

The matrix $\mathbf{H} = \mathbf{G}_{trn} + \theta\mathbf{I}$ in the above equation can be used to obtain predictions corresponding to un-observed data (testing set), $\mathbf{u}_{tst} = \{u_i\}$, $i = 1, 2, ..., n_{tst}$, by

$$\mathbf{B} = \mathbf{G}_{tst,trn}(\mathbf{G}_{trn} + \theta\mathbf{I})^{-1}$$

where $\mathbf{G}_{tst,trn}$ is the sub-matrix of **G** corresponding to the testing set (in rows) and training set (in columns).

Solutions are found using the GEMMA (Genome-wide Efficient Mixed Model Analysis) approach (Zhou & Stephens, 2012). First, the Brent's method is implemented to solve for the genetic/residual variances ratio (i.e., $1/\theta$) from the first derivative of the log-likelihood (either REML or ML). Then, variances σ_u^2 and σ_e^2 are calculated. Finally, **b** is obtained using Generalized Least Squares.

Value

Returns a list object that contains the elements:

- b: (vector) fixed effects solutions (including the intercept).
- u: (vector) total genetic values ($\mathbf{u} = \mathbf{Zg}$).
- g: (vector) genetic effects solutions.
- varU: random effect variance.
- varE: residual variance.
- h2: heritability.
- convergence: (logical) whether Brent's method converged.
- method: either 'REML' or 'ML' method used.

References

Zhou X, Stephens M (2012). Genome-wide efficient mixed-model analysis for association studies. *Nature Genetics*, **44**(7), 821-824

Examples

```
require(SFSI)
data(wheatHTP)

index = which(Y$trial %in% 1:20)    # Use only a subset of data
Y = Y[index,]
M = scale(M[index,])/sqrt(ncol(M))  # Subset and scale markers
G = tcrossprod(M)                   # Genomic relationship matrix
Y0 = scale(as.matrix(Y[,4:6]))      # Response variable

#-----------------------------------------------------
# Single-trait model
#-----------------------------------------------------
y = Y0[,1]

# Training and testing sets
tst = which(Y$trial %in% 1:3)
trn = seq_along(y)[-tst]

# Kinship-based model
fm1 = fitBLUP(y, K=G, trn=trn)

head(fm1$g)                  # Genetic effects
plot(y[tst],fm1$yHat[tst])   # Predicted vs observed values in testing set
cor(y[tst],fm1$yHat[tst])    # Prediction accuracy in testing set
cor(y[trn],fm1$yHat[trn])    # Prediction accuracy in training set
fm1$varU                     # Genetic variance
fm1$varE                     # Residual variance
fm1$h2                       # Heritability
fm1$b                        # Fixed effects
```

```
# Markers-based model
fm2 = fitBLUP(y, Z=M, trn=trn)
head(fm2$g)                   # Marker effects
all.equal(fm1$yHat, fm2$yHat)
fm2$varU                      # Genetic variance
fm2$varU*sum(apply(M,2,var))

#-----------------------------------------------
# Multiple response variables
#-----------------------------------------------
ID_geno = as.vector(row(Y0))
ID_trait = as.vector(col(Y0))
y = as.vector(Y0)

# Training and testing sets
tst = c(which(ID_trait==1)[Y$trial %in% 1:3],
        which(ID_trait==2)[Y$trial %in% 1:3],
        which(ID_trait==3)[Y$trial %in% 1:3])
trn = seq_along(y)[-tst]

# Across traits model
fm3 = fitBLUP(y, K=G, ID_geno=ID_geno, trn=trn)
plot(fm1$yHat,fm3$yHat[ID_trait==1])  # different from the single-trait model

# Within traits model: pass an index for traits
fm4 = fitBLUP(y, K=G, ID_geno=ID_geno, ID_trait=ID_trait, trn=trn)
plot(fm1$yHat,fm4$yHat[ID_trait==1])  # equal to the single-trait model
```

---

7. Genetic covariances computation

Pairwise Genetic Covariance

Description

Calculation of genetic and environmental (unstructured) covariances matrices from pairwise models of variables with the same experimental design

Usage

```
getGenCov(y, X = NULL, Z = NULL, K = NULL, trn = NULL,
          EVD = NULL, ID_geno, ID_trait, scale = TRUE,
          pairwise = TRUE, verbose = TRUE, ...)
```

Arguments

y	(numeric matrix) Response variable vector. It can be a matrix with each column representing a different response variable

X	(numeric matrix) Design matrix for fixed effects. When X=NULL a vector of ones is constructed only for the intercept (default)
Z	(numeric matrix) Design matrix for random effects. When Z=NULL an identity matrix is considered (default) thus **G = K**; otherwise **G** $=$ **Z K Z'** is used
K	(numeric matrix) Kinship relationship matrix
trn	(integer vector) Which elements from vector y are to be used for training. When trn = NULL, non-NA entries in vector y will be used as training set
EVD	(list) Eigenvectors and eigenvalues from eigenvalue decomposition (EVD) of **G** corresponding to training data
ID_geno	(character/integer) Vector with either names or indices mapping entries of the vector y to rows/columns of matrix G
ID_trait	(character/integer) Vector with either names or indices mapping entries of the vector y to different traits
scale	TRUE or FALSE to scale each column of y by their corresponding standard deviations so the resulting variables will have unit variance
pairwise	TRUE or FALSE to calculate pairwise genetic covariances for all columns in y. When pairwise = FALSE (default) covariances of the first column in y with the remaining columns (2,...,ncol(y)) are calculated
verbose	TRUE or FALSE to whether show progress
...	Other arguments passed to the function 'fitBLUP'

Details

Assumes that both $\mathbf{y}_1$ and $\mathbf{y}_2$ follow the basic linear mixed model that relates phenotypes with genetic values of the form

$$\mathbf{y}_1 = \mathbf{X}\mathbf{b}_1 + \mathbf{Z}\mathbf{g}_1 + \mathbf{e}_1$$

$$\mathbf{y}_2 = \mathbf{X}\mathbf{b}_2 + \mathbf{Z}\mathbf{g}_2 + \mathbf{e}_2$$

where $\mathbf{b}_1$ and $\mathbf{b}_2$ are the specific fixed effects, $\mathbf{g}_1$ and $\mathbf{g}_2$ are the specific genetic effects of the genotypes, $\mathbf{e}_1$ and $\mathbf{e}_2$ are the vectors of specific environmental residuals, and $\mathbf{X}$ and $\mathbf{Z}$ are common design matrices for the fixed and genetic effects, respectively. Genetic effects are assumed to follow a Normal distribution as $\mathbf{g}_1 \sim N(\mathbf{0}, \sigma^2_{u_1}\mathbf{K})$ and $\mathbf{g}_2 \sim N(\mathbf{0}, \sigma^2_{u_2}\mathbf{K})$, and environmental terms are assumed $\mathbf{e}_1 \sim N(\mathbf{0}, \sigma^2_{e_1}\mathbf{I})$ and $\mathbf{e}_2 \sim N(\mathbf{0}, \sigma^2_{e_2}\mathbf{I})$.

The genetic covariance $\sigma^2_{u_1u_2}$ is estimated from the formula for the variance for the sum of two variables as

$$\sigma^2_{u_1u_2} = \frac{1}{2}(\sigma^2_{u_3} - \sigma^2_{u_1} - \sigma^2_{u_2})$$

where $\sigma^2_{u_3}$ is the genetic variance of the variable $\mathbf{y}_3 = \mathbf{y}_1 + \mathbf{y}_2$ that also follows the same model as for $\mathbf{y}_1$ and $\mathbf{y}_2$.

Likewise, the environmental covariance $\sigma^2_{e_1e_2}$ is estimated as

$$\sigma^2_{e_1e_2} = \frac{1}{2}(\sigma^2_{e_3} - \sigma^2_{e_1} - \sigma^2_{e_2})$$

where $\sigma^2_{e_3}$ is the error variance of the variable $\mathbf{y}_3$.

Solutions are found using the function 'fitBLUP' (see `help(fitBLUP)`) to sequentialy fit mixed models for all the variables $\mathbf{y}_1$, $\mathbf{y}_2$ and $\mathbf{y}_3$.

Value

Returns a list object that contains the elements:

- `varU`: (vector) genetic variances.
- `varE`: (vector) error variances.
- `covU`: (vector) genetic covariances between response variable 1 and the rest.
- `covE`: (vector) environmental covariances between response variable 1 and the rest.

When `pairwise = TRUE`, `varU` and `varE` are matrices containing all variances (diagonal) and pairwise covariances (off diagonal)

Examples

```
require(SFSI)
data(wheatHTP)

index = which(Y$trial %in% 1:10)    # Use only a subset of data
Y = Y[index,]
M = scale(M[index,])/sqrt(ncol(M))  # Subset and scale markers
G = tcrossprod(M)                   # Genomic relationship matrix
Y0 = scale(as.matrix(Y[,4:7]))      # Response variable

ID_geno = as.vector(row(Y0))
ID_trait = as.vector(col(Y0))
y = as.vector(Y0)

# Pairwise covariance for all columns in y
fm = getGenCov(y=y, K=G, ID_geno=ID_geno, ID_trait=ID_trait)
fm$varU           # Genetic variances
fm$varE           # Residual variances
fm$varU+fm$varE   # Phenotypic covariance
var(Y0)           # Sample phenotypic covariance

# Covariances between the first and the rest: y[,1] and y[,2:4]
fm = getGenCov(y=y, K=G, ID_geno=ID_geno, ID_trait=ID_trait, pairwise=FALSE)
fm$covU           # Genetic covariance between y[,1] and y[,2:4]

# Containing some missing values:
# The model is estimated from common training data
YNA = Y0
YNA[Y$trial %in% 1:2, 1] = NA
YNA[Y$trial %in% 2:3, 3] = NA
y = as.vector(YNA)
```

```
fm = getGenCov(y=y, K=G, ID_geno=ID_geno, ID_trait=ID_trait)
fm$varU
fm$varE
```

---

8. SGP accuracy vs penalization plot

Accuracy vs penalization plot

Description

Accuracy as a function of the penalization plot for an object of the class 'SGP'

Usage

```
## S3 method for class 'SGP'
plot(..., x.stat = c("nsup","lambda"),
          y.stat = c("accuracy","MSE"),
          label = x.stat, nbreaks.x = 6)
```

Arguments

`...`	Other arguments to be passed: • One or more objects of the class 'SGP' • Optional arguments for method `plot`: 'xlab', 'ylab', 'main', 'lwd', 'xlim', 'ylim' • For multi-trait SGP, optional argument 'trait' to plot results for a specific trait
`x.stat`	(character) Either 'nsup' (number of non-zero regression coefficients entering in the prediction of a given testing individual) or 'lambda' (penalization parameter in log scale) to plot in the x-axis
`y.stat`	(character) Either 'accuracy' (correlation between observed and predicted values) or 'MSE' (mean squared error) to plot in the y-axis
`label`	(character) Similar to `x.stat` but to show the value in x-axis for which the y-axis is maximum
`nbreaks.x`	(integer) Number of breaks in the x-axis

Value

Creates a plot of either accuracy or MSE versus either the support set size (average number of predictors with non-zero regression coefficient) or versus lambda.

Examples

```
# See examples in
# help(SGP, package="SFSI")
```

Create partitions *Data partition into folds of the same size*

Description

Create a random data partition of size n into k non-overlapping folds of approximately the same size

Usage

```
get_folds(n, k = 5L, nCV = 1L, seed = NULL)
```

Arguments

n	(integer) Sample size
k	(integer) Number of folds
nCV	(integer) Number of different partitions to be created
seed	(integer vector) Optional seed for randomization (see help(set.seed)). It has to be of length equal to nCV

Value

Returns a matrix with n rows and nCV columns. Each column contains a partition with k folds.

Examples

```
require(SFSI)

# Create 5 different partitions into 10 folds
# for a sample size equal to 115
out = get_folds(n=115, k=10, nCV=5)

# Size of folds at first partition
table(out[,1])
```

Graphical Network from a numeric matrix *Graphical Network*

Description

Obtain a Graphical Network representation from a matrix where nodes are subjects in the rows and columns, and edges are obtained from the matrix entries

Usage

```
net(object, K = NULL,
    nsup = NULL, p.radius = 1.7,
    delta = .Machine$double.eps)
```

Arguments

`object`	Either a numeric matrix `X` or an object of the 'SGP' class. When the object is of the 'SGP' class the regression coefficients are used as `X`
`K`	(numeric matrix) Kinship relationship matrix among nodes
`nsup`	(numeric) For a SGP, average number of training individuals contributing to the prediction (with non-zero regression coefficient) of testing individuals. Default `nsup = NULL` will use the value of `nsup` that yielded the optimal accuracy
`p.radius`	(numeric) For a multi-trait SGP, a factor (x-folds radius) to separate each trait from the origin
`delta`	(numeric) Minumum value to determine nodes to be connected. Default is the machine precision (numerical zero)

Details

For a numeric matrix $\mathbf{X} = \{x_{ij}\}$ with `m` rows and `n` columns, a graphical network with `m + n` nodes is obtained by defining edges connecting subjects in rows with those in columns. An edge between subject in row `i` and subject in column `j` is determined if the corresponding (absolute) entry matrix is larger than certain value, i.e., $|x_{ij}| > \delta$.

For a symmetric matrix, only `m=n` nodes are considered with edges determined by the above diagonal entries of the matrix.

Nodes and edges are plotted in the cartesian plane according to the Fruchterman-Reingold algorithm. When a matrix **K** is provided, nodes are plotted according to the top 2 eigenvectors from the spectral value decomposition of $\mathbf{K} = \mathbf{U}\,\mathbf{D}\,\mathbf{U'}$.

When the object is a 'SGP' class object the edges are taken from the regression coefficients (associated to a specific `nsup` value) are used as matrix **X** with testing subjects in rows and training subjects in columns.

Value

Returns a plottable object of the class 'net' that can be visualized using 'plot' method

Examples

```
require(SFSI)
data(wheatHTP)

#---------------------------------------
# Net for an SGP object
#---------------------------------------
index = which(Y$trial %in% 1:6)     # Use only a subset of data
Y = Y[index,]
```

```
M = scale(M[index,])/sqrt(ncol(M))  # Subset and scale markers
G = tcrossprod(M)                   # Genomic relationship matrix
y = as.vector(scale(Y[,"E1"]))      # Scale response variable

# Training and testing sets
tst = which(Y$trial %in% 2)
trn = seq_along(y)[-tst]

fm = SGP(y, K=G, trn=trn, tst=tst)

nt = net(fm)          # Get the net
plot(nt)              # Plot the net
plot(nt, i=c(1,5))    # Show the first and fifth tst elements
plot(net(fm, nsup=10), show.names=c(TRUE,TRUE,FALSE))

#--------------------------------------
# Net for a numeric matrix
#--------------------------------------
B = as.matrix(coef(fm, nsup=10))
plot(net(B), curve=TRUE, set.size=c(3.5,1.5,1))

#--------------------------------------
# Net for a symmetric numeric matrix
#--------------------------------------
X = X_E1[,seq(1,ncol(X_E1),by=5)]
R2 = cor(X)^2  # An R2 matrix
plot(net(R2, delta=0.9))
```

Graphical network plot

Plotting a network

Description

Plot a Graphical Network obtained from a numeric matrix

Usage

```
## S3 method for class 'net'
plot(x, i = NULL, show.names = FALSE,
        group = NULL, group.shape = NULL,
        set.color = NULL, set.size = NULL,
        axis.labels = TRUE, curve = FALSE,
        bg.color = "white", unified = TRUE, ni = 36,
        line.color = "gray70", line.width = 0.3,
        legend.pos = "right", point.color = "gray20",
        sets = c("Testing","Supporting","Non-active"),
        circle = FALSE, ...)
```

Arguments

`x`	An object of the 'net' class as per the `net` function
`i`	(integer vector) Index subjects in rows to be shown in plot. Default `i=NULL` will consider all elements in rows
`show.names`	`TRUE` or `FALSE` to whether show node names given by the row/column names of the matrix used to make the net (see `help(net)`)
`group`	(data.frame) Column grouping for the subjects
`group.shape`	(integer vector) Shape of each level of the grouping column provided as `group`
`bg.color`	(character) Plot background color
`line.color`	(character) Color of lines connecting nodes in rows with those in columns
`line.width`	(numeric) Width of lines connecting nodes in rows with those in columns
`curve`	`TRUE` or `FALSE` to whether draw curve lines connecting nodes in rows with those in columns
`set.color`	(character vector) Color point of each type of node: row, 'active' column, and 'non-active' column, respectively
`set.size`	(numeric vector) Size of each type of node: row, 'active' column, and 'non-active' column, respectively
`axis.labels`	`TRUE` or `FALSE` to whether show labels in both axes
`unified`	`TRUE` or `FALSE` to whether show an unified plot or separated for each individual in 'testing'
`point.color`	(character) Color of the points in the plot
`ni`	(integer) Maximum number of row nodes that are plotted separated as indicated by `unified=FALSE`
`legend.pos`	(character) Either "right", topright","bottomleft","bottomright","topleft", or "none" indicating where the legend is positioned in the plot
`sets`	(character vector) Names of the types of node: row, 'active' column, and 'non-active' column, respectively
`circle`	`TRUE` or `FALSE` to whether draw a circle for each trait in a multi-trait 'SGP'
`...`	Other arguments for method `plot`: 'xlab', 'ylab', 'main'

Details

Plot a Graphical Network from a matrix where nodes are subjects in the rows and columns, and edges are obtained from the matrix entries. This Network is obtained using `net` function

Examples

```
# See examples in
# help(net, package="SFSI")
```

`Multi-trait SGP accuracy vs penalization plot`
Accuracy vs penalization from multi-trait SGP

Description

Visualizing results from an object of the class 'SGP'

Usage

```
multitrait.plot(object, trait_names = NULL,
                x.stat = c("nsup","lambda"),
                y.stat = c("accuracy","MSE"), label = x.stat,
                line.color = "orange", point.color = line.color,
                point.size = 1.2, nbreaks.x = 6, ...)
```

Arguments

`object`	An object of the class 'SGP' for a multi-trait case
`x.stat`	(character) Either 'nsup' (number of non-zero regression coefficients entering in the prediction of a given testing individual) or 'lambda' (penalization parameter in log scale) to plot in the x-axis
`y.stat`	(character) Either 'accuracy' (correlation between observed and predicted values) or 'MSE' (mean squared error) to plot in the y-axis
`label`	(character) Similar to `x.stat` but to show the value in x-axis for which the y-axis is maximum across traits
`point.color`, `line.color`	(character) Color of the points and lines
`point.size`	(numeric) Size of the points showing the maximum accuracy
`nbreaks.x`	(integer) Number of breaks in the x-axis
`trait_names`	(character) Names of traits to be shown in the plot
`...`	Other arguments for method `plot`: 'xlab', 'ylab', 'main', 'lwd', 'xlim', 'ylim'

Value

Creates a plot of either accuracy or MSE versus either the support set size (average number of predictors with non-zero regression coefficient) or versus lambda. This is done separately for each trait

Examples

```
# See examples in
# help(SGP, package="SFSI")
```

R-squared pruning *R-squared pruning*

Description

Pruning features using an R-squared threshold and maximum distance

Usage

```
Prune(X, alpha = 0.95,
      pos = NULL, d.max = NULL,
      centered = FALSE, scaled = FALSE,
      verbose = FALSE)
```

Arguments

X	(numeric matrix) A matrix with observations in rows and features (e.g., SNPs) in columns
alpha	(numeric) R-squared threshold used to determine connected sets
pos	(numeric vector) Optional vector with positions (e.g., bp) of features
d.max	(numeric) Maximum distance that connected sets are apart
centered	TRUE or FALSE whether columns in X are centered with mean zero
scaled	TRUE or FALSE whether columns in X are scaled with unit standard deviation
verbose	TRUE or FALSE to whether show progress

Details

The algorithm identifies sets of connected features as those that share an $R^2 > \alpha$ and retains only one feature (first appearance) for each set.

The sets can be limited to lie within a distance less or equal to a d.max value.

Value

Returns a list object that contains the elements:

- prune.in: (vector) indices of selected (unconnected) features.
- prune.out: (vector) indices of dropped out features.

Examples

```
require(SFSI)
data(wheatHTP)

index = c(154:156,201:205,306:312,381:387,540:544)
X = M[,index]          # Subset markers
colnames(X) = 1:ncol(X)
```

```
# See connected sets using R^2=0.8
R2thr = 0.8
R2 = cor(X)^2
nw1 = net(R2, delta=R2thr)
plot(nw1, show.names=TRUE)

# Get pruned features
res = Prune(X, alpha=R2thr)

# See selected (unconnected) features
nw2 = net(R2[res$prune.in,res$prune.in], delta=R2thr)
nw2$xy = nw1$xy[res$prune.in,]
plot(nw2, show.names=TRUE)
```

`Reading and combining SGP outputs`

Read and combine SGP outputs

Description

Read the output generated by the 'SGP' or 'SGP.CV' functions saved at the provided `save.at` parameter. It merges all partial files when data was splited according to argument `subset`. Function 'read_summary' reads the output generated after calling the method `summary`; multiple outputs are read if argument `path` is a vector and in this case, they are combined by averaging across all outputs.

Usage

```
read_SGP(path = ".", type = NULL,
         file.ext = ".RData", verbose = TRUE)

read_summary(path = ".", type = NULL,
             file.ext = ".RData", verbose = TRUE)
```

Arguments

`path`	(character) Path where output files were saved. This may include a prefix
`type`	(character) Either 'SGP' or 'SGP.CV' to collect results generated by either function of the same name
`file.ext`	(character) This is the file extension at which file was saved as per the argument `save.at`
`verbose`	`TRUE` or `FALSE` to whether printing details

Value

An object of the class 'SGP' for which methods `predict`, `plot` and `summary` exist

Examples

```
require(SFSI)
data(wheatHTP)

index = which(Y$trial %in% 1:10)    # Use only a subset of data
Y = Y[index,]
M = scale(M[index,])/sqrt(ncol(M))  # Subset and scale markers
G = tcrossprod(M)                   # Genomic relationship matrix
y = as.vector(scale(Y[,"E1"]))      # Scale response variable

# Training and testing sets
tst = which(Y$trial %in% 1:3)
trn = seq_along(y)[-tst]

path = paste0(tempdir(),"/testSGP_")

# Run the analysis into 4 subsets and save them at a given path
SGP(y, K=G, trn=trn, tst=tst, subset=c(1,4), save.at=path)
SGP(y, K=G, trn=trn, tst=tst, subset=c(2,4), save.at=path)
SGP(y, K=G, trn=trn, tst=tst, subset=c(3,4), save.at=path)
SGP(y, K=G, trn=trn, tst=tst, subset=c(4,4), save.at=path)

# Collect all results after completion
fm = read_SGP(path)

# Generate and save summary
summary(fm, save.at=path)
fm2 = read_summary(path)
```

Regression coefficients and predicted values in LASSO-type problems

LASSO methods

Description

Retrieving regression coefficients and predicted values from the 'solveEN' and 'LARS' functions' outputs

Usage

```
## S3 method for class 'LASSO'
coef(object, ...)

## S3 method for class 'LASSO'
predict(object, ...)
```

Arguments

`object`	An object of the class 'LASSO' returned either by the function 'LARS' or 'solveEN'
`...`	Other arguments: • `X` (numeric matrix) scores for as many predictors there are in `ncol(object$beta)` (in columns) for a desired number `n` of observations (in rows) • `iy` (integer vector) Optional index of columns of the matrix 'Gamma' to be returned in `coef` function • `ilambda` (integer) Optional to return regression coefficients associated to a specific penalty position • `nsup` (numeric) Optional to return regression coefficients associated to a given penalty that yield approximately 'nsup' non-zero coefficients

Value

Method `coef` returns a matrix that contains the regression coefficients (in rows) associated to each value of lambda (in columns). When the regression was applied to an object `Gamma` with more than one column, method `coef` returns a list

Method `predict` returns a matrix with predicted values $\mathbf{X}\boldsymbol{\beta}$ (in rows) for each value of lambda (in columns).

Examples

```
# See examples in
# help(solveEN, package="SFSI")
# help(LARS, package="SFSI")
```

Regression coefficients path

Coefficients path plot

Description

Coefficients evolution path plot from an object of the class 'LASSO' or 'SGP'

Usage

```
path.plot(object, K = NULL, i = NULL,
          prune = FALSE, cor.max = 0.97,
          lambda.min = .Machine$double.eps^0.5,
          nbreaks.x = 6, npaths.max = 5000, ...)
```

Arguments

`object`	An object of the 'LASSO' or 'SGP' class
`K`	(numeric matrix) Kinship relationships. Only needed for an object of the class 'SGP'
`i`	(integer vector) Index a response variable (columns of matrix `Gamma`) for an object of the class 'LASSO'. Index testing elements (stored in `object$tst`) for an object of the class 'SGP'. Default `i = NULL` will consider either all columns in matrix `Gamma` or all elements in `object$tst`, respectively
`prune`	`TRUE` or `FALSE` to whether prune within groups of correlated coefficients, keeping only one per group. A group of coefficients that are highly correlated are likely to overlap in the plot
`cor.max`	(numeric) Correlation threshold to prune within groups of correlated coefficients
`lambda.min`	(numeric) Minimum value of lambda to show in the plot as `-log(lambda)`
`nbreaks.x`	(integer) Number of breaks in the x-axis
`npaths.max`	(integer) Maximum number of paths defined by the number of predictors times the number of columns of matrix `Gamma` for an object of the class 'LASSO'. This correspond to the number of training elements (stored in `object$trn`) times the number of testing elements (stored in `object$tst`) for an object of the class 'SGP'
`...`	Other arguments for method `plot`: 'xlab', 'ylab', 'main', 'lwd'

Value

Returns the plot of the coefficients' evolution path along the regularization parameter

Examples

```
require(SFSI)
data(wheatHTP)

index = which(Y$trial %in% 1:6)       # Use only a subset of data
Y = Y[index,]
X = scale(X_E1[index,])               # Reflectance data
M = scale(M[index,])/sqrt(ncol(M))    # Subset and scale markers
G = tcrossprod(M)                     # Genomic relationship matrix
y = as.vector(scale(Y[,'E1']))        # Subset response variable

# Sparse phenotypic regression
fm = LARS(var(X),cov(X,y))

path.plot(fm)

# Sparse Genomic Prediction
fm = SGP(y, K=G, trn=12:length(y), tst=1:11)

path.plot(fm, prune=TRUE)
path.plot(fm, K=G, prune=TRUE, cor.max=0.9)
```

```
# Path plot for the first individual in testing set for the SGP
path.plot(fm, K=G, i=1)
```

Regression coefficients, predicted values and summary in SGP problems

SGP methods

Description

Useful methods for retrieving and summarizing important results from the 'SGP' function's output

Usage

```
## S3 method for class 'SGP'
coef(object, ...)

## S3 method for class 'SGP'
predict(object, ...)

## S3 method for class 'SGP'
summary(object, ...)
```

Arguments

`object`	An object of the class 'SGP'
`...`	Other arguments to be passed to `coef` method: • `nsup`: (numeric) Average (across testing individuals) number of non-zero regression coefficients. Only the coefficients for the lambda associated to `nsup` are returned as a 'matrix' with testing individuals in rows • `iy`: (integer vector) Index testing elements (stored in `object$tst`) to be considered. Only coefficients corresponding to the testing individuals `object$tst[iy]` are returned For `predict` and `summary` methods: • `y`: (numeric vector) An optional response vector

Value

Method `predict` returns a matrix with the predicted values for each individual in the testing set (in rows) for each value of lambda (in columns).

Method `coef` (list of matrices) returns the regression coefficients for each testing set individual (elements of the list). Each matrix contains the coefficients for each value of lambda (in rows) associated to each training set individual (in columns).

Method `summary` returns a list object containing:

- `lambda`: (vector) sequence of values of lambda used in the coefficients' estimation.
- `nsup`: (vector) Number of non-zero coefficients (across testing individuals) at each solution associated to each value of lambda.
- `accuracy`: (vector) correlation between observed and predicted values associated to each value of lambda.
- `MSE`: (vector) mean squared error associated to each value of lambda.
- `optCOR`: (vector) summary of the optimal SGP with maximum accuracy.
- `optMSE`: (vector) summary of the optimal SGP with minimum MSE.

Examples

```
# See examples in
# help(SGP, package="SFSI")
```

`Wheat data set` *Wheat dataset*

Description

The dataset consists of 1,092 inbred wheat lines grouped into 39 trials and grown during the 2013-2014 season at the Norman Borlaug experimental research station in Ciudad Obregon, Sonora, Mexico. Each trial consisted of 28 breeding lines that were arranged in an alpha-lattice design with three replicates and six sub-blocks. The trials were grown in four different environments:

- E1: Flat-Drought (sowing in flat with irrigation of 180 mm through drip system)
- E2: Bed-2IR (sowing in bed with 2 irrigations approximately 250 mm)
- E3: Bed-5IR (bed sowing with 5 normal irrigations)
- E4: Bed-EHeat (bed sowing 30 days before optimal planting date with 5 normal irrigations approximately 500 mm)

1. Phenotypic data. Measurements of grain yield (YLD) were reported as the total plot yield after maturity. Records for YLD are reported as adjusted means from which trial, replicate and sub-block effects were removed. Measurements for days to heading (DTH), days to maturity (DTM), and plant height (PH) were recorded only in the first replicate at each trial and thus no phenotype adjustment was made.

2. Reflectance data. Reflectance data was collected from the fields using both infrared and hyper-spectral cameras mounted on an aircraft on 9 different dates (time-points) between January 10 and March 27th, 2014. During each flight, data from 250 wavelengths ranging from 392 to 850 nm were collected for each pixel in the pictures. The average reflectance of all the pixels for each wavelength was calculated from each of the geo-referenced trial plots and reported as each line reflectance. Data for reflectance and Green NDVI and Red NDVI are reported as adjusted phenotypes from which trial, replicate and sub-block effects were removed. Each data-point matches to each data-point in phenotypic data.

3. Marker data. Lines were sequenced for GBS at 192-plexing on Illumina HiSeq2000 or HiSeq2500 with 1 x 100 bp reads. SNPs were called across all lines anchored to the genome assembly of

Chinese Spring (International Wheat Genome Sequencing Consortium 2014). Next, SNP were extracted and filtered so that lines >50% missing data were removed. Markers were recoded as –1, 0, and 1, corresponding to homozygous for the minor allele, heterozygous, and homozygous for the major allele, respectively. Next, markers with a minor allele frequency <0.05 and >15% of missing data were removed. Remaining SNPs with missing values were imputed using the mean of the observed marker genotypes at a given locus.

Adjusted un-replicated data. The `SFSI` R-package includes the `wheatHTP` dataset containing (un-replicated) only YLD from all environments E1,...,E4, and reflectance (latest time-point only) data from the environment E1 only. Marker data is also included in the dataset. The phenotypic and reflectance data are averages (line effects from mixed models) for 776 lines evaluated in 28 trials (with at least 26 lines each) for which marker information on 3,438 SNPs is available.

The full (replicated) data for all four environments, all traits, and all time-points can be found in the repository https://github.com/MarcooLopez/Data_for_Lopez-Cruz_et_al_2020.

Cross-validation partitions. One random partition of 4-folds was created for the 776 individuals (distributed into 28 trials). Data from 7 entire trials (25% of 28 the trials) were arbitrarily assigned to each of the 4 folds. The partition consist of an array of length 776 with indices 1, 2, 3, and 4 denoting the fold.

Genetic covariances. Multi-variate Gaussian mixed models were fitted to phenotypes. Bi-variate models were fitted to YLD with each of the 250 wavelengths from environment E1. Tetra-variate models were fitted for YLD from all environments. All models were fitted within each fold (provided partition) using scaled (null mean and unit variance) phenotypes from the remaining 3 folds as training data. Bayesian models were implemented using the 'Multitrait' function from the `BGLR` R-package with 40,000 iterations discarding 5,000 runs for burning. A marker-derived relationships matrix as in VanRaden (2008) was used to model between-individuals genetic effects. Between-traits genetic covariances were assumed unstructured, while residual covariances were assumed diagonal.

Genetic covariances between YLD and each wavelength (environment E1) are storaged in a matrix of 250 rows and 4 columns (folds). Genetic and residual covariances matrices among YLD within each environment are storaged in a list with 4 elements (folds).

Usage

```
data(wheatHTP)
```

Format

- `Y`: (matrix) phenotypic data for YLD in environments E1, E2, E3, and E4; and columns 'trial' and 'CV' (indicating the 4-folds partition).
- `M`: (matrix) marker data with SNPs in columns.
- `X_E1`: (matrix) reflectance data for time-point 9 in environment E1.
- `VI_E1`: (matrix) green and red NDVI for time-point 9 in environment E1.
- `genCOV_xy`: (matrix) genetic covariances between YLD and each reflectance trait, for each fold (in columns).
- `genCOV_yy`: (4-dimensional list) genetic covariances matrices for YLD among environments, for each fold.
- `resCOV_yy`: (4-dimensional list) residual covariances matrices for YLD among environments, for each fold.

Source

International Maize and Wheat Improvement Center (CIMMYT), Mexico.

References

Perez-Rodriguez P, de los Campos G (2014). Genome-wide regression and prediction with the BGLR statistical package. *Genetics*, **198**, 483–495.

VanRaden PM (2008). Efficient methods to compute genomic predictions. *Journal of Dairy Science*, **91**(11), 4414–4423.

Index