

Package ‘SFSI’

November 18, 2023

Title Sparse Family and Selection Index

Version 1.3.1

Date 2023-11-17

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](https://doi.org/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](https://doi.org/10.1093/genetics/iyab030)>.

LazyLoad true

Depends R (>= 3.6.0), ggplot2

Imports stats, scales, tensorEVD, parallel, reshape2, viridis, igraph

Suggests BGLR, knitr, rmarkdown

VignetteBuilder rmarkdown

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 2023-11-18 07:00:03 UTC

R topics documented:

| | |
|--|---|
| 1. Save/read binary files | 2 |
| 3. Covariance matrix transformations | 4 |

| | |
|--|----|
| 4. Solve an Elastic-Net problem | 5 |
| 5. Least Angle Regression (LARS) | 9 |
| 6. Sparse Selection Index (SSI) | 11 |
| 7. BLUP estimation from Linear Mixed Model | 16 |
| 8. Computing Genetic Covariances | 19 |
| 9. SSI accuracy vs penalization plot | 21 |
| Collecting SSI outputs | 22 |
| Graphical Network from a numeric matrix | 23 |
| Graphical network plot | 25 |
| Multitrait SSI accuracy vs penalization plot | 27 |
| R-squared pruning | 28 |
| Regression coefficients and fitted values in LASSO-type problems | 30 |
| Regression coefficients path | 31 |
| Summarize SSI results | 33 |
| Wheat data set | 34 |

| | |
|--------------|-----------|
| Index | 37 |
|--------------|-----------|

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)
data(wheatHTP)

# Read a numeric matrix
X = scale(X_E1)
X[1:10,1:5]

# 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)
X2[1:10,1:5]
max(abs(X-X2))          # No loss of precision
file.info(filename1)$size # Size of the file

# Read the single-precision matrix
X3 = readBinary(filename2)
X3[1:10,1:5]
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)
X2
# Equal to:
X[rows,cols]
```

3. 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 $\mathbf{x}_i = (x_{i1}, \dots, x_{in})'$ and $\mathbf{x}_j = (x_{j1}, \dots, x_{jn})'$, 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)
D[1:5,1:5]
# (it must equal (but faster) to:)
D0 = as.matrix(dist(X))
max(D-D0)

# Covariance to a correlation matrix
R = cov2cor2(A)
R[1:5,1:5]
# (it must equal (but faster) to:)
R0 = cov2cor(A)
max(R-R0)

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

4. 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,
```

```
precision.format = c("double","single"),
fileID = NULL, verbose = FALSE)
```

Arguments

| | |
|-------------------------------------|---|
| <code>Sigma</code> | (numeric matrix) Variance-covariance matrix of predictors |
| <code>Gamma</code> | (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 |
| <code>lambda</code> | (numeric vector) Penalization parameter sequence. Default is <code>lambda=NULL</code> , in this case a decreasing grid of ' <code>nlambda</code> ' lambdas will be generated starting from a maximum equal to $\max(\text{abs}(\text{Gamma})/\alpha)$ to a minimum equal to zero. If <code>alpha=0</code> the grid is generated starting from a maximum equal to 5 |
| <code>nlambda</code> | (integer) Number of lambdas generated when <code>lambda=NULL</code> |
| <code>lambda.min, lambda.max</code> | (numeric) Minimum and maximum value of lambda that are generated when <code>lambda=NULL</code> |
| <code>common.lambda</code> | 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 <code>common.lambda=TRUE</code> |
| <code>beta0</code> | (numeric vector) Initial value for the regression coefficients that the algorithm will update for <code>maxiter</code> iterations. If <code>beta0=NULL</code> a vector of zeros will be considered. These values will be used as starting values for the first lambda value |
| <code>alpha</code> | (numeric) Value between 0 and 1 for the weights given to the L1 and L2-penalties |
| <code>scale</code> | TRUE or FALSE to scale matrix Sigma for variables with unit variance and scale Gamma by the standard deviation (<code>sdX</code>) of the corresponding predictor taken from the diagonal of Sigma |
| <code>sdX</code> | (numeric vector) Scaling factor that will be used to scale the regression coefficients. When <code>scale = TRUE</code> 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 |
| <code>tol</code> | (numeric) Maximum error between two consecutive solutions of the CD algorithm to declare convergence |
| <code>maxiter</code> | (integer) Maximum number of iterations to run the CD algorithm at each lambda step before convergence is reached |
| <code>nsup.max</code> | (integer) Maximum number of non-zero coefficients in the last solution. Default <code>nsup.max=NULL</code> will calculate solutions for the entire lambda grid |
| <code>mc.cores</code> | (integer) Number of cores used. When <code>mc.cores > 1</code> , the analysis is run in parallel for each column of Gamma. Default is <code>mc.cores=1</code> |

| | |
|------------------|---|
| 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 | 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, $\mathbf{x}_i = (x_{i1}, \dots, x_{ip})'$ is a vector of p predictors assumed to have unit variance, $\boldsymbol{\beta} = (\beta_1, \dots, \beta_p)'$ 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 $j = 1, \dots, p$ 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.

Hoerl AE, Kennard RW (1970). Ridge Regression: Biased estimation for nonorthogonal problems. *Technometrics*, **12**(1), 55–67.

Tibshirani R (1996). Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society B*, **58**(1), 267–288.

Zou H, Hastie T (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society B*, **67**(2), 301–320.

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)

# Predicted values
yHat1 = fitted(fm, X=X[trn,]) # training data
yHat2 = fitted(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")
```

 5. 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, eps = .Machine$double.eps*100,
     scale = TRUE, sdx = NULL, mc.cores = 1L, save.at = NULL,
     precision.format = c("double", "single"),
     fileID = NULL, 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 |
| method | (character) Either: <ul style="list-style-type: none"> • '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 |
| eps | (numeric) An effective 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 |

| | |
|-------------------------------|--|
| <code>save.at</code> | (character) Path where regression coefficients are to be saved (this may include a prefix added to the files). Default <code>save.at=NULL</code> will no save the regression coefficients and they are returned in the output object |
| <code>fileID</code> | (character) Suffix added to the file name where regression coefficients are to be saved. Default <code>fileID=NULL</code> will automatically add sequential integers from 1 to the number of columns of <code>Gamma</code> |
| <code>precision.format</code> | (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 <code>save.at</code> is not <code>NULL</code> |
| <code>verbose</code> | TRUE or FALSE to whether printing each LARS step |

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, $\mathbf{x}_i = (x_{i1}, \dots, x_{ip})'$ is a vector of p predictors assumed to have unit variance, $\boldsymbol{\beta} = (\beta_1, \dots, \beta_p)'$ 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 `fitted` 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.

Friedman J, Hastie T, Tibshirani R (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, **33**(1), 1–22.

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

Tibshirani R (1996). Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society B*, **58**(1), 267–288.

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")

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

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

```
SSI.CV(y, X = NULL, b = NULL, Z = NULL, K, trn_tst = NULL,
    varU = NULL, varE = NULL, intercept = TRUE,
    alpha = 1, lambda = NULL, nlambdas = 100,
    lambda.min = .Machine$double.eps^0.5,
    common.lambda = TRUE, nCV = 1L, n folds = 5, seed = NULL,
    tol = 1E-4, maxiter = 500, method = c("REML", "ML"),
    name = NULL, mc.cores = 1L, verbose = TRUE)
```

Arguments

| | |
|------------|---|
| y | (numeric matrix) Response variable. It can contain >1 columns for multi-trait analysis |
| 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 $\mathbf{G} = \mathbf{K}$; otherwise $\mathbf{G} = \mathbf{Z} \mathbf{K} \mathbf{Z}'$ 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)). When y has >1 columns both varU and varE are matrices pairwise estimated (see help(getGenCov)) |
| intercept | TRUE or FALSE to whether fit an intercept. When FALSE, the model assumes a null intercept |
| trn_tst | (integer vector) Which elements from matrix y are in training/testing set (0: testing, 1: training, NA are ignored). Default trn_tst=NULL will consider all individuals as training |

| | |
|------------------|--|
| 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 when <code>saveAt</code> argument is provided and can be retrieved later using function 'collect' (see <code>help(collect)</code>). Default is <code>subset=NULL</code> for no subsetting, then the model is fitted for all testing data |
| 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 <code>lambda=NULL</code> , in this case a decreasing grid of <code>nlambda</code> <code>lambdas</code> will be generated starting from a maximum equal to <p style="text-align: center;">$\max(\text{abs}(G[\text{trn}, \text{tst}])/\alpha)$</p> to a minimum equal to zero. If <code>alpha=0</code> the grid is generated starting from a maximum equal to 5 |
| nlambda | (integer) Number of <code>lambdas</code> generated when <code>lambda=NULL</code> |
| lambda.min | (numeric) Minimum value of <code>lambda</code> in the generated grid when <code>lambda=NULL</code> |
| 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 <code>nfolds='n'</code> 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 <code>nCV=1</code> |
| common.lambda | TRUE or FALSE to whether computing the coefficients for a grid of <code>lambdas</code> common to all individuals in testing set or for a grid of <code>lambdas</code> specific to each individual in testing set. Default is <code>common.lambda=TRUE</code> |
| mc.cores | (integer) Number of cores used. When <code>mc.cores > 1</code> , the analysis is run in parallel for each testing set individual. Default is <code>mc.cores=1</code> |
| 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 <code>lambda</code> 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 <code>save.at=NULL</code> will no save any results and they are returned in the output object |
| 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 <code>save.at</code> is not NULL |
| method | (character) Either 'REML' (Restricted Maximum Likelihood) or 'ML' (Maximum Likelihood) to calculate variance components as per the function 'fit-BLUP' |
| name | (character) Name given to the output for tagging purposes. Default <code>name=NULL</code> will give the name of the method used |
| verbose | TRUE or FALSE to whether printing each step |

Details

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

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

where \mathbf{y} is a vector with the response, \mathbf{b} is the vector of fixed effects, \mathbf{g} is the vector of the genetic values of the genotypes, \mathbf{e} is the vector of environmental residuals, and \mathbf{X} and \mathbf{Z} are design matrices connecting the fixed and genetic effects with replicates. Genetic values 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 $\mathbf{u} = \mathbf{Z}\mathbf{g}$ will therefore follow $\mathbf{u} \sim N(\mathbf{0}, \sigma_u^2 \mathbf{G})$ where $\mathbf{G} = \mathbf{Z}\mathbf{K}\mathbf{Z}'$. 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 $\mathbf{u}_{tst} = (u_i)$, $i = 1, 2, \dots, n_{tst}$, for a testing set are estimated individual-wise using (as predictors) all available observations in a training set as

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

where β_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

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

where $\mathbf{G}_{trn, tst(i)}$ is the i^{th} column of the sub-matrix of \mathbf{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 $J(\beta_i)$ is a penalty function given by

$$1/2(1 - \alpha) \|\beta_i\|_2^2 + \alpha \|\beta_i\|_1$$

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

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

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

Value

Function 'SSI' returns a list object of the class 'SSI' for which methods coef, fitted, 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) product of the design matrix 'X' times the fixed effects solutions.
- u: (matrix) genetic values 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 'SSI.CV' returns a list object of length nCV of the class 'SSI.CV' for which methods plot and Optimal cross-validated penalization values can be obtained using thesummary method. Method plot is also available.

References

- Efron B, Hastie T, Johnstone I, Tibshirani R (2004). Least angle regression. *The Annals of Statistics*, **32**(2), 407–499.
- Friedman J, Hastie T, Höfling H, Tibshirani R (2007). Pathwise coordinate optimization. *The Annals of Applied Statistics*, **1**(2), 302–332.
- Hoerl AE, Kennard RW (1970). Ridge regression: biased estimation for nonorthogonal problems. *Technometrics*, **12**(1), 55–67.
- Lush JL (1947). Family merit an individual merit as bases for selection. Part I. *The American Naturalist*, **81**(799), 241–261.
- Tibshirani R (1996). Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society B*, **58**(1), 267–288.
- VanRaden PM (2008). Efficient methods to compute genomic predictions. *Journal of Dairy Science*, **91**(11), 4414–4423.
- Zou H, Hastie T (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society B*, **67**(2), 301–320

Examples

```
require(SFSI)
data(wheatHTP)

index = which(Y$trial %in% 1:8)      # 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

# Sets (testing=0, training=1)
trn_tst = ifelse(Y$trial %in% 1:2, 0, 1)

# Sparse selection index
# Fixed effects b and variances varU and varE are
# computed internally from training data
fm1 = SSI(y,K=G,trn_tst=trn_tst)
summary(fm1)$optCOR
varU = fm1$varU
```

```

varE = fm1$varE
b = fm1$b

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

# Fit the index with the obtained lambda
fm3 = SSI(y,K=G,varU=varU,varE=varE,b=b,trn_tst=trn_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

# Obtain an 'optimal' lambda by repeating the CV several times
fm22 = SSI.CV(y,K=G,varU=varU,varE=varE,b=b,trn_tst=trn_tst,nCV=5,name="5 5CV")
plot(fm22,fm2)

#-----
# Multi-trait SSI
#-----
y = scale(Y[,4:6])      # Response variable

# Sets (testing=0, training=1)
trn_tst = matrix(NA,ncol=ncol(y),nrow=nrow(y))
trn_tst[,1] = ifelse(Y$trial %in% 1:2, 0, 1)
trn_tst[,2] = ifelse(Y$trial %in% 1, 0, 1)
trn_tst[,3] = ifelse(Y$trial %in% 2, 0, 1)

fm = SSI(y, K=G, trn_tst=trn_tst, mc.cores=1)

```

7. 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,
        U = NULL, d = NULL, varU = NULL,
        varE = 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 matrix) Response variable. It can contain >1 columns, each of them will be fitted separately |
| 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 $\mathbf{G} = \mathbf{K}$; otherwise $\mathbf{G} = \mathbf{Z} \mathbf{K} \mathbf{Z}'$ is used |
| K | (numeric matrix) Kinship relationship matrix |
| U, d | (numeric matrix/vector) Eigenvectors and eigenvalues from eigen value decomposition of $\mathbf{G} = \mathbf{U} \mathbf{D} \mathbf{U}'$ |
| 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

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

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

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

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

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

where \mathbf{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} , by

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

where $\mathbf{G}_{tst,trn}$ is the sub-matrix of \mathbf{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, \mathbf{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) random 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

- VanRaden PM (2008). Efficient methods to compute genomic predictions. *Journal of Dairy Science*, **91**(11), 4414–4423.
- 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% 1:10)    # 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$trial %in% 1:2)
trn = which(!Y$trial %in% 1:2)

yNA <- y
yNA[tst] <- NA
fm = fitBLUP(yNA, K=G)
plot(y[tst], fm$u[tst])             # Predicted vs observed values in testing set
cor(y[tst], fm$u[tst])             # Prediction accuracy in testing set
cor(y[trn], fm$u[trn])             # Prediction accuracy in training set
fm$varU                             # Genetic variance
fm$varE                             # Residual variance
fm$h2                               # Heritability
fm$b                                # Intercept

```

8. Computing Genetic Covariances

Pairwise Genetic Covariance

Description

Pairwise genetic covariances for variables with the same experimental design

Usage

```

getGenCov(y, X = NULL, Z = NULL, K = NULL,
           U = NULL, d = NULL, scale = TRUE,
           pairwise=FALSE, verbose = TRUE, ...)

```

Arguments

y (numeric matrix) Response variable matrix. It should contain more than one column

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 $\mathbf{G} = \mathbf{K}$; otherwise $\mathbf{G} = \mathbf{Z} \mathbf{K} \mathbf{Z}'$ is used |
| K | (numeric matrix) Kinship relationship matrix |
| U | (numeric matrix) Eigenvectors from spectral value decomposition of $\mathbf{G} = \mathbf{U} \mathbf{D} \mathbf{U}'$ |
| d | (numeric vector) Eigenvalues from spectral value decomposition of $\mathbf{G} = \mathbf{U} \mathbf{D} \mathbf{U}'$ |
| 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

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

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

where \mathbf{b}_1 and \mathbf{b}_2 are the specific fixed effects, \mathbf{u}_1 and \mathbf{u}_2 are the specific genetic values 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 connecting the fixed and genetic effects with replicates. Genetic values are assumed to follow a Normal distribution as $\mathbf{u}_1 \sim N(\mathbf{0}, \sigma_{u_1}^2 \mathbf{K})$ and $\mathbf{u}_2 \sim N(\mathbf{0}, \sigma_{u_2}^2 \mathbf{K})$, and environmental terms are assumed $\mathbf{e}_1 \sim N(\mathbf{0}, \sigma_{e_1}^2 \mathbf{I})$ and $\mathbf{e}_2 \sim N(\mathbf{0}, \sigma_{e_2}^2 \mathbf{I})$.

The genetic covariance $\sigma_{u_1 u_2}^2$ is estimated from the formula for the variance for the sum of two variables as

$$\sigma_{u_1 u_2}^2 = \frac{1}{2}(\sigma_{u_3}^2 - \sigma_{u_1}^2 - \sigma_{u_2}^2)$$

where $\sigma_{u_3}^2$ 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 $\sigma_{e_1 e_2}^2$ is estimated as

$$\sigma_{e_1 e_2}^2 = \frac{1}{2}(\sigma_{e_3}^2 - \sigma_{e_1}^2 - \sigma_{e_2}^2)$$

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

Solutions are found using the function 'fitBLUP' (see help(fitBLUP)) to sequentially 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% 1:8) # 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 = scale(Y[,4:7]) # Response variable

# Covariances between the first and the rest: y[,1] and y[,2:4]
fm = getGenCov(y=y, K=G)
fm$varU # Genetic variances
fm$varE # Residual variances
fm$covU # Genetic covariance between y[,1] and y[,2:4]
fm$covU+fm$covE # Phenotypic covariance
cov(y)[1,-1] # Sample phenotypic covariance

# Pairwise covariance for all columns in y
fm = getGenCov(y=y, K=G, pairwise=TRUE)
fm$varU
fm$varE
```

 9. SSI accuracy vs penalization plot

Accuracy vs penalization plot

Description

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

Usage

```
## S3 method for class 'SSI'
plot(..., x.stat = c("nsup", "lambda"),
      y.stat = c("accuracy", "MSE"),
      nbreaks.x=7)
```

Arguments

| | |
|-----------|--|
| ... | Other arguments to be passed: <ul style="list-style-type: none"> • One or more objects of the class 'SSI' • Other arguments for method plot: 'xlab', 'ylab', 'main', 'lwd', 'xlim', 'ylim' |
| 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 |
| 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

```
require(SFSI)
data(wheatHTP)

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

# Sets (testing=0, training=1)
trn_tst = ifelse(Y$trial == 2, 0, 1)

fm = SSI(y,K=G,varU=0.4,varE=0.6,b=0,trn_tst=trn_tst)

plot(fm)
plot(fm, main=expression('corr('*y[obs]*','*y[pred]*') vs sparsity'))
plot(fm, y.stat="MSE",ylab='Mean Square Error', xlab='Sparsity')
```

Collecting SSI outputs

Collecting SSI outputs

Description

Collects all outputs saved at the provided save.at parameter from the SSI analysis when testing data was splited according to argument subset.

Usage

```
collect(path = "")
```

Arguments

path (character) Path where output files were saved, this may include a prefix

Value

An object of the class 'SSI' for which methods fitted, plot and summary exist

Examples

```
require(SFSI)
data(wheatHTP)

index = which(Y$trial %in% 1:8) # 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"])) # Response variable

# Sets (testing=0, training=1)
trn_tst = ifelse(Y$trial %in% 1:2, 0, 1)

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

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

# Collect all results after completion
fm = collect(path)
```

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

Arguments

| | |
|----------|--|
| object | Either a numeric matrix X or an object of the 'SSI' class. When the object is an 'SSI' the regression coefficients are used as X |
| K | (numeric matrix) Kinship relationship matrix among nodes |
| Z | (numeric matrix) Design matrix for random effects. When $Z=NULL$ an identity matrix is considered (default) thus $\mathbf{G} = \mathbf{K}$; otherwise $\mathbf{G} = \mathbf{Z} \mathbf{K} \mathbf{Z}'$ is used |
| nsup | (numeric) For a SSI, 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 SSI, a factor (x-folds radius) to separate each trait from the origin |
| delta | (numeric) Minimum 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 \mathbf{K} is provided, nodes are plotted according to the top 2 eigenvectors from the spectral value decomposition of $\mathbf{Z} \mathbf{K} \mathbf{Z}' = \mathbf{U} \mathbf{D} \mathbf{U}'$.

When the object is a 'SSI' object the edges are taken from the regression coefficients (associated to a specific nsup value) are used as matrix \mathbf{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 SSI 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
```

```

trn_tst = ifelse(Y$trial == 2, 0, 1)

fm = SSI(y,K=G,trn_tst=trn_tst)

tmp = net(fm)          # Get the net
plot(tmp)             # Plot the net
plot(tmp, i=c(1,2))   # 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.tick = 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.tick | (numeric) Tick 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 'SSI' |
| ... | 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

```
require(SFSI)
data(wheatHTP)

#-----
# Net for an SSI 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
```

```

trn_tst = ifelse(Y$trial == 2, 0, 1)

fm = SSI(y,K=G,trn_tst=trn_tst)

tmp = net(fm)          # Get the net
plot(tmp)              # Plot the net
plot(tmp, i=c(1,2))    # 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))

```

Multitrait SSI accuracy vs penalization plot

Accuracy vs penalization from Multitrait SSI

Description

Visualizing results from an object of the class 'SSI'

Usage

```

multitrait.plot(object, x.stat = c("nsup", "lambda"),
                y.stat = c("accuracy", "MSE"),
                line.color = "orange",
                nbreaks.x = 7, ...)

```

Arguments

| | |
|------------|---|
| object | An object of the class 'SSI' for a multitrait 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 |
| line.color | (character) Color of the lines |

`nbreaks.x` (integer) Number of breaks in the x-axis
 ... 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

```
require(SFSI)
data(wheatHTP)

index = which(Y$trial %in% 1:9) # 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 = scale(Y[,4:5]) # Response variable

# Sets (testing=0, training=1)
trn_tst = matrix(NA,ncol=ncol(y),nrow=nrow(y))
trn_tst[,1] = ifelse(Y$trial %in% 2, 0, 1)
trn_tst[,2] = ifelse(Y$trial %in% 2, 0, 1)

fm = SSI(y, K=G, trn_tst=trn_tst, mc.cores=1)
multitrait.plot(fm)
```

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)
```

Regression coefficients and fitted 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'
fitted(object, ...)
```

Arguments

| | |
|--------|--|
| object | An object of the class 'LASSO' returned either by the function 'LARS' or 'solveEN' |
| ... | Other arguments: <ul style="list-style-type: none"> • X (numeric matrix) scores for as many predictors there are in <code>ncol(object\$beta)</code> (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 `fitted` returns a matrix with fitted values $\mathbf{X}\beta$ (in rows) for each value of lambda (in columns).

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)

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

# Predicted values
yHat1 = fitted(fm, X=X[trn,]) # training data
yHat2 = fitted(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")

```

Regression coefficients path
Coefficients path plot

Description

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

Usage

```

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

```

Arguments

| | |
|--------|---|
| object | An object of the 'LASSO' or 'SSI' class |
| Z | (numeric matrix) Design matrix for random effects. When Z=NULL an identity matrix is considered (default) thus $\mathbf{G} = \mathbf{K}$; otherwise $\mathbf{G} = \mathbf{Z} \mathbf{K} \mathbf{Z}'$ is used. Only needed for a fm object of the class 'SSI' |
| K | (numeric matrix) Kinship relationships. This can be a name of a binary file where the matrix is stored. Only needed for a fm object of the class 'SSI' |

| | |
|-------------------------|---|
| <code>i</code> | (integer vector) Index a response variable (columns of matrix Gamma) for an object of the class 'LASSO'. Index testing elements (stored in <code>object\$tst</code>) for an object of the class 'SSI'. Default <code>i=NULL</code> will consider either all columns in matrix Gamma or all elements in <code>object\$tst</code> , respectively |
| <code>prune</code> | 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 |
| <code>cor.max</code> | (numeric) Correlation threshold to prune within groups of correlated coefficients |
| <code>lambda.min</code> | (numeric) Minimum value of lambda to show in the plot as $-\log(\lambda)$. This prevents $-\log(\lambda)$ going to infinite for near-zero lambda values |
| <code>nbreaks.x</code> | (integer) Number of breaks in the x-axis |
| <code>...</code> | Other arguments for method <code>plot</code> : '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 family index
trn_tst = ifelse(seq_along(y)<11,0,1)
fm = SSI(y,K=G,trn_tst=trn_tst)

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 SSI
path.plot(fm, K=G, i=1)
```

 Summarize SSI results *SSI methods*

Description

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

Usage

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

## S3 method for class 'SSI'
fitted(object, ...)

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

Arguments

| | |
|---------------------|---|
| <code>object</code> | An object of the class 'SSI' |
| <code>...</code> | Other arguments to be passed to <code>coef</code> method: <ul style="list-style-type: none"> <code>nsup</code>: (numeric) Average (across testing individuals) number of non-zero regression coefficients. Only the coefficients for the lambda associated to <code>nsup</code> are returned as a 'matrix' with testing individuals in rows <code>iy</code>: (integer vector) Index testing elements (stored in <code>object\$stst</code>) to be considered. Only coefficients corresponding to the testing individuals <code>object\$stst[i]</code> are returned |
| | For <code>fitted</code> and <code>summary</code> methods: <ul style="list-style-type: none"> <code>y</code>: (numeric vector) An optional response matrix |

Value

Method `fitted` 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 SSI with maximum accuracy.
- optMSE: (vector) summary of the SSI with minimum MSE.

Examples

```
require(SFSI)
data(wheatHTP)

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

# Sets (testing=0, training=1)
trn_tst = ifelse(Y$trial == 2, 0, 1)

fm = SSI(y,K=G,varU=0.4,varE=0.6,b=0,trn_tst=trn_tst)

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

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 stored in a matrix of 250 rows and 4 columns (folds). Genetic and residual covariances matrices among YLD within each environment are stored 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

- * **LARS**
 - 5. Least Angle Regression (LARS), [9](#)
- * **Prune**
 - R-squared pruning, [28](#)
- * **SSI**
 - 6. Sparse Selection Index (SSI), [11](#)
- * **datasets**
 - Wheat data set, [34](#)
- * **fitBLUP**
 - 7. BLUP estimation from Linear Mixed Model, [16](#)
 - 8. Computing Genetic Covariances, [19](#)
- * **solveEN**
 - 4. Solve an Elastic-Net problem, [5](#)
- 1. Save/read binary files, [2](#)
- 3. Covariance matrix transformations, [4](#)
- 4. Solve an Elastic-Net problem, [5](#)
- 5. Least Angle Regression (LARS), [9](#)
- 6. Sparse Selection Index (SSI), [11](#)
- 7. BLUP estimation from Linear Mixed Model, [16](#)
- 8. Computing Genetic Covariances, [19](#)
- 9. SSI accuracy vs penalization plot, [21](#)
- coef.LASSO (Regression coefficients and fitted values in LASSO-type problems), [30](#)
- coef.SSI (Summarize SSI results), [33](#)
- collect (Collecting SSI outputs), [22](#)
- Collecting SSI outputs, [22](#)
- cov2cor2 (3. Covariance matrix transformations), [4](#)
- cov2dist (3. Covariance matrix transformations), [4](#)
- fitBLUP (7. BLUP estimation from Linear Mixed Model), [16](#)
- fitted.LASSO (Regression coefficients and fitted values in LASSO-type problems), [30](#)
- fitted.SSI (Summarize SSI results), [33](#)
- genCOV_xy (Wheat data set), [34](#)
- genCOV_yy (Wheat data set), [34](#)
- getGenCov (8. Computing Genetic Covariances), [19](#)
- Graphical Network from a numeric matrix, [23](#)
- Graphical network plot, [25](#)
- LARS (5. Least Angle Regression (LARS)), [9](#)
- M (Wheat data set), [34](#)
- Multitrait SSI accuracy vs penalization plot, [27](#)
- multitrait.plot (Multitrait SSI accuracy vs penalization plot), [27](#)
- net (Graphical Network from a numeric matrix), [23](#)
- path.plot (Regression coefficients path), [31](#)
- plot.net (Graphical network plot), [25](#)
- plot.SSI (9. SSI accuracy vs penalization plot), [21](#)
- Prune (R-squared pruning), [28](#)
- R-squared pruning, [28](#)
- readBinary (1. Save/read binary files), [2](#)
- Regression coefficients and fitted values in LASSO-type problems, [30](#)

Regression coefficients path, [31](#)
resCOV_yy (Wheat data set), [34](#)

saveBinary (1. Save/read binary files),
[2](#)

solveEN (4. Solve an Elastic-Net
problem), [5](#)

SSI (6. Sparse Selection Index (SSI)),
[11](#)

Summarize SSI results, [33](#)
summary.SSI (Summarize SSI results), [33](#)

VI_E1 (Wheat data set), [34](#)

Wheat data set, [34](#)
wheatHTP (Wheat data set), [34](#)

X_E1 (Wheat data set), [34](#)

Y (Wheat data set), [34](#)