Package 'decorrelate'

July 22, 2025

Type Package

Title Decorrelation Projection Scalable to High Dimensional Data

Version 0.1.6.4

Date 2025-06-18

Description Data whitening is a widely used preprocessing step to remove correlation structure since statistical models often assume independence. Here we use a probabilistic model of the observed data to apply a whitening transformation. This Gaussian Inverse Wishart Empirical Bayes model substantially reduces computational complexity, and regularizes the eigen-values of the sample covariance matrix to improve out-of-sample performance.

License Artistic-2.0

Encoding UTF-8

URL https://gabrielhoffman.github.io/decorrelate/

BugReports https://github.com/GabrielHoffman/decorrelate/issues

Suggests knitr, pander, whitening, CCA, yacca, mvtnorm, ggplot2, cowplot, colorRamps, RUnit, latex2exp, clusterGeneration, rmarkdown

Depends R (>= 4.2.0), methods

Imports Rfast, irlba, graphics, Rcpp, CholWishart, Matrix, utils, stats

RoxygenNote 7.3.2

VignetteBuilder knitr

LinkingTo Rcpp, RcppArmadillo

NeedsCompilation yes

Author Gabriel Hoffman [aut, cre] (ORCID: <https://orcid.org/0000-0002-0957-0224>)

Maintainer Gabriel Hoffman <gabriel.hoffman@mssm.edu>

Repository CRAN

Date/Publication 2025-07-18 13:10:03 UTC

Contents

`autocorr.mat` *Create auto-correlation matrix*

Description

Create auto-correlation matrix

Usage

```
autocorr.mat(p, rho)
```

Arguments

`p`	dimension of matrix
`rho`	autocorrelation value

Value

auto-matrix of size p with parameter rho

Examples

```
# Create 4x4 matrix with correlation between adjacent enties is 0.9
autocorr.mat(4, .9)
```

`averageCorr` *Summarize correlation matrix*

Description

Summarize correlation matrix as a scalar scalar value, given its SVD and shrinkage parameter. `averageCorr()` computes the average correlation, and `averageCorrSq()` computes the average squared correlation, where both exclude the diagonal terms. `sumInverseCorr()` computes the sum of entries in the inverse correlation matrix to give the 'effective number of independent features'. `effVariance()` evaluates effective variance of the correlation (or covariance) matrix. These values can be computed using the correlation matrix using standard MLE, or EB shrinkage.

Usage

```
averageCorr(ecl, method = c("EB", "MLE"))

averageCorrSq(ecl, method = c("EB", "MLE"))

sumInverseCorr(ecl, method = c("EB", "MLE"), absolute = TRUE)

effVariance(ecl, method = c("EB", "MLE"))

tr(ecl, method = c("EB", "MLE"))
```

Arguments

`ecl`	estimate of correlation matrix from `eclairs()` storing U, d_1^2, λ and ν
`method`	compute average correlation for either the empirical Bayes (EB) shinken correlation matrix or the MLE correlation matrix
`absolute`	if `TRUE` (default) evaluate on absolute correlation matrix

Details

`tr()`: trace of the matrix. Sum of diagonals is the same as the sum of the eigen-values.

`averageCorr()`: The average correlation is computed by summing the off-diagonal values in the correlation matrix. The sum of all elements in a matrix is $g = \sum_{i,j} C_{i,j} = 1^T C 1$, where 1 is a vector of p elements with all entries 1. This last term is a quadratic form of the correlation matrix

that can be computed efficiently using the SVD and shrinkage parameter from `eclairs()`. Given the value of g, the average is computed by subtracting the diagonal values and dividing by the number of off-diagonal values: $(g-p)/(p(p-1))$.

`averageCorrSq()`: The average squared correlation is computed using only the eigen-values. Surprisingly, this is a function of the variance of the eigen-values. The is reviewed by Watanabe (2022) and Durand and Le Roux (2017). Letting λ_i be the i^{th} sample or shrunk eigen-value, and $\tilde{\lambda}$ be the mean eigen-value, then $\sum_i(\lambda_i - \tilde{\lambda})^2/p(p-1)\tilde{\lambda}^2$.

`sumInverseCorr()`: The 'effective number of independent features' is computed by summing the entires of the inverse covariance matrix. This has the form $\sum_{i,j} C^{-1}_{i,j} = 1^T C^{-1} 1$. This last term is a quadratic form of the correlation matrix that can be computed efficiently using the SVD and shrinkage parameter from `eclairs()` as described above.

`effVariance()`: Compute a metric of the amount of variation represented by a correlation (or covariance) matrix that is comparable across matrices of difference sizes. Proposed by Peña and Rodriguez (2003), the 'effective variance' is $|C|^{\frac{1}{p}}$ where C is a correlation (or covariance matrix) between p variables. The effective variance is the mean of the log eigen-values.

Value

value of summary statistic

References

Durand, J. L., & Le Roux, B. (2017). Linkage index of variables and its relationship with variance of eigenvalues in PCA and MCA. Statistica Applicata-Italian Journal of Applied Statistics, (2-3), 123-135.

Peña, D., & Rodriguez, J. (2003). Descriptive measures of multivariate scatter and linear dependence. Journal of Multivariate Analysis, 85(2), 361-374.

Watanabe, J. (2022). Statistics of eigenvalue dispersion indices: Quantifying the magnitude of phenotypic integration. Evolution, 76(1), 4-28.

Examples

```
library(Rfast)
n <- 200 # number of samples
p <- 800 # number of features

# create correlation matrix
Sigma <- matrix(.2, p, p)
diag(Sigma) <- 1

# draw data from correlation matrix Sigma
Y <- rmvnorm(n, rep(0, p), sigma = Sigma, seed = 1)
rownames(Y) <- paste0("sample_", seq(n))
colnames(Y) <- paste0("gene_", seq(p))

# eclairs decomposition
ecl <- eclairs(Y, compute = "cor")

# Average correlation value
```

```
averageCorr(ecl)

# Average squared correlation value
averageCorrSq(ecl)

# Sum elements in inverse correlation matrix
# Gives the effective number of independent features
sumInverseCorr(ecl)

# Effective variance
effVariance(ecl)
```

`cov_transform` *Estimate covariance matrix after applying transformation*

Description

Given covariance between features in the original data, estimate the covariance matrix after applying a transformation to each feature. Here we use the eclairs decomposition of the original covariance matrix, perform a parametric bootstrap and return the eclairs decomposition of the covariance matrix of the transformed data.

Usage

```
cov_transform(
  ecl,
  f,
  n.boot,
  lambda = NULL,
  compute = c("covariance", "correlation"),
  seed = NULL
)
```

Arguments

`ecl`	covariance/correlation matrix as an eclairs object
`f`	function specifying the transformation.
`n.boot`	number of parametric bootstrap samples. Increasing n gives more precise estimates.
`lambda`	shrinkage parameter. If not specified, it is estimated from the data.
`compute`	evaluate either the `"covariance"` or `"correlation"` of `X`
`seed`	If you want the same to be generated again use a seed for the generator, an integer number

Details

When the transformation is linear, these covariance matrices are the same.

Value

eclairs decomposition representing correlation/covariance on the transformed data

Examples

```
library(Rfast)

n <- 800 # number of samples
p <- 200 # number of features

# create correlation matrix
Sigma <- autocorr.mat(p, .9)

# sample matrix from MVN with covariance Sigma
Y <- rmvnorm(n, rep(0, p), sigma = Sigma, seed = 1)

# perform eclairs decomposition
ecl <- eclairs(Y)

# Parametric boostrap to estimate covariance
# after transformation

# transformation function
f <- function(x) log(x^2 + 1e-3)

# number of bootstrap samples
n_boot <- 5000

# Evaluate eclairs decomposition on boostrap samples
ecl2 <- cov_transform(ecl, f = f, n_boot, lambda = 1e-4)

# Get full covariance matrix from eclairs decomposition
C1 <- getCov(ecl2)

# Parametric boostrap samples directly from full covariance matrix
X <- rmvnorm(n_boot, rep(0, p), getCov(ecl))

# get covariance of transformed data
C2 <- cov(f(X))

# Evaluate differences
# small differences are due to Monte Carlo error from boostrap sampling
range(C1 - C2)

# Plot entries from two covariance estimates
par(pty = "s")
plot(C1, C2, main = "Concordance between covariances")
abline(0, 1, col = "red")

# Same above but compute eclairs for correlation matrix
#-------------------------------------------------------
```

```
# Evaluate eclairs decomposition on boostrap samples
ecl2 <- cov_transform(ecl, f = f, n_boot, compute = "correlation", lambda = 1e-4)

# Get full covariance matrix from eclairs decomposition
C1 <- getCor(ecl2)

# Parametric boostrap samples directly from full covariance matrix
X <- rmvnorm(n_boot, rep(0, p), getCov(ecl))

# get correlation of transformed data
C2 <- cor(f(X))

# Evaluate differences
# small differences are due to Monte Carlo error from boostrap sampling
range(C1 - C2)

# Plot entries from two correlation estimates
oldpar <- par(pty = "s")
plot(C1, C2, main = "Correlation between covariances")
abline(0, 1, col = "red")

par(oldpar)
```

`decorrelate` *Decorrelation projection*

Description

Efficient decorrelation projection using eclairs decomposition

Usage

```
decorrelate(X, ecl, lambda, transpose = FALSE, alpha = -1/2)
```

Arguments

`X`	matrix to be transformed so *columns* are independent
`ecl`	estimate of covariance/correlation matrix from eclairs storing U, d_1^2, λ and ν
`lambda`	specify lambda and override value from `ecl`
`transpose`	logical, (default FALSE) indicating if X should be transposed first
`alpha`	default = -1/2. Exponent of eigen-values

Details

Apply a decorrelation transform using the implicit covariance approach to avoid directly evaluating the covariance matrix

Value

a matrix following the decorrelation transformation

Examples

```
library(Rfast)

n <- 800 # number of samples
p <- 200 # number of features

# create correlation matrix
Sigma <- autocorr.mat(p, .9)

# draw data from correlation matrix Sigma
Y <- rmvnorm(n, rep(0, p), sigma = Sigma * 5.1, seed = 1)

# eclairs decomposition
ecl <- eclairs(Y)

# whitened Y
Y.transform <- decorrelate(Y, ecl)
#
```

`dmult` *Multiply by diagonal matrix*

Description

Multiply by diagonal matrix using efficient algorithm

Usage

```
dmult(M, v, side = c("left", "right"))
```

Arguments

`M`	matrix
`v`	vector with entries forming a diagonal matrix matching the dimensions of `M` depending on the value of `side`
`side`	is the matrix `M` `"left"` or `"right"` multiplied by the diagonal matrix

Details

Naively multiplying by a diagonal matrix with `p` entries takes $\mathcal{O}(p^2)$, even though must values in the diagonal matrix are zero. R has built in syntax to scale *columns*, so `diag(v) %*% M` can be evaluated with `v * M`. This is often difficult to remember, hard to look up, and scaling *rows* requires `t(t(M) * v)`. This is hard to read and write, and requires two transpose operations.

Here, `dmult()` uses `Rcpp` to evaluate the right multiplication efficiently, and uses `v * M` for left multiplication. This gives good performance and readability.

In principle, the `Rcpp` code can be modified to use BLAS for better performance by treating a `NumericMatrix` as a `C` array. But this is not currently a bottleneck

Value

matrix product

Examples

```
# right multiply
# mat %*% diag(v)
n <- 30
p <- 20
mat <- matrix(n * p, n, p)
v <- rnorm(p)

A <- dmult(mat, v, side = "right")
B <- mat %*% diag(v)
range(A - B)

# left multiply
# diag(v) %*% mat
n <- 30
p <- 20
mat <- matrix(n * p, p, n)
v <- rnorm(p)

A <- dmult(mat, v, side = "left")
B <- diag(v) %*% mat
range(A - B)
```

---

eclairs *Estimate covariance/correlation with low rank and shrinkage*

Description

Estimate the covariance/correlation between columns as the weighted sum of a low rank matrix and a scaled identity matrix. The weight acts to shrink the sample correlation matrix towards the identity matrix or the sample covariance matrix towards a scaled identity matrix with constant variance. An estimate of this form is useful because it is fast, and enables fast operations downstream. The method is based on the Gaussian Inverse Wishart Empirical Bayes (GIW-EB) model.

Usage

```
eclairs(
  X,
  k,
```

```
  lambda = NULL,
  compute = c("covariance", "correlation"),
  n.samples = nrow(X)
)
```

Arguments

X	data matrix with n samples as rows and p features as columns
k	the rank of the low rank component
lambda	shrinkage parameter. If not specified, it is estimated from the data.
compute	evaluate either the "covariance" or "correlation" of X
n.samples	number of samples data is from. Usually nrow(X), but can be other values in special cases.

Details

Compute U, d^2 to approximate the correlation matrix between columns of data matrix X by $Udiag(d^2(1-\lambda))U^T + I\nu\lambda$. When computing the covariance matrix, scale by the standard deviation of each feature.

Value

eclairs object storing:

U: orthonormal matrix with k columns representing the low rank component

dSq: eigen-values so that $Udiag(d^2)U^T$ is the low rank component

lambda: shrinkage parameter λ for the scaled diagonal component

sigma: standard deviations of input columns

nu: diagonal value, ν, of target matrix in shrinkage

n: number of samples (i.e. rows) in the original data

p: number of features (i.e. columns) in the original data

k: rank of low rank component

rownames: sample names from the original matrix

colnames: features names from the original matrix

method: method used for decomposition

call: the function call

Examples

```
library(Rfast)

n <- 800 # number of samples
p <- 200 # number of features

# create correlation matrix
Sigma <- autocorr.mat(p, .9)
```

```
# draw data from correlation matrix Sigma
Y <- rmvnorm(n, rep(0, p), sigma = Sigma * 5.1, seed = 1)
rownames(Y) <- paste0("sample_", seq(n))
colnames(Y) <- paste0("gene_", seq(p))

# eclairs decomposition: covariance
ecl <- eclairs(Y, compute = "covariance")

ecl

# eclairs decomposition: correlation
ecl <- eclairs(Y, compute = "correlation")

ecl
```

`eclairs-class` *Class eclairs*

Description

Class `eclairs`

Details

Object storing:

U: orthonormal matrix with k columns representing the low rank component

dSq: eigen-values so that $Udiag(d^2)U^T$ is the low rank component

lambda: shrinkage parameter λ for the scaled diagonal component

sigma: standard deviations of input columns

nu: diagonal value, ν, of target matrix in shrinkage

n: number of samples (i.e. rows) in the original data

p: number of features (i.e. columns) in the original data

k: rank of low rank component

rownames: sample names from the original matrix

colnames: features names from the original matrix

method: method used for decomposition

call: the function call

`eclairs_corMat` *Estimate covariance/correlation with low rank and shrinkage*

Description

Estimate covariance/correlation with low rank and shrinkage from the correlation matrix

Usage

```
eclairs_corMat(C, n, k = min(n, nrow(C)), lambda = NULL)
```

Arguments

`C`	sample correlation matrix between features
`n`	number of samples used to estimate the sample correlation matrix
`k`	the rank of the low rank component. Defaults to min of sample size and feature number, `min(n, p)`.
`lambda`	shrinkage parameter. If not specified, it is estimated from the data.

Value

eclairs object storing:

U: orthonormal matrix with k columns representing the low rank component

dSq: eigen-values so that $Udiag(d^2)U^T$ is the low rank component

lambda: shrinkage parameter λ for the scaled diagonal component

sigma: standard deviations of input columns

nu: diagonal value, ν, of target matrix in shrinkage

n: number of samples (i.e. rows) in the original data

p: number of features (i.e. columns) in the original data

k: rank of low rank component

rownames: sample names from the original matrix

colnames: features names from the original matrix

method: method used for decomposition

call: the function call

Examples

```
library(Rfast)

n <- 800 # number of samples
p <- 200 # number of features

# create correlation matrix
Sigma <- autocorr.mat(p, .9)

# draw data from correlation matrix Sigma
Y <- rmvnorm(n, rep(0, p), sigma = Sigma * 5.1, seed = 1)
rownames(Y) <- paste0("sample_", 1:n)
colnames(Y) <- paste0("gene_", 1:p)

# eclairs decomposition
eclairs(Y, compute = "correlation")

# eclairs decomposition from correlation matrix
eclairs_corMat(cor(Y), n = n)
```

`eclairs_sq` *Compute eclairs decomp of squared correlation matrix*

Description

Given the eclairs decomp of C, compute the eclairs decomp of C^2

Usage

```
eclairs_sq(ecl, rank1 = ecl$k, rank2 = Inf, varianceFraction = 1)
```

Arguments

`ecl`	estimate of correlation matrix from `eclairs()` storing U, d_1^2, λ and ν
`rank1`	use the first 'rank' singular vectors from the SVD. Using increasing rank1 will increase the accuracy of the estimation. But note that the computationaly complexity is O(P choose(rank, 2)), where P is the number of features in the dataset
`rank2`	rank of `eclairs()` decomposition returned
`varianceFraction`	fraction of variance to retain after truncating trailing eigen values

Details

Consider a data matrix X_{NxP} of P features and N samples where $N << P$. Let the columns of X be scaled so that $C_{PxP} = XX^T$. C is often too big to compute directly since it is O(P^2) and O(P^3) to invert. But we can compute the SVD of X in O(PN^2). The goal is to compute the SVD of the matrix C^2, given only the SVD of C in less than $O(P^2)$ time. Here we compute this SVD

of C^2 in $O(PN^4)$ time, which is tractible for small N. Moreover, if we use an SVD X = UDV^T with of rank R, we can approximate the SVD of C^2 in O(PR^4) using only D and V

Value

compute the eclairs of C^2

Examples

```
# Compute correlations directly and using eclairs decomp

n <- 600 # number of samples
p <- 100 # number of features

# create correlation matrix
Sigma <- autocorr.mat(p, .9)

# draw data from correlation matrix Sigma
Y <- Rfast::rmvnorm(n, rep(0, p), sigma = Sigma, seed = 1)
rownames(Y) <- paste0("sample_", seq(n))
colnames(Y) <- paste0("gene_", seq(p))

# correlation computed directly
C <- cor(Y)

# correlation from eclairs decomposition
ecl <- eclairs(Y, compute = "cor")
C.eclairs <- getCor(ecl, lambda = 0)

all.equal(C, C.eclairs)

# Correlation of Y^2
#-------------------

# exact quadratic way
C <- 2 * cor(Y)^2

# faster low rank
ecl2 <- eclairs_sq(ecl)
C.eclairs <- 2 * getCor(ecl2, lambda = 0)

all.equal(C.eclairs, C)
```

---

fastcca-class *Class fastcca*

Description

Class fastcca

Details

Object storing:

n.comp: number of canonical components

cors: canonical correlations

x.coefs: canonical coefficients for X

x.vars: canonical variates for X

y.coefs: canonical coefficients for Y

y.vars: canonical variates for Y

lambdas: shrinkage parameters from `eclairs`

`getCov` *Get full covariance/correlation matrix from eclairs*

Description

Get full covariance/correlation matrix from eclairs decomposition

Usage

```
getCov(ecl, lambda, ...)

getCor(ecl, lambda, ...)

## S4 method for signature 'eclairs'
getCov(ecl, lambda, ...)

## S4 method for signature 'eclairs'
getCor(ecl, lambda, ...)
```

Arguments

`ecl`	eclairs decomposition
`lambda`	shrinkage parameter for the convex combination.
`...`	other arguments

Details

The full matrix is computationally expensive to compute and uses a lot of memory for large p. So it is better to use decorrelate or mult_eclairs to perform projections in $O(np)$ time.

Value

p x p covariance/correlation matrix

Examples

```
library(Rfast)

n <- 800 # number of samples
p <- 200 # number of features

# create correlation matrix
Sigma <- autocorr.mat(p, .9)

# draw data from correlation matrix Sigma
Y <- rmvnorm(n, rep(0, p), sigma = Sigma * 5.1, seed = 1)
rownames(Y) <- paste0("sample_", seq(n))
colnames(Y) <- paste0("gene_", seq(p))

# eclairs decomposition
ecl <- eclairs(Y)

# extract covariance implied by eclairs decomposition
getCov(ecl)[1:3, 1:3]
```

`getShrinkageParams` *Estimate shrinkage parameter by empirical Bayes*

Description

Estimate shrinkage parameter by empirical Bayes

Usage

```
getShrinkageParams(ecl, k = ecl$k, minimum = 1e-04, lambda = NULL)
```

Arguments

`ecl`	`eclairs()` decomposition
`k`	number of singular vectors to use
`minimum`	minimum value of lambda
`lambda`	(default: NULL) If NULL, estimate lambda from data. Else evaluate logML using specified lambda value.

Details

Estimate shrinkage parameter for covariance matrix estimation using empirical Bayes method (Hannart and Naveau, 2014; Leday and Richardson 2019). The shrinage estimate of the covariance matrix is $(1-\lambda)\hat{\Sigma} + \lambda\nu I$, where $\hat{\Sigma}$ is the sample covariance matrix, given a value of λ. A large value of λ indicates more weight on the prior.

Value

value λ and ν indicating the shrinkage between sample and prior covariance matrices.

References

Hannart, A., & Naveau, P. (2014). Estimating high dimensional covariance matrices: A new look at the Gaussian conjugate framework. Journal of Multivariate Analysis, 131, 149-162.

Leday, G. G., & Richardson, S. (2019). Fast Bayesian inference in large Gaussian graphical models. Biometrics, 75(4), 1288-1298.

Examples

```
library(Rfast)

n <- 800 # number of samples
p <- 200 # number of features

# create correlation matrix
Sigma <- autocorr.mat(p, .9)

# draw data from correlation matrix Sigma
Y <- rmvnorm(n, rep(0, p), sigma = Sigma * 5.1, seed = 1)
rownames(Y) <- paste0("sample_", seq(n))
colnames(Y) <- paste0("gene_", seq(p))

# eclairs decomposition: covariance
ecl <- eclairs(Y, compute = "correlation")

# For full SVD
getShrinkageParams(ecl)

# For truncated SVD at k = 20
getShrinkageParams(ecl, k = 20)
```

---

getWhiteningMatrix *Get whitening matrix*

Description

Get whitening matrix implied by eclairs decompostion

Usage

```
getWhiteningMatrix(ecl, lambda)
```

Arguments

ecl	estimate of covariance/correlation matrix from eclairs storing U, d_1^2, λ and ν
lambda	specify lambda and override value from ecl

Value

whitening matrix

Examples

```
library(Rfast)

n <- 2000
p <- 3

Y <- matrnorm(n, p, seed = 1) * 10

# decorrelate with implicit whitening matrix
# give same result as explicity whitening matrix
ecl <- eclairs(Y, compute = "covariance")

# get explicit whitening matrix
W <- getWhiteningMatrix(ecl)

# apply explicit whitening matrix
Z1 <- tcrossprod(Y, W)

# use implicit whitening matrix
Z2 <- decorrelate(Y, ecl)

range(Z1 - Z2)
```

---

kappa,eclairs-method *Compute condition number*

Description

Compute condition number of matrix from eclairs decomposition

Usage

```
## S4 method for signature 'eclairs'
kappa(z, lambda = NULL)
```

Arguments

z	eclairs() decomposition
lambda	specify lambda to override value from z

Value

condition number of the correlation matrix. If z is a covariance matrix, kappa is only computed for the correlation component

Examples

```
library(Rfast)

n <- 800 # number of samples
p <- 200 # number of features

# create correlation matrix
Sigma <- autocorr.mat(p, .9)

# draw data from correlation matrix Sigma
Y <- rmvnorm(n, rep(0, p), sigma = Sigma * 5.1, seed = 1)
rownames(Y) <- paste0("sample_", seq(n))
colnames(Y) <- paste0("gene_", seq(p))

# eclairs decomposition
ecl <- eclairs(Y, compute = "correlation")

# compute condition number
kappa(ecl)
```

---

lm_each_eclairs *Fit linear model on each feature after decorrelating*

Description

Fit linear model on each feature after applying decorrelation projection to response and predictors.

Usage

```
lm_each_eclairs(
  formula,
  data,
  X,
  ecl,
  subset,
  weights,
  na.action,
  method = "qr",
  model = TRUE,
  x = FALSE,
  y = FALSE,
  qr = TRUE,
```

```
  singular.ok = TRUE,
  contrasts = NULL,
  offset,
  ...
)
```

Arguments

formula	an object of class 'formula' (or one that can be coerced to that class): a symbolic description of the model to be fitted.
data	a matrix or data.frame containing the variables in the model
X	matrix or data.frame where each column stores a predictor to be evaluated by the regression model one at a time. The i^{th} model includes `X[,i]` as a predictor.
ecl	estimate of covariance/correlation matrix from eclairs storing U, d_1^2, λ and ν
subset	same as for lm
weights	same as for lm
na.action	same as for lm
method	same as for lm
model	same as for lm
x	same as for lm
y	same as for lm
qr	same as for lm
singular.ok	same as for lm
contrasts	same as for lm
offset	same as for lm
...	other arguments passed to `lm()`

Value

data.frame with columns `beta`, `se`, `tsat`, `pvalue` storing results for regression model fit for each feature

Examples

```
library(Rfast)

n <- 800 # number of samples
p <- 200 # number of features

# create correlation matrix
Sigma <- autocorr.mat(p, .9)

# draw data from correlation matrix Sigma
Y <- rmvnorm(n, rep(0, p), sigma = Sigma * 5.1, seed = 1)
```

```
# eclairs decomposition
ecl <- eclairs(Y)

# simulate covariates
data <- data.frame(matrnorm(p, 2, seed = 1))
colnames(data) <- paste0("v", 1:2)

# simulate response
y <- rnorm(p)

# Simulate 1000 features to test
X <- matrnorm(p, 1000, seed = 1)
colnames(X) <- paste0("set_", seq(ncol(X)))

# Use linear model to test each feature stored as columns in X
res <- lm_each_eclairs(y ~ v1 + v2, data, X, ecl)

head(res)

# Analysis after non-linear transform
#-----------------------------------

# Apply function to transforme data
f <- function(x) log(x^2 + 0.001)

# evaluate covariance of transformed data
ecl_transform <- cov_transform(ecl, f, 100)

# Use linear model to test each feature stored as columns in X
# in data transformed by f()
res2 <- lm_each_eclairs(f(y) ~ v1 + v2, data, X, ecl_transform)

head(res)
```

`lm_eclairs` *Fit linear model after decorrelating*

Description

Fit linear model after applying decorrelation projection to response and predictors.

Usage

```
lm_eclairs(
  formula,
  data,
  ecl,
  subset,
  weights,
```

```
  na.action,
  method = "qr",
  model = TRUE,
  x = FALSE,
  y = FALSE,
  qr = TRUE,
  singular.ok = TRUE,
  contrasts = NULL,
  offset,
  ...
)
```

Arguments

formula	an object of class 'formula' (or one that can be coerced to that class): a symbolic description of the model to be fitted.
data	a matrix or data.frame containing the variables in the model
ecl	estimate of covariance/correlation matrix from eclairs storing U, d_1^2, λ and ν
subset	same as for lm
weights	same as for lm
na.action	same as for lm
method	same as for lm
model	same as for lm
x	same as for lm
y	same as for lm
qr	same as for lm
singular.ok	same as for lm
contrasts	same as for lm
offset	same as for lm
...	same as for lm

Details

This function fit a linear regression to the transformed response, and transformed design matrix. Note that the design matrix, not just the data.frame of variables is transformed so that 1) factors are transformed and 2) the intercept term is transformed.

Value

Object of class lm returned by function lm

Examples

```
library(Rfast)

n <- 800 # number of samples
p <- 200 # number of features

# create correlation matrix
Sigma <- autocorr.mat(p, .9)

# draw data from correlation matrix Sigma
Y <- rmvnorm(n, rep(0, p), sigma = Sigma * 5.1, seed = 1)

# eclairs decomposition
ecl <- eclairs(Y)

# simulate covariates
data <- data.frame(matrnorm(p, 2, seed = 1))
colnames(data) <- paste0("v", 1:2)

# simulate response
y <- rnorm(p)

# fit linear model on transformed data
lm_eclairs(y ~ v1 + v2, data, ecl)
```

`logDet` *Evaluate the log determinant*

Description

Evaluate the log determinant of the matrix

Usage

```
logDet(ecl, alpha = 1)
```

Arguments

`ecl`	estimate of covariance/correlation matrix from `eclairs()` storing U, d_1^2, λ and ν
`alpha`	exponent to be applied to eigen-values

Value

log determinant

Examples

```
library(Rfast)

n <- 800 # number of samples
p <- 200 # number of features

# create correlation matrix
Sigma <- autocorr.mat(p, .9)

# draw data from correlation matrix Sigma
Y <- rmvnorm(n, rep(0, p), sigma = Sigma * 5.1, seed = 1)
rownames(Y) <- paste0("sample_", seq(n))
colnames(Y) <- paste0("gene_", seq(p))

# eclairs decomposition
ecl <- eclairs(Y)

logDet(ecl)
```

---

mahalanobisDistance *Mahalanobis Distance*

Description

Mahalanobis Distance using `eclairs()` decomposition

Usage

```
mahalanobisDistance(ecl, X, lambda, center = FALSE)
```

Arguments

`ecl`	estimate of covariance/correlation matrix from `eclairs` storing U, d_1^2, λ and ν
`X`	data matrix
`lambda`	specify lambda and override value from 'ecl'
`center`	logical: should columns be centered internally

Details

Evaluate quadratic form $(X-\mu)^T\Sigma^{-1}(X-\mu)$ where covariance is estimated from finite sample

Value

array of distances

Examples

```
library(Rfast)

n <- 800 # number of samples
p <- 200 # number of features

# create correlation matrix
Sigma <- autocorr.mat(p, .9)

# draw data from correlation matrix Sigma
Y <- rmvnorm(n, rep(0, p), sigma = Sigma * 5.1, seed = 1)

# eclairs decomposition
ecl <- eclairs(Y)

# Mahalanobis distance after mean centering
Y_center <- scale(Y, scale = FALSE)
mu <- colMeans(Y)

# Standard R method
a <- mahalanobis(Y, mu, cov = cov(Y))

# distance using eclairs decomposition, no shrinage
b <- mahalanobisDistance(ecl, Y_center, lambda = 0)
range(a - b)

# with shrinkage
d <- mahalanobisDistance(ecl, Y_center)

# centering internally
e <- mahalanobisDistance(ecl, Y, center = TRUE)
range(d - e)
#
```

`mult_eclairs` *Multiply by eclairs matrix*

Description

Multiply by eclairs matrix using special structure to achieve linear instead of cubic time complexity.

Usage

```
mult_eclairs(X, U1, dSq1, lambda, nu, alpha, sigma, transpose = FALSE)
```

Arguments

X	matrix to be transformed so *columns* are independent
U1	orthonormal matrix with k columns representing the low rank component

dSq1	eigen values so that $U_1 diag(d_1^2) U_1^T$ is the low rank component
lambda	shrinkage parameter for the convex combination.
nu	diagonal value of target matrix in shrinkage
alpha	exponent to be evaluated
sigma	standard deviation of each feature
transpose	logical, (default FALSE) indicating if X should be transposed first

Details

Let $\Sigma = U_1 diag(d_1^2) U_1^T * (1 - \lambda) + diag(\nu\lambda, p)$, where λ shrinkage parameter for the convex combination between a low rank matrix and the diagonal matrix with values ν.

Evaluate $X\Sigma^\alpha$ using special structure of the eclairs decomposition in $O(k^2 p)$ when there are k components in the decomposition.

Value

a matrix product

`optimal_SVHT_coef` *Optimal Hard Threshold for Singular Values*

Description

A function for the calculation of the coefficient determining optimal location of hard threshold for matrix denoising by singular values hard thresholding when noise level is known or unknown. Recreation of MATLAB code by Matan Gavish and David Donoho.

Usage

```
optimal_SVHT_coef(beta, sigma_known = FALSE)
```

Arguments

beta	A single value or a vector that represents aspect ratio m/n of the matrix to be denoised. 0<beta<=1.
sigma_known	A logical value. TRUE if noise level known, FALSE if unknown.

Value

Optimal location of hard threshold, up the median data singular value (sigma unknown) or up to `sigma*sqrt(n)` (sigma known); a vector of the same dimension as `beta`, where `coef[i]` is the coefficient corresponding to `beta[i]`.

References

Gavish, M., & Donoho, D. L. (2014). The optimal hard threshold for singular values is 4/sqrt(3). IEEE Transactions on Information Theory, 60(8), 5040-5053.

plot,eclairs-method *Plot eclairs object*

Description

Plot eclairs object

Usage

```
## S4 method for signature 'eclairs'
plot(x, y, ...)
```

Arguments

x	eclairs object
y	extra argument, not used
...	additional arguments

Value

plot

quadForm *Evaluate quadratic form*

Description

Evaluate quadratic form

Usage

```
quadForm(ecl, A, alpha = -1/2)
```

Arguments

ecl	estimate of covariance/correlation matrix from `eclairs` storing U, d_1^2, λ and ν
A	matrix
alpha	default = -1/2. Exponent of eigen-values

Details

Evaluate quadratic form $A^T \Sigma^{2\alpha} A$

Value

scalar value

Examples

```
library(Rfast)
n <- 800 # number of samples
p <- 200 # number of features

# create correlation matrix
Sigma <- autocorr.mat(p, .9)

# draw data from correlation matrix Sigma
Y <- rmvnorm(n, rep(0, p), sigma = Sigma * 5.1, seed = 1)

# eclairs decomposition
ecl <- eclairs(Y)

# return scalar
quadForm(ecl, Y[1, ])

# return matrix
quadForm(ecl, Y[1:2, ])
```

`reform_decomp` *Recompute eclairs after dropping features*

Description

Recompute eclairs after dropping features

Usage

```
reform_decomp(ecl, k = ecl$k, drop = NULL)
```

Arguments

`ecl`	covariance/correlation matrix as an eclairs object
`k`	the rank of the low rank component
`drop`	array of variable names to drop.

Details

Reform the dataset from the eclairs decomposition, drop features, then recompute the eclairs decomposition. If the original SVD/eigen was truncated, then the reconstruction of the original data will be approximate. Note that the target shrinkage matrix is the same as in `ecl`, so ν is not recomputed from the retained features.

Value

eclairs decomposition for a subset of features

Examples

```
library(Rfast)

n <- 800 # number of samples
p <- 200 # number of features

# create correlation matrix
Sigma <- autocorr.mat(p, .9)

# draw data from correlation matrix Sigma
Y <- rmvnorm(n, rep(0, p), sigma = Sigma * 5.1, seed = 1)
rownames(Y) <- paste0("sample_", seq(n))
colnames(Y) <- paste0("gene_", seq(p))

# Correlation
#------------

# eclairs decomposition
Sigma.eclairs <- eclairs(Y, compute = "correlation")

# features to drop
drop <- paste0("gene_", 1:100)

# Compute SVD on subset of eclairs decomposition
ecl1 <- reform_decomp(Sigma.eclairs, drop = drop)

ecl1
```

`rmvnorm_eclairs` *Draw from multivariate normal and t distributions*

Description

Draw from multivariate normal and t distributions using eclairs decomposition

Usage

```
rmvnorm_eclairs(n, mu, ecl, v = Inf, seed = NULL)
```

Arguments

`n`	sample size
`mu`	mean vector
`ecl`	covariance matrix as an eclairs object
`v`	degrees of freedom, defaults to Inf. If finite, uses a multivariate t distribution
`seed`	If you want the same to be generated again use a seed for the generator, an integer number

Details

Draw from multivariate normal and t distributions using eclairs decomposition. If the (implied) covariance matrix is $p \times p$, the standard approach is $O(p^3)$. Taking advantage of the previously computed eclairs decomposition of rank k, this can be done in $O(pk^2)$.

Value

matrix where rows are samples from multivariate normal or t distribution where columns have covariance specified by `ecl`

Examples

```
library(Rfast)

n <- 800 # number of samples
p <- 200 # number of features

# create correlation matrix
Sigma <- autocorr.mat(p, .9)

# draw data from correlation matrix Sigma
Y <- rmvnorm(n, rep(0, p), sigma = Sigma, seed = 1)

# perform eclairs decomposition
ecl <- eclairs(Y)

# draw from multivariate normal
n <- 10000
mu <- rep(0, ncol(Y))

# using eclairs decomposition
X.draw1 <- rmvnorm_eclairs(n, mu, ecl)

# using full covariance matrix implied by eclairs model
X.draw2 <- rmvnorm(n, mu, getCov(ecl))

# assess difference betweeen covariances from two methods
range(cov(X.draw1) - cov(X.draw2))

# compare covariance to the covariance matrix used to simulated the data
range(cov(X.draw1) - getCov(ecl))
```

---

sv_threshold *Singular value thresholding*

Description

Singular value thresholding evaluates the optimal number of singular values to retain

Usage

```
sv_threshold(n, p, d)
```

Arguments

n	number of samples
p	number of features
d	singular values

Value

Number of singular values to retain

References

Gavish, M., & Donoho, D. L. (2014). The optimal hard threshold for singular values is 4/sqrt(3). IEEE Transactions on Information Theory, 60(8), 5040-5053.

Examples

```
# simulate data

n <- 500
p <- 5000
Y <- Rfast::matrnorm(n, p, seed = 1)

# SVD
dcmp <- svd(Y)

# how many components to retain
sv_threshold(n, p, dcmp$d)

# in this case the data has no structure, so no components are retained
```

---

whiten *Decorrelation projection + eclairs*

Description

Efficient decorrelation projection using eclairs decomposition

Usage

```
whiten(X, k = ncol(X), lambda = NULL)
```

Arguments

X	matrix to be transformed so *columns* are independent
k	the rank of the low rank component
lambda	specify lambda and override value estimated by eclairs()

Value

data rotated and scaled according to the regularized sample covariance of the input data

Examples

```
library(Rfast)

n <- 800 # number of samples
p <- 200 # number of features

# create correlation matrix
Sigma <- autocorr.mat(p, .9)

# draw data from correlation matrix Sigma
Y <- rmvnorm(n, rep(0, p), sigma = Sigma * 5.1, seed = 1)

# eclairs decomposition
ecl <- eclairs(Y)

# whitened Y
Y.transform <- decorrelate(Y, ecl)

# Combine eclairs and decorrelate into one step
Y.transform2 <- whiten(Y)
```

Index