Package 'GFORCE'

June 14, 2018

Type Package

Title Clustering and Inference Procedures for High-Dimensional Latent Variable Models

Version 0.1.2

Description A complete suite of computationally efficient methods for high dimensional clustering and inference problems in G-Latent Models (a type of Latent Variable Gaussian graphical model). The main feature is the FORCE (First-Order, Certifiable, Efficient) clustering algorithm which is a fast solver for a semi-definite programming (SDP) relaxation of the K-means problem. For certain types of graphical models (G-Latent Models), with high probability the algorithm not only finds the optimal clustering, but produces a certificate of having done so. This certificate, however, is model independent and so can also be used to certify data clustering problems. The 'GFORCE' package also contains implementations of inferential procedures for G-Latent graphical models using n-fold cross validation. Also included are native code implementations of other popular clustering methods such as Lloyd's algorithm with kmeans++ initialization and complete linkage hierarchical clustering. The FORCE method is due to Eisenach and Liu (2017) <arxiv:1806.00530>.

License GPL-2

LazyData TRUE

RoxygenNote 6.0.1

Imports MASS,lpSolve,stats

Suggests testthat

NeedsCompilation yes

Author Carson Eisenach [aut, cre]

Maintainer Carson Eisenach <eisenach@princeton.edu>

Repository CRAN

Date/Publication 2018-06-14 19:00:12 UTC

R topics documented:

`gforce.certify` *FORCE optimality certificate.*

Description

Given a proposed integer solution to the K-means SDP relaxation, this function attempts to construct a solution to the dual problem with matching objective value.

Usage

```
gforce.certify(sol, D, eps1 = 0.01, eps2 = 10^-7, Y_T_min = 0.01)
```

Arguments

`sol`	vector of length d. This contains the assignment of variables or points to clusters.
`D`	dxd matrix.
`eps1`	a scalar. It controls the stopping condition for the dual solution search.
`eps2`	a scalar. It controls the infeasibility tolerance for the dual solution to allow for numerical imprecision.
`Y_T_min`	a scalar. THe smallest Y_T that the function can return. Must be greater than 0.

Value

An object with the following components:

`Y_T` a numeric. The value of the variable `Y_T` in the dual solution found.

`Y_a` a d dimensional numeric vector. The value of the variable `Y_a` in the dual solution found.

`feasible` an integer. 1 signifies that `sol` is optimal, 0 otherwise.

References

C. Eisenach and H. Liu. Efficient, Certifiably Optimal High-Dimensional Clustering. *arXiv:1806.00530*, 2018.

Examples

```
K <- 5
n <- 50
d <- 50
dat <- gforce.generator(K,d,n,3,graph='scalefree')
sig_hat <- (1/n)*t(dat$X)%*%dat$X
gam_hat <- gforce.Gamma(dat$X)
D <- diag(gam_hat) - sig_hat
dual_cert <- gforce.certify(dat$group_assignments,D)
```

---

gforce.certify_adapt *FORCE optimality certificate (K is unknown).*

Description

Given a proposed integer solution to the adaptive K-means SDP relaxation, this function attempts to construct a solution to the dual problem with matching objective value.

Usage

```
gforce.certify_adapt(sol, D, eps1 = 10^-7)
```

Arguments

`sol`	vector of length d. This contains the assignment of variables or points to clusters.
`D`	dxd matrix.
`eps1`	a scalar. It controls the infeasibility tolerance for the dual solution to allow for numerical imprecision.

Value

An object with the following components:

`Y_a` a d dimensional numeric vector. The value of the variable `Y_a` in the dual solution found.

`feasible` an integer. 1 signifies that `sol` is optimal, 0 otherwise.

References

C. Eisenach and H. Liu. Efficient, Certifiably Optimal High-Dimensional Clustering. *arXiv:1806.00530*, 2018.

Examples

```
K <- 5
n <- 50
d <- 50
dat <- gforce.generator(K,d,n,3,graph='scalefree')
sig_hat <- (1/n)*t(dat$X)%*%dat$X
gam_hat <- gforce.Gamma(dat$X)
D <- diag(gam_hat) - sig_hat
dual_cert <- gforce.certify_adapt(dat$group_assignments,D)
```

`gforce.clust2mat` *Convert a clustering or grouping to partnership matrix.*

Description

Takes a clustering G and constructs $B(G)$.

Usage

```
gforce.clust2mat(clusters)
```

Arguments

`clusters`	length d vector. Assigns each variable or data point to a cluster. Cluster names can be numbers or strings.

Value

a dxd numeric array that contains the partnership matrix corresponding to `clusters`.

Examples

```
clusters <- c(1,1,1,2,2,2,3,3)
B_clust <- gforce.clust2mat(clusters)
```

gforce.confint2test *Convert confidence intervals to equivalent test statistics.*

Description

Can convert a 4D array encoding the confidence intervals for a precision matrix to standard normal test-statistics.

Usage

```
gforce.confint2test(conf_ints, alpha)
```

Arguments

conf_ints	$dxdx3$ array. Each $dxdx1$ slice is a symmetric matrix.
alpha	confidence level level of the confidence intervals.

Value

a dxd symmetric matrix of test statistics.

gforce.defaults *FORCE default tuning parameters.*

Description

Provides the default tuning parameters for gforce.FORCE.

Usage

```
gforce.defaults(d)
```

Arguments

d	dimension of random vector or number of datapoints.

Value

An object with following components

adapt_init_mode a numeric. Indicates which initialization mode to use for gforce.FORCE_adapt.

alpha a numeric. Gives the step size for the projected gradient descent updates.

dual_frequency an integer. Specifies how many gradient updates to perform between searches for a dual certificate.

`duality_gap` a numeric. If the duality gap can be shown to be less than `duality_gap`, the FORCE algorithm terminates.

`early_stop_mode` a numeric. `early_stop_mode == 1` indicates that the algorithm should use an early stopping rule.

`early_stop_lag` an integer. This indicates the number of iterations without sufficient improvement in objective value before early stopping.

`early_stop_eps` a numeric. Threshold for objective value improvement used to determine early stopping.

`eps_obj` a numeric. Specifies the precision required of the optimal solution to the eigenvalue maximization problem.

`finish_pgd` an integer. If `finish_pgd` is 1, then other stopping criteria are ignored and FORCE performs `max_iter` gradient updates.

`initial_mixing` a numeric between 0 and 1. Specifies how to construct the initial strictly feasible solution to the SDP relaxation.

`kmeans_iter` an integer. The number of times to run a K-means solver during each search for an optimal clustering and dual certificate.

`max_iter` an integer. The maximum number of gradient updates to perform.

`primal_only` an integer. `primal_only == 1` indicates that the algorithm should not search for a dual certificate.

`restarts` a vector of integers. This specifies the iterations at which to take the projection of the current iterate and restart the algorithm with that as the initial solution.

`verbose` an integer. Specifies the level of verbosity requested from gforce.FORCE.

Examples

```
opts <- gforce.defaults(20)
```

---

gforce.FDR_control *FDR Control Procedure.*

Description

Performs by default the Benjamini-Yekutieli FDR control procedure. Optionally, the Benjamini-Hochberg thresholding rule can be used instead. As input it takes a symmetric matrix of test statistics with standard normal null distributions. The number of hypotheses tested is $d(d-1)/2$.

Usage

```
gforce.FDR_control(test_stats, alpha, procedure = "BY")
```

Arguments

test_stats	dxd symmetric matrix of test statistics.
alpha	alpha level for the FDR control procedure.
procedure	a string. procedure == 'BY' indicates to use the Benjamini-Yekutieli thresholding rule. procedure == 'BH' indicates to use the Benjamini-Hochberg thresholding rule.

Value

An object with following components

reject_null dxd upper triangular matrix. TRUE entries indicate the null hypothesis should be rejected.

R_tau_hat an integer. Indicates the number of hypotheses rejected.

tau_hat a real number. Indicates a threshold above which the null hypothesis is rejected.

num_hypotheses an integer. Indicates the number of hypotheses tested.

References

Y. Benjamini and Y. Hochberg. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal of the Royal Statistical Society*, 1995.

Y. Benjamini and D. Yekutieli. The Control of the False Discovery Rate in Multiple Testing Under Dependency. *The Annals of Statistics*, 2001.

---

gforce.FORCE *FORCE K-means solver.*

Description

Solves the Peng-Wei K-means SDP Relaxation using the FORCE algorithm.

Usage

```
gforce.FORCE(D, K, force_opts = NULL, D_Kmeans = NULL, X0 = NULL,
  E = NULL, R_only = FALSE)
```

Arguments

D	a matrix D as defined above.
K	number of clusters.
force_opts	tuning parameters. NULL signifies defaults will be used.
D_Kmeans	matrix to be used for initial integer solution. NULL signifies that D will be used.
X0	initial iterate. NULL signifies that it will be generated randomly from D_Kmeans. If supplied, E must be supplied as well.

E	strictly feasible solutions. NULL signifies that it will be generated randomly. If supplied, X0 must be supplied as well.
R_only	logical expression. If R_only == FALSE, then the included native code implementation will be used. Otherwise, an R implementation is used.

Value

An object with following components

Z_T Final iterate of the projected gradient descent algorithm run on the smoothed eigenvalue problem.

B_Z_T Projection of Z_T to the border of the positive semi-definite cone.

B_Z_T_opt_val Objective value of the K-means SDP relaxation at B_Z_T.

Z_best Iterate with best objective value found during projected gradient descent on the smoothed eigenvalue problem.

B_Z_best Projection of Z_best to the border of the positive semi-definite cone.

B_Z_best_opt_val Objective value of the K-means SDP relaxation at B_Z_T.

km_best Best clustering in terms of objective value of the SDP relaxation. This is found by running Lloyd's algorithm on the rows of D_kmeans_matrix.

B_km Partnership matrix corresponding to km_best.

km_opt_val Objective value of the K-means SDP relaxation at B_km.

km_best_time Time elapsed (in seconds) until km_best was found.

km_iter_best Number of times a K-means algorithm was run before km_best was found.

km_iter_total Total number of calls to a K-means solver (such as Lloyd's algorithm).

dual_certified 1 if a dual certificate was found, and 0 otherwise.

dual_certified_grad_iter Number of gradient updates performed before a dual certificate was found.

dual_certified_time Time elapsed (in seconds) until dual certificate was found for B_km.

grad_iter_best Gradient iteration where Z_best was computed.

grad_iter_best_time Time elapsed (in seconds) when the update grad_iter_best was performed.

total_time Total time elapsed (in seconds) during call to gforce.FORCE.

References

C. Eisenach and H. Liu. Efficient, Certifiably Optimal High-Dimensional Clustering. *arXiv:1806.00530*, 2018.

J. Peng and Y. Wei. Approximating K-means-type Clustering via Semidefinite Programming. *SIAM Journal on Optimization*, 2007.

J. Renegar. Efficient first-order methods for linear programming and semidefinite programming. *arXiv:1409.5832*, 2014.

See Also

gforce.defaults

Examples

```
K <- 5
n <- 50
d <- 50
dat <- gforce.generator(K,d,n,3,graph='scalefree')
sig_hat <- (1/n)*t(dat$X)%*%dat$X
gam_hat <- gforce.Gamma(dat$X)
D <- diag(gam_hat) - sig_hat
res <- gforce.FORCE(D,K)
```

---

gforce.FORCE_adapt *FORCE K-means solver.*

Description

Solves a K-means SDP Relaxation using the FORCE algorithm when K is unknown.

Usage

```
gforce.FORCE_adapt(D, force_opts = NULL, D_Kmeans = NULL, X0 = NULL)
```

Arguments

D	a matrix D as defined above.
force_opts	tuning parameters. NULL signifies defaults will be used.
D_Kmeans	matrix to be used for initial integer solution. NULL signifies that D will be used.
X0	initial iterate. NULL signifies that it will be generated randomly from D_Kmeans. If supplied, E must be supplied as well.

Value

An object with following components

Z_T Final iterate of the projected gradient descent algorithm run on the smoothed eigenvalue problem.

B_Z_T Projection of Z_T to the border of the positive semi-definite cone.

B_Z_T_opt_val Objective value of the K-means SDP relaxation at B_Z_T.

Z_best Iterate with best objective value found during projected gradient descent on the smoothed eigenvalue problem.

B_Z_best Projection of Z_best to the border of the positive semi-definite cone.

B_Z_best_opt_val Objective value of the K-means SDP relaxation at B_Z_T.

`km_best` Best clustering in terms of objective value of the SDP relaxation. This is found by running a complete linkage clustering algorithm on the rows of the iterate `Z_t`.

`B_km` Partnership matrix corresponding to `km_best`.

`km_opt_val` Objective value of the K-means SDP relaxation at `B_km`.

`km_best_time` Time elapsed (in seconds) until `km_best` was found.

`dual_certified` 1 if a dual certificate was found, and 0 otherwise.

`dual_certified_grad_iter` Number of gradient updates performed before a dual certificate was found.

`dual_certified_time` Time elapsed (in seconds) until dual certificate was found for `B_km`.

`grad_iter_best` Gradient iteration where `Z_best` was computed.

`grad_iter_best_time` Time elapsed (in seconds) when the update `grad_iter_best` was performed.

`total_time` Total time elapsed (in seconds) during call to `gforce.FORCE`.

References

C. Eisenach and H. Liu. Efficient, Certifiably Optimal High-Dimensional Clustering. *arXiv:1806.00530*, 2018.

J. Peng and Y. Wei. Approximating K-means-type Clustering via Semidefinite Programming. *SIAM Journal on Optimization*, 2007.

J. Renegar. Efficient first-order methods for linear programming and semidefinite programming. *arXiv:1409.5832*, 2014.

See Also

`gforce.defaults`

Examples

```
K <- 5
n <- 50
d <- 50
dat <- gforce.generator(K,d,n,3,graph='scalefree')
sig_hat <- (1/n)*t(dat$X)%*%dat$X
gam_hat <- gforce.Gamma(dat$X)
D <- diag(gam_hat) - sig_hat
res <- gforce.FORCE_adapt(D)
```

gforce.Gamma *Estimates* Γ *for the PECOK SDP.*

Description

In particular, it returns in diagonal form the estimator Γ used to construct the PECOK penalized covariance estimator.

Usage

```
gforce.Gamma(X, par = FALSE, fast_estimator = FALSE, R_only = FALSE)
```

Arguments

X	nxd matrix. Each row represents a realization of a d dimensional random vector.
par	logical expression. If `par == TRUE`, then a multi-threaded version of the function is called. If `par == FALSE`, a single-threaded version is called.
fast_estimator	logical expression. If `fast_estimator == TRUE`, then the alternative estimator for $\hat{\Gamma}$ is used.
R_only	logical expression. If `R_only == TRUE`, then no native code is run. If `fast_estimator != TRUE` this is ignored.

Value

The estimator Γ as a d dimensional numeric vector.

References

F. Bunea, C. Giraud, M. Royer and N. Verzelen. PECOK: a convex optimization approach to variable clustering. *arXiv:1606.05100*, 2016.

Examples

```
dat <- gforce.generator(5,20,20,3)
gam_hat <- gforce.Gamma(dat$X)
```

`gforce.generator` *Data generator.*

Description

Generates n random samples from a G-Latent Variable Model. The caller can specify the graph structure on the latent variables via several parameters. The magnitude of the non-zero entries in the population precision matrix can also be specified. Observed variables are assigned uniformly at random to K groups with minimum size m.

Usage

```
gforce.generator(K, d, n, m, graph = "DeltaC", num_hubs = NULL,
  band_size = 3, cov_gap_mult = 1, error_base = 0.25, error_add = 0.25,
  corr_value = 0.3, normalize = TRUE)
```

Arguments

`K`	number of clusters.
`d`	dimension of the observed random vector.
`n`	number of samples.
`m`	minimal group size.
`graph`	latent graph structure. Can be 'scalefree', 'hub', 'band', 'identity' or 'DeltaC'.
`num_hubs`	number of hubs in the latent graph. Ignored unless `graph == 'hub'`.
`band_size`	size of bands in the latent graph. Ignored unless `graph=='band'`.
`cov_gap_mult`	scales the size of ΔC. Ignored unless `graph == 'DeltaC'`.
`error_base`	minimum variance of errors.
`error_add`	size of range of possible variances for errors.
`corr_value`	size of off diagonal entries in latent precision matrix.
`normalize`	logical. If `normalize == TRUE`, the covariance matrix for the latent graph will be normalized so that it is also a correlation matrix.

Value

An S3 object with the slots Z,E,X,group_assignments,CStar,Theta_Star

Examples

```
dat <- gforce.generator(5,20,20,3)
dat <- gforce.generator(10,100,100,3,graph='hub',num_hubs=2)
dat <- gforce.generator(10,100,100,3,graph='band',band_size=3)
```

gforce.glatent_confints *Confidence Intervals for Estimation in G-Latent Models.*

Description

Estimate the precision matrix and construct confidence intervals for the latent and group averages graphs. The user can either provide an estimate of the relevant covariance matrix or the data and cluster assignments. If cross validation is selected, the data and clusters must be provided.

Usage

```
gforce.glatent_confints(C_hat = NULL, X_vals = NULL, clusters = NULL,
  alpha = 0.05, graph = "latent", variance_estimator = "simple",
  use_cv = FALSE, cv_opts = NULL, lambda1 = NULL, lambda2 = NULL)
```

Arguments

C_hat	a dxd matrix. This is the estimated latent or group averages covariance matrix.
X_vals	a nxd matrix. This is a matrix where each row is a sample from the model.
clusters	a d dimensional integer vector. Contains the assignment of variables to groups.
alpha	a numeric. alpha is the confidence level.
graph	a string. It can either have value 'latent' or 'averages'.
variance_estimator	a string. It can either have value 'simple' or 'exact'.
use_cv	logical expression. Indicates whether or not to use cross validation.
cv_opts	an object. Contains options for cross validation procedure.
lambda1	a numeric. Parameter for the first optimization problem.
lambda2	a numeric. Parameter for the second optimization problem.

Value

a $dxdx3$ array. The first dxd slice is the lower confidence bound, the second the point estimate, and the third the upper confidence bound.

References

C. Eisenach, F. Bunea, Y. Ning, and C. Dinicu. Efficient, High-Dimensional Inference for Cluster-Based Graphical Models. *Manuscript submitted for publication*, 2018.

See Also

gforce.glatent_confints.cv_defaults

Examples

```
K <- 5
n <- 50
d <- 50
dat <- gforce.generator(K,d,n,3,graph='scalefree')
th_tilde <- gforce.glatent_confints(X_vals = dat$X,clusters = dat$group_assignments,
                                    use_cv = TRUE,graph='latent')
```

`gforce.glatent_confints.cv_defaults` *Default Cross Validation Options for Confidence Intervals.*

Description

Default Cross Validation Options for Confidence Intervals.

Usage

```
gforce.glatent_confints.cv_defaults()
```

Value

An object with following components

`grid_density` an integer. Indicates the number of values in the search grid.

`lambda1_min_exp` a numeric. Minimum exponent for values of λ_1 to search over.

`lambda1_max_exp` a numeric. Maximum exponent for values of λ_1 to search over.

`lambda2_min_exp` a numeric. Minimum exponent for values of λ_2 to search over.

`lambda2_max_exp` a numeric. Maximum exponent for values of λ_2 to search over.

`num_folds` an integer. The number of cross-validation folds to use.

See Also

`gforce.glatent_confints`

`gforce.hclust` *Hierarchical Clustering with Estimation of K.*

Description

Clusters n points of dimension m using a complete linkage algorithm and estimates K.

Usage

```
gforce.hclust(X = NULL, dists = NULL, R_only = FALSE)
```

Arguments

`X`	nxm matrix. Each row is treated as a point in R^m.
`dists`	nxn symmetric matrix. This encodes the distances between the n points.
`R_only`	logical expression. If `R_only == FALSE`, then the included native code implementation will be used. Otherwise, an R implementation is used.

Value

Returns an object with the components:

`K` an estimate of the number of clusters.

`clusters` a n dimensional integer vector. Entry i to the cluster assignment of the data point given by row i of `X`.

`MSE` a n dimensional vector of the mean squared errors of each choice of K.

References

D. Defays. An efficient algorithm for a complete link method. *The Computer Journal*, 1977.

Examples

```
m <- 10
n <- 10
X <- matrix(MASS::mvrnorm(m*n,rep(0,m*n),diag(m*n)), nrow = n)
hc_res <- gforce.hclust(X=X)
```

`gforce.hclust.agg2clust` *Hierarchical Clustering – Convert Agglomeration to clustering.*

Description

Hierarchical Clustering – Convert Agglomeration to clustering.

Usage

```
gforce.hclust.agg2clust(hc, K)
```

Arguments

`hc`	an object. This encodes the order of agglomeration.
`K`	an integer. `K` is the number of clusters

`gforce.hclust.agglomerate` *Hierarchical Clustering Agglomeration.*

Description

Hierarchical Clustering Agglomeration.

Usage

```
gforce.hclust.agglomerate(X = NULL, dists = NULL)
```

Arguments

`X`	nxm matrix. Each row is treated as a point in R^m.
`dists`	nxn symmetric matrix. This encodes the distances between the n points.

`gforce.kmeans` *K-means Clustering.*

Description

Solves the K-means problem using kmeans++ for the initialization and then runs Lloyd's algorithm.

Usage

```
gforce.kmeans(X, K, R_only = FALSE)
```

Arguments

`X`	nxm matrix. Each row is treated as a point in R^m.
`K`	integer. The number of clusters to group the data into.
`R_only`	logical expression. If `R_only == FALSE`, then the included native code implementation will be used. Otherwise, an R implementation is used.

Value

Returns an object with the components:

`clusters` a n dimensional integer vector. Entry i to the cluster assignment of the data point given by row i of `X`.

`centers` a Kxm numeric matrix. Row i corresponds to the center of cluster i.

`num_iters` an integer. Number of iterations of Lloyd's Algorithm.

`time` a numeric. Runtime of Lloyd's Algorithm.

References

S.P. Lloyd. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 1982.

D. Arthur and S. Vassilvitskii. k-means++: The Advantages of Careful Seeding. *SODA*, 2007.

Examples

```
m <- 10
n <- 10
X <- matrix(MASS::mvrnorm(m*n,rep(0,m*n),diag(m*n)), nrow = n)
km_res <- gforce.kmeans(X,3)
```

`gforce.kmeans_SDP_matrix` *K-means SDP Matrices.*

Description

Constructs the SDP constraint matrices so they can be passed to a black-box solver.

Usage

```
gforce.kmeans_SDP_matrix(D, K)
```

Arguments

`D`	$d \times d$ numeric matrix.
`K`	number of clusters.

`gforce.metrics` *Evaluates the correctness of a clustering solution.*

Description

This can be used for either data or variable clustering. Cluster names can be either strings or numbers. Arbitrary data types cannot be used.

Usage

```
gforce.metrics(true_clust, est_clust, method = "purity")
```

Arguments

`true_clust`	length d vector of cluster assignments. This represents the true, or reference, clustering.
`est_clust`	length d vector of cluster assignments. This represents the estimated clustering.
`method`	the method used to evaluate the quality of the clustering solution `est_clust`. The three options are `'purity'`, `'perfect'`, `'misclassified-points'`.

Value

Returns a numeric that represents the value of the chosen metric on the two clusterings `true_clust` and `est_clust`.

Examples

```
clust1 <- c(1,1,1,2,2,2,3,3,3)
clust2 <- c(1,1,2,1,2,2,3,3,3)
gforce.metrics(clust1,clust2,method='purity')
```

gforce.PECOK *Solve PECOK with FORCE.*

Description

Uses the FORCE algorithm to solve the PECOK SDP.

Usage

```
gforce.PECOK(K, X = NULL, D = NULL, sigma_hat = NULL, force_opts = NULL,
  X0 = NULL, E = NULL, gamma_par = FALSE)
```

Arguments

K	number of clusters.
X	nxd matrix. Either this or D must be specified.
D	dxd matrix. Either this or X must be specified.
sigma_hat	dxd matrix. If D is specified, this argument should be the estimated covariance matrix. It is not strictly necessary to provide it, but it should be for optimal performance. If X is specified, it will be ignored.
force_opts	tuning parameters. NULL signifies defaults will be used.
X0	initial iterate. NULL signifies that it will be generated randomly from D_Kmeans. If supplied, E must be supplied as well.
E	strictly feasible solutions. NULL signifies that it will be generated randomly. If supplied, X0 must be supplied as well.
gamma_par	logical expression. If gamma_par==TRUE, then if Γ is computed, a multi-threaded method is called, otherwise a single-threaded method is called.

References

C. Eisenach and H. Liu. Efficient, Certifiably Optimal High-Dimensional Clustering. *arXiv:1806.00530*, 2018.

J. Peng and Y. Wei. Approximating K-means-type Clustering via Semidefinite Programming. *SIAM Journal on Optimization*, 2007.

F. Bunea, C. Giraud, M. Royer and N. Verzelen. PECOK: a convex optimization approach to variable clustering. *arXiv:1606.05100*, 2016.

See Also

gforce.defaults

`gforce.PECOK_adapt` *Solve PECOK Adaptive SDP with FORCE.*

Description

Uses the FORCE algorithm to solve the PECOK SDP when K is unknown.

Usage

```
gforce.PECOK_adapt(X = NULL, D = NULL, sigma_hat = NULL,
  force_opts = NULL, X0 = NULL, gamma_par = FALSE)
```

Arguments

`X`	nxd matrix. Either this or `D` must be specified.
`D`	dxd matrix. Either this or `X` must be specified.
`sigma_hat`	dxd matrix. If `D` is specified, this argument should be the estimated covariance matrix. It is not strictly necessary to provide it, but it should be for optimal performance. If `X` is specified, it will be ignored.
`force_opts`	tuning parameters. `NULL` signifies defaults will be used.
`X0`	initial iterate. `NULL` signifies that it will be generated randomly from `D_Kmeans`. If supplied, `E` must be supplied as well.
`gamma_par`	logical expression. If `gamma_par==TRUE`, then if Γ is computed, a multi-threaded method is called, otherwise a single-threaded method is called.

References

C. Eisenach and H. Liu. Efficient, Certifiably Optimal High-Dimensional Clustering. *arXiv:1806.00530*, 2018.

J. Peng and Y. Wei. Approximating K-means-type Clustering via Semidefinite Programming. *SIAM Journal on Optimization*, 2007.

F. Bunea, C. Giraud, M. Royer and N. Verzelen. PECOK: a convex optimization approach to variable clustering. *arXiv:1606.05100*, 2016.

See Also

`gforce.defaults`

`gforce.scio` *SCIO Estimator.*

Description

Estimate the precision matrix with the SCIO estimator. This algorithm is due to Liu and Luo (2012). The implementation follows the active set strategy also used in the SCIO package.

Usage

```
gforce.scio(C, lambda, k = NULL, eps = 10^-6, max_iter = 10000,
  R_only = FALSE)
```

Arguments

`C`	a dxd numeric matrix. This is the matrix of which we seek the inverse.
`lambda`	a numeric. This is the sparsity penalty parameter.
`k`	an integer. Indicates the column of the inverse to compute.
`eps`	a numeric. A threshold used as a stopping criterion.
`max_iter`	an integer. The max number of iterations of the SCIO algorithm.
`R_only`	logical expression. If `R_only == FALSE`, then the included native code implementation will be used. Otherwise, an R implementation is used.

Value

a d dimensional numeric vector that is the kth column of the inverse of `C`.

References

T. Cai, W. Liu and X. Luo. A constrained l1 minimization approach to sparse precision matrix estimation. *Journal of the American Statistical Association*, 2011.

Examples

```
C <- diag(5)
theta_1 <- gforce.scio(C,0.01,1)
```

Index