R documentation

of 'AUC.Rd' etc.

May 17, 2019

AUC *Calculates the AUC of an ROC curve.*

Description

Calculates the area under an ROC curve (AUC).

Usage

```
AUC(tpfp)
```

Arguments

tpfp	A matrix with two columns, the true positive and the false positive rates.

Value

A number between 0 and 1, the area under the curve (AUC).

Examples

```
n <- 40
p <- 50
mu <- rep(0, p)
tol <- 1e-8
K <- cov_cons(mode="sub", p=p, seed=1, spars=0.2, eig=0.1, subgraphs=10)
true_edges <- which(abs(K) > tol & diag(p) == 0)
dm <- 1 + (1-1/(1+4*exp(1)*max(6*log(p)/n, sqrt(6*log(p)/n))))
set.seed(1)
x <- tmvtnorm::rtmvnorm(n, mean = mu, sigma = solve(K),
       lower = rep(0, p), upper = rep(Inf, p), algorithm = "gibbs",
       burn.in.samples = 100, thinning = 10)
est <- estimate(x, "trun_gaussian", elts=NULL, centered=TRUE,
         symmetric="symmetric", lambda_length=100, mode="min_pow",
         param1=1, param2=3, diagonal_multiplier=dm)
# Apply tp_fp to each estimated edges set for each lambda
TP_FP <- t(sapply(est$edgess, function(edges){tp_fp(edges, true_edges, p)}))
par(mfrow=c(1,1), mar=c(5,5,5,5))
```

```
auc <- AUC(TP_FP)
plot(c(), c(),  ylim=c(0,1), xlim=c(0,1), cex.lab=1,
  main="ROC curve, AUC",round(auc,4)), xlab="False Positives",
  ylab="True Positives")
points(TP_FP[,2], TP_FP[,1], type="l")
points(c(0,1), c(0,1), type = "l", lty = 2)
```

---

avgrocs *Takes the vertical average of ROC curves.*

Description

Takes the vertical average of ROC curves using algorithm 3 from Fawcett (2006). The resulting ROC curve preserves the average AUC.

Usage

```
avgrocs(rocs, num_true_edges, p)
```

Arguments

rocs	A list of ROC curves, each of which is a matrix with two columns corresponding to the true positive and false positive rates, respectively.
num_true_edges	A positive integer, the number of true edges
p	A positive integer, the dimension

Value

The averaged ROC curve, a matrix with 2 columns and (p^2-p-num_true_edges+1) rows.

References

Tom Fawcett. An introduction to ROC analysis. Pattern Recognition Letters, 27(8):861–874, 2006.

Examples

```
n <- 40
p <- 50
mu <- rep(0, p)
tol <- 1e-8
K <- cov_cons(mode="sub", p=p, seed=1, spars=0.2, eig=0.1, subgraphs=10)
true_edges <- which(abs(K) > tol & diag(p) == 0)
dm <- 1 + (1-1/(1+4*exp(1)*max(6*log(p)/n, sqrt(6*log(p)/n))))
ROCs <- list()
par(mfrow=c(2,2), mar=c(5,5,5,5))
for (i in 1:3){
  set.seed(i)
  x <- tmvtnorm::rtmvnorm(n, mean = mu, sigma = solve(K),
         lower = rep(0, p), upper = rep(Inf, p), algorithm = "gibbs",
         burn.in.samples = 100, thinning = 10)
  est <- estimate(x, "trun_gaussian", elts=NULL, centered=TRUE,
           symmetric="symmetric", lambda_length=100, mode="min_pow",
           param1=1, param2=3, diag=dm)
```

```
  # Apply tp_fp to each estimated edges set for each lambda
  TP_FP <- t(sapply(est$edgess, function(edges){tp_fp(edges, true_edges, p)}))
  ROCs[[i]] <- TP_FP
  plot(c(), c(),  ylim=c(0,1), xlim=c(0,1), cex.lab=1,
    main=paste("ROC, trial ",i,", AUC ",round(AUC(TP_FP),4),sep=""),
    xlab="False Positives", ylab="True Positives")
  points(TP_FP[,2], TP_FP[,1], type="l")
  points(c(0,1), c(0,1), type = "l", lty = 2)
}
average_ROC <- avgrocs(ROCs, length(true_edges), p)
plot(c(), c(),  ylim=c(0,1), xlim=c(0,1), cex.lab=1,
  main=paste("Average ROC, AUC",round(AUC(average_ROC),4)),
  xlab="False Positives", ylab="True Positives")
points(average_ROC[,2], average_ROC[,1], type="l")
points(c(0,1), c(0,1), type = "l", lty = 2)
```

`compare_two_results` *Compares two lists returned from estimate().*

Description

Compares two lists returned from `estimate()`.

Usage

```
compare_two_results(res, res2)
```

Arguments

`res`	A res list returned from `estimate()`.
`res2`	A res list returned from `estimate()`.

Value

A list of numbers all of which should be close to 0 if `res` and `res2` are expected to be the same.

`compare_two_sub_results` *Compares two lists returned from get_results().*

Description

Compares two lists returned from `get_results()`.

Usage

```
compare_two_sub_results(res, res2)
```

Arguments

`res`	A res list returned from `get_results()`.
`res2`	A res list returned from `get_results()`.

Value

A list of numbers all of which should be close to 0 if `res` and `res2` are expected to be the same.

`cov_cons` *Random generator of inverse covariance matrices.*

Description

Random generator of inverse covariance matrices.

Usage

```
cov_cons(mode, p, seed = NULL, spars = 1, eig = 0.1, subgraphs = 1)
```

Arguments

`mode`	A string, see details.
`p`	A positive integer >= 2, the dimension.
`seed`	A number, the seed for the generator. Ignored if `NULL` or `mode == "band"` or `mode == "chain"`.
`spars`	A number, see details. Ignored if `mode == "chain"`. Default to 1.
`eig`	A positive number, the minimum eigenvalue of the returned matrix. Default to 0.1.
`subgraphs`	A positive integer, the number of subgraphs for the `"sub"` mode. Note that `p` must be divisible by `subgraphs`.

Details

The function generates an inverse covariance matrix according to the `mode` argument as follows. The diagonal entries of the matrix are set to the same value such that the minimum eigenvalue of the returned matrix is equal to `eig`.

Takes the `Q` matrix from the `QR` decomposition of a `p` by `p` random matrix with independent $Normal(0, 1)$ entries, and calculates $Q' diag(ev) Q$. Randomly zeros out its upper triangular entries using independent uniform Bernoulli(`spars`) variables, and then symmetrizes the matrix using the upper triangular part.

"randsub" Constructs a block diagonal matrix with `subgraphs` disconnected subgraphs with equal number of nodes. In each subgraph, takes each entry independently from $Uniform(0.5, 1)$, and randomly zeros out its upper triangular entries using independent uniform Bernoulli(`spars`) variables, and finally symmetrizes the matrix using the upper triangular part.

"er" Constructs an Erdős–Rényi game with probability `spars`, and sets the edges to independent $Uniform(0.5, 1)$ variables, and finally symmetrizes the matrix using the lower triangular entries.

"band" Constructs a banded matrix so that the `(i,j)`-th matrix is nonzero if and only if $|i-j| <= spars$, and is equal to $1 - |i - j|/(spars + 1)$ if $i! = j$.

"chain" A chain graph, where the `(i,j)`-th matrix is nonzero if and only if $|i - j| <= 1$, and is equal to 0.5 if $|i - j| == 1$. A special case of the `"band"` construction with `spars` equal to 1.

Value

A p by p inverse covariance matrix. See details.

References

Lina Lin, Mathias Drton, and Ali Shojaie. Estimation of high-dimensional graphical models using regularized score matching. Electron. J. Stat., 10(1):806–854, 2016. \insertRefRpack:bibtexRdpack \insertReflin16genscore

Examples

```
p <- 100
K1 <- cov_cons("random", p, seed = 1, spars = 0.05, eig = 0.1)
K2 <- cov_cons("sub", p, seed = 2, spars = 0.5, eig = 0.1, subgraphs=10)
K3 <- cov_cons("er", p, seed = 3, spars = 0.05, eig = 0.1)
K4 <- cov_cons("band", p, spars = 2, eig = 0.1)
K5 <- cov_cons("chain", p, eig = 0.1)
```

---

crbound_mu *The Cramér-Rao lower bound (times n) for estimating the mean parameter from a univariate truncated normal sample with known variance parameter.*

Description

The Cramér-Rao lower bound (times n) on the variance for estimating the mean parameter mu from a univariate truncated normal sample, assuming the true variance parameter sigmasq is known.

Usage

```
crbound_mu(mu, sigmasq)
```

Arguments

mu	The mean parameter.
sigmasq	The variance parameter.

Details

The Cramér-Rao lower bound in this case is defined as $\sigma^4/var(X - \mu)$.

Value

A number, the Cramér-Rao lower bound.

`crbound_sigma` *The Cramér-Rao lower bound (times* `n`*) for estimating the variance parameter from a univariate truncated normal sample with known mean parameter.*

Description

The Cramér-Rao lower bound (times `n`) on the variance for estimating the variance parameter `sigmasq` from a univariate truncated normal sample, assuming the true mean parameter `mu` is known.

Usage

```
crbound_sigma(mu, sigmasq)
```

Arguments

`mu`	The mean parameter.
`sigmasq`	The variance parameter.

Details

The Cramér-Rao lower bound in this case is defined as $4\sigma^8/var((X-\mu)^2)$.

Value

A number, the Cramér-Rao lower bound .

`diff_lists` *Computes the sum of absolute differences between two lists.*

Description

Computes the sum of absolute differences between two lists using diff_vecs().

Usage

```
diff_lists(l1, l2, name = NULL)
```

Arguments

`l1`	A list.
`l2`	A list.
`name`	A string, default to `NULL`. If not `NULL`, computes the differences in the `l1[[name]]` and `l2[[name]]`.

Value

Returns the sum of absolute differences between `l1` and `l2` if `name` is `NULL`, or that between `l1[[name]]` and `l2[[name]]` otherwise. If `name` is not `NULL` and if `name` is in exactly one of `l1` and `l2`, returns `Inf`; if `name` is in neither, returns `NA`. Exception: Returns a positive integer if the two elements compared hold `NA`, `NULL` or `Inf` values in different places.

diff_vecs — *Computes the sum of absolute differences in the finite non-NA/NULL elements between two vectors.*

Description

Computes the sum of absolute differences in the finite non-NA/NULL elements between two vectors.

Usage

```
diff_vecs(l1, l2, relative = FALSE)
```

Arguments

l1	A vector.
l2	A vector.
relative	A boolean, default to FALSE. If TRUE, returns the relative difference (sum of absolute differences divided by the elementwise minimum between l1 and l2).

Value

The sum of (relative) absolute differences in l1 and l2, or a positive integer if two vectors differ in length or hold NA, NULL or Inf values in different places.

eBIC — *eBIC score with or without refitting.*

Description

Calculates the eBIC score both with and without refitting an unpenalized model restricted to the estimated support.

Usage

```
eBIC(res, elts, BIC_refit = TRUE, gammas = c(0, 0.5, 1))
```

Arguments

res	A list of results returned by get_results().
elts	A list, elements necessary for calculations returned by get_elts().
BIC_refit	A boolean, whether to get the BIC scores by refitting an unpenalized model restricted to the estimated edges, with lambda1=0, lambda2=0 and diagonal_multiplier=1. Default to TRUE.
gammas	Optional. A number of a vector of numbers. The γ parameter in eBIC. Default to c(0,0.5,1).

Value

A vector of length `2*length(gammas)`. The first `length(gammas)` numbers are the eBIC scores without refitting for each gamma value, and the rest are those with refitting if `BIC_refit == TRUE`, or `Inf` if `BIC_refit == FALSE`.

Examples

```
if (!requireNamespace("tmvtnorm", quietly = TRUE)){
  stop("Please install package \"tmvtnorm\" first.", call. = FALSE)
}
require(tmvtnorm)
n <- 50
p <- 30
h_hp <- get_h_hp("min_pow", 1, 3)
mu <- rep(0, p)
K <- diag(p)
dm <- 1 + (1-1/(1+4*exp(1)*max(6*log(p)/n, sqrt(6*log(p)/n))))
x <- tmvtnorm::rtmvnorm(n, mean = mu, sigma = solve(K),
       lower = rep(0, p), upper = rep(Inf, p), algorithm = "gibbs",
       burn.in.samples = 100, thinning = 10)

elts_NC_NP <- get_elts(h_hp$h, h_hp$hp, x, setting="trun_gaussian",
                centered=FALSE, profiled=FALSE, diag=dm)
res_nc_np <- get_results(elts_NC_NP, symmetric="symmetric",
               lambda1=0.35, lambda2=2, previous_res=NULL,
               is_refit=FALSE)
eBIC(res_nc_np, elts_NC_NP, BIC_refit=TRUE, gammas=c(0,0.5,1))
```

`estimate` *The main function for the generalized score-matching estimator for graphical models.*

Description

The main function for the generalized score-matching estimator for graphical models.

Usage

```
estimate(x, setting, elts = NULL, centered = TRUE,
  symmetric = "symmetric", scale = "norm", lambda1s = NULL,
  lambda_length = NULL, lambda_ratio = Inf, mode = NULL,
  param1 = NULL, param2 = NULL, h = NULL, hp = NULL,
  verbose = TRUE, verbosetext = "", tol = 1e-06, maxit = 1000,
  BIC_refit = TRUE, warmstart = TRUE, diagonal_multiplier = NULL,
  eBIC_gammas = c(0, 0.5, 1), return_raw = FALSE)
```

Arguments

`x`	A matrix, the data.
`setting`	A string that indicates the setting, must be one of `"exp"`, `"gamma"`, `"gaussian"`, `"trun_gaussian"`, or of the form `"ab_NUM1_NUM2"`, where `NUM1` is the `a` value and `NUM2` is the `b` value.

`elts` A list (optional), elements necessary for calculations returned by get_elts().

`centered` A boolean, whether in the centered setting (assume $\boldsymbol{\mu} = \boldsymbol{\eta} = 0$) or not. Default to `TRUE`.

`symmetric` A string. If equals `"symmetric"`, estimates the minimizer $\mathbf{K}$ over all symmetric matrices; if `"and"` or `"or"`, use the "and"/"or" rule to get the support. Default to `"symmetric"`.

`scale` A string indicating the scaling method. If contains `"sd"`, columns are scaled by standard deviation; if contains `"norm"`, columns are scaled by l2 norm; if contains `"center"` and `setting == "gaussian"`, columns are centered to have mean zero. Default to `"norm"`.

`lambda1s` A vector of lambdas, the penalty parameter for K.

`lambda_length` An integer >= 2, the number of lambda1s. Ignored if `lambda1s` is provided, otherwise a grid of lambdas is automatically chosen so that the results range from an empty graph to a complete graph. Default to `10` if neither `lambda1s` nor `lambda_length` is provided.

`lambda_ratio` A positive number, the fixed ratio between $\lambda_{\mathbf{K}}$ and $\lambda_{\boldsymbol{\eta}}$, if $\lambda_{\boldsymbol{\eta}} \neq 0$ (non-profiled) in the non-centered setting.

`mode` A string, the class of the `h` function. Ignored if `elts`, or `h` and `hp` are provided, or if `setting == "gaussian"`.

`param1` A number, the first parameter to the `h` function. Ignored if `elts`, or `h` and `hp` are provided, or if `setting == "gaussian"`.

`param2` A number, the second parameter (may be optional depending on `mode`) to the `h` function. Ignored if `elts`, or `h` and `hp` are provided, or if `setting == "gaussian"`.

`h` A function, the h function. Must evaluate to 0 at 0. Ignored if `elts` is provided, or if `setting == "gaussian"`.

`hp` A function, the derivative of the h function. Must be provided if `h` is provided, or if `setting == "gaussian"`.

`verbose` Optional. A boolean, whether to output intermediate results.

`verbosetext` Optional. A string, text to be added to the end of each printout if `verbose == TRUE`.

`tol` Optional. A number, the tolerance parameter. Default to `1e-6`.

`maxit` Optional. A positive integer, the maximum number of iterations for each fit. Default to `1000`.

`BIC_refit` A boolean, whether to get the BIC scores by refitting an unpenalized model restricted to the estimated edges, with `lambda1=lambda2=0` and `diagonal_multiplier=1`. Default to `TRUE`.

`warmstart` Optional. A boolean, whether to use the results from a previous (larger) lambda as a warm start for each new lambda. Default to `TRUE`.

`diagonal_multiplier`
A number >= 1, the diagonal multiplier. Optional and ignored if elts is provided. If `ncol(x) > ncol(n)`, a value strictly larger than 1 is recommended. Default to $1 + \left(1 - \left(1 + 4e \max\left(6 \log p/n, \sqrt{6 \log p/n}\right)\right)^{-1}\right)$.

`eBIC_gammas` Optional. A number of a vector of numbers. The γ parameter in eBIC. Default to `c(0,0.5,1)`.

`return_raw` A boolean, whether to return the raw estimates of `K`. Default to `FALSE`.

Value

edgess	A list of vectors of integers: indices of the non-zero edges.
etas	If applicable, a lambda_length*p matrix of eta estimates with the i-th row corresponding to the i-th lambda1, and may contain NAs after the first lambda that gives the complete graph. Otherwise NULL.
BICs	A lambda_length by length(eBIC_gammas) matrix of raw eBIC scores (without refitting). May contain Infs for rows after the first lambda that gives the complete graph.
BIC_refits	NULL if BIC_refit == FALSE, otherwise a lambda_length by length(eBIC_gammas) matrix of refitted eBIC scores, obtained by refitting unpenalized models restricted to the estimated edges. May contain Infs for rows after the first lambda that gives the graph restricted to which an unpenalized model does not have a solution (loss unbounded from below).
lambda1s	A vector of numbers of length lambda_length: the grid of lambda1s over which the estimates are obtained.
lambda2s	A vector of numbers of length lambda_length: the grid of lambda2s over which the estimates are obtained, if applicable, otherwise NULL.
converged	A vector of booleans of length lambda_length: indicators of convergence for each fit. May contain 0s for all lambdas after the first lambda that gives the complete graph.
iters	A vector of integers of length lambda_length: the number of iterations run for each fit. May contain 0s for all lambdas after the first lambda that gives the complete graph.
raw_estimate	An empty list if return_raw == FALSE, otherwise a list that contains lambda_length estimates for K of size ncol(x)*ncol(x). May contain ncol(x)*ncol(x) matrices of NAs for all lambdas after the first lambda that gives the complete graph.

Examples

```
if (!requireNamespace("tmvtnorm", quietly = TRUE)){
  stop("Please install package \"tmvtnorm\" first.", call. = FALSE)
}
require(tmvtnorm)
n <- 50
p <- 30
mu <- rep(0, p)
K <- diag(p)
lambda1s <- c(0.01,0.1,0.2,0.3,0.4,0.5)
dm <- 1 + (1-1/(1+4*exp(1)*max(6*log(p)/n, sqrt(6*log(p)/n))))
x <- tmvtnorm::rtmvnorm(n, mean = mu, sigma = solve(K),
       lower = rep(0, p), upper = rep(Inf, p), algorithm = "gibbs",
       burn.in.samples = 100, thinning = 10)

## Centered estimates, no elts or h provided, mode and params provided
est1 <- estimate(x, "trun_gaussian", elts=NULL, centered=TRUE,
          symmetric="symmetric", lambda1s=lambda1s, mode="min_pow",
          param1=1, param2=3, diag=dm, return_raw=TRUE)

h_hp <- get_h_hp("min_pow", 1, 3)
## Centered estimates, no elts provided, h provided; equivalent to est1
est2 <- estimate(x, "trun_gaussian", elts=NULL, centered=TRUE,
          symmetric="symmetric", lambda1s=lambda1s, h=h_hp$h,
```

```
          hp=h_hp$hp, diag=dm, return_raw=TRUE)

elts_C <- get_elts(h_hp$h, h_hp$hp, x, setting="trun_gaussian",
            centered=TRUE, diag=dm)
## Centered estimates, elts provided; equivalent to est1 and est2
est3 <- estimate(x, "trun_gaussian", elts=elts_C,
          symmetric="symmetric", lambda1s=lambda1s, diag=NULL,
          return_raw=TRUE)

## Noncentered estimates with Inf penalty on eta; equivalent to est1~3
est4 <- estimate(x, "trun_gaussian", elts=NULL, centered=FALSE,
          lambda_ratio=0, symmetric="symmetric", lambda1s=lambda1s,
          h=h_hp$h, hp=h_hp$hp, diag=dm, return_raw=TRUE)
compare_two_results(est1, est2) ## Should be almost all 0
compare_two_results(est1, est3) ## Should be almost all 0
sum(abs(est4$etas)) ## Should be 0 since non-centered with lambda ratio 0 is equivalent to centered
est4$etas <- NULL ## But different from est1 in that the zero etas are returned in est4
compare_two_results(est1, est4) ## Should be almost all 0


## Profiled estimates, no elts or h provided, mode and params provided
est5 <- estimate(x, "trun_gaussian", elts=NULL, centered=FALSE,
          lambda_ratio=Inf, symmetric="or", lambda1s=lambda1s,
          mode="min_pow", param1=1, param2=3, diag=dm, return_raw=TRUE)

## Profiled estimates, no elts provided, h provided; equivalent to est5
est6 <- estimate(x, "trun_gaussian", elts=NULL, centered=FALSE,
          lambda_ratio=Inf, symmetric="or", lambda1s=lambda1s,
          h=h_hp$h, hp=h_hp$hp, diag=dm, return_raw=TRUE)

elts_NC_P <- get_elts(h_hp$h, h_hp$hp, x, setting="trun_gaussian",
                centered=FALSE, profiled=TRUE, diag=dm)
## Profiled estimates, elts provided; equivalent to est5~6
est7 <- estimate(x, "trun_gaussian", elts=elts_NC_P, centered=FALSE,
          lambda_ratio=Inf, symmetric="or", lambda1s=lambda1s,
          diagonal_multiplier=NULL, return_raw=TRUE)
compare_two_results(est5, est6) ## Should be almost all 0
compare_two_results(est5, est7) ## Should be almost all 0


## Non-centered estimates, no elts or h provided, mode and params provided
est8 <- estimate(x, "trun_gaussian", elts=NULL, centered=FALSE,
          lambda_ratio=2, symmetric="and", lambda_length=100,
          mode="min_pow", param1=1, param2=3, diag=dm, return_raw=TRUE)

## Non-centered estimates, no elts provided, h provided; equivalent to est5
est9 <- estimate(x, "trun_gaussian", elts=NULL, centered=FALSE,
          lambda_ratio=2, symmetric="and", lambda_length=100,
          h=h_hp$h, hp=h_hp$hp, diag=dm, return_raw=TRUE)

elts_NC_NP <- get_elts(h_hp$h, h_hp$hp, x, setting="trun_gaussian", centered=FALSE,
                profiled=FALSE, diag=dm)
## Non-centered estimates, elts provided; equivalent to est8~9
est10 <- estimate(x, "trun_gaussian", elts=elts_NC_NP,
           centered=FALSE, lambda_ratio=2, symmetric="and",
           lambda_length=100, diag=NULL, return_raw=TRUE)
compare_two_results(est8, est9) ## Should be almost all 0
```

```
compare_two_results(est8, est10) ## Should be almost all 0
```

`find_max_ind` *Finds the max index in a vector that does not exceed a target number.*

Description

Finds the max index in a vector that does not exceed a target number.

Usage

```
find_max_ind(vals, target, start = 1)
```

Arguments

`vals`	A vector of numbers.
`target`	A number. Must not be smaller than `vals[start]`.
`start`	A number, the starting index; default to 1. Must be such that `vals[start] <= target`.

Value

The max index `i` such that `vals[i] <= target` and `i >= start`.

Examples

```
for (i in 1:100) {
   vals <- 1:i
   for (start in 1:i)
      for (target in seq(start, i+0.5, by=0.5))
         if (find_max_ind(vals, target, start) != floor(target))
            stop()
}
```

`get_crit_nopenalty` *Minimized loss for unpenalized restricted asymmetric models.*

Description

Analytic solution of the minimized loss for an unpenalized asymmetric model restricted to a given support. Does not work if `symmetric == "symmetric"`.

Usage

```
get_crit_nopenalty(elts, exclude = NULL, exclude_eta = NULL,
  previous_res = NULL)
```

Arguments

elts	A list, elements necessary for calculations returned by get_elts().
exclude	Optional. A p*p binary matrix or a p^2 binary vector, where 1 indicates the entry in K was estimated to 0 in the previous estimate. Default to NULL.
exclude_eta	Optional. A p-binary vector, similar to exclude. Default to NULL.
previous_res	Optional. A list, the returned list by get_results() run previously with another lambda value. Default to NULL.

Details

If previous_res is provided, exclude and exclude_eta must be NULL or be consistent with the estimated support in previous_res. If previous_res and exclude are both NULL, assume all edges are present. The same applies to the non-profiled non-centered case when previous_res and exclude_eta are both NULL.

Value

A number, the refitted loss.

Examples

```
if (!requireNamespace("tmvtnorm", quietly = TRUE)){
  stop("Please install package \"tmvtnorm\" first.", call. = FALSE)
}
require(tmvtnorm)
n <- 50
p <- 30
h_hp <- get_h_hp("min_pow", 1, 3)
mu <- rep(0, p)
K <- diag(p)
dm <- 1 + (1-1/(1+4*exp(1)*max(6*log(p)/n, sqrt(6*log(p)/n))))
x <- tmvtnorm::rtmvnorm(n, mean = mu, sigma = solve(K),
       lower = rep(0, p), upper = rep(Inf, p), algorithm = "gibbs",
       burn.in.samples = 100, thinning = 10)

elts_NC_NP <- get_elts(h_hp$h, h_hp$hp, x, setting="trun_gaussian",
                centered=FALSE, profiled=FALSE, diag=dm)
res_nc_np <- get_results(elts_NC_NP, symmetric="symmetric", lambda1=0.35,
               lambda2=2, previous_res=NULL, is_refit=FALSE)
get_crit_nopenalty(elts_NC_NP, previous_res=res_nc_np)
```

---

get_elts *The function wrapper to get the elements necessary for calculations for all settings.*

Description

The function wrapper to get the elements necessary for calculations for all settings.

Usage

```
get_elts(h, hp, x, setting, centered = TRUE,
  profiled_if_noncenter = TRUE, scale = "norm",
  diagonal_multiplier = 1, use_C = TRUE,
  tol = .Machine$double.eps^0.5)
```

Arguments

h	A function, the h function. Must evaluate to 0 at 0. Ignored if `elts` is provided.
hp	A function, the derivative of the h function. Must be provided if `h` is provided.
x	A matrix, the data matrix.
setting	A string that indicates the setting, must be one of `"exp"`, `"gamma"`, `"gaussian"`, `"trun_gaussian"`, or of the form `"ab_NUM1_NUM2"`, where `NUM1` is the `a` value and `NUM2` is the `b` value.
centered	A boolean, whether in the centered setting(assume $\boldsymbol{\mu} = \boldsymbol{\eta} = 0$) or not. Default to `TRUE`.
profiled_if_noncenter	A boolean, whether in the profiled setting ($\lambda_{\boldsymbol{\eta}} = 0$) if noncentered. Parameter ignored if `centered=TRUE`. Default to `TRUE`.
scale	A string indicating the scaling method. If contains `"sd"`, columns are scaled by standard deviation; if contains `"norm"`, columns are scaled by l2 norm; if contains `"center"` and `setting == "gaussian"`, columns are centered to have mean zero. Default to `"norm"`.
diagonal_multiplier	A number >= 1, the diagonal multiplier.
use_C	Optional. A boolean, use C (`TRUE`) or R (`FALSE`) functions for computation. Default to `TRUE`. Ignored if `setting == "gaussian"`.
tol	Optional. A positive number. If `setting != "gaussian"`, function stops if any entry if smaller than -tol, and all entries between -tol and 0 are set to tol, for numerical stability and to avoid violating the assumption that $h(\mathbf{x}) > 0$ almost surely.

Details

Computes the $\boldsymbol{\Gamma}$ matrix and the $\boldsymbol{g}$ vector for generalized score matching.

Here, $\boldsymbol{\Gamma}$ is block-diagonal, and in the non-profiled non-centered setting, the j-th block is composed of $\boldsymbol{\Gamma}_{\mathbf{KK},j}$, $\boldsymbol{\Gamma}_{\mathbf{K}\boldsymbol{\eta},j}$ and its transpose, and finally $\boldsymbol{\Gamma}_{\boldsymbol{\eta}\boldsymbol{\eta},j}$. In the centered case, only $\boldsymbol{\Gamma}_{\mathbf{KK},j}$ is computed. In the profiled non-centered case,

$$\boldsymbol{\Gamma}_j \equiv \boldsymbol{\Gamma}_{\mathbf{KK},j} - \boldsymbol{\Gamma}_{\mathbf{K}\boldsymbol{\eta},j}\boldsymbol{\Gamma}_{\boldsymbol{\eta}\boldsymbol{\eta},j}^{-1}\boldsymbol{\Gamma}_{\mathbf{K}\boldsymbol{\eta}}^{\top}.$$

Similarly, in the non-profiled non-centered setting, $\boldsymbol{g}$ can be partitioned p parts, each with a p-vector $\boldsymbol{g}_{\mathbf{K},j}$ and a scalar $g_{\boldsymbol{\eta},j}$. In the centered setting, only $\boldsymbol{g}_{\mathbf{K},j}$ is needed. In the profiled non-centered case,

$$\boldsymbol{g}_j \equiv \boldsymbol{g}_{\mathbf{K},j} - \boldsymbol{\Gamma}_{\mathbf{K}\boldsymbol{\eta},j}\boldsymbol{\Gamma}_{\boldsymbol{\eta}\boldsymbol{\eta},j}^{-1} g_{\boldsymbol{\eta},j}.$$

The formulae for the pieces above are

$$\boldsymbol{\Gamma}_{\mathbf{KK},j} \equiv \frac{1}{n}\sum_{i=1}^{n} h\left(X_j^{(i)}\right) {X_j^{(i)}}^{2a-2} {\boldsymbol{X}^{(i)}}^{a} {\boldsymbol{X}^{(i)}}^{a\top},$$

$$\boldsymbol{\Gamma}_{\mathbf{K}\boldsymbol{\eta},j} \equiv -\frac{1}{n}\sum_{i=1}^{n} h\left(X_j^{(i)}\right) {X_j^{(i)}}^{a+b-2} {\boldsymbol{X}^{(i)}}^{a},$$

$$\boldsymbol{\Gamma}_{\boldsymbol{\eta}\boldsymbol{\eta},j} \equiv \frac{1}{n}\sum_{i=1}^{n} h\left(X_j^{(i)}\right) {X_j^{(i)}}^{2b-2},$$

$$\boldsymbol{g}_{\mathbf{K},j} \equiv \frac{1}{n}\sum_{i=1}^{n} \left(h'\left(X_j^{(i)}\right) {X_j^{(i)}}^{a-1} + (a-1)h\left(X_j^{(i)}\right) {X_j^{(i)}}^{a-2}\right) {\boldsymbol{X}^{(i)}}^{a} + ah\left(X_j^{(i)}\right) {X_j^{(i)}}^{2a-2} \boldsymbol{e}_{j,p},$$

$$\boldsymbol{g}_{\boldsymbol{\eta},j} \equiv \frac{1}{n}\sum_{i=1}^{n} -h'\left(X_j^{(i)}\right) {X_j^{(i)}}^{b-1} - (b-1)h\left(X_j^{(i)}\right) {X_j^{(i)}}^{b-2},$$

where $\boldsymbol{e}_{j,p}$ is the p-vector with 1 at the j-th position and 0 elsewhere.

In the profiled non-centered setting, the function also returns t_1 and t_2 defined as

$$\boldsymbol{t}_1 \equiv \boldsymbol{\Gamma}_{\boldsymbol{\eta}\boldsymbol{\eta}}^{-1}\boldsymbol{g}_{\boldsymbol{\eta}}, \quad \boldsymbol{t}_2 \equiv \boldsymbol{\Gamma}_{\boldsymbol{\eta}\boldsymbol{\eta}}^{-1}\boldsymbol{\Gamma}_{\mathbf{K}\boldsymbol{\eta}}^{\top},$$

so that $\hat{\boldsymbol{\eta}} = \boldsymbol{t}_1 - \boldsymbol{t}_2 \mathrm{vec}(\hat{\mathbf{K}})$.

Value

A list that contains the elements necessary for estimation.

`n` The sample size.

`p` The dimension.

`centered` The centered setting or not. Same as input.

`scale` The scaling method. Same as input.

`diagonal_multiplier`
The diagonal multiplier. Same as input.

`diagonals_with_multiplier`
A vector that contains the diagonal entries of $\boldsymbol{\Gamma}$ after applying the multiplier.

`setting` The setting. Same as input.

`g_K` The $\boldsymbol{g}$ vector. In the non-profiled non-centered setting, this is the $\boldsymbol{g}$ sub-vector corresponding to $\mathbf{K}$. A p^2-vector. Not returned if `setting == "gaussian"` since it is just $diag(p)$.

`Gamma_K` The $\boldsymbol{\Gamma}$ matrix with no diagonal multiplier. In the non-profiled non-centered setting, this is the $\boldsymbol{\Gamma}$ sub-matrix corresponding to $\mathbf{K}$. A vector of length p^2 if `setting == "gaussian"` or p^3 otherwise.

`g_eta` Returned in the non-profiled non-centered setting. The $\boldsymbol{g}$ sub-vector corresponding to $\boldsymbol{\eta}$. A p-vector. Not returned if `setting == "gaussian"` since it is just $numeric(p)$.

`Gamma_K_eta` Returned in the non-profiled non-centered setting. The $\boldsymbol{\Gamma}$ sub-matrix corresponding to interaction between $\mathbf{K}$ and $\boldsymbol{\eta}$. If `setting == "gaussian"`, returns a vector of length p, or p^2 otherwise.

`Gamma_eta` Returned in the non-profiled non-centered setting. The $\boldsymbol{\Gamma}$ sub-matrix corresponding to $\boldsymbol{\eta}$. A p-vector. Not returned if `setting == "gaussian"` since it is just `rep(1,p)`.

`t1,t2` Returned in the profiled non-centered setting, where the $\boldsymbol{\eta}$ estimate can be retrieved from $\boldsymbol{t_1} - \boldsymbol{t_2}\hat{\mathbf{K}}$ after appropriate resizing.

Examples

```
if (!requireNamespace("mvtnorm", quietly = TRUE)){
  stop("Please install package \"mvtnorm\" first.", call. = FALSE)
}
if (!requireNamespace("tmvtnorm", quietly = TRUE)){
  stop("Please install package \"tmvtnorm\" first.", call. = FALSE)
}
require(mvtnorm)
require(tmvtnorm)
n <- 50
p <- 30
h_hp <- get_h_hp("min_pow", 1, 3)
mu <- rep(0, p)
K <- diag(p)
diagonal_multiplier <- 1 + (1-1/(1+4*exp(1)*max(6*log(p)/n, sqrt(6*log(p)/n))))
x <- tmvtnorm::rtmvnorm(n, mean = mu, sigma = solve(K),
       lower = rep(0, p), upper = rep(Inf, p), algorithm = "gibbs",
       burn.in.samples = 100, thinning = 10)
h_hp <- get_h_hp("min_pow", 1, 3)
get_elts(h_hp$h, h_hp$hp, x, setting="trun_gaussian",
  centered=TRUE, scale="norm", diag=1.5)
get_elts(h_hp$h, h_hp$hp, x, setting="ab_0.7_1.2",
  centered=FALSE, profiled=FALSE, scale="sd", diag=1.9)

x <- mvtnorm::rmvnorm(n, mean=mu, sigma=solve(K))
get_elts(NULL, NULL, x, setting="gaussian", centered=FALSE,
  profiled=FALSE, scale="center_norm", diag=1.3)
```

`get_elts_ab` *The R implementation to get the elements necessary for calculations for general a and b.*

Description

The R implementation to get the elements necessary for calculations for general a and b.

Usage

```
get_elts_ab(hx, hpx, x, a, b, setting, centered = TRUE,
  profiled_if_noncenter = TRUE, scale = "norm",
  diagonal_multiplier = 1)
```

Arguments

hx	A matrix, $h(\mathbf{x})$, should be of the same dimension as x.
hpx	A matrix, $h'(\mathbf{x})$, should be of the same dimension as x.
x	A matrix, the data matrix.
a	A number, must be strictly larger than $b/2$.
b	A number, must be >= 0.
setting	A string that indicates the setting. Returned without being checked or used in the function body.

`centered` A boolean, whether in the centered setting (assume $\boldsymbol{\mu} = \boldsymbol{\eta} = 0$) or not. Default to `TRUE`.

`profiled_if_noncenter` A boolean, whether in the profiled setting ($\lambda_{\boldsymbol{\eta}} = 0$) if noncentered. Parameter ignored if `centered=TRUE`. Default to `TRUE`.

`scale` A string indicating the scaling method. Returned without being checked or used in the function body. Default to `"norm"`.

`diagonal_multiplier` A number >= 1, the diagonal multiplier.

Details

Computes the $\boldsymbol{\Gamma}$ matrix and the $\boldsymbol{g}$ vector for generalized score matching.

Here, $\boldsymbol{\Gamma}$ is block-diagonal, and in the non-profiled non-centered setting, the j-th block is composed of $\boldsymbol{\Gamma}_{\mathbf{KK},j}$, $\boldsymbol{\Gamma}_{\mathbf{K}\boldsymbol{\eta},j}$ and its transpose, and finally $\boldsymbol{\Gamma}_{\boldsymbol{\eta}\boldsymbol{\eta},j}$. In the centered case, only $\boldsymbol{\Gamma}_{\mathbf{KK},j}$ is computed. In the profiled non-centered case,

$$\boldsymbol{\Gamma}_j \equiv \boldsymbol{\Gamma}_{\mathbf{KK},j} - \boldsymbol{\Gamma}_{\mathbf{K}\boldsymbol{\eta},j}\boldsymbol{\Gamma}_{\boldsymbol{\eta}\boldsymbol{\eta},j}^{-1}\boldsymbol{\Gamma}_{\mathbf{K}\boldsymbol{\eta}}^{\top}.$$

Similarly, in the non-profiled non-centered setting, $\boldsymbol{g}$ can be partitioned p parts, each with a p-vector $\boldsymbol{g}_{\mathbf{K},j}$ and a scalar $g_{\boldsymbol{\eta},j}$. In the centered setting, only $\boldsymbol{g}_{\mathbf{K},j}$ is needed. In the profiled non-centered case,

$$\boldsymbol{g}_j \equiv \boldsymbol{g}_{\mathbf{K},j} - \boldsymbol{\Gamma}_{\mathbf{K}\boldsymbol{\eta},j}\boldsymbol{\Gamma}_{\boldsymbol{\eta}\boldsymbol{\eta},j}^{-1} g_{\boldsymbol{\eta},j}.$$

The formulae for the pieces above are

$$\boldsymbol{\Gamma}_{\mathbf{KK},j} \equiv \frac{1}{n}\sum_{i=1}^{n} h\left(X_j^{(i)}\right) {X_j^{(i)}}^{2a-2} {\boldsymbol{X}^{(i)}}^{a} {\boldsymbol{X}^{(i)}}^{a\top},$$

$$\boldsymbol{\Gamma}_{\mathbf{K}\boldsymbol{\eta},j} \equiv -\frac{1}{n}\sum_{i=1}^{n} h\left(X_j^{(i)}\right) {X_j^{(i)}}^{a+b-2} {\boldsymbol{X}^{(i)}}^{a},$$

$$\boldsymbol{\Gamma}_{\boldsymbol{\eta}\boldsymbol{\eta},j} \equiv \frac{1}{n}\sum_{i=1}^{n} h\left(X_j^{(i)}\right) {X_j^{(i)}}^{2b-2},$$

$$\boldsymbol{g}_{\mathbf{K},j} \equiv \frac{1}{n}\sum_{i=1}^{n} \left(h'\left(X_j^{(i)}\right) {X_j^{(i)}}^{a-1} + (a-1)h\left(X_j^{(i)}\right) {X_j^{(i)}}^{a-2}\right) {\boldsymbol{X}^{(i)}}^{a} + ah\left(X_j^{(i)}\right) {X_j^{(i)}}^{2a-2} \boldsymbol{e}_{j,p},$$

$$\boldsymbol{g}_{\boldsymbol{\eta},j} \equiv \frac{1}{n}\sum_{i=1}^{n} -h'\left(X_j^{(i)}\right) {X_j^{(i)}}^{b-1} - (b-1)h\left(X_j^{(i)}\right) {X_j^{(i)}}^{b-2},$$

where $\boldsymbol{e}_{j,p}$ is the p-vector with 1 at the j-th position and 0 elsewhere.

In the profiled non-centered setting, the function also returns t_1 and t_2 defined as

$$\boldsymbol{t}_1 \equiv \boldsymbol{\Gamma}_{\boldsymbol{\eta}\boldsymbol{\eta}}^{-1}\boldsymbol{g}_{\boldsymbol{\eta}}, \quad \boldsymbol{t}_2 \equiv \boldsymbol{\Gamma}_{\boldsymbol{\eta}\boldsymbol{\eta}}^{-1}\boldsymbol{\Gamma}_{\mathbf{K}\boldsymbol{\eta}}^{\top},$$

so that $\hat{\boldsymbol{\eta}} = \boldsymbol{t}_1 - \boldsymbol{t}_2 \mathrm{vec}(\hat{\mathbf{K}})$.

Value

A list that contains the elements necessary for estimation.

n	The sample size.
p	The dimension.
centered	The centered setting or not. Same as input.
scale	The scaling method. Same as input.
diagonal_multiplier	The diagonal multiplier. Same as input.
diagonals_with_multiplier	A vector that contains the diagonal entries of $\mathbf{\Gamma}$ after applying the multiplier.
setting	The setting. Same as input.
g_K	The $\boldsymbol{g}$ vector. In the non-profiled non-centered setting, this is the $\boldsymbol{g}$ sub-vector corresponding to $\mathbf{K}$.
Gamma_K	The $\mathbf{\Gamma}$ matrix with no diagonal multiplier. In the non-profiled non-centered setting, this is the $\mathbf{\Gamma}$ sub-matrix corresponding to $\mathbf{K}$.
g_eta	Returned in the non-profiled non-centered setting. The $\boldsymbol{g}$ sub-vector corresponding to $\boldsymbol{\eta}$.
Gamma_K_eta	Returned in the non-profiled non-centered setting. The $\mathbf{\Gamma}$ sub-matrix corresponding to interaction between $\mathbf{K}$ and $\boldsymbol{\eta}$.
Gamma_eta	Returned in the non-profiled non-centered setting. The $\mathbf{\Gamma}$ sub-matrix corresponding to $\boldsymbol{\eta}$.
t1,t2	Returned in the profiled non-centered setting, where the $\boldsymbol{\eta}$ estimate can be retrieved from $\boldsymbol{t_1} - \boldsymbol{t_2}\hat{\mathbf{K}}$ after appropriate resizing.

Examples

```
if (!requireNamespace("tmvtnorm", quietly = TRUE)){
  stop("Please install package \"tmvtnorm\" first.", call. = FALSE)
}
require(tmvtnorm)
n <- 50
p <- 30
h_hp <- get_h_hp("min_pow", 1, 3)
mu <- rep(0, p)
K <- diag(p)
x <- tmvtnorm::rtmvnorm(n, mean = mu, sigma = solve(K),
       lower = rep(0, p), upper = rep(Inf, p), algorithm = "gibbs",
       burn.in.samples = 100, thinning = 10)
h_hp <- get_h_hp("min_pow", 1, 3)
hx <- t(apply(x, 1, h_hp$h))
hpx <- t(apply(x, 1, h_hp$hp))
get_elts_ab(hx, hpx, x, a=0.5, b=0.5, setting="trun_gaussian",
            centered=TRUE, scale="norm", diag=1.5)
get_elts_ab(hx, hpx, x, a=0.7, b=1.2, setting="ab_0.7_1.2",
            centered=FALSE, profiled=FALSE, scale="sd", diag=1.9)
```

`get_elts_exp` *The R implementation to get the elements necessary for calculations for the exponential square-root setting (a=0.5, b=0.5).*

Description

The R implementation to get the elements necessary for calculations for the exponential square-root setting (a=0.5, b=0.5).

Usage

```
get_elts_exp(hx, hpx, x, centered = TRUE, profiled_if_noncenter = TRUE,
  scale = "norm", diagonal_multiplier = 1)
```

Arguments

`hx`	A matrix, $h(\mathbf{x})$, should be of the same dimension as `x`.
`hpx`	A matrix, $h'(\mathbf{x})$, should be of the same dimension as `x`.
`x`	A matrix, the data matrix.
`centered`	A boolean, whether in the centered setting (assume $\boldsymbol{\mu} = \boldsymbol{\eta} = 0$) or not. Default to `TRUE`.
`profiled_if_noncenter`	A boolean, whether in the profiled setting ($\lambda_{\boldsymbol{\eta}} = 0$) if noncentered. Parameter ignored if centered=TRUE. Default to `TRUE`.
`scale`	A string indicating the scaling method. Returned without being checked or used in the function body. Default to `"norm"`.
`diagonal_multiplier`	A number >= 1, the diagonal multiplier.

Details

For details on the returned values, please refer to `get_elts_ab` or `get_elts`.

Value

A list that contains the elements necessary for estimation.

`n`	The sample size.
`p`	The dimension.
`centered`	The centered setting or not. Same as input.
`scale`	The scaling method. Same as input.
`diagonal_multiplier`	The diagonal multiplier. Same as input.
`diagonals_with_multiplier`	A vector that contains the diagonal entries of $\boldsymbol{\Gamma}$ after applying the multiplier.
`setting`	The setting `"exp"`.
`g_K`	The $\boldsymbol{g}$ vector. In the non-profiled non-centered setting, this is the $\boldsymbol{g}$ sub-vector corresponding to $\mathbf{K}$.

Gamma_K	The $\mathbf{\Gamma}$ matrix with no diagonal multiplier. In the non-profiled non-centered setting, this is the $\mathbf{\Gamma}$ sub-matrix corresponding to $\mathbf{K}$.
g_eta	Returned in the non-profiled non-centered setting. The $\boldsymbol{g}$ sub-vector corresponding to $\boldsymbol{\eta}$.
Gamma_K_eta	Returned in the non-profiled non-centered setting. The $\mathbf{\Gamma}$ sub-matrix corresponding to interaction between $\mathbf{K}$ and $\boldsymbol{\eta}$.
Gamma_eta	Returned in the non-profiled non-centered setting. The $\mathbf{\Gamma}$ sub-matrix corresponding to $\boldsymbol{\eta}$.
t1,t2	Returned in the profiled non-centered setting, where the $\boldsymbol{\eta}$ estimate can be retrieved from $\boldsymbol{t_1} - \boldsymbol{t_2}\hat{\mathbf{K}}$ after appropriate resizing.

Examples

```
if (!requireNamespace("tmvtnorm", quietly = TRUE)){
  stop("Please install package \"tmvtnorm\" first.", call. = FALSE)
}
require(tmvtnorm)
n <- 50
p <- 30
h_hp <- get_h_hp("min_pow", 1, 3)
mu <- rep(0, p)
K <- diag(p)
x <- tmvtnorm::rtmvnorm(n, mean = mu, sigma = solve(K),
       lower = rep(0, p), upper = rep(Inf, p), algorithm = "gibbs",
       burn.in.samples = 100, thinning = 10)
h_hp <- get_h_hp("min_pow", 1, 3)
hx <- t(apply(x, 1, h_hp$h))
hpx <- t(apply(x, 1, h_hp$hp))
get_elts_exp(hx, hpx, x, centered=TRUE, scale="norm", diag=1.5)
get_elts_exp(hx, hpx, x, centered=FALSE, profiled=FALSE, scale="sd", diag=1.9)
```

---

get_elts_gamma *The R implementation to get the elements necessary for calculations for the gamma setting (a=0.5, b=0).*

Description

The R implementation to get the elements necessary for calculations for the gamma setting (a=0.5, b=0).

Usage

```
get_elts_gamma(hx, hpx, x, centered = TRUE,
  profiled_if_noncenter = TRUE, scale = "norm",
  diagonal_multiplier = 1)
```

Arguments

hx	A matrix, $h(\mathbf{x})$, should be of the same dimension as x.
hpx	A matrix, $h'(\mathbf{x})$, should be of the same dimension as x.
x	A matrix, the data matrix.

`centered` A boolean, whether in the centered setting (assume $\boldsymbol{\mu} = \boldsymbol{\eta} = 0$) or not. Default to `TRUE`.

`profiled_if_noncenter` A boolean, whether in the profiled setting ($\lambda_{\boldsymbol{\eta}} = 0$) if noncentered. Parameter ignored if centered=TRUE. Default to `TRUE`.

`scale` A string indicating the scaling method. Returned without being checked or used in the function body. Default to `"norm"`.

`diagonal_multiplier` A number >= 1, the diagonal multiplier.

Details

For details on the returned values, please refer to `get_elts_ab` or `get_elts`.

Value

A list that contains the elements necessary for estimation.

`n` The sample size.

`p` The dimension.

`centered` The centered setting or not. Same as input.

`scale` The scaling method. Same as input.

`diagonal_multiplier` The diagonal multiplier. Same as input.

`diagonals_with_multiplier` A vector that contains the diagonal entries of $\boldsymbol{\Gamma}$ after applying the multiplier.

`setting` The setting `"gamma"`.

`g_K` The $\boldsymbol{g}$ vector. In the non-profiled non-centered setting, this is the $\boldsymbol{g}$ sub-vector corresponding to $\mathbf{K}$.

`Gamma_K` The $\boldsymbol{\Gamma}$ matrix with no diagonal multiplier. In the non-profiled non-centered setting, this is the $\boldsymbol{\Gamma}$ sub-matrix corresponding to $\mathbf{K}$.

`g_eta` Returned in the non-profiled non-centered setting. The $\boldsymbol{g}$ sub-vector corresponding to $\boldsymbol{\eta}$.

`Gamma_K_eta` Returned in the non-profiled non-centered setting. The $\boldsymbol{\Gamma}$ sub-matrix corresponding to interaction between $\mathbf{K}$ and $\boldsymbol{\eta}$.

`Gamma_eta` Returned in the non-profiled non-centered setting. The $\boldsymbol{\Gamma}$ sub-matrix corresponding to $\boldsymbol{\eta}$.

`t1,t2` Returned in the profiled non-centered setting, where the $\boldsymbol{\eta}$ estimate can be retrieved from $\boldsymbol{t_1} - \boldsymbol{t_2}\hat{\mathbf{K}}$ after appropriate resizing.

Examples

```
if (!requireNamespace("tmvtnorm", quietly = TRUE)){
  stop("Please install package \"tmvtnorm\" first.", call. = FALSE)
}
require(tmvtnorm)
n <- 50
p <- 30
h_hp <- get_h_hp("min_pow", 1, 3)
mu <- rep(0, p)
```

```
K <- diag(p)
x <- tmvtnorm::rtmvnorm(n, mean = mu, sigma = solve(K),
       lower = rep(0, p), upper = rep(Inf, p), algorithm = "gibbs",
       burn.in.samples = 100, thinning = 10)
h_hp <- get_h_hp("min_pow", 1, 3)
hx <- t(apply(x, 1, h_hp$h))
hpx <- t(apply(x, 1, h_hp$hp))
get_elts_gamma(hx, hpx, x, centered=TRUE, scale="norm", diag=1.5)
get_elts_gamma(hx, hpx, x, centered=FALSE, profiled=FALSE, scale="sd", diag=1.9)
```

`get_elts_gauss` *The R implementation to get the elements necessary for calculations for the untruncated gaussian setting.*

Description

The R implementation to get the elements necessary for calculations for the untruncated gaussian setting.

Usage

```
get_elts_gauss(x, centered = TRUE, profiled_if_noncenter = TRUE,
  scale = "norm", diagonal_multiplier = 1)
```

Arguments

`x`	A matrix, the data matrix.
`centered`	A boolean, whether in the centered setting (assume $\boldsymbol{\mu} = \boldsymbol{\eta} = 0$) or not. Default to `TRUE`.
`profiled_if_noncenter`	A boolean, whether in the profiled setting ($\lambda_{\boldsymbol{\eta}} = 0$) if noncentered. Parameter ignored if `centered==TRUE`. Default to `TRUE`.
`scale`	A string indicating the scaling method. Returned without being checked or used in the function body. Default to `"norm"`.
`diagonal_multiplier`	A number >= 1, the diagonal multiplier.

Details

For details on the returned values, please refer to `get_elts_ab` or `get_elts`.

Value

A list that contains the elements necessary for estimation.

`n`	The sample size.
`p`	The dimension.
`centered`	The centered setting or not. Same as input.
`scale`	The scaling method. Same as input.
`diagonal_multiplier`	The diagonal multiplier. Same as input.

diagonals_with_multiplier
: A vector that contains the diagonal entries of $\mathbf{\Gamma}$ after applying the multiplier.

setting
: The setting "gaussian".

Gamma_K
: The $\mathbf{\Gamma}$ matrix with no diagonal multiplier. In the non-profiled non-centered setting, this is the $\mathbf{\Gamma}$ sub-matrix corresponding to $\mathbf{K}$. Except for the *profiled* setting, this is $\mathbf{x}\mathbf{x}^\top/n$.

Gamma_K_eta
: Returned in the non-profiled non-centered setting. The $\mathbf{\Gamma}$ sub-matrix corresponding to interaction between $\mathbf{K}$ and $\boldsymbol{\eta}$. The minus column means of x.

t1,t2
: Returned in the profiled non-centered setting, where the$\boldsymbol{\eta}$ estimate can be retrieved from $\boldsymbol{t_1} - \boldsymbol{t_2}\hat{\mathbf{K}}$ after appropriate resizing.

Examples

```
if (!requireNamespace("mvtnorm", quietly = TRUE)){
  stop("Please install package \"mvtnorm\" first.", call. = FALSE)
}
require(mvtnorm)
n <- 50
p <- 30
h_hp <- get_h_hp("min_pow", 1, 3)
mu <- rep(0, p)
K <- diag(p)
x <- mvtnorm::rmvnorm(n, mean=mu, sigma=solve(K))
get_elts_gauss(x, centered=TRUE, scale="norm", diag=1.5)
get_elts_gauss(x, centered=FALSE, profiled=FALSE, scale="sd", diag=1.9)
```

---

get_elts_trun_gauss *The R implementation to get the elements necessary for calculations for the truncated gaussian setting (a=1, b=1).*

Description

The R implementation to get the elements necessary for calculations for the truncated gaussian setting (a=1, b=1).

Usage

```
get_elts_trun_gauss(hx, hpx, x, centered = TRUE,
  profiled_if_noncenter = TRUE, scale = "norm",
  diagonal_multiplier = 1)
```

Arguments

hx
: A matrix, $h(\mathbf{x})$, should be of the same dimension as x.

hpx
: A matrix, $h'(\mathbf{x})$, should be of the same dimension as x.

x
: A matrix, the data matrix.

centered
: A boolean, whether in the centered setting (assume $\boldsymbol{\mu} = \boldsymbol{\eta} = 0$) or not. Default to TRUE.

profiled_if_noncenter
: A boolean, whether in the profiled setting ($\lambda_{\boldsymbol{\eta}} = 0$) if noncentered. Parameter ignored if centered=TRUE. Default to TRUE.

`scale` A string indicating the scaling method. Returned without being checked or used in the function body. Default to `"norm"`.

`diagonal_multiplier`
A number >= 1, the diagonal multiplier.

Details

For details on the returned values, please refer to `get_elts_ab` or `get_elts`.

Value

A list that contains the elements necessary for estimation.

`n` The sample size.

`p` The dimension.

`centered` The centered setting or not. Same as input.

`scale` The scaling method. Same as input.

`diagonal_multiplier`
The diagonal multiplier. Same as input.

`diagonals_with_multiplier`
A vector that contains the diagonal entries of $\mathbf{\Gamma}$ after applying the multiplier.

`setting` The setting `"trun_gaussian"`.

`g_K` The $\boldsymbol{g}$ vector. In the non-profiled non-centered setting, this is the $\boldsymbol{g}$ sub-vector corresponding to $\mathbf{K}$.

`Gamma_K` The $\mathbf{\Gamma}$ matrix with no diagonal multiplier. In the non-profiled non-centered setting, this is the $\mathbf{\Gamma}$ sub-matrix corresponding to $\mathbf{K}$.

`g_eta` Returned in the non-profiled non-centered setting. The $\boldsymbol{g}$ sub-vector corresponding to $\boldsymbol{\eta}$.

`Gamma_K_eta` Returned in the non-profiled non-centered setting. The $\mathbf{\Gamma}$ sub-matrix corresponding to interaction between $\mathbf{K}$ and $\boldsymbol{\eta}$.

`Gamma_eta` Returned in the non-profiled non-centered setting. The $\mathbf{\Gamma}$ sub-matrix corresponding to $\boldsymbol{\eta}$.

`t1,t2` Returned in the profiled non-centered setting, where the $\boldsymbol{\eta}$ estimate can be retrieved from $\boldsymbol{t_1} - \boldsymbol{t_2}\hat{\mathbf{K}}$ after appropriate resizing.

Examples

```
if (!requireNamespace("tmvtnorm", quietly = TRUE)){
  stop("Please install package \"tmvtnorm\" first.", call. = FALSE)
}
require(tmvtnorm)
n <- 50
p <- 30
h_hp <- get_h_hp("min_pow", 1, 3)
mu <- rep(0, p)
K <- diag(p)
x <- tmvtnorm::rtmvnorm(n, mean = mu, sigma = solve(K),
       lower = rep(0, p), upper = rep(Inf, p), algorithm = "gibbs",
       burn.in.samples = 100, thinning = 10)
h_hp <- get_h_hp("min_pow", 1, 3)
hx <- t(apply(x, 1, h_hp$h))
```

```
hpx <- t(apply(x, 1, h_hp$hp))
get_elts_trun_gauss(hx, hpx, x, centered=TRUE, scale="norm", diag=1.5)
get_elts_trun_gauss(hx, hpx, x, centered=FALSE, profiled=FALSE, scale="sd", diag=1.9)
```

`get_h_hp` *Generator of h and hp functions.*

Description

Generator of h and hp (h') functions.

Usage

```
get_h_hp(mode, para = NULL, para2 = NULL)
```

Arguments

mode	A string, see details.
para	May be optional. A number, the first parameter. Default to NULL.
para2	May be optional. A number, the second parameter. Default to NULL.

Details

The mode parameter can be chosen from the options listed below along with the corresponding definitions of h under appropriate choices of para and para2 parameters. Unless otherwise noted, para and para2, must both be strictly positive if provided, and are set to 1 if not provided. Functions h and hp should only be applied to non-negative values x and this is not enforced or checked by the functions.

asinh An asinh function $\boldsymbol{h}(\boldsymbol{x}) = \mathrm{asinh}(\mathrm{para} \cdot \boldsymbol{x}) = \log\left(\mathrm{para} \cdot \boldsymbol{x} + \sqrt{(\mathrm{para} \cdot \boldsymbol{x})^2 + 1}\right)$. Unbounded and takes one parameter. Equivalent to `min_asinh(x, para, Inf)`.

cosh A shifted cosh function $\boldsymbol{h}(\boldsymbol{x}) = \cosh(\mathrm{para} \cdot \boldsymbol{x}) - 1$. Unbounded and takes one parameter. Equivalent to `min_cosh(x, para, Inf)`.

exp A shifted exponential function $\boldsymbol{h}(\boldsymbol{x}) = \exp(\mathrm{para} \cdot \boldsymbol{x}) - 1$. Unbounded and takes one parameter. Equivalent to `min_exp(x, para, Inf)`.

identity The identity function $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{x}$. Unbounded and does not take any parameter. Equivalent to `pow(x, 1)` or `min_pow(x, 1, Inf)`.

log_pow A power function on a log scale $\boldsymbol{h}(\boldsymbol{x}) = \log(1 + \boldsymbol{x})^{\mathrm{para}}$. Unbounded and takes one parameter. Equivalent to `min_log_pow(x, para, Inf)`.

mcp Treating λ=para, γ=para2, the step-wise MCP function applied element-wise: $\lambda x - x^2/(2\gamma)$ if $x \leq \lambda\gamma$, or $\gamma\lambda^2/2$ otherwise. Bounded and takes two parameters.

min_asinh A truncated asinh function applied element-wise: $\min(\mathrm{asinh}(\mathrm{para} \cdot \boldsymbol{x}), \mathrm{para}_2)$. Bounded and takes two parameters.

min_cosh A truncated shifted cosh function applied element-wise: $\min(\cosh(\mathrm{para} \cdot \boldsymbol{x}) - 1, \mathrm{para}_2)$. Bounded and takes two parameters.

min_exp A truncated shifted exponential function applied element-wise: $\boldsymbol{h}(\boldsymbol{x}) = \min(\exp(\mathrm{para} \cdot \boldsymbol{x}) - 1, \mathrm{para}_2)$. Bounded and takes two parameters.

min_log_pow A truncated power on a log scale applied element-wise: $\boldsymbol{h}(\boldsymbol{x}) = \min(\log(1 + \boldsymbol{x}), \mathrm{para}_2)^{\mathrm{para}}$. Bounded and takes two parameters.

`min_pow` A truncated power function applied element-wise: $\boldsymbol{h}(\boldsymbol{x}) = \min(\boldsymbol{x}, \text{para}_2)^{\text{para}}$. Bounded and takes two parameters.

`min_sinh` A truncated sinh function applied element-wise: $\min(\sinh(\text{para} \cdot \boldsymbol{x}), \text{para}_2)$. Bounded and takes two parameters.

`min_softplus` A truncated shifted softplus function applied element-wise: $\min(\log(1+\exp(\text{para} \cdot \boldsymbol{x})) - \log(2), \text{para}_2)$. Bounded and takes two parameters.

`pow` A power function $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{x}^{\text{para}}$. Unbounded and takes two parameter. Equivalent to `min_pow(x, para, Inf)`.

`scad` Treating λ=para, γ=para2, the step-wise SCAD function applied element-wise: λx if $x \leq \lambda$, or $(2\gamma\lambda x - x^2 - \lambda^2)/(2(\gamma - 1))$ if $\lambda < x < \gamma\lambda$, or $\lambda^2(\gamma + 1)/2$ otherwise. Bounded and takes two parameters, where `para2` must be larger than 1, and will be set to 2 by default if not provided.

`sinh` A sinh function $\boldsymbol{h}(\boldsymbol{x}) = \sinh(\text{para} \cdot \boldsymbol{x})$. Unbounded and takes one parameter. Equivalent to `min_sinh(x, para, Inf)`.

`softplus` A shifted softplus function $\boldsymbol{h}(\boldsymbol{x}) = \log(1 + \exp(\text{para} \cdot \boldsymbol{x})) - \log(2)$. Unbounded and takes one parameter. Equivalent to `min_softplus(x, para, Inf)`.

`tanh` A tanh function $\boldsymbol{h}(\boldsymbol{x}) = \tanh(\text{para} \cdot \boldsymbol{x})$. Bounded and takes one parameter.

`truncated_sin` A truncated sin function applied element-wise: $\sin(\text{para} \cdot x)$ if $\text{para} \cdot x \leq \pi/2$, or 1 otherwise. Bounded and takes one parameter.

`truncated_tan` A truncated tan function applied element-wise: $\tan(\text{para} \cdot x)$ if $\text{para} \cdot x \leq \pi/4$, or 1 otherwise. Bounded and takes one parameter.

Value

A list containing two functions `h` and `hp` (h').

Examples

```
get_h_hp("mcp", 2, 4)
get_h_hp("min_log_pow", 1, log(1+3))
get_h_hp("min_pow", 1, 3)
get_h_hp("min_softplus")
```

`get_results` *Estimate* $\mathbf{K}$ *and* $\boldsymbol{\eta}$ *using elts from* `get_elts()` *given one* λ*_*$\mathbf{K}$ *(and* λ*_*$\boldsymbol{\eta}$ *if non-profiled non-centered) and applying warm-start with strong screening rules.*

Description

Estimate $\mathbf{K}$ and $\boldsymbol{\eta}$ using elts from `get_elts()` given one $\lambda_{\mathbf{K}}$ (and $\lambda_{\boldsymbol{\eta}}$ if non-profiled non-centered) and applying warm-start with strong screening rules.

Usage

```
get_results(elts, symmetric, lambda1, lambda2 = 0, tol = 1e-06,
  maxit = 10000, previous_res = NULL, is_refit = FALSE)
```

Arguments

elts	A list, elements necessary for calculations returned by get_elts().
symmetric	A string. If equals "symmetric", estimates the minimizer $\mathbf{K}$ over all symmetric matrices; if "and" or "or", use the "and"/"or" rule to get the support.
lambda1	A number, the penalty parameter for $\mathbf{K}$.
lambda2	A number, the penalty parameter for $\boldsymbol{\eta}$. Default to 0. Cannot be Inf if non-profiled non-centered.
tol	Optional. A number, the tolerance parameter.
maxit	Optional. A positive integer, the maximum number of iterations.
previous_res	Optional. A list or NULL, the returned list by this function run previously with another lambda value.
is_refit	A boolean, in the refit mode for BIC estimation if TRUE. If TRUE, lambda1, previous_lambda1 and lambda2 are all set to 0, and estimation is restricted to entries in exclude that are 0.

Value

converged	A boolean indicating convergence.
crit	A number, the final penalized loss.
edges	A vector of the indices of entries in the K estimate that are non-zero.
eta	A p-vector, the eta estimate. Returned only if elts$centered == FALSE.
eta_support	A vector of the indices of entries in the eta estimate that are non-zero. Returned only if elts$centered == FALSE && elts$profiled_if_noncenter == TRUE.
iters	An integer, number of iterations run.
K	A p*p matrix, the K estimate.
n	An integer, the number of samples.
p	An integer, the dimension.
is_refit,lambda1,maxit,previous_lambda1,symmetric,tol	Same as in the input.
lambda2	Same as in the input, and returned only if elts$centered == FALSE and elts$profiled_if_noncenter == FALSE.

Examples

```
if (!requireNamespace("tmvtnorm", quietly = TRUE)){
  stop("Please install package \"tmvtnorm\" first.", call. = FALSE)
}
require(tmvtnorm)
n <- 50
p <- 30
h_hp <- get_h_hp("min_pow", 1, 3)
mu <- rep(0, p)
K <- diag(p)
dm <- 1 + (1-1/(1+4*exp(1)*max(6*log(p)/n, sqrt(6*log(p)/n))))
x <- tmvtnorm::rtmvnorm(n, mean = mu, sigma = solve(K),
       lower = rep(0, p), upper = rep(Inf, p), algorithm = "gibbs",
       burn.in.samples = 100, thinning = 10)
```

```
elts_NC_NP <- get_elts(h_hp$h, h_hp$hp, x, setting="trun_gaussian",
                centered=FALSE, profiled=FALSE, scale="norm", diag=dm)
test_nc_np <- get_results(elts_NC_NP, symmetric="symmetric", lambda1=0.35,
                lambda2=2, previous_res=NULL, is_refit=FALSE)
test_nc_np2 <- get_results(elts_NC_NP, symmetric="and", lambda1=0.25,
                 lambda2=2, previous_res=test_nc_np, is_refit=FALSE)

elts_NC_P <- get_elts(h_hp$h, h_hp$hp, x, setting="trun_gaussian",
               centered=FALSE, profiled=TRUE, scale="norm", diag=dm)
test_nc_p <- get_results(elts_NC_P, symmetric="symmetric",
               lambda1=0.35, lambda2=NULL, previous_res=NULL, is_refit=FALSE)

elts_C <- get_elts(h_hp$h, h_hp$hp, x, setting="trun_gaussian",
               centered=TRUE, scale="norm", diag=dm)
test_c <- get_results(elts_C, symmetric="or", lambda1=0.35,
               lambda2=NULL, previous_res=NULL, is_refit=FALSE)
```

---

get_safe_log_h_hp *Asymptotic log of* h *and* hp *functions for large* x *for some modes.*

Description

Asymptotic log of h and hp functions for large x for some modes.

Usage

```
get_safe_log_h_hp(mode, para)
```

Arguments

mode	A string, the class of the h function.
para	A number, the first parameter to the h function.

Value

A list of two functions, logh and loghp.

---

get_trun *The truncation point for* h.

Description

The truncation point for h.

Usage

```
get_trun(mode, param1, param2)
```

Arguments

mode	A string, the class of the h function.
param1	A number, the first parameter to the h function.
param2	A number, the second parameter (may be optional depending on mode) to the h function.

Value

Returns the truncation point (the point where h becomes constant and hp becomes 0) for some selected modes.

---

lambda_max *Analytic solution for the minimum λ_$\mathbf{K}$ that gives the empty graph.*

Description

Analytic solution for the minimum $\lambda_{\mathbf{K}}$ that gives the empty graph. In the non-centered setting the bound is not tight, as it is such that both $\mathbf{K}$ and $\boldsymbol{\eta}$ are empty. The bound is also not tight if `symmetric == "and"`.

Usage

```
lambda_max(elts, symmetric, lambda_ratio = Inf)
```

Arguments

elts	A list, elements necessary for calculations returned by `get_elts()`.
symmetric	A string. If equals `"symmetric"`, estimates the minimizer $\mathbf{K}$ over all symmetric matrices; if `"and"` or `"or"`, use the "and"/"or" rule to get the support.
lambda_ratio	A positive number (or `Inf`), the fixed ratio $\lambda_{\mathbf{K}}$ and $\lambda_{\boldsymbol{\eta}}$, if $\lambda_{\boldsymbol{\eta}} \neq 0$ (non-profiled) in the non-centered setting.

Value

A number, the smallest lambda that produces the empty graph in the centered case, or that gives zero solutions for $\mathbf{K}$ and $\boldsymbol{\eta}$ in the non-centered case. If `symmetric == "and"`, it is not a tight bound for the empty graph.

Examples

```
if (!requireNamespace("tmvtnorm", quietly = TRUE)){
  stop("Please install package \"tmvtnorm\" first.", call. = FALSE)
}
require(tmvtnorm)
n <- 50
p <- 30
h_hp <- get_h_hp("min_pow", 1, 3)
mu <- rep(0, p)
K <- diag(p)
dm <- 1 + (1-1/(1+4*exp(1)*max(6*log(p)/n, sqrt(6*log(p)/n))))
x <- tmvtnorm::rtmvnorm(n, mean = mu, sigma = solve(K),
```

```
       lower = rep(0, p), upper = rep(Inf, p), algorithm = "gibbs",
       burn.in.samples = 100, thinning = 10)


elts_NC_NP <- get_elts(h_hp$h, h_hp$hp, x, setting="trun_gaussian",
                centered=FALSE, profiled=FALSE, diag=dm)

# Exact analytic solution for the smallest lambda such that K and eta are both zero,
#  but not a tight bound for K only
lambda_max(elts_NC_NP, "symmetric", 2)
# Use the upper bound as a starting point for numerical search
test_lambda_bounds2(elts_NC_NP, "symmetric", lambda_ratio=2, lower = FALSE,
     lambda_start = lambda_max(elts_NC_NP, "symmetric", 2))

# Exact analytic solution for the smallest lambda such that K and eta are both zero,
#  but not a tight bound for K only
lambda_max(elts_NC_NP, "or", 2)
# Use the upper bound as a starting point for numerical search
test_lambda_bounds2(elts_NC_NP, "or", lambda_ratio=2, lower = FALSE,
     lambda_start = lambda_max(elts_NC_NP, "or", 2))

# An upper bound, not tight.
lambda_max(elts_NC_NP, "and", 2)
# Use the upper bound as a starting point for numerical search
test_lambda_bounds2(elts_NC_NP, "and", lambda_ratio=2, lower = FALSE,
     lambda_start = lambda_max(elts_NC_NP, "and", 2))


elts_NC_P <- get_elts(h_hp$h, h_hp$hp, x, setting="trun_gaussian",
              centered=FALSE, profiled=TRUE, diag=dm)
# Exact analytic solution
lambda_max(elts_NC_P, "symmetric")
# Numerical solution, should be close to the analytic solution
test_lambda_bounds2(elts_NC_P, "symmetric", lambda_ratio=Inf, lower = FALSE,
     lambda_start = NULL)

# Exact analytic solution
lambda_max(elts_NC_P, "or")
# Numerical solution, should be close to the analytic solution
test_lambda_bounds2(elts_NC_P, "or", lambda_ratio=Inf, lower = FALSE,
     lambda_start = NULL)

# An upper bound, not tight
lambda_max(elts_NC_P, "and")
# Use the upper bound as a starting point for numerical search
test_lambda_bounds2(elts_NC_P, "and", lambda_ratio=Inf, lower = FALSE,
     lambda_start = lambda_max(elts_NC_P, "and"))
```

`mu_sigmasqhat` *Estimates the mu and sigma squared parameters from a univariate truncated normal sample.*

Description

Estimates the mu and sigma squared parameters from a univariate truncated normal sample.

Usage

```
mu_sigmasqhat(x, mode, param1, param2, mu = NULL, sigmasq = NULL)
```

Arguments

x	A vector, the data.
mode	A string, the class of the h function.
param1	A number, the first parameter to the h function.
param2	A number, the second parameter (may be optional depending on mode) to the h function.
mu	A number, may be NULL. If NULL, an estimate will be given; otherwise, the value will be treated as the known true mu parameter and is used to calculate an estimate for sigmasq, if sigmasq is NULL.
sigmasq	A number, may be NULL. If NULL, an estimate will be given; otherwise, the value will be treated as the known true sigmasq parameter and is used to calculate an estimate for mu, if mu is NULL.

Details

If both mu and sigmasq are provided, they are returned immediately. If neither is provided, the estimates are given as

$$[1/\sigma^2, \mu/\sigma^2] = \left\{\sum_{i=1}^{n} h(X_i)[X_i, -1][X_i, -1]^\top\right\}^{-1} \left\{\sum_{i=1}^{n} [h(X_i) + h'(X_i)X_i, -h'(X_i)]\right\}.$$

If only sigmasq is provided, the estimate for mu is given as

$$\sum_{i=1}^{n} [h(X_i)X_i - \sigma^2 h'(X_i)] / \sum_{i=1}^{n} h(X_i).$$

If only mu is given, the estimate for sigmasq is given as

$$\sum_{i=1}^{n} h(X_i)(X_i - \mu)^2 / \sum_{i=1}^{n} [h(X_i) + h'(X_i)(X_i - \mu)].$$

Value

A vector that contains the mu and the sigmasq estimates.

---

output *Helper function for outputting if verbose.*

Description

Helper function for outputting if verbose.

Usage

```
output(out, verbose, verbosetext)
```

Arguments

out	Text string.
verbose	Boolean.
verbosetext	Text string.

Value

If verbose == TRUE, outputs a string that concatenates verbosetext and out.

---

rab_arms_R *Random data generator from general* a-b *distributions.*

Description

Random data generator from general a-b graphs using adaptive rejection metropolis sampling (ARMS).

Usage

```
rab_arms_R(n, a, b, eta, K, seed = NULL, burn_in = 1000,
  thinning = 1000, verbose = TRUE)
```

Arguments

n	A number, number of observations.
a	A number, must be strictly larger than $b/2$.
b	A number, must be >= 0.
eta	A vector, the linear part in the distribution.
K	A strictly co-positive square matrix, the interaction matrix.
seed	Optional. A number, the seed for the random generator.
burn_in	Optional. A positive integer, the number of burn-in samples in ARMS to be discarded.
thinning	Optional. A positive integer, thinning factor in ARMS. Samples are taken at iteration steps $\mathrm{burn\_in} + 1$, $\mathrm{burn\_in} + 1 + \mathrm{thinning}$, ..., $\mathrm{burn\_in} + 1 + (n-1) * \mathrm{thinning}$. Default to 1000.
verbose	Optional. A boolean. If TRUE, prints a progress bar showing the progress. Defaults to TRUE.

Details

Randomly generates n samples from the p-variate a-b distributions with parameters $\boldsymbol{\eta}$ and $\mathbf{K}$, where p is the length of $\boldsymbol{\eta}$ or the dimension of the square matrix $\mathbf{K}$.

The a-b distribution is proportional to

$$\exp\left(-\frac{1}{2a}\boldsymbol{x}^{a\top}\mathbf{K}\boldsymbol{x}^{a} + \boldsymbol{\eta}^{\top}\frac{\boldsymbol{x}^{b}-\mathbf{1}_{p}}{b}\right)$$

on the non-negative orthant of $\mathbf{R}^{p}$.

Value

An $n \times p$ matrix of samples, where p is the length of `eta`.

Examples

```
p <- 30
set.seed(1)
eta <- stats::rnorm(p)*0.5
K <- cov_cons("er", p, seed = 3, spars = 0.05, eig = 0.1)
rab_arms_R(n=100, a=1.5, b=2.3, eta=eta, K=K,
  seed=1, burn_in=100, thinning=1000, verbose=TRUE)
rab_arms_R(n=100, a=1.2, b=0.9, eta=eta, K=K,
  seed=2, burn_in=200, thinning=2000, verbose=TRUE)
```

---

ran_mat *Random generator of matrices with given eigenvalues.*

Description

Random generator of matrices with given eigenvalues.

Usage

```
ran_mat(p, ev = stats::runif(p, 0, 10), seed = NULL)
```

Arguments

p	A positive integer >= 2, the dimension.
ev	A vector of length p, the eigenvalues of the output matrix.
seed	A number, the seed for the generator. Ignored if NULL.

Details

The function randomly fills a p by p matrix with independent $Normal(0, 1)$ entries, takes the Q matrix from its QR decomposition, and returns $Q' diag(ev) Q$.

Value

A p by p matrix whose eigenvalues equal to ev.

Examples

```
p <- 30
eigen_values <- (0.1*p-1+1:p)/p
K <- ran_mat(p, ev=eigen_values, seed=1)
sort(eigen(K)$val)-eigen_values
```

refit *Loss for a refitted (restricted) unpenalized model*

Description

Refits an unpenalized model restricted to the estimated edges, with `lambda1=0`, `lambda2=0` and `diagonal_multiplier=1`. Returns `Inf` if no unique solution exists (when the loss is unbounded from below or has infinitely many minimizers).

Usage

```
refit(res, elts)
```

Arguments

`res`	A list of results returned by `get_results()`.
`elts`	A list, elements necessary for calculations returned by `get_elts()`.

Details

Currently the function only returns `Inf` when the maximum node degree is >= the sample size, which is a sufficient and necessary condition for inexistence of a unique solution with probability 1 if `symmetric != "symmetric"`. In practice it is also a sufficient and necessary condition for most cases and `symmetric == "symmetric"`.

Value

A number, the loss of the refitted model.

Examples

```
if (!requireNamespace("tmvtnorm", quietly = TRUE)){
  stop("Please install package \"tmvtnorm\" first.", call. = FALSE)
}
require(tmvtnorm)
n <- 50
p <- 30
h_hp <- get_h_hp("min_pow", 1, 3)
mu <- rep(0, p)
K <- diag(p)
dm <- 1 + (1-1/(1+4*exp(1)*max(6*log(p)/n, sqrt(6*log(p)/n))))
x <- tmvtnorm::rtmvnorm(n, mean = mu, sigma = solve(K),
       lower = rep(0, p), upper = rep(Inf, p), algorithm = "gibbs",
       burn.in.samples = 100, thinning = 10)

elts_NC_NP <- get_elts(h_hp$h, h_hp$hp, x, setting="trun_gaussian",
                centered=FALSE, profiled=FALSE, diag=dm)
res_nc_np <- get_results(elts_NC_NP, symmetric="symmetric",
               lambda1=0.35, lambda2=2, previous_res=NULL, is_refit=FALSE)
refit(res_nc_np, elts_NC_NP)

elts_NC_P <- get_elts(h_hp$h, h_hp$hp, x, setting="trun_gaussian",
               centered=FALSE, profiled=TRUE, diag=dm)
```

```
res_nc_p <- get_results(elts_NC_P, symmetric="symmetric",
              lambda1=0.35, lambda2=NULL, previous_res=NULL, is_refit=FALSE)
refit(res_nc_p, elts_NC_P)

elts_C <- get_elts(h_hp$h, h_hp$hp, x, setting="trun_gaussian",
            centered=TRUE, diag=dm)
res_c <- get_results(elts_C, symmetric="or", lambda1=0.35,
           lambda2=NULL, previous_res=NULL, is_refit=FALSE)
refit(res_c, elts_C)
```

`rexp_gamma_reject_R` *Random data generator from exp or gamma graphs.*

Description

Random data generator from exponential square-root ($a = 0.5, b = 0.5$) or gamma ($a = 0.5, b = 0$) graphs using basic rejection sampling.

Usage

```
rexp_gamma_reject_R(n, gamm, eta, K, seed = NULL, burn_in = 1000,
  thinning = 1000, max_iter = 1e+05, verbose = TRUE)
```

Arguments

`n`	A number, number of observations.
`gamm`	A boolean, generate from gamma (`TRUE`) or exponential square-root (`FALSE`) graphs.
`eta`	A vector, the linear part in the distribution.
`K`	A strictly co-positive square matrix, the interaction matrix.
`seed`	Optional. A number, the seed for the random generator.
`burn_in`	Optional. A positive integer, the number of burn-in samples in Gibbs sampling to be discarded.
`thinning`	Optional. A positive integer, thinning factor in Gibbs sampling. Samples are taken at iteration steps $\text{burn\_in}+1$, $\text{burn\_in}+1+\text{thinning}$, ..., $\text{burn\_in}+1+(n-1)*\text{thinning}$. Default to `1000`.
`max_iter`	Optional. A positive integer, the maximum number of proposals for each sample. Default to `100000`.
`verbose`	Optional. A boolean. If `TRUE`, prints a progress bar showing the progress. Defaults to `TRUE`.

Details

Note: rab_arms_R() with a=0.5 and b=0.5 (exp) or b=0 (gamma) may be a more stable generator.

Randomly generates `n` samples from the `p`-variate exponential square-root or gamma distributions with parameters `eta` and `K`, where `p` is the length of `eta` or the dimension of the square matrix `K`.

The exponential square-root distribution is proportional to

$$\exp\left(-\sqrt{\boldsymbol{x}}^\top \mathbf{K}\sqrt{\boldsymbol{x}} + 2\boldsymbol{\eta}^\top\sqrt{\boldsymbol{x}}\right)$$

on the non-negative orthant of $\mathbf{R}^p$.

The gamma distribution is proportional to

$$\exp\left(-\sqrt{\boldsymbol{x}}^\top \mathbf{K}\sqrt{\boldsymbol{x}} + \boldsymbol{\eta}^\top \log(\boldsymbol{x})\right)$$

on the non-negative orthant of $\mathbf{R}^p$.

Value

An $n \times p$ matrix of samples, where p is the length of `eta`.

Examples

```
p <- 30
set.seed(1)
eta <- stats::rnorm(p)*0.5
eta[eta <= -1] <- abs(eta[eta <= -1])
K <- cov_cons("er", p, seed = 3, spars = 0.05, eig = 0.1)
rexp_gamma_reject_R(n=1000, gamm=TRUE, eta=eta, K=K, seed=1,
  burn_in=100, thinning=1000, max_iter=1e+05, verbose=TRUE)
rexp_gamma_reject_R(n=1000, gamm=FALSE, eta=eta, K=K, seed=2,
  burn_in=200, thinning=2000, max_iter=1e+05, verbose=TRUE)
```

---

test_lambda_bounds *Searches for a tight bound for λ_$\boldsymbol{K}$ that gives the empty or complete graph starting from a given lambda with a given step size*

Description

Searches for the smallest lambda that gives the empty graph (if `lower == FALSE`) or the largest that gives the complete graph (if `lower == TRUE`) starting from the given lambda, each time updating by multiplying or dividing by `step` depending on the search direction.

Usage

```
test_lambda_bounds(elts, symmetric, lambda = 1, lambda_ratio = 1,
  step = 2, lower = TRUE, verbose = TRUE, tol = 1e-06,
  maxit = 10000, cur_res = NULL)
```

Arguments

`elts`	A list, elements necessary for calculations returned by `get_elts()`.
`symmetric`	A string. If equals `"symmetric"`, estimates the minimizer $\mathbf{K}$ over all symmetric matrices; if `"and"` or `"or"`, use the "and"/"or" rule to get the support
`lambda`	A number, the initial searching point for $\lambda_{\mathbf{K}}$.
`lambda_ratio`	A positive number (or `Inf`), the fixed ratio $\lambda_{\mathbf{K}}$ and $\lambda_{\boldsymbol{\eta}}$, if $\lambda_{\boldsymbol{\eta}} \neq 0$ (non-profiled) in the non-centered setting.
`step`	A number, the multiplicative constant applied to lambda at each iteration. Must be strictly larger than 1.

lower	A boolean. If TRUE, finds the largest possible lambda that gives the complete graph (a $lower$ bound). If FALSE, finds the smallest possible lambda that gives the empty graph (an $upper$ bound).
verbose	Optional. A boolean. If TRUE, prints out the lambda value at each iteration.
tol	Optional. A number, the tolerance parameter.
maxit	Optional. A positive integer, the maximum number of iterations in model fitting for each lambda.
cur_res	Optional. A list, current results returned from a previous lambda. If provided, used as a warm start. Default to NULL.

Value

lambda	A number, the best lambda that produces the desired number of edges. 1e-10 or 1e15 is returned if out of bound.
cur_res	A list, results for this lambda. May be NULL if lambda is out of bound.

Examples

```
if (!requireNamespace("tmvtnorm", quietly = TRUE)){
  stop("Please install package \"tmvtnorm\" first.", call. = FALSE)
}
#require(tmvtnorm)
#n <- 50
#p <- 30
#h_hp <- get_h_hp("min_pow", 1, 3)
#mu <- rep(0, p)
#K <- diag(p)
#dm <- 1 + (1-1/(1+4*exp(1)*max(6*log(p)/n, sqrt(6*log(p)/n))))
#x <- tmvtnorm::rtmvnorm(n, mean = mu, sigma = solve(K),
#       lower = rep(0, p), upper = rep(Inf, p), algorithm = "gibbs",
#       burn.in.samples = 100, thinning = 10)

#elts_NC_NP <- get_elts(h_hp$h, h_hp$hp, x, setting="trun_gaussian",
#                centered=FALSE, profiled=FALSE, diag=dm)
#lambda_cur_res <- test_lambda_bounds(elts_NC_NP, "symmetric", lambda=1,
#                       lambda_ratio=1, step=1.5, lower=TRUE, cur_res=NULL)
#lambda_cur_res2 <- test_lambda_bounds(elts_NC_NP, "symmetric", lambda=1,
#                   lambda_ratio=1, step=1.5, lower=FALSE, cur_res=lambda_cur_res$cur_res)
```

---

test_lambda_bounds2 *Searches for a tight bound for $\lambda\_\boldsymbol{K}$ that gives the empty or complete graph starting from a given lambda*

Description

Searches for the smallest lambda that gives the empty graph (if lower == FALSE) or the largest that gives the complete graph (if lower == TRUE) starting from the given lambda.

Usage

```
test_lambda_bounds2(elts, symmetric, lambda_ratio = Inf, lower = TRUE,
  verbose = TRUE, tol = 1e-06, maxit = 10000, lambda_start = NULL)
```

Arguments

elts	A list, elements necessary for calculations returned by get_elts().
symmetric	A string. If equals "symmetric", estimates the minimizer $\mathbf{K}$ over all symmetric matrices; if "and" or "or", use the "and"/"or" rule to get the support
lambda_ratio	A positive number (or Inf), the fixed ratio $\lambda_{\mathbf{K}}$ and $\lambda_{\boldsymbol{\eta}}$, if $\lambda_{\boldsymbol{\eta}} \neq 0$ (non-profiled) in the non-centered setting.
lower	A boolean. If TRUE, finds the largest possible lambda that gives the complete graph (a $lower$ bound). If FALSE, finds the smallest possible lambda that gives the empty graph (an $upper$ bound).
verbose	Optional. A boolean. If TRUE, prints out the lambda value at each iteration.
tol	Optional. A number, the tolerance parameter.
maxit	Optional. A positive integer, the maximum number of iterations in model fitting for each lambda.
lambda_start	Optional. A number, the starting point for searching. If NULL, set to 1e-4 if lower == TRUE, or 1 if lower == FALSE.

Details

This function calls test_lambda_bounds three times with step set to 10, 10^(1/4), 10^(1/16), respectively.

Value

A number, the best lambda that produces the desired number of edges. 1e-10 or 1e15 is returned if out of bound.

Examples

```
if (!requireNamespace("tmvtnorm", quietly = TRUE)){
  stop("Please install package \"tmvtnorm\" first.", call. = FALSE)
}
require(tmvtnorm)
n <- 50
p <- 30
h_hp <- get_h_hp("min_pow", 1, 3)
mu <- rep(0, p)
K <- diag(p)
dm <- 1 + (1-1/(1+4*exp(1)*max(6*log(p)/n, sqrt(6*log(p)/n))))
x <- tmvtnorm::rtmvnorm(n, mean = mu, sigma = solve(K),
       lower = rep(0, p), upper = rep(Inf, p), algorithm = "gibbs",
       burn.in.samples = 100, thinning = 10)

elts_NC_NP <- get_elts(h_hp$h, h_hp$hp, x, setting="trun_gaussian",
                centered=FALSE, profiled=FALSE, diag=dm)
test_lambda_bounds2(elts_NC_NP, "symmetric", lambda_ratio=2,
     lower=TRUE, lambda_start=NULL)
test_lambda_bounds2(elts_NC_NP, "symmetric", lambda_ratio=2,
     lower=FALSE, lambda_start=1.0)
```

tp_fp *Calculates the true and false positive rates given the estimated and true edges.*

Description

Calculates the true and false positive rates given the estimated and true edges.

Usage

```
tp_fp(edges, true_edges, p)
```

Arguments

edges	A vector of indices corresponding to the estimated edges. Should not contain the diagonals.
true_edges	A vector of indices corresponding to the true edges.
p	A positive integer, the dimension.

Value

A vector containing the true positive rate and the false positive rate.

Examples

```
n <- 40
p <- 50
mu <- rep(0, p)
tol <- 1e-8
K <- cov_cons(mode="sub", p=p, seed=1, spars=0.2, eig=0.1, subgraphs=10)
true_edges <- which(abs(K) > tol & diag(p) == 0)
dm <- 1 + (1-1/(1+4*exp(1)*max(6*log(p)/n, sqrt(6*log(p)/n))))
set.seed(1)
x <- tmvtnorm::rtmvnorm(n, mean = mu, sigma = solve(K),
       lower = rep(0, p), upper = rep(Inf, p), algorithm = "gibbs",
       burn.in.samples = 100, thinning = 10)
est <- estimate(x, "trun_gaussian", elts=NULL, centered=TRUE,
         symmetric="symmetric", lambda_length=100, mode="min_pow",
         param1=1, param2=3, diagonal_multiplier=dm,)
# Apply tp_fp to each estimated edges set for each lambda
TP_FP <- t(sapply(est$edgess, function(edges){tp_fp(edges, true_edges, p)}))
par(mfrow=c(1,1), mar=c(5,5,5,5))
plot(c(), c(),  ylim=c(0,1), xlim=c(0,1), cex.lab=1, main = "ROC curve",
  xlab="False Positives", ylab="True Positives")
points(TP_FP[,2], TP_FP[,1], type="l")
points(c(0,1), c(0,1), type = "l", lty = 2)
```

varhat — *Asymptotic variance (times* n*) of the estimator for* mu *or* sigmasq *for the univariate truncated normal assuming the other parameter is known.*

Description

Asymptotic variance (times n) of the estimator for mu or sigmasq for the univariate truncated normal assuming the other parameter is known.

Usage

```
varhat(mu, sigmasq, mode, param1, param2, est_mu)
```

Arguments

mu	A number, the true mu parameter.
sigmasq	A number, the true sigmasq parameter.
mode	A string, the class of the h function.
param1	A number, the first parameter to the h function.
param2	A number, the second parameter (may be optional depending on mode) to the h function.
est_mu	A boolean. If TRUE, returns the asymptotic variance of muhat assuming sigmasq is known; if FALSE, returns the asymptotic variance of sigmasqhat assuming mu is known.

Details

The estimates may be off from the empirical variance, or may even be Inf or NaN if "mode" is one of "cosh", "exp", and "sinh") as the functions grow too fast. If est_mu == TRUE, the function numerically calculates

$$E\left[\sigma^2 h^2(X) + \sigma^4 h'^2(X)\right] / E^2[h(X)],$$

and if est_mu == FALSE, the function numerically calculates

$$E\left[\left(2\sigma^6 h^2(X) + \sigma^8 h'^2(X)\right)(X-\mu)^2\right] / E^2\left[h(X)(X-\mu)^2\right],$$

where E is the expectation over the true distribution $TN(\mu, \sigma)$ of X.

Value

A number, the asymptotic variance.

Examples

```
varhat(0, 1, "min_log_pow", 1, 1, TRUE)
varhat(0.5, 4, "min_pow", 1, 1, TRUE)
```

Index