Package 'pedmod'

September 11, 2022

Type Package

Title Pedigree Models

Version 0.2.4

Maintainer Benjamin Christoffersen <boennecd@gmail.com>

Description Provides functions to estimate mixed probit models using, for instance, pedigree data like in <doi:10.1002/sim.1603>. The models are also commonly called liability threshold models. The approximation is based on direct log marginal likelihood approximations like the randomized Quasi-Monte Carlo suggested by <doi:10.1198/106186002394> with a similar procedure to approximate the derivatives. The minimax tilting method suggested by <doi:10.1111/rssb.12162> is also supported. Graph-based methods are also provided that can be used to simplify pedigrees.

License GPL-3

Encoding UTF-8

RoxygenNote 7.2.0

URL https://github.com/boennecd/pedmod

BugReports https://github.com/boennecd/pedmod/issues

LinkingTo Rcpp, RcppArmadillo, BH, testthat, psqn

Imports Rcpp, alabama

Suggests testthat, mvtnorm, xml2, knitr, rmarkdown, R.rsp, abind, kinship2, igraph, TruncatedNormal, numDeriv

Depends R (>= 3.5.0)

VignetteBuilder R.rsp

SystemRequirements C++17

NeedsCompilation yes

Author Benjamin Christoffersen [cre, aut] (<https://orcid.org/0000-0002-7182-1346>), Alan Genz [cph], Frank Bretz [cph], Bjoern Bornkamp [cph],

Torsten Hothorn [cph],
Christophe Dutang [cph],
Diethelm Wuertz [cph],
R-core [cph],
Leo Belzile [cph],
Zdravko Botev [cph]

Repository CRAN

Date/Publication 2022-09-11 10:00:02 UTC

R topics documented:

`biconnected_components`

Finds the Biconnected Components

Description

Finds the biconnected components and the cut vertices (articulation points) using the methods suggested by Hopcroft et al. (1973).

Usage

```
biconnected_components(from, to)

biconnected_components_pedigree(id, father.id, mother.id)
```

Arguments

from	integer vector with one of the vertex ids.
to	integer vector with one of the vertex ids.
id	integer vector with the child id.
father.id	integer vector with the father id. May be NA if it is missing.
mother.id	integer vector with the mother id. May be NA if it is missing.

Value

A list with vectors of vertices in each biconnected component. An attribute called "cut_verices" contains the cut vertices in each biconnected component.

References

Hopcroft, J., & Tarjan, R. (1973). *Algorithm 447: efficient algorithms for graph manipulation*. Communications of the ACM, 16(6), 372-378.

See Also

block_cut_tree and max_balanced_partition.

Examples

```
# example of a data set in pedigree and graph form
library(pedmod)
dat_pedigree <- data.frame(
  id = 1:48,
  mom = c(
    NA, NA, 2L, 2L, 2L, NA, NA, 7L, 7L, 7L, 3L, 3L, 3L, 3L, NA, 15L, 15L, 43L,
    18L, NA, NA, 21L, 21L, 9L, 9L, 9L, 9L, NA, NA, 29L, 29L, 29L, 30L, 30L, NA,
    NA, 36L, 36L, 36L, 38L, 38L, NA, NA, 43L, 43L, 43L, 32L, 32L),
  dad = c(NA, NA, 1L, 1L, 1L, NA, NA, 6L, 6L, 6L, 8L, 8L, 8L, 8L, NA, 4L, 4L,
          42L, 5L, NA, NA, 20L, 20L, 22L, 22L, 22L, 22L, NA, NA, 28L, 28L, 28L,
          23L, 23L, NA, NA, 35L, 35L, 35L, 31L, 31L, NA, NA, 42L, 42L, 42L,
          45L, 45L),
  sex = c(1L, 2L, 2L, 1L, 1L, 1L, 2L, 1L, 2L, 2L, 2L, 1L, 1L, 1L, 2L, 2L, 2L,
          2L, 1L, 1L, 2L, 1L, 1L, 2L, 1L, 1L, 2L, 1L, 2L, 2L, 1L, 2L, 2L, 2L,
          1L, 2L, 1L, 2L, 1L, 2L, 1L, 1L, 2L, 2L, 1L, 1L, 2L, 2L))

dat <- list(
  to = c(
    3L, 4L, 5L, 8L, 9L, 10L, 11L, 12L, 13L, 14L, 16L, 17L, 18L, 19L, 22L, 23L,
    24L, 25L, 26L, 27L, 30L, 31L, 32L, 33L, 34L, 37L, 38L, 39L, 40L, 41L, 44L,
    45L, 46L, 47L, 48L, 3L, 4L, 5L, 8L, 9L, 10L, 11L, 12L, 13L, 14L, 16L, 17L,
    18L, 19L, 22L, 23L, 24L, 25L, 26L, 27L, 30L, 31L, 32L, 33L, 34L, 37L, 38L,
    39L, 40L, 41L, 44L, 45L, 46L, 47L, 48L),
  from = c(
    1L, 1L, 1L, 6L, 6L, 6L, 8L, 8L, 8L, 8L, 4L, 4L, 42L, 5L, 20L, 20L, 22L, 22L,
    22L, 22L, 28L, 28L, 28L, 23L, 23L, 35L, 35L, 35L, 31L, 31L, 42L, 42L, 42L,
    45L, 45L, 2L, 2L, 2L, 7L, 7L, 7L, 3L, 3L, 3L, 3L, 15L, 15L, 43L, 18L, 21L,
```

```
    21L, 9L, 9L, 9L, 9L, 29L, 29L, 29L, 30L, 30L, 36L, 36L, 36L, 38L, 38L, 43L,
    43L, 43L, 32L, 32L))

# they give the same
out_pedigree <- biconnected_components_pedigree(
  id = dat_pedigree$id, father.id = dat_pedigree$dad,
  mother.id = dat_pedigree$mom)
out <- biconnected_components(dat$to, dat$from)
all.equal(out_pedigree, out)
```

`block_cut_tree` *Creates a Block-cut Tree Like Object*

Description

Creates a block-cut tree like structure computed using the method suggested by Hopcroft et al. (1973).

Usage

```
block_cut_tree(from, to)

block_cut_tree_pedigree(id, father.id, mother.id)
```

Arguments

`from`	integer vector with one of the vertex ids.
`to`	integer vector with one of the vertex ids.
`id`	integer vector with the child id.
`father.id`	integer vector with the father id. May be `NA` if it is missing.
`mother.id`	integer vector with the mother id. May be `NA` if it is missing.

Value

A tree structure where each node is represented as list that contains the vertices in the biconnected component, the cut_vertices, and the node's leafs.

References

Hopcroft, J., & Tarjan, R. (1973). *Algorithm 447: efficient algorithms for graph manipulation*. Communications of the ACM, 16(6), 372-378.

See Also

`biconnected_components` and `max_balanced_partition`.

Examples

```
# example of a data set in pedigree and graph form
library(pedmod)
dat_pedigree <- data.frame(
  id = 1:48,
  mom = c(
    NA, NA, 2L, 2L, 2L, NA, NA, 7L, 7L, 7L, 3L, 3L, 3L, 3L, NA, 15L, 15L, 43L,
    18L, NA, NA, 21L, 21L, 9L, 9L, 9L, 9L, NA, NA, 29L, 29L, 29L, 30L, 30L, NA,
    NA, 36L, 36L, 36L, 38L, 38L, NA, NA, 43L, 43L, 43L, 32L, 32L),
  dad = c(NA, NA, 1L, 1L, 1L, NA, NA, 6L, 6L, 6L, 8L, 8L, 8L, 8L, NA, 4L, 4L,
          42L, 5L, NA, NA, 20L, 20L, 22L, 22L, 22L, 22L, NA, NA, 28L, 28L, 28L,
          23L, 23L, NA, NA, 35L, 35L, 35L, 31L, 31L, NA, NA, 42L, 42L, 42L,
          45L, 45L),
  sex = c(1L, 2L, 2L, 1L, 1L, 1L, 2L, 1L, 2L, 2L, 2L, 1L, 1L, 1L, 2L, 2L, 2L,
          2L, 1L, 1L, 2L, 1L, 1L, 2L, 1L, 1L, 2L, 1L, 2L, 2L, 1L, 2L, 2L, 2L,
          1L, 2L, 1L, 2L, 1L, 2L, 1L, 1L, 2L, 2L, 1L, 1L, 2L, 2L))

dat <- list(
  to = c(
    3L, 4L, 5L, 8L, 9L, 10L, 11L, 12L, 13L, 14L, 16L, 17L, 18L, 19L, 22L, 23L,
    24L, 25L, 26L, 27L, 30L, 31L, 32L, 33L, 34L, 37L, 38L, 39L, 40L, 41L, 44L,
    45L, 46L, 47L, 48L, 3L, 4L, 5L, 8L, 9L, 10L, 11L, 12L, 13L, 14L, 16L, 17L,
    18L, 19L, 22L, 23L, 24L, 25L, 26L, 27L, 30L, 31L, 32L, 33L, 34L, 37L, 38L,
    39L, 40L, 41L, 44L, 45L, 46L, 47L, 48L),
  from = c(
    1L, 1L, 1L, 6L, 6L, 6L, 8L, 8L, 8L, 8L, 4L, 4L, 42L, 5L, 20L, 20L, 22L, 22L,
    22L, 22L, 28L, 28L, 28L, 23L, 23L, 35L, 35L, 35L, 31L, 31L, 42L, 42L, 42L,
    45L, 45L, 2L, 2L, 2L, 7L, 7L, 7L, 3L, 3L, 3L, 3L, 15L, 15L, 43L, 18L, 21L,
    21L, 9L, 9L, 9L, 9L, 29L, 29L, 29L, 30L, 30L, 36L, 36L, 36L, 38L, 38L, 43L,
    43L, 43L, 32L, 32L))

# they give the same
out_pedigree <- block_cut_tree_pedigree(
  id = dat_pedigree$id, father.id = dat_pedigree$dad,
  mother.id = dat_pedigree$mom)
out <- block_cut_tree(dat$to, dat$from)
all.equal(out_pedigree, out)
```

---

eval_pedigree_ll *Approximate the Log Marginal Likelihood*

Description

Approximate the log marginal likelihood and the derivatives with respect to the model parameters.

Usage

```
eval_pedigree_ll(
```

```
  ptr,
  par,
  maxvls,
  abs_eps,
  rel_eps,
  indices = NULL,
  minvls = -1L,
  do_reorder = TRUE,
  use_aprx = FALSE,
  n_threads = 1L,
  cluster_weights = NULL,
  standardized = FALSE,
  method = 0L,
  use_tilting = FALSE,
  vls_scales = NULL
)

eval_pedigree_grad(
  ptr,
  par,
  maxvls,
  abs_eps,
  rel_eps,
  indices = NULL,
  minvls = -1L,
  do_reorder = TRUE,
  use_aprx = FALSE,
  n_threads = 1L,
  cluster_weights = NULL,
  standardized = FALSE,
  method = 0L,
  use_tilting = FALSE,
  vls_scales = NULL
)

eval_pedigree_hess(
  ptr,
  par,
  maxvls,
  abs_eps,
  rel_eps,
  indices = NULL,
  minvls = -1L,
  do_reorder = TRUE,
  use_aprx = FALSE,
  n_threads = 1L,
  cluster_weights = NULL,
  standardized = FALSE,
```

```
  method = 0L,
  use_tilting = FALSE,
  vls_scales = NULL
)
```

Arguments

ptr	object from pedigree_ll_terms or pedigree_ll_terms_loadings.
par	numeric vector with parameters. For an object from pedigree_ll_terms these are the fixed effect coefficients and log scale parameters. The log scale parameters should be last. For an object from pedigree_ll_terms_loadings these are the fixed effects and the coefficients for scale parameters.
maxvls	maximum number of samples in the approximation for each marginal likelihood term.
abs_eps	absolute convergence threshold.
rel_eps	relative convergence threshold.
indices	zero-based vector with indices of which log marginal likelihood terms to include. Use NULL if all indices should be used.
minvls	minimum number of samples for each marginal likelihood term. Negative values provides a default which depends on the dimension of the integration.
do_reorder	TRUE if a heuristic variable reordering should be used. TRUE is likely the best value.
use_aprx	TRUE if a less precise approximation of pnorm and qnorm should be used. This may reduce the computation time while not affecting the result much.
n_threads	number of threads to use.
cluster_weights	numeric vector with weights for each cluster. Use NULL if all clusters have weight one.
standardized	logical for whether to use the standardized or direct parameterization. See standardized_to_direct and the vignette at vignette("pedmod", package = "pedmod").
method	integer with the method to use. Zero yields randomized Korobov lattice rules while one yields scrambled Sobol sequences.
use_tilting	TRUE if the minimax tilting method suggested by Botev (2017) should be used. See doi:10.1111/rssb.12162.
vls_scales	can be a numeric vector with a positive scalar for each cluster. Then vls_scales[i] * minvls and vls_scales[i] * maxvls is used for cluster i rather than minvls and maxvls. Set vls_scales = NULL if the latter should be used.

Details

eval_pedigree_hess is only implemented for objects from pedigree_ll_terms.

Value

`eval_pedigree_ll`: a scalar with the log marginal likelihood approximation. It has an attribute called `"n_fails"` which shows the number of log marginal likelihood term approximations which do not satisfy the `abs_eps` and `rel_eps` criteria and an attribute called `std` with a standard error estimate based on the delta rule.

`eval_pedigree_grad`: a vector with the derivatives with respect to `par`. An attribute called `"logLik"` contains the log marginal likelihood approximation. There will also be `"n_fails"` attribute like for `eval_pedigree_ll` and an attribute called `"std"` which first element is the standard error estimate of the log likelihood based on the delta method and the last elements are the standard error estimates of the gradient. The latter ignores the Monte Carlo error from the likelihood approximation.

`eval_pedigree_hess`: a matrix with the hessian with respect to `par`. An attribute called `"logLik"` contains the log marginal likelihood approximation and an attribute called `"grad"` contains the gradient· The attribute `"hess_org"` contains the Hessian with the scale parameter on the identity scale rather than the log scale. `"vcov"` and `"vcov_org"` are the covariance matrices from the hessian and `"hess_org"`.

Examples

```
# three families as an example
fam_dat <- list(
  list(
    y = c(FALSE, TRUE, FALSE, FALSE),
    X = structure(c(
      1, 1, 1, 1, 1.2922654151273, 0.358134905909256, -0.734963997107464,
      0.855235473516044, -1.16189500386223, -0.387298334620742,
      0.387298334620742, 1.16189500386223),
      .Dim = 4:3, .Dimnames = list( NULL, c("(Intercept)", "X1", ""))),
    rel_mat = structure(c(
      1, 0.5, 0.5, 0.125, 0.5, 1, 0.5, 0.125, 0.5, 0.5,
      1, 0.125, 0.125, 0.125, 0.125, 1), .Dim = c(4L, 4L)),
    met_mat = structure(c(1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1),
                        .Dim = c(4L, 4L))),
  list(
    y = c(FALSE, FALSE, FALSE),
    X = structure(c(
      1, 1, 1, -0.0388728997202442, -0.0913782435233639,
      -0.0801619722392612, -1, 0, 1), .Dim = c(3L, 3L)),
    rel_mat = structure(c(
      1, 0.5, 0.125, 0.5, 1, 0.125, 0.125, 0.125, 1), .Dim = c(3L, 3L)),
    met_mat = structure(c(
      1, 1, 0, 1, 1, 0, 0, 0, 1), .Dim = c(3L, 3L))),
  list(
    y = c(TRUE, FALSE),
    X = structure(c(
      1, 1, 0.305275750370738, -1.49482995913648,  -0.707106781186547,
      0.707106781186547),
      .Dim = 2:3, .Dimnames = list( NULL, c("(Intercept)", "X1", ""))),
    rel_mat = structure(c(1, 0.5,  0.5, 1), .Dim = c(2L, 2L)),
    met_mat = structure(c(1, 1, 1, 1), .Dim = c(2L,  2L))))
```

```
# get the data into the format needed for the package
dat_arg <- lapply(fam_dat, function(x){
  # we need the following for each family:
  #   y: the zero-one outcomes.
  #   X: the design matrix for the fixed effects.
  #   scale_mats: list with the scale matrices for each type of effect.
  list(y = as.numeric(x$y), X = x$X,
       scale_mats = list(x$rel_mat, x$met_mat))
})

# get a pointer to the C++ object
ptr <- pedigree_ll_terms(dat_arg, max_threads = 1L)

# approximate the log marginal likelihood
beta <- c(-1, 0.3, 0.2) # fixed effect coefficients
scs <- c(0.5, 0.33)     # scales parameters

set.seed(44492929)
system.time(ll1 <- eval_pedigree_ll(
  ptr = ptr, par = c(beta, log(scs)), abs_eps = -1, maxvls = 1e5,
  rel_eps = 1e-5, minvls = 2000, use_aprx = FALSE))
ll1 # the approximation

# with the approximation of pnorm and qnorm
system.time(ll2 <- eval_pedigree_ll(
  ptr = ptr, par = c(beta, log(scs)), abs_eps = -1, maxvls = 1e5,
  rel_eps = 1e-5, minvls = 2000, use_aprx = TRUE))
all.equal(ll1, ll2, tolerance = 1e-5)

# cluster weights can be used as follows to repeat the second family three
# times and remove the third
system.time(deriv_w_weight <- eval_pedigree_grad(
  ptr = ptr, par = c(beta, log(scs)), abs_eps = -1, maxvls = 1e6,
  rel_eps = 1e-3, minvls = 2000, use_aprx = TRUE,
  cluster_weights = c(1, 3, 0)))

# the same as manually repeating second cluster and not including the third
dum_dat <- dat_arg[c(1, 2, 2, 2)]
dum_ptr <- pedigree_ll_terms(dum_dat, 1L)
system.time(deriv_dum <- eval_pedigree_grad(
  ptr = dum_ptr, par = c(beta, log(scs)), abs_eps = -1, maxvls = 1e6,
  rel_eps = 1e-3, minvls = 2000, use_aprx = TRUE))
all.equal(deriv_dum, deriv_w_weight, tolerance = 1e-3)

# the hessian is computed on the scale parameter scale rather than on the
# log of the scale parameters
system.time(hess_w_weight <- eval_pedigree_hess(
  ptr = ptr, par = c(beta, log(scs)), abs_eps = -1, maxvls = 1e6,
  rel_eps = 1e-3, minvls = 2000, use_aprx = TRUE,
  cluster_weights = c(1, 3, 0)))

system.time(hess_dum <- eval_pedigree_hess(
```

```
  ptr = dum_ptr, par = c(beta, log(scs)), abs_eps = -1, maxvls = 1e6,
  rel_eps = 1e-3, minvls = 2000, use_aprx = TRUE))
attr(hess_w_weight, "n_fails") <- attr(hess_dum, "n_fails") <- NULL
all.equal(hess_w_weight, hess_dum, tolerance = 1e-3)

# the results are consistent with the gradient output
all.equal(attr(deriv_dum, "logLik"), attr(hess_dum, "logLik"),
          tolerance = 1e-5)

hess_grad <- attr(hess_dum, "grad")
all.equal(hess_grad, deriv_dum, check.attributes = FALSE,
          tolerance = 1e-3)

# with loadings
dat_arg_loadings <- lapply(fam_dat, function(x){
  list(y = as.numeric(x$y), X = x$X, Z = x$X[, 1:2],
       scale_mats = list(x$rel_mat, x$met_mat))
})

ptr_loadings <-
  pedigree_ll_terms_loadings(dat_arg_loadings, max_threads = 1L)

scs <- c(log(0.5) / 2, 0.1, log(0.33) / 2, 0.2) # got more scales parameters
eval_pedigree_ll(
  ptr = ptr_loadings, par = c(beta, scs), abs_eps = -1, maxvls = 1e4,
  rel_eps = 1e-3, minvls = 2000, use_aprx = TRUE)
eval_pedigree_grad(
  ptr = ptr_loadings, par = c(beta, scs), abs_eps = -1, maxvls = 1e4,
  rel_eps = 1e-3, minvls = 2000, use_aprx = TRUE)

# can recover the result from before
scs <- c(log(0.5) / 2, 0, log(0.33) / 2, 0)
ll3 <- eval_pedigree_ll(
  ptr = ptr_loadings, par = c(beta, scs), abs_eps = -1, maxvls = 1e4,
  rel_eps = 1e-3, minvls = 2000, use_aprx = TRUE)
all.equal(ll1, ll3, tolerance = 1e-5)
```

`max_balanced_partition` *Finds an Approximately Balanced Connected Partition*

Description

Uses the method suggested by Chlebíková (1996) to construct an approximate maximally balanced connected partition. A further refinement step can be made to reduce the cost of the cut edges. See `vignette("pedigree_partitioning", package = "pedmod")` for further details.

Usage

```
max_balanced_partition(
  from,
  to,
  weight_data = NULL,
  edge_weights = NULL,
  slack = 0,
  max_kl_it_inner = 50L,
  max_kl_it = 10000L,
  trace = 0L,
  check_weights = TRUE,
  do_reorder = FALSE
)

max_balanced_partition_pedigree(
  id,
  father.id,
  mother.id,
  id_weight = NULL,
  father_weight = NULL,
  mother_weight = NULL,
  slack = 0,
  max_kl_it_inner = 50L,
  max_kl_it = 10000L,
  trace = 0L,
  check_weights = TRUE,
  do_reorder = FALSE
)
```

Arguments

`from`	integer vector with one of the vertex ids.
`to`	integer vector with one of the vertex ids.
`weight_data`	list with two elements called `"id"` for the id and `"weight"` for the vertex weight. All vertices that are not in this list have a weight of one. Use `NULL` if all vertices have a weight of one.
`edge_weights`	numeric vector with weights for each edge. Needs to have the same length as `from` and `to`. Use `NULL` if all edges should have a weight of one.
`slack`	fraction between zero and 0.5 for the allowed amount of deviation from the balance criterion that is allowed to reduce the cost of the cut edges.
`max_kl_it_inner`	maximum number of moves to consider in each iteration when `slack > 0`.
`max_kl_it`	maximum number of iterations to use when reducing the cost of the cut edges. Typically the method converges quickly and this argument is not needed.
`trace`	integer where larger values yields more information printed to the console during the procedure.

check_weights	logical for whether to check the weights in each biconnected component. This may fail if the graph is not connected in which case the results will likely be wrong. It may also fail for large graphs because of floating-point arithmetic. The latter is not an error and the reason for this argument.
do_reorder	logical for whether the implementation should reorder the vertices. This may reduce the computation time for some data sets.
id	integer vector with the child id.
father.id	integer vector with the father id. May be NA if it is missing.
mother.id	integer vector with the mother id. May be NA if it is missing.
id_weight	numeric vector with the weight to use for each vertex (individual). NULL yields a weight of one for all.
father_weight	weights of the edges created between the fathers and the children. Use NULL if all should have a weight of one.
mother_weight	weights of the edges created between the mothers and the children. Use NULL if all should have a weight of one.

Value

A list with the following elements:

balance_criterion	value of the balance criterion.
removed_edges	2D integer matrix with the removed edges.
set_1,set_2	The two sets in the partition.

References

Chlebíková, J. (1996). *Approximating the maximally balanced connected partition problem in graphs*. Information Processing Letters, 60(5), 225-230.

Hopcroft, J., & Tarjan, R. (1973). *Algorithm 447: efficient algorithms for graph manipulation*. Communications of the ACM, 16(6), 372-378.

See Also

biconnected_components, block_cut_tree, and unconnected_partition.

Examples

```
# example of a data set in pedigree and graph form
library(pedmod)
dat_pedigree <- data.frame(
  id = 1:48,
  mom = c(
    NA, NA, 2L, 2L, 2L, NA, NA, 7L, 7L, 7L, 3L, 3L, 3L, 3L, NA, 15L, 15L, 43L,
    18L, NA, NA, 21L, 21L, 9L, 9L, 9L, 9L, NA, NA, 29L, 29L, 29L, 30L, 30L, NA,
    NA, 36L, 36L, 36L, 38L, 38L, NA, NA, 43L, 43L, 43L, 32L, 32L),
  dad = c(NA, NA, 1L, 1L, 1L, NA, NA, 6L, 6L, 6L, 8L, 8L, 8L, 8L, NA, 4L, 4L,
          42L, 5L, NA, NA, 20L, 20L, 22L, 22L, 22L, 22L, NA, NA, 28L, 28L, 28L,
```

```
          23L, 23L, NA, NA, 35L, 35L, 35L, 31L, 31L, NA, NA, 42L, 42L, 42L,
          45L, 45L),
  sex = c(1L, 2L, 2L, 1L, 1L, 1L, 2L, 1L, 2L, 2L, 2L, 1L, 1L, 1L, 2L, 2L, 2L,
          2L, 1L, 1L, 2L, 1L, 1L, 2L, 1L, 1L, 2L, 1L, 2L, 2L, 1L, 2L, 2L, 2L,
          1L, 2L, 1L, 2L, 1L, 2L, 1L, 1L, 2L, 2L, 1L, 1L, 2L, 2L))

dat <- list(
  to = c(
    3L, 4L, 5L, 8L, 9L, 10L, 11L, 12L, 13L, 14L, 16L, 17L, 18L, 19L, 22L, 23L,
    24L, 25L, 26L, 27L, 30L, 31L, 32L, 33L, 34L, 37L, 38L, 39L, 40L, 41L, 44L,
    45L, 46L, 47L, 48L, 3L, 4L, 5L, 8L, 9L, 10L, 11L, 12L, 13L, 14L, 16L, 17L,
    18L, 19L, 22L, 23L, 24L, 25L, 26L, 27L, 30L, 31L, 32L, 33L, 34L, 37L, 38L,
    39L, 40L, 41L, 44L, 45L, 46L, 47L, 48L),
  from = c(
    1L, 1L, 1L, 6L, 6L, 6L, 8L, 8L, 8L, 8L, 4L, 4L, 42L, 5L, 20L, 20L, 22L, 22L,
    22L, 22L, 28L, 28L, 28L, 23L, 23L, 35L, 35L, 35L, 31L, 31L, 42L, 42L, 42L,
    45L, 45L, 2L, 2L, 2L, 7L, 7L, 7L, 3L, 3L, 3L, 3L, 15L, 15L, 43L, 18L, 21L,
    21L, 9L, 9L, 9L, 9L, 29L, 29L, 29L, 30L, 30L, 36L, 36L, 36L, 38L, 38L, 43L,
    43L, 43L, 32L, 32L))

# the results may be different because of different orders!
out_pedigree <- max_balanced_partition_pedigree(
  id = dat_pedigree$id, father.id = dat_pedigree$dad,
  mother.id = dat_pedigree$mom)
out <- max_balanced_partition(dat$to, dat$from)

all.equal(out_pedigree$balance_criterion, out$balance_criterion)
all.equal(out_pedigree$removed_edges, out$removed_edges)
```

`mvndst` *Multivariate Normal Distribution CDF and Its Derivative*

Description

Provides an approximation of the multivariate normal distribution CDF over a hyperrectangle and the derivative with respect to the mean vector and the covariance matrix.

Usage

```
mvndst(
  lower,
  upper,
  mu,
  sigma,
  maxvls = 25000L,
  abs_eps = 0.001,
  rel_eps = 0L,
  minvls = -1L,
```

```
  do_reorder = TRUE,
  use_aprx = FALSE,
  method = 0L,
  n_sequences = 8L,
  use_tilting = FALSE
)

mvndst_grad(
  lower,
  upper,
  mu,
  sigma,
  maxvls = 25000L,
  abs_eps = 0.001,
  rel_eps = 0L,
  minvls = -1L,
  do_reorder = TRUE,
  use_aprx = FALSE,
  method = 0L,
  n_sequences = 8L,
  use_tilting = FALSE
)
```

Arguments

lower	numeric vector with lower bounds.
upper	numeric vector with upper bounds.
mu	numeric vector with means.
sigma	covariance matrix.
maxvls	maximum number of samples in the approximation.
abs_eps	absolute convergence threshold.
rel_eps	relative convergence threshold.
minvls	minimum number of samples. Negative values provides a default which depends on the dimension of the integration.
do_reorder	TRUE if a heuristic variable reordering should be used. TRUE is likely the best value.
use_aprx	TRUE if a less precise approximation of pnorm and qnorm should be used. This may reduce the computation time while not affecting the result much.
method	integer with the method to use. Zero yields randomized Korobov lattice rules while one yields scrambled Sobol sequences.
n_sequences	number of randomized quasi-Monte Carlo sequences to use. More samples yields a better estimate of the error but a worse approximation. Eight is used in the original Fortran code. If one is used then the error will be set to zero because it cannot be estimated.
use_tilting	TRUE if the minimax tilting method suggested by Botev (2017) should be used. See doi:10.1111/rssb.12162.

Value

`mvndst`: An approximation of the CDF. The `"n_it"` attribute shows the number of integrand evaluations, the `"inform"` attribute is zero if the requested precision is achieved, and the `"abserr"` attribute shows 3.5 times the estimated standard error.

`mvndst_grad`: A list with

- `likelihood`: the likelihood approximation.
- `d_mu`: the derivative with respect to the the mean vector.
- `d_sigma`: the derivative with respect to the covariance matrix ignoring the symmetry (i.e. working the n^2 parameters with n being the dimension rather than the $n(n+1)/2$ free parameters).

Examples

```
# simulate covariance matrix and the upper bound
set.seed(1)
n <- 10L
S <- drop(rWishart(1L, 2 * n, diag(n) / 2 / n))
u <- rnorm(n)

system.time(pedmod_res <- mvndst(
    lower = rep(-Inf, n), upper = u, sigma = S, mu = numeric(n),
    maxvls = 1e6, abs_eps = 0, rel_eps = 1e-4, use_aprx = TRUE))
pedmod_res

# compare with mvtnorm
if(require(mvtnorm)){
    mvtnorm_time <- system.time(mvtnorm_res <- mvtnorm::pmvnorm(
        upper = u, sigma = S, algorithm = GenzBretz(
            maxpts = 1e6, abseps = 0, releps = 1e-4)))
    cat("mvtnorm_res:\n")
    print(mvtnorm_res)

    cat("mvtnorm_time:\n")
    print(mvtnorm_time)
}

# with titling
system.time(pedmod_res <- mvndst(
    lower = rep(-Inf, n), upper = u, sigma = S, mu = numeric(n),
    maxvls = 1e6, abs_eps = 0, rel_eps = 1e-4, use_tilting = TRUE))
pedmod_res

# compare with TruncatedNormal
if(require(TruncatedNormal)){
    TruncatedNormal_time <- system.time(
        TruncatedNormal_res <- TruncatedNormal::pmvnorm(
            lb = rep(-Inf, n), ub = u, sigma = S,
            B = attr(pedmod_res, "n_it"), type = "qmc"))
    cat("TruncatedNormal_res:\n")
    print(TruncatedNormal_res)
```

```
    cat("TruncatedNormal_time:\n")
    print(TruncatedNormal_time)
}

# check the gradient
system.time(pedmod_res <- mvndst_grad(
  lower = rep(-Inf, n), upper = u, sigma = S, mu = numeric(n),
  maxvls = 1e5, minvls = 1e5, abs_eps = 0, rel_eps = 1e-4, use_aprx = TRUE))
pedmod_res

# compare with numerical differentiation. Should give the same up to Monte
# Carlo and finite difference error
if(require(numDeriv)){
  num_res <- grad(
    function(par){
      set.seed(1)
      mu <- head(par, n)
      S[upper.tri(S, TRUE)] <- tail(par, -n)
      S[lower.tri(S)] <- t(S)[lower.tri(S)]
      mvndst(
        lower = rep(-Inf, n), upper = u, sigma = S, mu = mu,
        maxvls = 1e4, minvls = 1e4, abs_eps = 0, rel_eps = 1e-4,
        use_aprx = TRUE)
    }, c(numeric(n), S[upper.tri(S, TRUE)]),
    method.args = list(d = .01, r = 2))

  d_mu <- head(num_res, n)
  d_sigma <- matrix(0, n, n)
  d_sigma[upper.tri(d_sigma, TRUE)] <- tail(num_res, -n)
  d_sigma[upper.tri(d_sigma)] <- d_sigma[upper.tri(d_sigma)] / 2
  d_sigma[lower.tri(d_sigma)] <- t(d_sigma)[lower.tri(d_sigma)]

  cat("numerical derivatives\n")
  print(rbind(numDeriv = d_mu,
              pedmod = pedmod_res$d_mu))
  print(d_sigma)
  cat("\nd_sigma from pedmod\n")
  print(pedmod_res$d_sigma) # for comparison
}
```

`pedigree_ll_terms` *Get a C++ Object for Log Marginal Likelihood Approximations*

Description

Constructs an object needed for `eval_pedigree_ll` and `eval_pedigree_grad`.

Usage

```
pedigree_ll_terms(data, max_threads = 1L, n_sequences = 8L)

pedigree_ll_terms_loadings(data, max_threads = 1L, n_sequences = 8L)
```

Arguments

data	list where each element is a list for a cluster with an: • "X" element with the design matrix for the fixed effect, • "Z" element with the design matrix for the loadings of the effects (only needed for pedigree_ll_terms_loadings), • "y" element with the zero-one outcomes, and • "scale_mats" element with a list where each element is a scale/correlation matrix for a particular type of effect.
max_threads	maximum number of threads to use.
n_sequences	number of randomized quasi-Monte Carlo sequences to use. More samples yields a better estimate of the error but a worse approximation. Eight is used in the original Fortran code. If one is used then the error will be set to zero because it cannot be estimated.

Details

An intercept column is not added to the X matrices like what lm.fit and glm.fit do. Thus, it is often important that the user adds an intercept column to these matrices as it is hardly ever justified to not include the intercept (the exceptions being e.g. when splines are used which include the intercept and with certain dummy designs). This equally holds for the Z matrices with pedigree_ll_terms_loadings.

pedigree_ll_terms_loadings relax the assumption that the scale parameter is the same for all individuals. pedigree_ll_terms_loadings and pedigree_ll_terms yield the same model if "Z" is an intercept column for all families but with a different parameterization. In this case, pedigree_ll_terms will be faster. See vignette("pedmod", "pedmod") for examples of using pedigree_ll_terms_loadings.

Examples

```
# three families as an example
fam_dat <- list(
  list(
    y = c(FALSE, TRUE, FALSE, FALSE),
    X = structure(c(
      1, 1, 1, 1, 1.2922654151273, 0.358134905909256, -0.734963997107464,
      0.855235473516044, -1.16189500386223, -0.387298334620742,
      0.387298334620742, 1.16189500386223),
      .Dim = 4:3, .Dimnames = list( NULL, c("(Intercept)", "X1", ""))),
    rel_mat = structure(c(
      1, 0.5, 0.5, 0.125, 0.5, 1, 0.5, 0.125, 0.5, 0.5,
      1, 0.125, 0.125, 0.125, 0.125, 1), .Dim = c(4L, 4L)),
    met_mat = structure(c(1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1),
```

```
                        .Dim = c(4L, 4L))),
  list(
    y = c(FALSE, FALSE, FALSE),
    X = structure(c(
      1, 1, 1, -0.0388728997202442, -0.0913782435233639,
      -0.0801619722392612, -1, 0, 1), .Dim = c(3L, 3L)),
    rel_mat = structure(c(
      1, 0.5, 0.125, 0.5, 1, 0.125, 0.125, 0.125, 1), .Dim = c(3L, 3L)),
    met_mat = structure(c(
      1, 1, 0, 1, 1, 0, 0, 0, 1), .Dim = c(3L, 3L))),
  list(
    y = c(TRUE, FALSE),
    X = structure(c(
      1, 1, 0.305275750370738, -1.49482995913648,  -0.707106781186547,
      0.707106781186547),
      .Dim = 2:3, .Dimnames = list( NULL, c("(Intercept)", "X1", ""))),
    rel_mat = structure(c(1, 0.5,  0.5, 1), .Dim = c(2L, 2L)),
    met_mat = structure(c(1, 1, 1, 1), .Dim = c(2L,  2L))))

# get the data into the format needed for the package
dat_arg <- lapply(fam_dat, function(x){
  # we need the following for each family:
  #   y: the zero-one outcomes.
  #   X: the design matrix for the fixed effects.
  #   scale_mats: list with the scale matrices for each type of effect.
  list(y = as.numeric(x$y), X = x$X,
       scale_mats = list(x$rel_mat, x$met_mat))
})

# get a pointer to the C++ object
ptr <- pedigree_ll_terms(dat_arg, max_threads = 1L)

# get the argument for a the version with loadings
dat_arg_loadings <- lapply(fam_dat, function(x){
  list(y = as.numeric(x$y), X = x$X, Z = x$X[, 1:2],
       scale_mats = list(x$rel_mat, x$met_mat))
})

ptr <- pedigree_ll_terms_loadings(dat_arg_loadings, max_threads = 1L)
```

---

pedmod_opt *Optimize the Log Marginal Likelihood*

Description

Optimizes eval_pedigree_ll and eval_pedigree_grad using a passed optimization function.

Usage

```
pedmod_opt(
  ptr,
  par,
  maxvls,
  abs_eps,
  rel_eps,
  opt_func = NULL,
  seed = 1L,
  indices = NULL,
  minvls = -1L,
  do_reorder = TRUE,
  use_aprx = FALSE,
  n_threads = 1L,
  cluster_weights = NULL,
  fix = NULL,
  standardized = FALSE,
  method = 0L,
  use_tilting = FALSE,
  vls_scales = NULL,
  ...
)

pedmod_start(
  ptr,
  data,
  maxvls = 1000L,
  abs_eps = 0,
  rel_eps = 0.01,
  seed = 1L,
  indices = NULL,
  scale_max = 9,
  minvls = 100L,
  do_reorder = TRUE,
  use_aprx = TRUE,
  n_threads = 1L,
  cluster_weights = NULL,
  standardized = FALSE,
  method = 0L,
  sc_start = NULL,
  use_tilting = FALSE,
  vls_scales = NULL
)

pedmod_start_loadings(
  ptr,
  data,
  indices = NULL,
```

```
  cluster_weights = NULL,
  sc_start_invariant = NULL
)
```

Arguments

ptr	object from pedigree_ll_terms or pedigree_ll_terms_loadings.
par	starting values passed to opt_func.
maxvls	maximum number of samples in the approximation for each marginal likelihood term.
abs_eps	absolute convergence threshold for eval_pedigree_ll and eval_pedigree_grad.
rel_eps	rel_eps convergence threshold for eval_pedigree_ll and eval_pedigree_grad.
opt_func	function to perform minimization with arguments like optim. BFGS is used with optim if this argument is NULL.
seed	seed to pass to set.seed before each gradient and function evaluation. Use NULL if the seed should not be fixed.
indices	zero-based vector with indices of which log marginal likelihood terms to include. Use NULL if all indices should be used.
minvls	minimum number of samples for each marginal likelihood term. Negative values provides a default which depends on the dimension of the integration.
do_reorder	TRUE if a heuristic variable reordering should be used. TRUE is likely the best value.
use_aprx	TRUE if a less precise approximation of pnorm and qnorm should be used. This may reduce the computation time while not affecting the result much.
n_threads	number of threads to use.
cluster_weights	numeric vector with weights for each cluster. Use NULL if all clusters have weight one.
fix	integer vector with indices of par to fix. This is useful for computing profile likelihoods. NULL yields all parameters.
standardized	logical for whether to use the standardized or direct parameterization. See standardized_to_direct and the vignette at vignette("pedmod", package = "pedmod").
method	integer with the method to use. Zero yields randomized Korobov lattice rules while one yields scrambled Sobol sequences.
use_tilting	TRUE if the minimax tilting method suggested by Botev (2017) should be used. See doi:10.1111/rssb.12162.
vls_scales	can be a numeric vector with a positive scalar for each cluster. Then vls_scales[i] * minvls and vls_scales[i] * maxvls is used for cluster i rather than minvls and maxvls. Set vls_scales = NULL if the latter should be used.
...	Arguments passed to opt_func.
data	the list that was passed to pedigree_ll_terms or pedigree_ll_terms_loadings.

`scale_max` the maximum value for the scale parameters. Sometimes, the optimization method tends to find large scale parameters and get stuck. Setting a maximum solves this.

`sc_start` starting value for the scale parameters. Use `NULL` if you have no value to start with.

`sc_start_invariant` scale parameter(s) like sc_start. It is the value that all individuals should have (i.e. not one that varies by individual).

Details

`pedmod_start` and `pedmod_start_loadings` yield starting values which can be used for `pedmod_opt`. The methods are based on a heuristics.

Value

`pedmod_opt`: The output from the `opt_func` argument. Thus, if `fix` is supplied then this is optimal values of only `par[-fix]` with `par[fix]` being fixed to the inputs. Thus, the length is only the number of non-fixed parameters.

`pedmod_start`: A `list` with:

- par: the starting value.
- beta_no_rng: the fixed effects MLEs without random effects.
- logLik_no_rng: the log maximum likelihood without random effects.
- logLik_est: the likelihood at par.

`pedmod_start_loadings`: A `list` with:

- par: the starting value.
- beta_no_rng: the fixed effects MLEs without random effects.
- logLik_no_rng: the log maximum likelihood without random effects.

See Also

pedmod_sqn.

Examples

```
# we simulate outcomes with an additive genetic effect. The kinship matrix is
# the same for all families and given by
K <- matrix(c(
  0.5  , 0    , 0.25 , 0   , 0.25 , 0   , 0.125 , 0.125 , 0.125 , 0.125 ,
  0    , 0.5  , 0.25 , 0   , 0.25 , 0   , 0.125 , 0.125 , 0.125 , 0.125 ,
  0.25 , 0.25 , 0.5  , 0   , 0.25 , 0   , 0.25  , 0.25  , 0.125 , 0.125 ,
  0    , 0    , 0    , 0.5 , 0    , 0   , 0.25  , 0.25  , 0     , 0     ,
  0.25 , 0.25 , 0.25 , 0   , 0.5  , 0   , 0.125 , 0.125 , 0.25  , 0.25  ,
  0    , 0    , 0    , 0   , 0    , 0.5 , 0     , 0     , 0.25  , 0.25  ,
  0.125, 0.125, 0.25 , 0.25, 0.125, 0   , 0.5   , 0.25  , 0.0625, 0.0625,
```

```
  0.125, 0.125, 0.25 , 0.25, 0.125, 0   , 0.25  , 0.5   , 0.0625, 0.0625,
  0.125, 0.125, 0.125, 0   , 0.25 , 0.25, 0.0625, 0.0625, 0.5   , 0.25  ,
  0.125, 0.125, 0.125, 0   , 0.25 , 0.25, 0.0625, 0.0625, 0.25  , 0.5
), 10)

# simulates a data set.
#
# Args:
#   n_fams: number of families.
#   beta: the fixed effect coefficients.
#   sig_sq: the scale parameter.
sim_dat <- function(n_fams, beta = c(-1, 1, 2), sig_sq = 3){
  # setup before the simulations
  Cmat <- 2 * K
  n_obs <- NROW(K)
  Sig <- diag(n_obs) + sig_sq * Cmat
  Sig_chol <- chol(Sig)

  # simulate the data
  out <- replicate(
    n_fams, {
      # simulate covariates
      X <- cbind(`(Intercept)` = 1, Continuous = rnorm(n_obs),
                 Binary = runif(n_obs) > .5)

      # assign the linear predictor + noise
      eta <- drop(X %*% beta) + drop(rnorm(n_obs) %*% Sig_chol)

      # return the list in the format needed for the package
      list(y = as.numeric(eta > 0), X = X, scale_mats = list(Cmat))
    }, simplify = FALSE)

  # add attributes with the true values and return
  attributes(out) <- list(beta = beta, sig_sq = sig_sq)
  out
}

# simulate the data
set.seed(1)
dat <- sim_dat(100L)

# fit the model
ptr <- pedigree_ll_terms(dat, max_threads = 1L)
start <- pedmod_start(ptr = ptr, data = dat, n_threads = 1L)
fit <- pedmod_opt(ptr = ptr, par = start$par, n_threads = 1L, use_aprx = TRUE,
                  maxvls = 5000L, minvls = 1000L, abs_eps = 0, rel_eps = 1e-3)
fit$par # the estimate
-fit$value # the log maximum likelihood
start$logLik_no_rng # the log maximum likelihood without the random effects
```

pedmod_profile *Computes Profile Likelihood Based Confidence Intervals*

Description

Computes likelihood ratio based confidence intervals for one the parameters in the model.

Usage

```
pedmod_profile(
  ptr,
  par,
  delta,
  maxvls,
  minvls = -1L,
  alpha = 0.05,
  abs_eps,
  rel_eps,
  which_prof,
  indices = NULL,
  maxvls_start = max(100L, as.integer(ceiling(maxvls/5))),
  minvls_start = if (minvls < 0) minvls else minvls/5,
  do_reorder = TRUE,
  use_aprx = FALSE,
  n_threads = 1L,
  cluster_weights = NULL,
  method = 0L,
  seed = 1L,
  verbose = FALSE,
  max_step = 15L,
  standardized = FALSE,
  use_tilting = FALSE,
  vls_scales = NULL,
  ...
)
```

Arguments

ptr	object from pedigree_ll_terms or pedigree_ll_terms_loadings.
par	numeric vector with the maximum likelihood estimator e.g. from pedmod_opt.
delta	numeric scalar with an initial step to take. Subsequent steps are taken by 2^(<iteration number> - 1) * delta. Two times the standard error is a good value or a guess thereof. Hessian approximations are not implemented as of this writing and therefore the user needs to provide some guess.
maxvls	maximum number of samples in the approximation for each marginal likelihood term.

minvls	minimum number of samples for each marginal likelihood term. Negative values provides a default which depends on the dimension of the integration.
alpha	numeric scalar with the confidence level required.
abs_eps	absolute convergence threshold for eval_pedigree_ll and eval_pedigree_grad.
rel_eps	rel_eps convergence threshold for eval_pedigree_ll and eval_pedigree_grad.
which_prof	integer scalar with index of the parameter which the profile likelihood curve should be computed for.
indices	zero-based vector with indices of which log marginal likelihood terms to include. Use NULL if all indices should be used.
maxvls_start, minvls_start	number of samples to use when finding the initial values for the optimization.
do_reorder	TRUE if a heuristic variable reordering should be used. TRUE is likely the best value.
use_aprx	TRUE if a less precise approximation of pnorm and qnorm should be used. This may reduce the computation time while not affecting the result much.
n_threads	number of threads to use.
cluster_weights	numeric vector with weights for each cluster. Use NULL if all clusters have weight one.
method	integer with the method to use. Zero yields randomized Korobov lattice rules while one yields scrambled Sobol sequences.
seed	seed to pass to set.seed before each gradient and function evaluation. Use NULL if the seed should not be fixed.
verbose	logical for whether output should be printed to the console during the estimation of the profile likelihood curve.
max_step	integer scalar with the maximum number of steps to take in either directions.
standardized	logical for whether to use the standardized or direct parameterization. See standardized_to_direct and the vignette at vignette("pedmod", package = "pedmod").
use_tilting	TRUE if the minimax tilting method suggested by Botev (2017) should be used. See doi:10.1111/rssb.12162.
vls_scales	can be a numeric vector with a positive scalar for each cluster. Then vls_scales[i] * minvls and vls_scales[i] * maxvls is used for cluster i rather than minvls and maxvls. Set vls_scales = NULL if the latter should be used.
...	arguments passed on to pedmod_opt.

Value

A list with the following elements:

confs	2D numeric vector with the profile likelihood based confidence interval.
xs	the points at which the profile likelihood is evaluated.
p_log_Lik	the log profile likelihood values at xs.
data	list with the returned objects from pedmod_opt.

See Also

pedmod_opt, pedmod_sqn, pedmod_profile_prop, and pedmod_profile_nleq

Examples

```
# we simulate outcomes with an additive genetic effect. The kinship matrix is
# the same for all families and given by
K <- matrix(c(
  0.5  , 0    , 0.25 , 0   , 0.25 , 0   , 0.125 , 0.125 , 0.125 , 0.125 ,
  0    , 0.5  , 0.25 , 0   , 0.25 , 0   , 0.125 , 0.125 , 0.125 , 0.125 ,
  0.25 , 0.25 , 0.5  , 0   , 0.25 , 0   , 0.25  , 0.25  , 0.125 , 0.125 ,
  0    , 0    , 0    , 0.5 , 0    , 0   , 0.25  , 0.25  , 0     , 0     ,
  0.25 , 0.25 , 0.25 , 0   , 0.5  , 0   , 0.125 , 0.125 , 0.25  , 0.25  ,
  0    , 0    , 0    , 0   , 0    , 0.5 , 0     , 0     , 0.25  , 0.25  ,
  0.125, 0.125, 0.25 , 0.25, 0.125, 0   , 0.5   , 0.25  , 0.0625, 0.0625,
  0.125, 0.125, 0.25 , 0.25, 0.125, 0   , 0.25  , 0.5   , 0.0625, 0.0625,
  0.125, 0.125, 0.125, 0   , 0.25 , 0.25, 0.0625, 0.0625, 0.5   , 0.25  ,
  0.125, 0.125, 0.125, 0   , 0.25 , 0.25, 0.0625, 0.0625, 0.25  , 0.5
), 10)

# simulates a data set.
#
# Args:
#   n_fams: number of families.
#   beta: the fixed effect coefficients.
#   sig_sq: the scale parameter.
sim_dat <- function(n_fams, beta = c(-1, 1, 2), sig_sq = 3){
  # setup before the simulations
  Cmat <- 2 * K
  n_obs <- NROW(K)
  Sig <- diag(n_obs) + sig_sq * Cmat
  Sig_chol <- chol(Sig)

  # simulate the data
  out <- replicate(
    n_fams, {
      # simulate covariates
      X <- cbind(`(Intercept)` = 1, Continuous = rnorm(n_obs),
                 Binary = runif(n_obs) > .5)

      # assign the linear predictor + noise
      eta <- drop(X %*% beta) + drop(rnorm(n_obs) %*% Sig_chol)

      # return the list in the format needed for the package
      list(y = as.numeric(eta > 0), X = X, scale_mats = list(Cmat))
    }, simplify = FALSE)

  # add attributes with the true values and return
  attributes(out) <- list(beta = beta, sig_sq = sig_sq)
  out
}
```

```
# simulate the data
set.seed(1)
dat <- sim_dat(100L)

# fit the model
ptr <- pedigree_ll_terms(dat, max_threads = 1L)
start <- pedmod_start(ptr = ptr, data = dat, n_threads = 1L)
fit <- pedmod_opt(ptr = ptr, par = start$par, n_threads = 1L, use_aprx = TRUE,
                  maxvls = 5000L, minvls = 1000L, abs_eps = 0, rel_eps = 1e-3)
fit$par # the estimate

# 90% likelihood ratio based confidence interval for the log of the scale
# parameter
prof_out <- pedmod_profile(ptr = ptr, fit$par, delta = .4, maxvls = 5000L,
                           minvls = 1000L, alpha = .1, which_prof = 4L,
                           abs_eps = 0, rel_eps = 1e-3, verbose = TRUE)
exp(prof_out$confs) # the confidence interval

# plot the log profile likelihood
plot(exp(prof_out$xs), prof_out$p_log_Lik, pch = 16,
     xlab = expression(sigma), ylab = "log profile likelihood")
abline(v = exp(prof_out$confs), lty = 2)
```

`pedmod_profile_nleq` *Computes Profile Likelihood Based Confidence Intervals for a Nonlinear Transformation of the Variables*

Description

Computes Profile Likelihood Based Confidence Intervals for a Nonlinear Transformation of the Variables

Usage

```
pedmod_profile_nleq(
  ptr,
  par,
  maxvls,
  minvls = -1L,
  alpha = 0.05,
  abs_eps,
  rel_eps,
  heq,
  heq_bounds = c(-Inf, Inf),
  delta,
  indices = NULL,
```

```
  maxvls_start = max(100L, as.integer(ceiling(maxvls/5))),
  minvls_start = if (minvls < 0) minvls else minvls/5,
  do_reorder = TRUE,
  use_aprx = FALSE,
  n_threads = 1L,
  cluster_weights = NULL,
  method = 0L,
  seed = 1L,
  verbose = FALSE,
  max_step = 15L,
  use_tilting = FALSE,
  vls_scales = NULL,
  control.outer = list(itmax = 100L, method = "BFGS", kkt2.check = FALSE, trace =
    FALSE),
  control.optim = list(fnscale = get_n_terms(ptr)),
  ...
)
```

Arguments

ptr	object from pedigree_ll_terms or pedigree_ll_terms_loadings.
par	numeric vector with the maximum likelihood estimator e.g. from pedmod_opt.
maxvls	maximum number of samples in the approximation for each marginal likelihood term.
minvls	minimum number of samples for each marginal likelihood term. Negative values provides a default which depends on the dimension of the integration.
alpha	numeric scalar with the confidence level required.
abs_eps	absolute convergence threshold for eval_pedigree_ll and eval_pedigree_grad.
rel_eps	rel_eps convergence threshold for eval_pedigree_ll and eval_pedigree_grad.
heq	function that returns a one dimensional numerical vector which should be profiled. It does not need to evaluate to zero at the maximum likelihood estimator.
heq_bounds	two dimensional numerical vector with bounds for heq.
delta	numeric scalar with an initial step to take. Subsequent steps are taken by 2^(<iteration number> - 1) * delta. Two times the standard error is a good value or a guess thereof. Hessian approximations are not implemented as of this writing and therefore the user needs to provide some guess.
indices	zero-based vector with indices of which log marginal likelihood terms to include. Use NULL if all indices should be used.
maxvls_start, minvls_start	number of samples to use when finding the initial values for the optimization.
do_reorder	TRUE if a heuristic variable reordering should be used. TRUE is likely the best value.
use_aprx	TRUE if a less precise approximation of pnorm and qnorm should be used. This may reduce the computation time while not affecting the result much.
n_threads	number of threads to use.

`cluster_weights`
: numeric vector with weights for each cluster. Use `NULL` if all clusters have weight one.

`method`
: integer with the method to use. Zero yields randomized Korobov lattice rules while one yields scrambled Sobol sequences.

`seed`
: seed to pass to `set.seed` before each gradient and function evaluation. Use `NULL` if the seed should not be fixed.

`verbose`
: logical for whether output should be printed to the console during the estimation of the profile likelihood curve.

`max_step`
: integer scalar with the maximum number of steps to take in either directions.

`use_tilting`
: `TRUE` if the minimax tilting method suggested by Botev (2017) should be used. See doi:10.1111/rssb.12162.

`vls_scales`
: can be a numeric vector with a positive scalar for each cluster. Then `vls_scales[i] * minvls` and `vls_scales[i] * maxvls` is used for cluster `i` rather than `minvls` and `maxvls`. Set `vls_scales = NULL` if the latter should be used.

`control.outer, control.optim, ...`
: arguments passed to `auglag`

See Also

`pedmod_opt`, `pedmod_sqn`, `pedmod_profile`, and `pedmod_profile_prop`.

Examples

```
# similar examples to that in help("pedmod_profile_prop")
K <- matrix(c(
  0.5  , 0    , 0.25 , 0   , 0.25 , 0   , 0.125 , 0.125 , 0.125 , 0.125 ,
  0    , 0.5  , 0.25 , 0   , 0.25 , 0   , 0.125 , 0.125 , 0.125 , 0.125 ,
  0.25 , 0.25 , 0.5  , 0   , 0.25 , 0   , 0.25  , 0.25  , 0.125 , 0.125 ,
  0    , 0    , 0    , 0.5 , 0    , 0   , 0.25  , 0.25  , 0     , 0     ,
  0.25 , 0.25 , 0.25 , 0   , 0.5  , 0   , 0.125 , 0.125 , 0.25  , 0.25  ,
  0    , 0    , 0    , 0   , 0    , 0.5 , 0     , 0     , 0.25  , 0.25  ,
  0.125, 0.125, 0.25 , 0.25, 0.125, 0   , 0.5   , 0.25  , 0.0625, 0.0625,
  0.125, 0.125, 0.25 , 0.25, 0.125, 0   , 0.25  , 0.5   , 0.0625, 0.0625,
  0.125, 0.125, 0.125, 0   , 0.25 , 0.25, 0.0625, 0.0625, 0.5   , 0.25  ,
  0.125, 0.125, 0.125, 0   , 0.25 , 0.25, 0.0625, 0.0625, 0.25  , 0.5
), 10)

C <- matrix(c(
  1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
  0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
  0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
  0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
```

```
  0, 0, 0, 0, 0, 0, 0, 0, 1, 1
), 10L)

# simulates a data set.
#
# Args:
#   n_fams: number of families.
#   beta: the fixed effect coefficients.
#   sig_sq: the scale parameters.
sim_dat <- function(n_fams, beta = c(-1, 1, 2), sig_sq = c(3, 1)){
  # setup before the simulations
  Cmat <- 2 * K
  n_obs <- NROW(K)
  Sig <- diag(n_obs) + sig_sq[1] * Cmat + sig_sq[2] * C
  Sig_chol <- chol(Sig)

  # simulate the data
  out <- replicate(
    n_fams, {
      # simulate covariates
      X <- cbind(`(Intercept)` = 1, Continuous = rnorm(n_obs),
                 Binary = runif(n_obs) > .5)

      # assign the linear predictor + noise
      eta <- drop(X %*% beta) + drop(rnorm(n_obs) %*% Sig_chol)

      # return the list in the format needed for the package
      list(y = as.numeric(eta > 0), X = X,
           scale_mats = list(genetic = Cmat, environment = C))
    }, simplify = FALSE)

  # add attributes with the true values and return
  attributes(out) <- list(beta = beta, sig_sq = sig_sq)
  out
}

# simulate the data
set.seed(1)
dat <- sim_dat(200L)

# fit the model
ptr <- pedigree_ll_terms(dat, max_threads = 2L)
start <- pedmod_start(ptr = ptr, data = dat, n_threads = 2L)
fit <- pedmod_opt(ptr = ptr, par = start$par, use_aprx = TRUE, n_threads = 2L,
                  maxvls = 5000L, minvls = 1000L, abs_eps = 0, rel_eps = 1e-3)
fit$par # the estimate

# 90% likelihood ratio based confidence interval for the proportion of variance
# of the genetic effect
heq <- function(par){
 vars <- exp(tail(par, 2))
 vars[1] / (1 + sum(vars))
}
```

```
heq(fit$par)
prof_out_nleq <- pedmod_profile_nleq(
  ptr = ptr, fit$par, maxvls = 2500L, minvls = 500L, alpha = .1,
  abs_eps = 0, rel_eps = 1e-3, verbose = TRUE, use_aprx = TRUE,
  heq = heq, heq_bounds = c(0, Inf), delta = .2, n_threads = 2L)
prof_out_nleq$confs # the confidence interval for the proportion

# plot the log profile likelihood
plot(prof_out_nleq$xs, prof_out_nleq$p_log_Lik, pch = 16,
     xlab = "proportion of variance", ylab = "log profile likelihood")
abline(v = prof_out_nleq$confs, lty = 2)
```

---

pedmod_profile_prop *Computes Profile Likelihood Based Confidence Intervals for the Proportion of Variance*

Description

Constructs a likelihood ratio based confidence intervals for the proportion of variance for one of the effects.

Usage

```
pedmod_profile_prop(
  ptr,
  par,
  maxvls,
  minvls = -1L,
  alpha = 0.05,
  abs_eps,
  rel_eps,
  which_prof,
  indices = NULL,
  maxvls_start = max(100L, as.integer(ceiling(maxvls/5))),
  minvls_start = if (minvls < 0) minvls else minvls/5,
  do_reorder = TRUE,
  use_aprx = FALSE,
  n_threads = 1L,
  cluster_weights = NULL,
  method = 0L,
  seed = 1L,
  verbose = FALSE,
  max_step = 15L,
  opt_func = NULL,
  use_tilting = FALSE,
  vls_scales = NULL,
```

```
  bound = c(0.01, 0.99),
  ...
)
```

Arguments

ptr	object from pedigree_ll_terms.
par	numeric vector with the maximum likelihood estimator e.g. from pedmod_opt.
maxvls	maximum number of samples in the approximation for each marginal likelihood term.
minvls	minimum number of samples for each marginal likelihood term. Negative values provides a default which depends on the dimension of the integration.
alpha	numeric scalar with the confidence level required.
abs_eps	absolute convergence threshold for eval_pedigree_ll and eval_pedigree_grad.
rel_eps	rel_eps convergence threshold for eval_pedigree_ll and eval_pedigree_grad.
which_prof	the index of the random effect which proportion of variance should be profiled.
indices	zero-based vector with indices of which log marginal likelihood terms to include. Use NULL if all indices should be used.
maxvls_start, minvls_start	number of samples to use when finding the initial values for the optimization.
do_reorder	TRUE if a heuristic variable reordering should be used. TRUE is likely the best value.
use_aprx	TRUE if a less precise approximation of pnorm and qnorm should be used. This may reduce the computation time while not affecting the result much.
n_threads	number of threads to use.
cluster_weights	numeric vector with weights for each cluster. Use NULL if all clusters have weight one.
method	integer with the method to use. Zero yields randomized Korobov lattice rules while one yields scrambled Sobol sequences.
seed	seed to pass to set.seed before each gradient and function evaluation. Use NULL if the seed should not be fixed.
verbose	logical for whether output should be printed to the console during the estimation of the profile likelihood curve.
max_step	integer scalar with the maximum number of steps to take in either directions.
opt_func	function to perform minimization with arguments like optim. BFGS is used with optim if this argument is NULL.
use_tilting	TRUE if the minimax tilting method suggested by Botev (2017) should be used. See doi:10.1111/rssb.12162.
vls_scales	can be a numeric vector with a positive scalar for each cluster. Then vls_scales[i] * minvls and vls_scales[i] * maxvls is used for cluster i rather than minvls and maxvls. Set vls_scales = NULL if the latter should be used.
bound	boundaries for the limits of the proportion. Has to be in between $(0, 1)$. This is useful particularly if the optimization fails to work on the default values.
...	arguments passed to opt_func.

Details

The function is only useful when there is more than one type of random effect. If not, then pedmod_profile can be used because of the scale invariance of the likelihood ratio.

Value

A list like pedmod_profile.

See Also

pedmod_opt, pedmod_sqn, pedmod_profile, and pedmod_profile_nleq.

Examples

```
# we simulate outcomes with an additive genetic effect and a childhood
# environment effect. The kinship matrix is the same for all families and
# given by
K <- matrix(c(
  0.5  , 0    , 0.25 , 0   , 0.25 , 0   , 0.125 , 0.125 , 0.125 , 0.125 ,
  0    , 0.5  , 0.25 , 0   , 0.25 , 0   , 0.125 , 0.125 , 0.125 , 0.125 ,
  0.25 , 0.25 , 0.5  , 0   , 0.25 , 0   , 0.25  , 0.25  , 0.125 , 0.125 ,
  0    , 0    , 0    , 0.5 , 0    , 0   , 0.25  , 0.25  , 0     , 0     ,
  0.25 , 0.25 , 0.25 , 0   , 0.5  , 0   , 0.125 , 0.125 , 0.25  , 0.25  ,
  0    , 0    , 0    , 0   , 0    , 0.5 , 0     , 0     , 0.25  , 0.25  ,
  0.125, 0.125, 0.25 , 0.25, 0.125, 0   , 0.5   , 0.25  , 0.0625, 0.0625,
  0.125, 0.125, 0.25 , 0.25, 0.125, 0   , 0.25  , 0.5   , 0.0625, 0.0625,
  0.125, 0.125, 0.125, 0   , 0.25 , 0.25, 0.0625, 0.0625, 0.5   , 0.25  ,
  0.125, 0.125, 0.125, 0   , 0.25 , 0.25, 0.0625, 0.0625, 0.25  , 0.5
), 10)

# the scale matrix for the childhood environment effect is also the same and
# given by
C <- matrix(c(
  1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
  0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
  0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
  0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
  0, 0, 0, 0, 0, 0, 0, 0, 1, 1
), 10L)

# simulates a data set.
#
# Args:
#   n_fams: number of families.
#   beta: the fixed effect coefficients.
#   sig_sq: the scale parameters.
```

```
sim_dat <- function(n_fams, beta = c(-1, 1, 2), sig_sq = c(3, 1)){
  # setup before the simulations
  Cmat <- 2 * K
  n_obs <- NROW(K)
  Sig <- diag(n_obs) + sig_sq[1] * Cmat + sig_sq[2] * C
  Sig_chol <- chol(Sig)

  # simulate the data
  out <- replicate(
    n_fams, {
      # simulate covariates
      X <- cbind(`(Intercept)` = 1, Continuous = rnorm(n_obs),
                 Binary = runif(n_obs) > .5)

      # assign the linear predictor + noise
      eta <- drop(X %*% beta) + drop(rnorm(n_obs) %*% Sig_chol)

      # return the list in the format needed for the package
      list(y = as.numeric(eta > 0), X = X,
           scale_mats = list(genetic = Cmat, environment = C))
    }, simplify = FALSE)

  # add attributes with the true values and return
  attributes(out) <- list(beta = beta, sig_sq = sig_sq)
  out
}

# simulate the data
set.seed(1)
dat <- sim_dat(200L)

# fit the model
ptr <- pedigree_ll_terms(dat, max_threads = 1L)
start <- pedmod_start(ptr = ptr, data = dat, n_threads = 1L)
fit <- pedmod_opt(ptr = ptr, par = start$par, n_threads = 1L, use_aprx = TRUE,
                  maxvls = 5000L, minvls = 1000L, abs_eps = 0, rel_eps = 1e-3)
fit$par # the estimate

# 90% likelihood ratio based confidence interval for the proportion of variance
# of the genetic effect
prof_out <- pedmod_profile_prop(
  ptr = ptr, fit$par, maxvls = 5000L, minvls = 1000L, alpha = .1,
  which_prof = 1L, abs_eps = 0, rel_eps = 1e-3, verbose = TRUE)
prof_out$confs # the confidence interval for the proportion

# plot the log profile likelihood
keep <- c(-1L, -length(prof_out$xs))
plot(prof_out$xs[keep], prof_out$p_log_Lik[keep], pch = 16,
     xlab = "proportion of variance", ylab = "log profile likelihood")
abline(v = prof_out$confs, lty = 2)
```

pedmod_sqn *Optimize the Log Marginal Likelihood Using a Stochastic Quasi-Newton Method*

Description

Optimizes eval_pedigree_ll and eval_pedigree_grad using a stochastic quasi-Newton method.

Usage

```
pedmod_sqn(
  ptr,
  par,
  maxvls,
  abs_eps,
  rel_eps,
  step_factor,
  n_it,
  n_grad_steps,
  indices = NULL,
  minvls = -1L,
  n_grad = 50L,
  n_hess = 500L,
  do_reorder = TRUE,
  use_aprx = FALSE,
  n_threads = 1L,
  cluster_weights = NULL,
  fix = NULL,
  standardized = FALSE,
  minvls_hess = minvls,
  maxvls_hess = maxvls,
  abs_eps_hess = abs_eps,
  rel_eps_hess = rel_eps,
  verbose = FALSE,
  method = 0L,
  check_every = 2L * n_grad_steps,
  use_tilting = FALSE,
  vls_scales = NULL
)
```

Arguments

ptr	object from pedigree_ll_terms.
par	starting values.
maxvls	maximum number of samples in the approximation for each marginal likelihood term.

abs_eps	absolute convergence threshold for eval_pedigree_ll and eval_pedigree_grad.
rel_eps	rel_eps convergence threshold for eval_pedigree_ll and eval_pedigree_grad.
step_factor	factor used for the step size. The step size is step_factor divided by the iteration number.
n_it	number of stochastic gradient steps to make.
n_grad_steps	number of stochastic gradient steps to make between each Hessian approximation update.
indices	zero-based vector with indices of which log marginal likelihood terms to include. Use NULL if all indices should be used.
minvls	minimum number of samples for each marginal likelihood term. Negative values provides a default which depends on the dimension of the integration.
n_grad	number of log marginal likelihood terms to include in the stochastic gradient step.
n_hess	number of log marginal likelihood terms to include in the gradients used for the Hessian approximation update. This is set to the entire sample (or indices) if this is greater than or equal to half the number of log marginal likelihood terms.
do_reorder	TRUE if a heuristic variable reordering should be used. TRUE is likely the best value.
use_aprx	TRUE if a less precise approximation of pnorm and qnorm should be used. This may reduce the computation time while not affecting the result much.
n_threads	number of threads to use.
cluster_weights	numeric vector with weights for each cluster. Use NULL if all clusters have weight one.
fix	integer vector with indices of par to fix. This is useful for computing profile likelihoods. NULL yields all parameters.
standardized	logical for whether to use the standardized or direct parameterization. See standardized_to_direct and the vignette at vignette("pedmod", package = "pedmod").
minvls_hess	minvls argument to use when updating the Hessian approximation.
maxvls_hess	maxvls argument to use when updating the Hessian approximation.
abs_eps_hess	abs_eps argument to use when updating the Hessian approximation.
rel_eps_hess	rel_eps argument to use when updating the Hessian approximation.
verbose	logical for whether to print output during the estimation.
method	integer with the method to use. Zero yields randomized Korobov lattice rules while one yields scrambled Sobol sequences.
check_every	integer for the number of gradient steps between checking that the likelihood did increase. If not, the iterations are reset and the step-size is halved.
use_tilting	TRUE if the minimax tilting method suggested by Botev (2017) should be used. See doi:10.1111/rssb.12162.
vls_scales	can be a numeric vector with a positive scalar for each cluster. Then vls_scales[i] * minvls and vls_scales[i] * maxvls is used for cluster i rather than minvls and maxvls. Set vls_scales = NULL if the latter should be used.

Details

The function uses a stochastic quasi-Newton method like suggested by Byrd et al. (2016) with a few differences: Differences in gradients are used rather than Hessian-vector products, BFGS rather than L-BFGS is used because the problem is typically low dimensional, and damped BFGS updates are used (see e.g. chapter 18 of Nocedal and Wright, 2006).

Separate arguments for the gradient approximation in the Hessian update are provided as one may want a more precise approximation for these gradients. `step_factor` likely depends on the other parameters and the data set and should be altered.

Value

A list with the following elements:

`par`	estimated parameters.
`omegas`	parameter estimates after each iteration.
`H`	Hessian approximation in the quasi-Newton method. It should not be treated as the Hessian.

References

Byrd, R. H., Hansen, S. L., Nocedal, J., & Singer, Y. (2016). *A stochastic quasi-Newton method for large-scale optimization*. SIAM Journal on Optimization, 26(2), 1008-1031.

Nocedal, J., & Wright, S. (2006). *Numerical optimization*. Springer Science & Business Media.

See Also

`pedmod_opt` and `pedmod_start`.

Examples

```
# we simulate outcomes with an additive genetic effect. The kinship matrix is
# the same for all families and given by
K <- matrix(c(
  0.5  , 0    , 0.25 , 0   , 0.25 , 0   , 0.125 , 0.125 , 0.125 , 0.125 ,
  0    , 0.5  , 0.25 , 0   , 0.25 , 0   , 0.125 , 0.125 , 0.125 , 0.125 ,
  0.25 , 0.25 , 0.5  , 0   , 0.25 , 0   , 0.25  , 0.25  , 0.125 , 0.125 ,
  0    , 0    , 0    , 0.5 , 0    , 0   , 0.25  , 0.25  , 0     , 0     ,
  0.25 , 0.25 , 0.25 , 0   , 0.5  , 0   , 0.125 , 0.125 , 0.25  , 0.25  ,
  0    , 0    , 0    , 0   , 0    , 0.5 , 0     , 0     , 0.25  , 0.25  ,
  0.125, 0.125, 0.25 , 0.25, 0.125, 0   , 0.5   , 0.25  , 0.0625, 0.0625,
  0.125, 0.125, 0.25 , 0.25, 0.125, 0   , 0.25  , 0.5   , 0.0625, 0.0625,
  0.125, 0.125, 0.125, 0   , 0.25 , 0.25, 0.0625, 0.0625, 0.5   , 0.25  ,
  0.125, 0.125, 0.125, 0   , 0.25 , 0.25, 0.0625, 0.0625, 0.25  , 0.5
), 10)

# simulates a data set.
#
# Args:
```

```
#   n_fams: number of families.
#   beta: the fixed effect coefficients.
#   sig_sq: the scale parameter.
sim_dat <- function(n_fams, beta = c(-1, 1, 2), sig_sq = 3){
  # setup before the simulations
  Cmat <- 2 * K
  n_obs <- NROW(K)
  Sig <- diag(n_obs) + sig_sq * Cmat
  Sig_chol <- chol(Sig)

  # simulate the data
  out <- replicate(
    n_fams, {
      # simulate covariates
      X <- cbind(`(Intercept)` = 1, Continuous = rnorm(n_obs),
                 Binary = runif(n_obs) > .5)

      # assign the linear predictor + noise
      eta <- drop(X %*% beta) + drop(rnorm(n_obs) %*% Sig_chol)

      # return the list in the format needed for the package
      list(y = as.numeric(eta > 0), X = X, scale_mats = list(Cmat))
    }, simplify = FALSE)

  # add attributes with the true values and return
  attributes(out) <- list(beta = beta, sig_sq = sig_sq)
  out
}

# simulate the data
set.seed(1)
dat <- sim_dat(100L)

# fit the model
ptr <- pedigree_ll_terms(dat, max_threads = 1L)
start <- pedmod_start(ptr = ptr, data = dat, n_threads = 1L)
fit <- pedmod_sqn(ptr = ptr, par = start$par, n_threads = 1L, use_aprx = TRUE,
                  maxvls = 5000L, minvls = 1000L, abs_eps = 0, rel_eps = 1e-3,
                  n_grad_steps = 20L, step_factor = 1, n_grad = 10L,
                  n_hess = 50L, check_every = 50L, n_it = 1000L)
fit$par # maximum likelihood estimate
# the maximum likelihood
eval_pedigree_ll(ptr = ptr, fit$par, maxvls = 5000L, abs_eps = 0,
                 rel_eps = 1e-3, minvls = 1000L)
```

---

standardized_to_direct

Transform Between Parameterizations

Description

Transform the parameters between the parameterizations that are used in the package.

Usage

```
standardized_to_direct(par, n_scales, jacobian = FALSE)

direct_to_standardized(par, n_scales)
```

Arguments

par	concatenated vector with the fixed effect slopes and the scale parameters that should be transformed.
n_scales	integer with the number of scale parameters.
jacobian	logical indicating if the Jacobian matrix of transformation should be computed.

Value

`standardized_to_direct`: returns the parameters using the direct parameterizations. See `vignette("pedmod", package = "pedmod")` for the definition. There is an attribute called 'variance proportions' with the proportion of variance of each effect assuming that all the scale matrices are correlation matrices. There is an attribute called jacobian with the Jacobian matrix if `jacobian` is `TRUE`.

`direct_to_standardized`: the parameters using the standardized parameterizations. See `vignette("pedmod", package = "pedmod")` for the definition.

Examples

```
# transform backwards and forwards
set.seed(1)
smp <- runif(10, -1, 1)
res <- standardized_to_direct(smp, 2L, jacobian = TRUE)
back_val <- direct_to_standardized(res, 2L)

all.equal(smp, back_val, check.attributes = FALSE)
res
```

---

unconnected_partition *Finds an Approximately Balanced Partition*

Description

Finds an Approximately Balanced Partition

Usage

```
unconnected_partition(
  from,
  to,
  weight_data = NULL,
  edge_weights = NULL,
  slack = 0,
  max_kl_it_inner = 50L,
  max_kl_it = 10000L,
  trace = 0L,
  init = integer()
)

unconnected_partition_pedigree(
  id,
  father.id,
  mother.id,
  id_weight = NULL,
  father_weight = NULL,
  mother_weight = NULL,
  slack = 0,
  max_kl_it_inner = 50L,
  max_kl_it = 10000L,
  trace = 0L,
  init = integer()
)
```

Arguments

`from`	integer vector with one of the vertex ids.
`to`	integer vector with one of the vertex ids.
`weight_data`	list with two elements called `"id"` for the id and `"weight"` for the vertex weight. All vertices that are not in this list have a weight of one. Use `NULL` if all vertices have a weight of one.
`edge_weights`	numeric vector with weights for each edge. Needs to have the same length as `from` and `to`. Use `NULL` if all edges should have a weight of one.
`slack`	fraction between zero and 0.5 for the allowed amount of deviation from the balance criterion that is allowed to reduce the cost of the cut edges.
`max_kl_it_inner`	maximum number of moves to consider in each iteration when `slack > 0`.
`max_kl_it`	maximum number of iterations to use when reducing the cost of the cut edges. Typically the method converges quickly and this argument is not needed.
`trace`	integer where larger values yields more information printed to the console during the procedure.
`init`	integer vector with ids that one of the two sets in the partition should start out with.

id	integer vector with the child id.
father.id	integer vector with the father id. May be NA if it is missing.
mother.id	integer vector with the mother id. May be NA if it is missing.
id_weight	numeric vector with the weight to use for each vertex (individual). NULL yields a weight of one for all.
father_weight	weights of the edges created between the fathers and the children. Use NULL if all should have a weight of one.
mother_weight	weights of the edges created between the mothers and the children. Use NULL if all should have a weight of one.

Value

A list with the following elements:

balance_criterion	value of the balance criterion.
removed_edges	2D integer matrix with the removed edges.
set_1,set_2	The two sets in the partition.

See Also

max_balanced_partition.

Examples

```
# example of a data set in pedigree and graph form
library(pedmod)
dat_pedigree <- data.frame(
  id = 1:48,
  mom = c(
    NA, NA, 2L, 2L, 2L, NA, NA, 7L, 7L, 7L, 3L, 3L, 3L, 3L, NA, 15L, 15L, 43L,
    18L, NA, NA, 21L, 21L, 9L, 9L, 9L, 9L, NA, NA, 29L, 29L, 29L, 30L, 30L, NA,
    NA, 36L, 36L, 36L, 38L, 38L, NA, NA, 43L, 43L, 43L, 32L, 32L),
  dad = c(NA, NA, 1L, 1L, 1L, NA, NA, 6L, 6L, 6L, 8L, 8L, 8L, 8L, NA, 4L, 4L,
          42L, 5L, NA, NA, 20L, 20L, 22L, 22L, 22L, 22L, NA, NA, 28L, 28L, 28L,
          23L, 23L, NA, NA, 35L, 35L, 35L, 31L, 31L, NA, NA, 42L, 42L, 42L,
          45L, 45L),
  sex = c(1L, 2L, 2L, 1L, 1L, 1L, 2L, 1L, 2L, 2L, 2L, 1L, 1L, 1L, 2L, 2L, 2L,
          2L, 1L, 1L, 2L, 1L, 1L, 2L, 1L, 1L, 2L, 1L, 2L, 2L, 1L, 2L, 2L, 2L,
          1L, 2L, 1L, 2L, 1L, 2L, 1L, 1L, 2L, 2L, 1L, 1L, 2L, 2L))

dat <- list(
  to = c(
    3L, 4L, 5L, 8L, 9L, 10L, 11L, 12L, 13L, 14L, 16L, 17L, 18L, 19L, 22L, 23L,
    24L, 25L, 26L, 27L, 30L, 31L, 32L, 33L, 34L, 37L, 38L, 39L, 40L, 41L, 44L,
    45L, 46L, 47L, 48L, 3L, 4L, 5L, 8L, 9L, 10L, 11L, 12L, 13L, 14L, 16L, 17L,
    18L, 19L, 22L, 23L, 24L, 25L, 26L, 27L, 30L, 31L, 32L, 33L, 34L, 37L, 38L,
    39L, 40L, 41L, 44L, 45L, 46L, 47L, 48L),
  from = c(
    1L, 1L, 1L, 6L, 6L, 6L, 8L, 8L, 8L, 8L, 4L, 4L, 42L, 5L, 20L, 20L, 22L, 22L,
```

```
    22L, 22L, 28L, 28L, 28L, 23L, 23L, 35L, 35L, 35L, 31L, 31L, 42L, 42L, 42L,
    45L, 45L, 2L, 2L, 2L, 7L, 7L, 7L, 3L, 3L, 3L, 3L, 15L, 15L, 43L, 18L, 21L,
    21L, 9L, 9L, 9L, 9L, 29L, 29L, 29L, 30L, 30L, 36L, 36L, 36L, 38L, 38L, 43L,
    43L, 43L, 32L, 32L))

# the results may be different because of different orders!
out_pedigree <- unconnected_partition_pedigree(
  id = dat_pedigree$id, father.id = dat_pedigree$dad,
  mother.id = dat_pedigree$mom)
out <- unconnected_partition(dat$to, dat$from)

all.equal(out_pedigree$balance_criterion, out$balance_criterion)
all.equal(out_pedigree$removed_edges, out$removed_edges)
```

Index