Package 'scorematchingad'

November 30, 2025

Title Score Matching Estimation by Automatic Differentiation

Version 0.1.6

Description Hyvärinen's score matching (Hyvärinen, 2005) <https://jmlr.org/papers/v6/hyvarinen05a.html> is a useful estimation technique when the normalising constant for a probability distribution is difficult to compute. This package implements score matching estimators using automatic differentiation in the 'CppAD' library <https://github.com/coin-or/CppAD> and is designed for quickly implementing score matching estimators for new models. Also available is general robustification (Windham, 1995) <https://www.jstor.org/stable/2346159>. Already in the package are estimators for directional distributions (Mardia, Kent and Laha, 2016) <doi:10.48550/arXiv.1604.08470> and the flexible Polynomially-Tilted Pairwise Interaction model for compositional data. The latter estimators perform well when there are zeros in the compositions (Scealy and Wood, 2023) <doi:10.1080/01621459.2021.2016422>, even many zeros (Scealy, Hingee, Kent, and Wood, 2024) <doi:10.1007/s11222-024-10412-w>. A partial interface to CppAD's ADFun objects is also available.

License GPL (>= 3)

Encoding UTF-8

LazyData true

ByteCompile true

RoxygenNote 7.3.2

Suggests testthat, ks, movMF, cubature, simdd, numDeriv

Depends R (>= 3.5.0)

Imports RcppEigen (>= 0.3.3.7), MCMCpack, optimx, FixedPoint, Rdpack, Rcpp (>= 1.0.9), methods, stats, utils, rlang (>= 1.1.0)

RdMacros Rdpack

LinkingTo Rcpp (>= 1.0.9), RcppEigen (>= 0.3.3.7)

Config/testthat/edition 3

URL https://github.com/kasselhingee/scorematchingad

BugReports https://github.com/kasselhingee/scorematchingad/issues

NeedsCompilation yes

Author Kassel Liam Hingee [aut, cre] (ORCID:
<https://orcid.org/0000-0001-9894-2407>),
Janice Scealy [aut] (ORCID: <https://orcid.org/0000-0002-9718-869X>),
Bradley M. Bell [cph]

Maintainer Kassel Liam Hingee <kassel.hingee@anu.edu.au>

Repository CRAN

Date/Publication 2025-11-30 03:30:02 UTC

Contents

---

scorematchingad-package

scorematchingad: Score Matching Estimation by Automatic Differentiation

---

Description

Hyvärinen's score matching (Hyvärinen, 2005) https://jmlr.org/papers/v6/hyvarinen05a.html is a useful estimation technique when the normalising constant for a probability distribution is difficult to compute. This package implements score matching estimators using automatic differentiation in the 'CppAD' library https://github.com/coin-or/CppAD and is designed for quickly implementing score matching estimators for new models. Also available is general robustification (Windham, 1995) https://www.jstor.org/stable/2346159. Already in the package are estimators for directional distributions (Mardia, Kent and Laha, 2016) doi:10.48550/arXiv.1604.08470 and the flexible Polynomially-Tilted Pairwise Interaction model for compositional data. The latter estimators perform well when there are zeros in the compositions (Scealy and Wood, 2023) doi:10.1080/01621459.2021.2016422, even many zeros (Scealy, Hingee, Kent, and Wood, 2024) doi:10.1007/s1122202410412w. A partial interface to CppAD's ADFun objects is also available.

Details

This package's main features are

- A general capacity to implement score matching estimators that use algorithmic differentiation to avoid tedious manual algebra. The package uses `CppAD` and `Eigen` to differentiate model densities and compute the score matching discrepancy function (see `scorematchingtheory`). The score matching discrepancy is usually minimised by solving a quadratic equation, but a method for solving numerically (through `optimx::Rcgmin()`) is also included. New models can be fitted with the help of `tape_uld()` in a similar fashion to models in the `TMB` package. New manifolds or new transforms require small alterations to the source code of this package.
- Score matching estimators for the Polynomially-Tilted Pairwise Interaction (PPI) model (Scealy and Wood 2023; Scealy et al. 2024). See function `ppi()`.
- Score matching and hybrid score matching estimators for von Mises Fisher, Bingham and Fisher-Bingham directional distributions (Mardia et al. 2016). See `vMF()`, `Bingham()` and `FB()`.
- Implementation of a modification of Windham's robustifying method (Windham 1995) for many exponential family distributions. See `Windham()`. For some models the density approaches infinity at some locations, creating difficulties for the weights in Windham's original method (Scealy et al. 2024).
- An interface of `CppAD`'s `ADFun` tape objects. See `Rcpp_ADFun`.

For an introduction to score matching estimation, see `scorematchingtheory`.

Acknowledgements

Colleagues Andrew T. A. Wood and John T. Kent played important roles in developing the statistical ideas and theory for score matching estimation for the PPI model (Scealy et al. 2024).

We developed this package on Ngunnawal and Ngambri Country. We thank the Country for its influence.

Author(s)

Maintainer: Kassel Liam Hingee <kassel.hingee@anu.edu.au> (ORCID)

Authors:

- Janice Scealy (ORCID)

Other contributors:

- Bradley M. Bell [copyright holder]

References

Amaral GJA, Dryden IL, Wood ATA (2007). "Pivotal Bootstrap Methods for k-Sample Problems in Directional Statistics and Shape Analysis." *Journal of the American Statistical Association*, **102**(478), 695–707. 27639898, http://www.jstor.org/stable/27639898.

Bell B (2023). "CppAD: A Package for Differentiation of C++ Algorithms." https://github.com/coin-or/CppAD.

Hyvärinen A (2005). "Estimation of Non-Normalized Statistical Models by Score Matching." *Journal of Machine Learning Research*, **6**(24), 695–709. https://jmlr.org/papers/v6/hyvarinen05a.html.

Hyvärinen A (2007). "Some extensions of score matching." *Computational Statistics & Data Analysis*, **51**(5), 2499–2512. doi:10.1016/j.csda.2006.09.003.

Liu S, Kanamori T, Williams DJ (2019). "Estimating Density Models with Truncation Boundaries using Score Matching." doi:10.48550/arXiv.1910.03834.

Mardia K (2018). "A New Estimation Methodology for Standard Directional Distributions." In *2018 21st International Conference on Information Fusion (FUSION)*, 724–729. doi:10.23919/ICIF.2018.8455640.

Mardia KV, Jupp PE (2000). *Directional Statistics*, Probability and Statistics. Wiley, Great Britain. ISBN 0-471-95333-4.

Mardia KV, Kent JT, Laha AK (2016). "Score matching estimators for directional distributions." doi:10.48550/arXiv.1604.08470.

Martin I, Uh H, Supali T, Mitreva M, Houwing-Duistermaat JJ (2019). "The mixed model for the analysis of a repeated-measurement multivariate count data." *Statistics in Medicine*, **38**(12), 2248–2268. doi:10.1002/sim.8101.

Scealy JL, Hingee KL, Kent JT, Wood ATA (2024). "Robust score matching for compositional data." *Statistics and Computing*, **34**, 93. doi:10.1007/s1122202410412w.

Scealy JL, Wood ATA (2023). "Score matching for compositional distributions." *Journal of the American Statistical Association*, **118**(543), 1811–1823. doi:10.1080/01621459.2021.2016422.

Windham MP (1995). "Robustifying Model Fitting." *Journal of the Royal Statistical Society. Series B (Methodological)*, **57**(3), 599–609. 2346159, http://www.jstor.org/stable/2346159.

Wiria AE, Prasetyani MA, Hamid F, Wammes LJ, Lell B, Ariawan I, Uh HW, Wibowo H, Djuardi Y, Wahyuni S, Sutanto I, May L, Luty AJ, Verweij JJ, Sartono E, Yazdanbakhsh M, Supali T (2010). "Does treatment of intestinal helminth infections influence malaria?" *BMC Infectious Diseases*, **10**, 77. doi:10.1186/147123341077.

Yu S, Drton M, Shojaie A (2019). "Generalized Score Matching for Non-Negative Data." *Journal of Machine Learning Research*, **20**(76), 1–70. https://jmlr.org/papers/v20/18-278.html.

Yu S, Drton M, Shojaie A (2020). "Generalized Score Matching for General Domains." doi:10.48550/arXiv.2009.11428.

See Also

Useful links:

- https://github.com/kasselhingee/scorematchingad
- Report bugs at https://github.com/kasselhingee/scorematchingad/issues

---

`avgrange` *Average Across Range of a Tape*

Description

Creates a CppAD tape that is the average of the returned values of `pfun`. For creating this tape, the values of `pfun$dyntape` and `pfun$xtape` are used.

Usage

```
avgrange(pfun)
```

Arguments

`pfun`	An `Rcpp_ADFun` object.

Value

An `Rcpp_ADFun` object.

See Also

Other tape builders: `fixdynamic()`, `fixindependent()`, `keeprange()`, `tape_Hessian()`, `tape_Jacobian()`, `tape_bdryw()`, `tape_gradoffset()`, `tape_logJacdet()`, `tape_smd()`, `tape_swap()`, `tape_uld()`

`Bingham` *Score Matching Estimators for the Bingham Distribution*

Description

Score matching estimators for the Bingham distribution's parameter matrix. Two methods are available: a full score matching method that estimates the parameter matrix directly and a hybrid method by Mardia et al. (2016) that uses score matching to estimate just the eigenvalues of the parameter matrix.

Usage

```
Bingham(Y, A = NULL, w = rep(1, nrow(Y)), method = "Mardia")
```

Arguments

`Y`	A matrix of multivariate observations in Cartesian coordinates. Each row is a multivariate measurement (i.e. each row corresponds to an individual).
`A`	For full score matching only: if supplied, then NA elements of `A` are estimated and the other elements are fixed. For identifiability the final element of `diag(A)` must be `NA`.
`w`	An optional vector of weights for each measurement in `Y`
`method`	Either "Mardia" or "hybrid" for the hybrid score matching estimator from Mardia et al. (2016) or "smfull" for the full score matching estimator.

Details

The Bingham distribution has a density proportional to

$$\exp(z^T A z),$$

where A is a symmetric matrix and the trace (sum of the diagonals) of A is zero for identifiability (p181, Mardia and Jupp 2000).

The full score matching method estimates all elements of A directly except the final element of the diagonal, which is calculated from the sum of the other diagonal elements to ensure that the trace of A is zero.

The method by Mardia et al. (2016) first calculates the maximum-likelihood estimate of the eigenvectors G of A. The observations `Y` are then standardised to `Y`G. This standardisation corresponds to diagonalising A where the eigenvalues of A become the diagonal elements of the new A. The diagonal elements of the new A are then estimated using score matching, with the final diagonal element calculated from the sum of the other elements. See Mardia et al. (2016) for details.

Value

A list of `est`, `SE` and `info`.

- `est` contains the estimated matrix `A` and a vector form, `paramvec`, of `A` (ordered according to `c(diag(A)[1:(p-1)], A[upper.tri(A)])`). For the Mardia method, the estimated eigenvalues of `A` (named `evals`) and eigenvectors of `A` (named `G`) are also returned.
- `SE` contains estimates of the standard errors if computed. See `cppad_closed()`.
- `info` contains a variety of information about the model fitting procedure and results.

References

Mardia KV, Jupp PE (2000). *Directional Statistics*, Probability and Statistics. Wiley, Great Britain. ISBN 0-471-95333-4.

Mardia KV, Kent JT, Laha AK (2016). "Score matching estimators for directional distributions." doi:10.48550/arXiv.1604.08470.

See Also

Other directional model estimators: `FB()`, `vMF()`, `vMF_robust()`

Examples

```
p <- 4
A <- rsymmetricmatrix(p)
A[p,p] <- -sum(diag(A)[1:(p-1)]) #to satisfy the trace = 0 constraint
if (requireNamespace("simdd")){
  Y <- simdd::rBingham(100, A)
  Bingham(Y, method = "Mardia")
}
```

`cppad_closed` *Score Matching Estimator for Quadratic-Form Score Matching Discrepancies*

Description

For a tape of a quadratic-form score matching discrepancy function, calculates the vector of parameters such that the gradient of the score matching discrepancy is zero. Also estimates standard errors and covariance. Many score matching discrepancy functions have a quadratic form (see `scorematchingtheory`).

Usage

```
cppad_closed(
  smdtape,
  Y,
  Yapproxcentres = NA * Y,
  w = rep(1, nrow(Y)),
  approxorder = 10
)
```

Arguments

smdtape	A tape (Rcpp_ADFun object) of a score matching discrepancy function that has *quadratic form*. Test for quadratic form using testquadratic(). The smdtape's independent variables are assumed to be the model parameters to fit and the smdtape's dynamic parameter is a (multivariate) measurement.
Y	A matrix of multivariate observations. Each row is an observation. The number of columns of Y must be smdtape$size_dyn_ind.
Yapproxcentres	A matrix of Taylor approximation centres for rows of Y that require approximation. NA for rows that do not require approximation.
w	Weights for each observation.
approxorder	The order of Taylor approximation to use.

Details

When the score matching discrepancy function is of quadratic form, then the gradient of the score matching discrepancy is zero at $H^{-1}b$, where H is the average of the Hessian of the score matching discrepancy function evaluated at each measurement and b is the average of the gradient offset (see quadratictape_parts()) evaluated at each measurement. Both the Hessian and the gradient offset are constant with respect to the model parameters for quadratic-form score matching discrepancy functions.

Standard errors are estimated using the Godambe information matrix (aka sandwich method) and are only computed when the weights are constant. The estimate of the negative of the sensitivity matrix $-G$ is the average of the Hessian of smdtape evaluated at each observation in Y. The estimate of the variability matrix J is the sample covariance (denominator of $n-1$) of the gradient of smdtape evaluated at each of the observations in Y for the estimated θ. The estimated variance of the estimator is then as $G^{-1}JG^{-1}/n$, where n is the number of observations.

Taylor approximation is available because boundary weight functions and transformations of the measure in Hyvärinen divergence can remove singularities in the model log-likelihood, however evaluation at these singularities may still involve computing intermediate values that are unbounded. If the singularity is ultimately removed, then Taylor approximation from a nearby location will give a very accurate evaluation at the removed singularity.

See Also

Other generic score matching tools: Windham(), cppad_search(), tape_smd()

Examples

```
smdtape <- tape_smd("sim", "sqrt", "sph", "ppi",
              ytape = rep(1/3, 3),
              usertheta = ppi_paramvec(p = 3),
              bdryw = "minsq", acut = 0.01,
              verbose = FALSE
              )$smdtape
Y <- rppi_egmodel(100)
cppad_closed(smdtape, Y$sample)
```

`cppad_search` *Iterative Score Matching Estimator Using Conjugate-Gradient Descent*

Description

Uses conjugate gradient descent to search for a vector of parameters such that gradient of the score matching discrepancy is within tolerance of zero. Also estimates standard errors and covariance.

Usage

```
cppad_search(
  smdtape,
  theta,
  Y,
  Yapproxcentres = NA * Y,
  w = rep(1, nrow(Y)),
  approxorder = 10,
  control = list(tol = 1e-15, checkgrad = TRUE)
)
```

Arguments

`smdtape`	A tape (`Rcpp_ADFun` object) of a score matching discrepancy function. The `smdtape`'s independent variables are assumed to be the model parameters to fit and the `smdtape`'s dynamic parameter is a (multivariate) measurement.
`theta`	The starting parameter set
`Y`	A matrix of multivariate observations. Each row is an observation. The number of columns of `Y` must be `smdtape$size_dyn_ind`.
`Yapproxcentres`	A matrix of Taylor approximation centres for rows of Y that require approximation. `NA` for rows that do not require approximation.
`w`	Weights for each observation.
`approxorder`	The order of Taylor approximation to use.
`control`	Control parameters passed to `optimx::Rcgmin()`

Details

The score matching discrepancy function and gradient of the score matching function are passed to `optimx::Rcgmin()`. The call to `optimx::Rcgmin()` uses the *sum* of observations (as opposed to the mean) to reduce floating point inaccuracies. This has implications for the meaning of the control parameters passed to `Rcgmin()` (e.g. `tol`). The results are converted into averages so the use of sums can be ignored when not setting control parameters, or studying the behaviour of Rcgmin.

Standard errors use the Godambe information matrix (aka sandwich method) and are only computed when the weights are constant. The estimate of the sensitivity matrix G is the negative of the average over the Hessian of `smdtape` evaluated at each observation in `Y`. The estimate of the variability matrix J is then the sample covariance (denominator of $n-1$) of the gradient of `smdtape` evaluated at each of the observations in `Y` for the estimated θ. The variance of the estimator is then estimated as $G^{-1}JG^{-1}/n$, where `n` is the number of observations.

Taylor approximation is available because boundary weight functions and transformations of the measure in Hyvärinen divergence can remove singularities in the model log-likelihood, however evaluation at these singularities may still involve computing intermediate values that are unbounded. If the singularity is ultimately removed, then Taylor approximation from a nearby location will give a very accurate evaluation at the removed singularity.

See Also

Other generic score matching tools: `Windham()`, `cppad_closed()`, `tape_smd()`

Examples

```
smdtape <- tape_smd("sim", "sqrt", "sph", "ppi",
              ytape = rep(1/3, 3),
              usertheta = ppi_paramvec(p = 3),
              bdryw = "minsq", acut = 0.01,
              verbose = FALSE
              )$smdtape
Y <- rppi_egmodel(100)
cppad_search(smdtape, 0.9 * Y$theta, Y$sample)
sum((smvalues_wsum(smdtape, Y$sample, Y$theta)$grad/nrow(Y$sample))^2)
```

---

dppi *Improper Log-Density of the PPI Model*

Description

Compute the **natural logarithm** of the improper density for the PPI model for the given matrix of measurements `Y`. Rows with negative values or with a sum that differs from `1` by more than `1E-15` are assigned a value of `-Inf`.

Usage

```
dppi(Y, ..., paramvec = NULL)
```

Arguments

Y	A matrix of measurements in the simplex. Each row is a multivariate measurement.
...	Arguments passed on to ppi_paramvec
	AL Either NULL, a p-1 x p-1 symmetric matrix, a number, or "diag". If NULL then all A_L elements will be set to NA. If a single number, then A_L will be fixed as a matrix of the given value. If "diag" then the non-diagonal elements of A_L will be fixed to 0, and the diagonal will be NA.
	bL Either NULL, a number, or a vector of length p-1. If NULL, then all elements of b_L will be set to NA. If a single number, then b_L will be fixed at the supplied value.
	beta Either NULL, a number, or a vector of length p. If NULL then all elements of β will be set to NA. If a single number then the β elements will be fixed at the given number.
	betaL Either NULL, a number, or a vector of length p-1. If NULL then the 1...(p-1)th β elements will be set to NA. If a single number then the 1...(p-1)th β elements will be fixed at the given number.
	betap Either NULL or a number. If NULL then the pth element of β will be set to NA, and ppi() will estimate it. If a number, then the pth element of β will be fixed at the given value.
	p The number of components. If NULL then p will be inferred from other inputs.
	Astar The A^* matrix (a p by p symmetric matrix)
paramvec	The PPI parameter vector, created easily using ppi_paramvec() and also returned by ppi(). Use paramvec instead of ....

Details

The value calculated by dppi is

$$z_L^T A_L z_L + b_L^T z_L + \beta^T \log(z),$$

where z is the multivariate observation (i.e. a row of Y), and z_L omits the final element of z.

See Also

Other PPI model tools: ppi(), ppi_param_tools, ppi_robust(), rppi()

Examples

```
m <- rppi_egmodel(10)
dppi(m$sample, paramvec = m$theta)
```

evaltape *Evaluate a CppAD Tape Many Times*

Description

Evaluates a tape exactly or approximately for an array of provided variable values and dynamic parameter values. The function `evaltape_wsum()` computes the weighted sum of each column of the `evaltape()` result.

Usage

```
evaltape(tape, xmat, pmat, xcentres = NA * xmat, approxorder = 10)

evaltape_wsum(
  tape,
  xmat,
  pmat,
  w = NULL,
  xcentres = NA * xmat,
  approxorder = 10
)
```

Arguments

`tape`	An `Rcpp_ADFun` object (i.e. a tape of a function).
`xmat`	A matrix of (multivariate) independent variables where each represents a single independent variable vector. Or a single independent variable vector that is used for all rows of `pmat`.
`pmat`	A matrix of dynamic parameters where each row specifies a new set of values for the dynamic parameters of `tape`. Or a single vector of dynamic parameters to use for all rows of `xmat`.
`xcentres`	A matrix of approximation for Taylor approximation centres for `xmat`. Use values of `NA` for rows that do not require Taylor approximation.
`approxorder`	Order of Taylor approximation
`w`	Weights to apply to each row of `xmat` for computing the weighted sum. If `NULL` then each row is given a weight of `1`.

Details

Approximation is via Taylor approximation of the independent variable around the approximation centre provided in `xcentres`.

Value

A matrix, each row corresponding to the evaluation of the same row in `xmat`, `pmat` and `xcentres`.

See Also

Other tape evaluators: quadratictape_parts(), smvalues(), testquadratic()

Examples

```
u <- rep(1/3, 3)
tapes <- tape_smd("sim", "sqrt", "sph", "ppi",
              ytape = u,
              usertheta = ppi_paramvec(p = 3),
              bdryw = "minsq", acut = 0.01,
              verbose = FALSE
              )
evaltape(tapes$lltape, u, rppi_egmodel(1)$theta)
evaltape(tapes$smdtape, rppi_egmodel(1)$theta, u)
evaltape(tapes$lltape, rbind(c(0, 0, 1), c(0,0,1)),
         rppi_egmodel(1)$theta,
         xcentres = rbind(c(0.0005, 0.0005, 0.999), NA))
```

FB *Estimate the Fisher-Bingham Distribution*

Description

Estimates parameters for the Fisher-Bingham distribution using score-matching.

Usage

```
FB(Y, km = NULL, A = NULL)
```

Arguments

Y	A matrix of multivariate observations in Cartesian coordinates. Each row is a multivariate measurement (i.e. each row corresponds to an individual).
km	Optional. A vector of the same length as the dimension, representing the parameter vector for the von Mises-Fisher component (i.e. the $\kappa\mu$ see vMF()). If supplied, the non-NA elements are fixed.
A	Optional. The Bingham matrix. If supplied the non-NA elements of the Bingham matrix are fixed. The final element of the diagonal of A must be NA as the software calculates this value to ensure the trace of the Bingham matrix is zero.

Details

The density of the Fisher-Bingham distribution is proportional to

$$\exp(z^T A z + \kappa \mu^T z),$$

where A is a matrix as in the Bingham distribution, and κ and μ are the concentration and mean direction, respectively, as in the von Mises-Fisher distribution.

Warning: Slow Convergence with Sample Size

Score matching estimates of all elements of A and $\kappa\mu$ converge slowly with sample size. Even with a million simulated measurements, the gradient of the score matching discrepancy at the true parameters can have size (L2 Euclidean norm) more than 0.001, which is substantially non-zero.

See Also

Other directional model estimators: `Bingham()`, `vMF()`, `vMF_robust()`

Examples

```
p <- 3
A <- rsymmetricmatrix(p, -10, 10)
A[p,p] <- -sum(diag(A)[1:(p-1)]) #to satisfy the trace = 0 constraint
m <- runif(p, -10, 10)
m <- m / sqrt(sum(m^2))
if (requireNamespace("simdd")){
  Y <- simdd::rFisherBingham(1000, 2 * m, A)
  FB(Y)
}
```

`fixdynamic` *Fix Dynamic Parameters of a Tape*

Description

Retapes an existing CppAD tape but with some of original dynamic parameters fixed to specified values. For creating this tape, the values of `pfun$xtape` is used.

Usage

```
fixdynamic(pfun, theta, isfixed)
```

Arguments

`pfun`	An Rcpp_ADFun object.
`theta`	A numerical vector specifying the value of all dynamic parameters of `pfun`. Some of these will be fixed according to `isfixed`, the remainder will remain dynamic.
`isfixed`	A boolean vector same length as `theta`. `TRUE` values are fixed at the value of `theta`, `FALSE` values are left dynamic.

Value

An Rcpp_ADFun object.

See Also

Other tape builders: avgrange(), fixindependent(), keeprange(), tape_Hessian(), tape_Jacobian(), tape_bdryw(), tape_gradoffset(), tape_logJacdet(), tape_smd(), tape_swap(), tape_uld()

`fixindependent` *Fix Independent Arguments of a Tape*

Description

Retapes an existing CppAD tape but with some of original independent arguments fixed to specified values. For creating this tape, the values of `pfun$dyntape` are used.

Usage

```
fixindependent(pfun, x, isfixed)
```

Arguments

`pfun`	An Rcpp_ADFun object.
`x`	A numerical vector specifying the value of all independent arguments of `pfun`. Some of these will be fixed according to `isfixed`, the remainder will remain as independent arguments.
`isfixed`	A boolean vector same length as `x`. `TRUE` values are fixed at the value of `x`, `FALSE` values are left as independent arguments.

Value

An Rcpp_ADFun object.

See Also

Other tape builders: avgrange(), fixdynamic(), keeprange(), tape_Hessian(), tape_Jacobian(), tape_bdryw(), tape_gradoffset(), tape_logJacdet(), tape_smd(), tape_swap(), tape_uld()

`keeprange` *Reduce Range Dimension of a Tape*

Description

Retapes an existing CppAD tape omitting some of the returned elements. For creating this tape, the values of `pfun$dyntape` and `pfun$xtape` are used.

Usage

```
keeprange(pfun, keep)
```

Arguments

pfun	An Rcpp_ADFun object.
keep	Integers (lowest of 1, highest of pfun$range) specifying which elements of the range to keep. To keep all pass keep = seq(1, pfun$range).

Value

An Rcpp_ADFun object.

See Also

Other tape builders: avgrange(), fixdynamic(), fixindependent(), tape_Hessian(), tape_Jacobian(), tape_bdryw(), tape_gradoffset(), tape_logJacdet(), tape_smd(), tape_swap(), tape_uld()

---

microbiome *16s Microbiome Data for Soil-Transmitted Helminths*

Description

The microbiome data contains paired DNA samples from before treatment and 21 months after treatment for helminth infections (Martin et al. 2019). This data was analysed by Martin et al. (2019) and a further subset was studied by Scealy and Wood (2023). The data are from a study into the effect of helminth infections on the course of malaria infections (ImmunoSPIN-Malaria) in the Nangapanda subdistrict, Indonesia (Wiria et al. 2010). As part of the study, some participants were given 400mg of albendazole every three months for 1.5 years, remaining participants were given a placebo (Wiria et al. 2010).

Usage

```
microbiome
```

Format

A dataframe with 300 rows (two rows per individual) and 31 columns:

IndividualID An integer uniquely specifying the individual.

Year The collection year for the sample. 2008 for before treatment. 2010 for after treatment.

Sex 1 if female, 0 otherwise.

Treatment TRUE if individual given 400mg of albendazole every three months for 1.5 years, FALSE otherwise.

Age Age at first sample.

ct_Al A Helminth measurement: The qPCR cycle threshold (CT) for *Ascaris lumbricoides* (large roundworm). *Ascaris lumbricoides* can be considered present if the value is 30 or less.

ct_Na A Helminth measurement: The qPCR cycle threshold (CT) for *Necator americanus* (a hookworm). *Necator americanus* can be considered present if the value is 30 or less.

ct_Ad A Helminth measurement: The qPCR cycle threshold (CT) for *Ancylostoma duodenale* (a hookworm). *Ancylostoma duodenale* can be considered present if the value is 30 or less.

micr_Tt A Helminth measurement: The presence of *Trichuris trichiura* as determined by microscopy. A value of `TRUE` means *Trichuris trichiura* was detected.

Helminth A Helminth measurement: If any of the above helminths were detected then `TRUE`, otherwise `FALSE`.

Remaining columns Count prevalence of 18 bacterial phyla and 2 unclassified columns.

Details

The measurements in the data come from stool samples before and after treatment. Gut microbiome prevalence was measured using 16s rRNA 454 sequencing (Martin et al. 2019). Helminth infections were detected by PCR or microscopy (Martin et al. 2019).

The subset studied by Scealy and Wood (2023) contained only the measurements from before treatment, and only those individuals with a helminth infection. These measurements can be obtained by running

```
microbiome[(microbiome$Year == 2008) & microbiome$Helminth, ]
```

Two further individuals (`IndividualID` of `2079` and `2280`) were deemed outliers by Scealy and Wood (2023).

Modifications from the Source

The `microbiome` data was created from the file `S1_Table.xlsx` hosted on `Nematode.net` at `http://nematode.net/Data/environmental_interaction/S1_Table.xlsx` using the below code.

```
microbiome <- readxl::read_excel("S1_Table.xlsx",
  range = "A3:AE303") #avoids the genus data, keeping - only phyla
metacolnames <- readxl::read_excel("S1_Table.xlsx",
  range = "A2:J2",
  col_names = FALSE)
colnames(microbiome)[1:ncol(metacolnames)] <- metacolnames[1, ]
colnames(microbiome)[2] <- "Year"
microbiome[, 11] <- (microbiome$ct_Al <= 30) | (microbiome$ct_Na <= 30) |
  (microbiome$ct_Ad <= 30) | (microbiome$ct_St <= 30) |
  (microbiome$micr_Tt == 1)
colnames(microbiome)[11] <- "Helminth"
microbiome <- microbiome |>
  dplyr::mutate(across(c(1,2,3,12:31), as.integer)) |>
  dplyr::mutate(micr_Tt = as.logical(micr_Tt),
                Treatment = as.logical(Treatment)) |>
  dplyr::rename(IndividualID = `Individual ID`)
microbiome <- as.data.frame(microbiome)
```

Source

`http://nematode.net/Data/environmental_interaction/S1_Table.xlsx` from `http://nematode.net`. `S1_Table.xlsx` was created by Dr. Bruce A Rosa for Martin et al. (2019). Permission to share this data was obtained from Dr. Bruce Rosa and Dr. Ivonne Martin.

References

Martin I, Uh H, Supali T, Mitreva M, Houwing-Duistermaat JJ (2019). "The mixed model for the analysis of a repeated-measurement multivariate count data." *Statistics in Medicine*, **38**(12), 2248–2268. doi:10.1002/sim.8101.

Scealy JL, Wood ATA (2023). "Score matching for compositional distributions." *Journal of the American Statistical Association*, **118**(543), 1811–1823. doi:10.1080/01621459.2021.2016422.

Wiria AE, Prasetyani MA, Hamid F, Wammes LJ, Lell B, Ariawan I, Uh HW, Wibowo H, Djuardi Y, Wahyuni S, Sutanto I, May L, Luty AJ, Verweij JJ, Sartono E, Yazdanbakhsh M, Supali T (2010). "Does treatment of intestinal helminth infections influence malaria?" *BMC Infectious Diseases*, **10**, 77. doi:10.1186/147123341077.

`ppi` *Estimation of Polynomially-Tilted Pairwise Interaction (PPI) Model*

Description

Estimates the parameters of the Polynomially-Tilted Pairwise Interaction (PPI) model (Scealy and Wood 2023) for compositional data. By default `ppi()` uses `cppad_closed()` to find estimate. For many situations a hard-coded implementation of the score matching estimator is also available.

For a given parameter vector `evalparam`, `ppi_smvalues()` computes the score matching discrepancy, the gradient and the Hessian of the score matching discrepancy (see `smvalues()`) and the gradient offset of the score matching discrepancy (see `quadratictape_parts()` and `tape_gradoffset()`).

Usage

```
ppi(
  Y,
  paramvec = NULL,
  trans,
  method = "closed",
  w = rep(1, nrow(Y)),
  constrainbeta = FALSE,
  bdryw = "ones",
  acut = NULL,
  bdrythreshold = 1e-10,
  shiftsize = bdrythreshold,
  approxorder = 10,
  control = list(tol = 1e-15, checkgrad = TRUE),
  paramvec_start = NULL
)

ppi_smvalues(
  Y,
  paramvec = NULL,
```

```
  evalparam,
  trans,
  method = "closed",
  w = rep(1, nrow(Y)),
  bdryw = "ones",
  acut = NULL,
  bdrythreshold = 1e-10,
  shiftsize = bdrythreshold,
  approxorder = 10,
  average = TRUE
)
```

Arguments

Y	A matrix of measurements. Each row is a compositional measurement (i.e. each row sums to 1 and has non-negative elements).
paramvec	Optionally a vector of the PPI models parameters. NA-valued elements of this vector are estimated and non-NA values are fixed. Generate paramvec easily using ppi_paramvec(). If NULL then all elements of A_L, b_L and β are estimated.
trans	The name of the transformation of the manifold in Hyvärinen divergence (See scorematchingtheory): "clr" (centred log ratio), "alr" (additive log ratio), "sqrt" or "none".
method	"closed" uses CppAD to solve in closed form the a quadratic score matching discrepancy using cppad_closed(). "hardcoded" uses hardcoded implementations. "iterative" uses cppad_search() (which uses CppAD and optimx::Rcgmin()) to iteratively find the minimum of the weighted Hyvärinen divergence.
w	Weights for each observation, if different observations have different importance. Used by Windham() and ppi_robust() for robust estimation.
constrainbeta	If TRUE, elements of β that are less than -1 are converted to -1 + 1E-7.
bdryw	The boundary weight function for down weighting measurements as they approach the manifold boundary. Either "ones", "minsq" or "prodsq". See details and tape_bdryw_inbuilt().
acut	The threshold a_c in bdryw to avoid over-weighting measurements interior to the simplex
bdrythreshold	iterative or closed methods only. For measurements within bdrythreshold of the simplex boundary a Taylor approximation is applied by shifting the measurement shiftsize towards the center of the simplex.
shiftsize	iterative or closed methods only. For measurements within bdrythreshold of the simplex boundary a Taylor approximation is applied by shifting the measurement shiftsize towards the center of the simplex.
approxorder	iterative or closed methods only. Order of the Taylor approximation for measurements on the boundary of the simplex.
control	iterative only. Passed to optimx::Rcgmin() to control the iterative solver.
paramvec_start	iterative method only. The starting guess for Rcgmin. Generate paramvec_start easily using ppi_paramvec().

`evalparam` The parameter set to evaluate the score matching values. This is different to `paramvec`, which specifies which parameters to estimate. All elements of `evalparam` must be non-NA, and any parameters fixed by `paramvec` must have the same value in `evalparam`.

`average` If TRUE return the (weighted average) of the measurements, otherwise return the values for each measurement.

Details

Estimation may be performed via transformation of the measure in Hyvärinen divergence from Euclidean space to the simplex (inverse of the additive log ratio transform), from a hyperplane to the simplex (inverse of the centred log ratio transform), from the positive quadrant of the sphere to the simplex (inverse of the square root transform), or without any transformation. In the latter two situations there is a boundary and *weighted Hyvärinen divergence* (Equation 7, Scealy and Wood 2023) is used. Properties of the estimator using the square root transform were studied by Scealy and Wood (2023). Properties of the estimator using the additive log ratio transform were studied by Scealy et al. (2024).

There are three boundary weight functions available (see `tape_bdryw_inbuilt()` for more details): "ones", "minsq", "prodsq".

Scealy and Wood (Theorem 1, Scealy and Wood 2023) prove that minimising the weighted Hyvärinen Divergence is equivalent to minimising $\psi(f, f_0)$ (See `scorematchingtheory`) when the boundary weight function is smooth or for the functions "minsq" and "prodsq" above when the manifold is the simplex or positive orthant of a sphere.

Hard-coded estimators are available for the following situations:

- Square root transformation ("sqrt") with the "minsq" boundary weight function:
  - full parameter vector (`paramvec` not provided)
  - `paramvec` fixes only the final element of β
  - `paramvec` fixes all elements of β
  - `paramvec` fixes $b_L = 0$ and provides fixed values of β
  - `paramvec` fixes $A_L = 0$ and $b_L = 0$, leaving β to be fitted.
- Square root transformation ("sqrt") with the "prodsq" boundary weight function:
  - `paramvec` fixes all elements of β
  - `paramvec` fixes $b_L = 0$ and provides fixed values of β
  - `paramvec` fixes $A_L = 0$ and $b_L = 0$, leaving β to be fitted.
- The additive log ratio transformation ("alr") using the final component on the denominator, with $b_L = 0$ and fixed final component of β.

Value

`ppi()` returns: A list of `est`, `SE` and `info`.

- `est` contains the estimates in vector form, `paramvec`, and as A_L, b_L and β.
- `SE` contains estimates of the standard errors if computed. See `cppad_closed()`.
- `info` contains a variety of information about the model fitting procedure and results.

`ppi_smvalues()` returns a list of

- `obj` the score matching discrepancy value
- `grad` the gradient of the score matching discrepancy
- `hess` the Hessian of the score matching discrepancy
- `offset` gradient offset (see `quadratictape_parts()`)

PPI Model

The PPI model density is proportional to

$$\exp(z_L^T A_L z_L + b_L^T z_L) \prod_{i=1}^{p} z_i^{\beta_i},$$

where p is the dimension of a compositional measurement z, and z_L is z without the final (pth) component. A_L is a $p-1 \times p-1$ symmetric matrix that controls the covariance between components. b_L is a $p-1$ vector that controls the location within the simplex. The ith component β_i of β controls the concentration of density when z_i is close to zero: when $\beta_i \geq 0$ there is no concentration and β_i is hard to identify; when $\beta_i < 0$ then the probability density of the PPI model increases unboundedly as z_i approaches zero, with the increasing occurring more rapidly and sharply the closer β_i is to -1.

References

Scealy JL, Hingee KL, Kent JT, Wood ATA (2024). "Robust score matching for compositional data." *Statistics and Computing*, **34**, 93. doi:10.1007/s1122202410412w.

Scealy JL, Wood ATA (2023). "Score matching for compositional distributions." *Journal of the American Statistical Association*, **118**(543), 1811–1823. doi:10.1080/01621459.2021.2016422.

See Also

Other PPI model tools: `dppi()`, `ppi_param_tools`, `ppi_robust()`, `rppi()`

Examples

```
model <- rppi_egmodel(100)
estalr <- ppi(model$sample,
              paramvec = ppi_paramvec(betap = -0.5, p = ncol(model$sample)),
              trans = "alr")
estsqrt <- ppi(model$sample,
              trans = "sqrt",
              bdryw = "minsq", acut = 0.1)
```

ppi_cW *Quickly Generate a Vector of Windham Exponents for the PPI Model*

Description

These functions help to quickly generate a set of Windham exponents for use in `ppi_robust()` or `Windham()`. Rows and columns of A_L and b_L corresponding to components with strong concentrations of probability mass near zero have non-zero constant tuning exponent, and all other elements have a tuning constant of zero. All elements of β have a tuning exponent of zero.

The function `ppi_cW_auto()` automatically detects concentrations near zero by fitting a PPI distribution with $A_L = 0$ and $b_L = 0$ (i.e. a Dirichlet distribution) with the centred log-ratio transformation of the manifold.

Usage

```
ppi_cW(cW, ...)

ppi_cW_auto(cW, Y)
```

Arguments

`cW`	The value of the non-zero Windham tuning exponents.
`...`	Values of `TRUE` or `FALSE` in the same order of the components specifying that a component has probability mass concentrated near zero.
`Y`	A matrix of observations

Details

The Windham robustifying method involves weighting observations by a function of the proposed model density (Windham 1995). Scealy et al. (2024) found that only some of the tuning constants should be non-zero: the tuning exponents corresponding to β should be zero to avoid infinite weights;and to improve efficiency any rows or columns of A_L corresponding to components without concentrations of probability mass (i.e. outliers can't exist) should have exponents of zero. Scealy et al. (2024) set the remaining tuning exponents to a constant.

Value

A vector of the same length as the parameter vector of the PPI model. Elements of A_L will have a value of `cW` if both their row and column component has probability mass concentrated near zero. Similarly, elements of b_L will have a value of `cW` if their row corresponds to a component that has a probability mass concentrated near zero. All other elements are zero.

References

Scealy JL, Hingee KL, Kent JT, Wood ATA (2024). "Robust score matching for compositional data." *Statistics and Computing*, **34**, 93. doi:10.1007/s1122202410412w.

Windham MP (1995). "Robustifying Model Fitting." *Journal of the Royal Statistical Society. Series B (Methodological)*, **57**(3), 599–609. 2346159, http://www.jstor.org/stable/2346159.

Examples

```
Y <- rppi_egmodel(100)$sample
ppi_cW_auto(0.01, Y)
ppi_cW(0.01, TRUE, TRUE, FALSE)
```

`ppi_mmmm` *A PPI Score-Matching Marginal Moment Matching Estimator (dimension=3 only)*

Description

Computes a marginal moment matching estimator (Section 6.2, Scealy and Wood 2023), which assumes β is a known vector with the same value in each element, and $b_L = 0$. Only A_L is estimated.

Usage

```
ppi_mmmm(Y, ni, beta0, w = rep(1, nrow(Y)))
```

Arguments

`Y`	Count data, each row is a multivariate observation.
`ni`	The total for each sample (sum across rows)
`beta0`	$\beta = \beta_0$ is the same for each component.
`w`	Weights for each observation. Useful for weighted estimation in `Windham()`.

Details

$\beta = \beta_0$ is fixed and not estimated. b_L is fixed at zero. See (Section 6.2 and A.8 of Scealy and Wood 2023). The boundary weight function in the score matching discrepancy is the unthresholded product weight function

$$h(z)^2 = \min\left(\prod_{j=1}^{p} z_j^2, a_c^2\right).$$

Value

A vector of estimates for A_L entries (diagonal and off diagonal).

References

Scealy JL, Wood ATA (2023). "Score matching for compositional distributions." *Journal of the American Statistical Association*, **118**(543), 1811–1823. doi:10.1080/01621459.2021.2016422.

`ppi_param_tools` *PPI Parameter Tools*

Description

The default parameterisation of the PPI model is a symmetric covariance-like matrix A_L, a location-like vector b_L and a set of Dirichlet exponents β. For `p` components, A_L has `p-1` rows, b_L is a vector with `p-1` elements and β is a vector with `p` elements. For score matching estimation this form of the parameters must be converted into a single parameter vector using `ppi_paramvec()`. `ppi_paramvec()` also includes easy methods to set parameters to `NA` for estimation with `ppi()` (in `ppi()` the NA-valued elements are estimated and all other elements are fixed). The reverse of `ppi_paramvec()` is `ppi_parammats()`. An alternative parametrisation of the PPI model uses a single `p` by `p` matrix A^* instead of A_L and b_L, and for identifiability A^* is such that $1^T A^* 1 = 0$ where $1 = (1, 1, ..., 1)$ and $0 = (0, 0, ..., 0)$ (Scealy and Wood 2023). Convert between parametrisations using `ppi_toAstar()` and `ppi_fromAstar()`.

Usage

```
ppi_paramvec(
  p = NULL,
  AL = NULL,
  bL = NULL,
  Astar = NULL,
  beta = NULL,
  betaL = NULL,
  betap = NULL
)

ppi_parammats(paramvec)

ppi_toAstar(AL, bL)

ppi_fromAstar(Astar)
```

Arguments

`p`	The number of components. If `NULL` then `p` will be inferred from other inputs.
`AL`	Either `NULL`, a p-1 x p-1 symmetric matrix, a number, or "diag". If NULL then all A_L elements will be set to NA. If a single number, then A_L will be fixed as a matrix of the given value. If "diag" then the non-diagonal elements of A_L will be fixed to 0, and the diagonal will be `NA`.

bL	Either NULL, a number, or a vector of length p-1. If NULL, then all elements of b_L will be set to NA. If a single number, then b_L will be fixed at the supplied value.
Astar	The A^* matrix (a p by p symmetric matrix)
beta	Either NULL, a number, or a vector of length p. If NULL then all elements of β will be set to NA. If a single number then the β elements will be fixed at the given number.
betaL	Either NULL, a number, or a vector of length p-1. If NULL then the 1...(p-1)th β elements will be set to NA. If a single number then the 1...(p-1)th β elements will be fixed at the given number.
betap	Either NULL or a number. If NULL then the pth element of β will be set to NA, and ppi() will estimate it. If a number, then the pth element of β will be fixed at the given value.
paramvec	A PPI parameter vector, typically created by ppi_paramvec() or as an output of ppi().

Details

ppi_paramvec() returns a vector starting with the diagonal elements of A_L, then the off-diagonal elements extracted by upper.tri() (which extracts elements of A_L along each row, left to right, then top to bottom), then b_L, then β.

The Astar parametrisation rewrites the PPI density as proportional to

$$\exp(z^T A^* z) \prod_{i=1}^{p} z_i^{\beta_i},$$

where A^* (Astar) is a p by p matrix. Because z lies in the simplex (in particular $\sum z_i = 1$), the density is the same regardless of the value of $1^T A^* 1$=sum(Astar), where 1 is the vector of ones. Thus A_L and b_L specify A^* up to an additive factor. In the conversion ppi_toAstar(), A^* is returned such that $1^T A^* 1 = 0$. NULL values or NA elements are not allowed for ppi_toAstar() and ppi_fromAstar().

Value

ppi_paramvec(): a vector of length $p + (p-1)(2 + (p-1)/2)$.

ppi_parammats(): A named list of A_L, b_L, and β.

ppi_toAstar(): The matrix A^*.

ppi_fromAstar(): A list of the matrix A_L, the vector b_L and a discarded constant.

PPI Model

The PPI model density is proportional to

$$\exp(z_L^T A_L z_L + b_L^T z_L) \prod_{i=1}^{p} z_i^{\beta_i},$$

where p is the dimension of a compositional measurement z, and z_L is z without the final (pth) component. A_L is a $p-1 \times p-1$ symmetric matrix that controls the covariance between components.

b_L is a $p-1$ vector that controls the location within the simplex. The ith component β_i of β controls the concentration of density when z_i is close to zero: when $\beta_i \geq 0$ there is no concentration and β_i is hard to identify; when $\beta_i < 0$ then the probability density of the PPI model increases unboundedly as z_i approaches zero, with the increasing occurring more rapidly and sharply the closer β_i is to -1.

See Also

Other PPI model tools: `dppi()`, `ppi()`, `ppi_robust()`, `rppi()`

Examples

```
ppi_paramvec(AL = "diag", bL = 0, betap = -0.5, p = 3)
vec <- ppi_paramvec(AL = rsymmetricmatrix(2), beta = c(-0.8, -0.7, 0))
ppi_parammats(vec)
Astar <- rWishart(1, 6, diag(3))[,,1]
ppi_fromAstar(Astar)
ppi_toAstar(ppi_fromAstar(Astar)$AL, ppi_fromAstar(Astar)$bL)
```

---

ppi_robust *Robustly Estimate Parameters of the PPI Distribution*

Description

`ppi_robust()` uses `Windham()` and `ppi()` to estimate a PPI distribution robustly. There are many arguments to the `ppi()` function and we highly recommend testing your arguments on `ppi()` first before running `ppi_robust()`.

`ppi_robust_alrgengamma()` performs the Windham robustification algorithm exactly as described in Scealy et al. (2024) for score matching via log-ratio transform of the PPI model with $b_L = 0$. This function calls the more general `Windham()` and `ppi()`.

Usage

```
ppi_robust(Y, cW, ...)

ppi_robust_alrgengamma(
  Y,
  cW,
  ...,
  fpcontrol = list(Method = "Simple", ConvergenceMetricThreshold = 1e-10)
)
```

Arguments

`Y`	A matrix of measurements. Each row is a measurement, each component is a dimension of the measurement.
`cW`	A vector of robustness tuning constants. Easy to build using `ppi_cW()` and `ppi_cW_auto()`. See `Windham()` for more details on `cW`.

...	Passed to `Windham()` and on to `ppi()`.
fpcontrol	A named list of control arguments to pass to `FixedPoint::FixedPoint()` for the fixed point iteration.

Details

`ppi_robust_alrgengamma()`: must fit a PPI model via additive-log ratio transform of the simplex with $b_L = 0$ fixed and the final element of β fixed. The default convergence metric and threshold are different to the default for `ppi_robust()` to match the implementation in (Scealy et al. 2024): convergence is measured by the change in the first element of β, and convergence is reached when the change is smaller than `1E-6`. Override this behaviour by specifying the elements `ConvergenceMetric` and `ConvergenceMetricThreshold` in a list passed as `fpcontrol`. `Windham()` is called with `alternative_populationinverse = TRUE`.

Value

A list:

- `est` The estimated parameters in vector form (`paramvec`) and as `AL`, `bL` and `beta`.
- `SE` "Not calculated." Returned for consistency with other estimators.
- `info` Information returned in the `optim` slot of `Windham()`. Includes the final weights in `finalweights`.

References

Scealy JL, Hingee KL, Kent JT, Wood ATA (2024). "Robust score matching for compositional data." *Statistics and Computing*, **34**, 93. doi:10.1007/s1122202410412w.

See Also

Other PPI model tools: `dppi()`, `ppi()`, `ppi_param_tools`, `rppi()`

Other Windham robustness functions: `Windham()`, `vMF_robust()`

Examples

```
set.seed(7)
model <- rppi_egmodel(100)
estsqrt <- ppi_robust(model$sample,
  cW = ppi_cW_auto(0.01, model$sample),
  paramvec_start = model$theta,
  trans = "sqrt", bdryw = "minsq", acut = 0.1)
set.seed(14)
model <- rppi_egmodel(100)
ppi_robust_alrgengamma(model$sample,
   cW = ppi_cW_auto(0.01, model$sample),
   paramvec = ppi_paramvec(betap = -0.5, p = ncol(model$sample)))
```

print,Rcpp_ADFun *Print or show a summary of an Rcpp_ADFun*

Description

Both `print()` and `show()` will display a summary of a Rcpp_ADFun object.

Usage

```
## S4 method for signature 'Rcpp_ADFun'
print(x, ...)

## S4 method for signature 'Rcpp_ADFun'
show(object)
```

Arguments

`x`	An object of class Rcpp_ADFun.
`...`	Passed to `format()`.
`object`	An object of class Rcpp_ADFun.

Details

The `show()` method overrides the default `show()` method for `Rcpp::C++Object` objects from the `Rcpp` package.

quadratictape_parts *Evaluate the Hessian and Gradient Offset of a Taped Quadratic Function*

Description

When the score matching discrepancy function is quadratic then the gradient of the score matching discrepancy function can be written using the Hessian and an offset term. This can be useful for solving for the situation when the gradient is zero. The Hessian and offset term are computed using `CppAD` tapes. Taylor approximation can be used for locations at removed singularities (i.e. where intermediate values are unbounded). `quadratictape_parts()` will error if `testquadratic(tape)` returns `FALSE`.

Usage

```
quadratictape_parts(tape, tmat, tcentres = NA * tmat, approxorder = 10)
```

Arguments

tape	A tape of a quadratic function where the independent and dynamic parameters correspond to the x and t in the details section, respectively. For score matching tape should be a tape of the score matching discrepancy function $A(z) + B(z) + C(z)$ in scorematchingtheory with z the *dynamic parameters* and the model parameters the *independent variable* (which is the usual for the return of tape_smd()).
tmat	A matrix of vectors corresponding to values of t (see details). Each row corresponds to a vector. For score matching, these vectors are measurements.
tcentres	A matrix of Taylor approximation centres for rows of tmat that require approximation. NA for rows that do not require approximation.
approxorder	The order of the Taylor approximation to use.

Details

A quadratic function can be written

$$f(x;t) = \frac{1}{2}x^T W(t)x + b(t)^T x + c,$$

where t is considered a vector that is constant with respect to the differentiation. The Hessian of the function is with respect to x is

$$Hf(x;t) = \frac{1}{2}(W(t) + W(t)^T).$$

The gradient of the function with respect to x can then be written

$$\Delta f(x;t) = Hf(x;t)x + b(t)^T x,$$

where the Hessian and offset $b(t)$ depend only on t.

The functions here evaluate the Hessian and offset $b(t)$ for many values of t. Tapes of the Hessian and gradient offset are created using tape_Hessian() and tape_gradoffset() respectively. These tapes are then evaluated for every row of tmat. When the corresponding tcentres row is not NA, then approximate (but very accurate) results are calculated using Taylor approximation around the location given by the row of tcentres.

For score matching x is the set of model parameters and the vector t is a (multivariate) measurement.

Value

A list of

- offset Array of offsets $b(t)$, each row corresponding to a row in tmat
- Hessian Array of vectorised $Hf(x;t)$ (see tape_Hessian()), each row corresponding to a row in tmat. For each row, obtain the Hessian in matrix format by using matrix(ncol = length(tape$xtape)).

See Also

Other tape evaluators: evaltape(), smvalues(), testquadratic()

Examples

```
u <- rep(1/3, 3)
smdtape <- tape_smd("sim", "sqrt", "sph", "ppi",
              ytape = u,
              usertheta = ppi_paramvec(p = 3),
              bdryw = "minsq", acut = 0.01,
              verbose = FALSE
              )$smdtape
quadratictape_parts(smdtape,
  tmat = rbind(u, c(1/4, 1/4, 1/2)))
```

Rcpp_ADFun-class *A Class for CppAD Tapes*

Description

Objects of type Rcpp_ADFun contain a tape of a C++ function (which has class ADFun in CppAD). These tapes are a record of operations performed by a function. Tapes can be evaluated and differentiated, and have properties (such as domain and range dimensions). Tapes also have dynamic parameters that can be updated. This class, Rcpp_ADFun uses reference semantics, so that copies all point to the same object and changes modify in place (i.e. changes modify the same object). Properties and methods of an Rcpp_ADFun object are accessed via $.

Details

An object of class Rcpp_ADFun wraps an ADFun object from CppAD. Many of the properties and behaviour of an Rcpp_ADFun object come directly from ADFun objects so more details and context can be found by looking at the ADFun object help in the CppAD help. The methods eval(), Jac() and Hes() have been added by scorematchingad as there were many cases where this seemed like an easier way to evaluate a tape.

Default printing of an Rcpp_ADFun object gives a short summary of the object, see print,Rcpp_ADFun.

Tapes cannot be saved from session to session.

Methods - Tape Properties:

- $size_order Number of Taylor coefficient orders, per variable and direction, currently calculated and stored in the object.
- $domain Dimension of the domain space (i.e., length of the independent variables vector x).
- $range Dimension of the range space (i.e., length of the vector returned by $eval()).
- $size_dyn_ind Number of independent dynamic parameters (i.e., length of the vector of dynamic parameters dyn).
- $name A name for the tape (may be empty). This is yet to incorporate the CppAD function_name property.
- $xtape The values of the independent variables used for the initial taping.
- $dyntape The values of the dynamic parameters used for the initial taping.

- `$get_check_for_nan()` Debugging: Return whether the tape is configured to check for NaN values during computation. The check for NaN only occurs if the `C++` compilation enables debugging.
- `$set_check_for_nan(bool)` Set whether the tape should check for NaN values during computation (only effective if C++ debugging is enabled).
- `$parameter(i)` Check if the `i`th component of the range corresponds to a constant parameter. Indexing is by the `C++` default, that is the first component has index `0`, the last component has index `$range - 1`.
- `$new_dynamic(dyn)` Specify new values for the dynamic parameters.

Methods - Tape Evaluation:

- `$eval(x, dyn)` Evaluate the function at new values of the variables and dynamic parameters. Returns a vector of length `$range`.
- `$Jac(x, dyn)` Compute the Jacobian at new values of the variables and dynamic parameters. Returns a vector of length `$range * $domain` arranged so that the first `$domain` elements correspond to the gradient of the first element of the range. The next `$domain` elements correspond to the gradient of the second element of the range, and so on.
- `$Hes(x, dyn)` Compute the Hessian of the first element of the range at new values of the variables and dynamic parameters. Returns a vector of length `$domain * $domain` where the `j*n + l` element corresponds to differentiating with respect to the `l`th element of the domain, then with respect to the `j`th element of the domain, with `n` the size of the domain.
- `$Jacobian(x)` Evaluate the Jacobian of the function at the current set of dynamic parameters.
- `$Hessiani(x, i)` Evaluate the Hessian for the `i`-th element of the range (where `i = 0, 1, ...`). Returns a vector arranged the same as `$Hes()`.
- `$Hessian0(x)` Evaluate the Hessian for the first element of the range (like `$Hes()` but uses the current values of the dynamic parameters). Returns a vector arranged the same as `$Hes()`.
- `$Hessianw(x, w)` Evaluate the Hessian for a weighted sum of the range. Returns a vector arranged the same as `$Hes()`.
- `$forward(q, x)` Perform forward mode evaluation for the specified Taylor coefficient order `q`. See the `CppAD` help for more.

Method Arguments

- `x` A vector of independent variables.
- `dyn` A vector of dynamic parameters.
- `q` Taylor coefficient order for evaluating derivatives with `$forward()`.
- `i` Index of range result. `i = 0, 1, ..., $range - 1`.
- `bool` Either `TRUE` or `FALSE` to set `check_for_nan` behaviour using `$set_check_for_nan()`.
- `w` Weights assigned to each element of the range, for use with `$Hessianw()`.

Extends

Extends class `C++Object` from the `Rcpp` package (`Rcpp::C++Object`), which is a `reference class`. For those familiar with `C++`, an object of class `Rcpp_ADFun` contains a pointer to a `CppAD ADFun` object.

Introduction to CppAD Tapes

This package uses version 2024000.5 of the algorithmic differentiation library CppAD (Bell 2023) to build score matching estimators. Full help for CppAD can be found at https://cppad.readthedocs.io/.

When using CppAD one first creates a *tape* of the basic (*atomic*) operations of a function. The atomic operations include multiplication, division, addition, sine, cosine, exponential and many more. These tapes can then be used for evaluating the function and its derivatives, and generating further tapes through argument swapping, differentiation and composition (see for example `tape_swap()` and `tape_Jacobian()`). Tapes can have both *independent* variables and *dynamic* parameters, and the differentiation occurs with respect to the independent variables. The atomic operations within a function are taped by following the function evaluation on example values for the variables and parameters, so care must be taken with any conditional (e.g. if-then) operations, and CppAD has a special tool for this called CondExp (short for `conditional expressions`).

The result of taping, called a *tape*, is exposed as an object of class Rcpp_ADFun, which contains a CppAD ADFun object. Although the algorithmic differentiation is with respect to the independent variables, a new tape (see `tape_swap()`) can be created where the dynamic parameters become independent variables. For the purposes of score matching, there are also *fixed* parameters, which are the elements of the model's parameter vector that are given and not estimated.

The example values used for taping are saved in the `$xtape` and `$dyntape` properties of Rcpp_ADFun objects.

Warning: multiple CPU

Each time a tape is evaluated the corresponding C++ object is altered. Parallel use of the same ADFun object thus requires care and is not tested. For now I recommend creating a new ADFun object for each CPU.

Improvements

A few methods for CppAD ADFun objects are not yet available through Rcpp_ADFun objects. These ones would be nice to include:

- `optimize()`
- `function_name_set()` and `function_name_get()` working with `$name`
- `Reverse()`

Examples

```
tape <- tape_uld_inbuilt("dirichlet", c(0.1, 0.4, 0.5), c(-0.5, -0.4, -0.2))
# Convenient evaluation
tape$eval(x = c(0.2, 0.3, 0.5), dyn = c(-0.1, -0.1, -0.5))
tape$Jac(x = c(0.2, 0.3, 0.5), dyn = c(-0.1, -0.1, -0.5))
matrix(tape$Hes(x = c(0.2, 0.3, 0.5), dyn = c(-0.1, -0.1, -0.5)), nrow = tape$domain)

# Properties
tape$domain
tape$range
tape$size_dyn_ind
```

```
tape$name
tape$xtape
tape$dyntape
tape$size_order
tape$new_dynamic(dyn = c(-0.1, -0.1, -0.5))
tape$parameter(0)
tape$set_check_for_nan(FALSE)
tape$get_check_for_nan()

# Further methods
tape$forward(order = 0, x = c(0.2, 0.3, 0.5))
tape$Jacobian(x = c(0.2, 0.3, 0.5))
tape$Hessiani(x = c(0.2, 0.3, 0.5), i = 0)
tape$Hessian0(x = c(0.2, 0.3, 0.5))
tape$Hessianw(x = c(0.2, 0.3, 0.5), w = c(2))
```

`rppi` *Simulate from a PPI Model*

Description

Given parameters of the PPI model, generates independent samples.

Usage

```
rppi(n, ..., paramvec = NULL, maxden = 4, maxmemorysize = 1e+05)

rppi_egmodel(n, maxden = 4)
```

Arguments

`n`	Number of samples to generate
`...`	Arguments passed on to `ppi_paramvec`
	`AL` Either `NULL`, a p-1 x p-1 symmetric matrix, a number, or "diag". If NULL then all A_L elements will be set to NA. If a single number, then A_L will be fixed as a matrix of the given value. If "diag" then the non-diagonal elements of A_L will be fixed to 0, and the diagonal will be `NA`.
	`bL` Either `NULL`, a number, or a vector of length p-1. If `NULL`, then all elements of b_L will be set to `NA`. If a single number, then b_L will be fixed at the supplied value.
	`beta` Either `NULL`, a number, or a vector of length p. If NULL then all elements of β will be set to `NA`. If a single number then the β elements will be fixed at the given number.
	`betaL` Either `NULL`, a number, or a vector of length p-1. If `NULL` then the 1...(p-1)th β elements will be set to `NA`. If a single number then the 1...(p-1)th β elements will be fixed at the given number.

betap Either NULL or a number. If NULL then the pth element of β will be set to NA, and ppi() will estimate it. If a number, then the pth element of β will be fixed at the given value.

p The number of components. If NULL then p will be inferred from other inputs.

Astar The A^* matrix (a p by p symmetric matrix)

paramvec The PPI parameter vector, created easily using ppi_paramvec() and also returned by ppi(). Use paramvec instead of ....

maxden This is the constant $log(C)$ in (Appendix A.1.3 Scealy and Wood 2023).

maxmemorysize Advanced use. The maximum size, in bytes, for matrices containing simulated Dirichlet samples. The default of 1E5 corresponds to 100 mega bytes.

Details

We recommend running rppi() a number of times to ensure the choice of maxden is good. rppi() will error when maxden is too low.

The simulation uses a rejection-sampling algorithm with Dirichlet proposal (Appendix A.1.3 Scealy and Wood 2023). Initially n Dirichlet proposals are generated. After rejection there are fewer samples remaining, say n^*. The ratio n^*/n is used to guess the number of new Dirichlet proposals to generate until n samples of the PPI model are reached.

Advanced use: The number of Dirichlet proposals created at a time is limited such that the matrices storing the Dirchlet proposals are always smaller than maxmemorysize bytes (give or take a few bytes for wrapping). Larger maxmemorysize leads to faster simulation so long as maxmemorysize bytes are reliably contiguously available in RAM.

Value

A matrix with n rows and p columns. Each row is an independent draw from the specified PPI distribution.

rppi_egmodel returns a list:

- sample A matrix of the simulated samples (n rows)
- p The number of components of the model
- theta The PPI parameter vector
- AL The A_L parameter matrix
- bL The b_L parameter vector
- beta The β parameter vector

Functions

- rppi_egmodel(): Simulates the 3-component PPI model from (Section 2.3, Scealy and Wood 2023) and returns both simulations and model parameters.

References

Scealy JL, Wood ATA (2023). "Score matching for compositional distributions." *Journal of the American Statistical Association*, **118**(543), 1811–1823. doi:10.1080/01621459.2021.2016422.

See Also

Other PPI model tools: dppi(), ppi(), ppi_param_tools, ppi_robust()

Examples

```
beta0=c(-0.8, -0.8, -0.5)
AL = diag(nrow = 2)
bL = c(2, 3)
samp <- rppi(100,beta=beta0,AL=AL,bL=bL)
rppi_egmodel(1000)
```

---

rsymmetricmatrix *Quickly Generate a Symmetric Matrix for Testing and Examples*

Description

A simple function for generating a symmetric matrix for use in examples. The diagonal, and upper-triangular elements of the matrix are simulated independently from a uniform distribution. The lower-triangle of the output matrix is copied from the upper-triangle. These matrices **do not** represent the full range of possible symmetric matrices.

Usage

```
rsymmetricmatrix(p, min = 0, max = 1)
```

Arguments

p	The desired dimension of the matrix
min	The minimum of the uniform distribution.
max	The maximum of the uniform distribution

Value

A p x p symmetric matrix.

Examples

```
rsymmetricmatrix(5)
```

`scorematchingtheory` *Introduction to Score Matching*

Description

This package includes score matching estimators for particular distributions and a general capacity to implement additional score matching estimators. Score matching is a popular estimation technique when normalising constants for the proposed model are difficult to calculate or compute. Score matching was first developed by Hyvärinen (2005) and was further developed for subsets of Euclidean space (Hyvärinen 2007; Yu et al. 2019; Yu et al. 2020; Liu et al. 2019), Riemannian manifolds (Mardia et al. 2016; Mardia 2018), and Riemannian manifolds with boundary (Scealy and Wood 2023). In this help entry we briefly describe score matching estimation.

Score Matching in General

In the most general form (Riemannian manifolds with boundary) score matching minimises the weighted Hyvärinen divergence (Equation 7, Scealy and Wood 2023)

$$\phi(f, f_0) = \frac{1}{2}\int_M f_0(z)h(z)^2 \left\| P(z)\Big(\nabla_z \log(f) - \nabla_z \log(f_0)\Big)\right\|^2 dM(z),$$

where

- M is the manifold, isometrically embedded in Euclidean space, and $dM(z)$ is the unnormalised uniform measure on M.
- $P(z)$ is the matrix that projects points onto the tangent space of the manifold at z, which is closely related to to Riemannian metric of M.
- f_0 is the density of the data-generating process, defined with respect to $dM(z)$.
- f is the density of a posited model, again defined with respect to $dM(z)$.
- $h(z)$ is a function, termed the *boundary weight function*, that is zero on the boundary of M and smooth (Section 3.2, Scealy and Wood 2023) or potentially piecewise smooth.
- ∇_z is the Euclidean gradient operator that differentiates with respect to z.
- $\|\cdot\|$ is the Euclidean norm.

Note that, because $P(z)$ is the projection matrix, $\left\| P(z)\Big(\nabla_z \log(f) - \nabla_z \log(f_0)\Big)\right\|^2$ is the natural inner product of the gradient of the log ratio of f and f_0.

When the density functions f and f_0 are smooth and positive inside M, and the boundary weight function is smooth or of particular forms for specific manifolds (Section 3.2, Scealy and Wood 2023), then minimising the weighted Hyvärinen divergence $\phi(f, f_0)$ is equivalent to minimising the score matching discrepancy (Theorem 1, Scealy and Wood 2023)

$$\psi(f, f_0) = \int f_0(z)\big(A(z) + B(z) + C(z)\big)dM(z),$$

where

$$A(z) = \frac{1}{2}h(z)^2 \left(\nabla_z \log(f)\right)^T P(z) \left(\nabla_z \log(f)\right),$$

$$B(z) = h(z)^2 \Delta_z \log(f),$$

$$C(z) = \left(\nabla_z h(z)^2)\right)^T P(z) \left(\nabla_z \log(f)\right),$$

and Δ is the Laplacian operator. We term

$$A(z) + B(z) + C(z)$$

the *score matching discrepancy function*.

We suspect that (Theorem 1, Scealy and Wood 2023) holds more generally for nearly all realistic continuous and piecewise-smooth boundary weight functions, although no proof exists to our knowledge.

When n independent observations from f_0 are available, the integration in $\psi(f, f_0)$ can be approximated by an average over the observations,

$$\psi(f, f_0) \approx \hat{\psi}(f, f_0) = \frac{1}{n} \sum_{i=1}^{n} A(z_i) + B(z_i) + C(z_i).$$

If we parameterise a family of models f_θ according to a vector of parameters θ, then the *score matching estimate* is the θ that minimises $\hat{\psi}(f_\theta, f_0)$. In general, the score matching estimate must be found via numerical optimisation techniques, such as in the function `cppad_search()`. However, when the family of models is a canonical exponential family then often $\hat{\psi}(f_\theta, f_0)$ and the score matching discrepancy function is a quadratic function of θ (Mardia 2018) and the minimum has a closed-form solution found by `cppad_closed()`.

Note that when M has a few or more dimensions, the calculations of $A(z)$, $B(z)$ and $C(z)$ can become cumbersome. This package uses `CppAD` to automatically compute $A(z)$, $B(z)$ and $C(z)$, and the quadratic simplification if it exists.

Transformations

Hyvärinen divergence $\phi(f, f_0)$ is sensitive to transformations of the measure $dM(z)$, including transforming the manifold. That is, transforming the manifold M changes the divergence between distributions and changes the minimum of $\hat{\psi}(f_\theta, f_0)$. The transformation changes measure $dM(z)$, the divergence and the estimator but does *not* transform the data.

For example, many different transformations of the simplex (i.e. compositional data) are possible (Appendix A.3, Scealy et al. 2024). Hyvärinen divergences that use the sphere, obtained from the simplex by a square root, have different behaviour to Hyvärinen divergence using Euclidean space obtained from the simplex using logarithms (Scealy et al. 2024). The estimator for the latter does not apply logarithms to the observations, in fact the estimator involves only polynomials of the observed compositions (Scealy et al. 2024).

The variety of estimator behaviour available through different transformations was a major motivator for this package as each transformation has different $A(z)$, $B(z)$ and $C(z)$, and without automatic differentiation, implementation of the score matching estimator in each case would require a huge programming effort.

References

Hyvärinen A (2005). "Estimation of Non-Normalized Statistical Models by Score Matching." *Journal of Machine Learning Research*, **6**(24), 695–709. https://jmlr.org/papers/v6/hyvarinen05a.html.

Hyvärinen A (2007). "Some extensions of score matching." *Computational Statistics & Data Analysis*, **51**(5), 2499–2512. doi:10.1016/j.csda.2006.09.003.

Liu S, Kanamori T, Williams DJ (2019). "Estimating Density Models with Truncation Boundaries using Score Matching." doi:10.48550/arXiv.1910.03834.

Mardia K (2018). "A New Estimation Methodology for Standard Directional Distributions." In *2018 21st International Conference on Information Fusion (FUSION)*, 724–729. doi:10.23919/ICIF.2018.8455640.

Mardia KV, Kent JT, Laha AK (2016). "Score matching estimators for directional distributions." doi:10.48550/arXiv.1604.08470.

Scealy JL, Hingee KL, Kent JT, Wood ATA (2024). "Robust score matching for compositional data." *Statistics and Computing*, **34**, 93. doi:10.1007/s1122202410412w.

Scealy JL, Wood ATA (2023). "Score matching for compositional distributions." *Journal of the American Statistical Association*, **118**(543), 1811–1823. doi:10.1080/01621459.2021.2016422.

Yu S, Drton M, Shojaie A (2019). "Generalized Score Matching for Non-Negative Data." *Journal of Machine Learning Research*, **20**(76), 1–70. https://jmlr.org/papers/v20/18-278.html.

Yu S, Drton M, Shojaie A (2020). "Generalized Score Matching for General Domains." doi:10.48550/arXiv.2009.11428.

`smvalues` *Compute Score Matching Discrepancy Value, Gradient, and Hessian*

Description

Computes a range of relevant information for investigating score matching estimators.

Usage

```
smvalues(smdtape, xmat, pmat, xcentres = NA * xmat, approxorder = 10)

smvalues_wsum(
  tape,
  xmat,
  pmat,
  w = NULL,
```

```
  xcentres = NA * xmat,
  approxorder = 10
)
```

Arguments

smdtape	A taped score matching discrepancy. Most easily created by tape_smd().
xmat	A matrix of (multivariate) independent variables where each represents a single independent variable vector. Or a single independent variable vector that is used for all rows of pmat.
pmat	A matrix of dynamic parameters where each row specifies a new set of values for the dynamic parameters of tape. Or a single vector of dynamic parameters to use for all rows of xmat.
xcentres	A matrix of approximation for Taylor approximation centres for xmat. Use values of NA for rows that do not require Taylor approximation.
approxorder	Order of Taylor approximation
tape	An Rcpp_ADFun object (i.e. a tape of a function).
w	Weights to apply to each row of xmat for computing the weighted sum. If NULL then each row is given a weight of 1.

Details

Computes the score matching discrepancy function from scorematchingtheory or weighted sum of the score matching discrepancy function. The gradient and Hessian are returned as arrays of row-vectors with each row corresponding to a row in xmat and pmat. Convert a Hessian row-vector to a matrix using `matrix(ncol = length(smdtape$xtape))`.

Value

A list of

- obj the score matching discrepancy values
- grad the gradient of the score matching discrepancy
- hess the Hessian of the score matching discrepancy

See Also

Other tape evaluators: evaltape(), quadratictape_parts(), testquadratic()

Examples

```
m <- rppi_egmodel(100)
smdtape <- tape_smd("sim", "sqrt", "sph", "ppi",
              ytape = rep(1/m$p, m$p),
              usertheta = ppi_paramvec(beta = m$beta),
              bdryw = "minsq", acut = 0.01)$smdtape
smvalues(smdtape, xmat = m$sample, pmat = m$theta[1:5])
smvalues_wsum(smdtape, m$sample, m$theta[1:5])$grad/nrow(m$sample)
```

tape_bdryw *Generate a tape of a custom boundary weight function*

Description

Generate a tape of a boundary weight function, which is useful for specifying the boundary of manifolds in score matching. Use `tape_bdryw()` to specify a custom function using C++ code much like `TMB::compile()`. Use `tape_bdryw_inbuilt()` for tapes of inbuilt functions implemented in this package.

Usage

```
tape_bdryw_inbuilt(name, x, acut)

tape_bdryw(fileORcode = "", x, Cppopt = NULL)
```

Arguments

`name`	Name of an inbuilt function. See details.
`x`	Vector giving location in the manifold.
`acut`	The a_c threshold used by the "minsq" and "prodsq" built-in functions. See details.
`fileORcode`	A character string giving the path name of a file containing the unnormalised log-density definition *OR* code. `fileORcode` will be treated as a file name if `fileORcode` contains no new line characters ('\n' or '\r\n') and has a file extension detected by `tools::file_ext()`.
`Cppopt`	List of named options passed to `Rcpp::sourceCpp()`

Details

For `tape_bdryw_inbuilt()`, currently available functions are:

- The function "ones" applies no weights and should be used whenever the manifold does not have a boundary.
$$h(x)^2 = 1.$$
- The function "minsq" is the minima-based boundary weight function (Equation 12, Scealy and Wood 2023)
$$h(x)^2 = \min(x_1^2, x_2^2, ..., x_p^2, a_c^2).$$
where x_j is the jth component of x.
- The function "prodsq" is the product-based (Equation 9, Scealy and Wood 2023)
$$h(x)^2 = \min(\prod_{j=1}^{p} x_j^2, a_c^2).$$
where x_j is the jth component of x.

The "minsq" and "prodsq" functions are useful when the manifold is the positive orthant, the p-dimensional unit sphere restricted to the positive orthant, or the unit simplex. (scealy2023sc) prove that both "minsq" and "prodsq" can be used for score matching the PPI model on the simplex or positive orthant of the sphere.

The function `tape_bdryw()` uses `Rcpp::sourceCpp()` to generate a tape of a function defined in C++.

The result result is NOT safe to save or pass to other CPUs in a parallel operation.

Value

A list of three objects

- `fun` a function that evaluates the function directly
- `tape` a tape of the function
- `file` the temporary file storing the final source code passed to `Rcpp::sourceCpp()`

Writing the `fileORcode` Argument

The code (possibly in the file pointed to by `fileORcode`) must be `C++` that uses only `CppAD` and `Eigen`, which makes it very similar to the requirements of the input to `TMB::compile()` (which also uses `CppAD` and `Eigen`).

The start of code should always be `"a1type fname(const veca1 &x){"` where `fname` is your chosen name of the function, `x` represents a point in the manifold. This specifies that the function will a single two vector argument (of type `veca1`) and will return a single numeric value (`a1type`).

The type `a1type` is a double with special ability for being taped by `CppAD`. The `veca1` type is a vector of `a1type` elements, with the vector wrapping supplied by the `Eigen` C++ package (that is an `Eigen` matrix with 1 column and dynamic number of rows). In place operations like `+=` on `a1type`, `veca1` and similar have not worked for me (compile errors); fortunately the efficiency of in place operations is irrelevant for taping operations.

The body of the function must use operations from Eigen and/or CppAD, prefixed by `Eigen::` and `CppAD::` respectively. There are no easy instructions for writing these as it is genuine `C++` code, which can be very opaque to those unfamiliar with `C++`. However, recently ChatGPT and claude.ai have been able to very quickly translating `R` functions to `C++` functions (KLH has been telling these A.I. to use Eigen and CppAD, and giving the definitions of `a1type` and `veca1`). I've found the quick reference pages for for `Eigen` useful. Limited unary and binary operations are available directly from `CppAD` without `Eigen`.

Non-smooth functions should be used with care, but can be specified using `CppAD`'s CondExp functions.

See Also

Other tape builders: `avgrange()`, `fixdynamic()`, `fixindependent()`, `keeprange()`, `tape_Hessian()`, `tape_Jacobian()`, `tape_gradoffset()`, `tape_logJacdet()`, `tape_smd()`, `tape_swap()`, `tape_uld()`

Examples

```
## Not run:
out <- tape_bdrw(
"a1type myminsq(const veca1 &x){
veca1 xsq = x.array().square();
a1type minval(0.1 * 0.1);
for(size_t i=0;i<x.size();i++){
  minval = CppAD::CondExpLe(xsq[i], minval, xsq[i], minval);
}
return(minval);
}",
   rep(0.2, 5))
out$fun(c(0.01, 0.2, 0.2, 0.2, 0.2))
out$tape$forward(0, c(0.1, 0.2, 0.2, 0.2, 0.2))
out$tape$Jacobian(c(0.1, 0.2, 0.2, 0.2, 0.2))
out$tape$name

## End(Not run)
```

`tape_gradoffset` *Tape the Gradient Offset of a Quadratic CppAD Tape*

Description

Tape the Gradient Offset of a Quadratic CppAD Tape

Usage

```
tape_gradoffset(pfun)
```

Arguments

`pfun`	An `Rcpp_ADFun` object.

Details

A quadratic function can be written as

$$f(x;\theta) = \frac{1}{2}x^T W(\theta)x + b(\theta)^T x + c.$$

The gradient of $f(x;\theta)$ with respect to x is

$$\Delta f(x;\theta) = \frac{1}{2}(W(\theta) + W(\theta)^T)x + b(\theta).$$

The Hessian is

$$Hf(x;\theta) = \frac{1}{2}(W(\theta) + W(\theta)^T),$$

which does not depend on x, so the gradient of the function can be rewritten as

$$\Delta f(x;\theta) = Hf(x;\theta)x + b(\theta)^T.$$

The tape calculates $b(\theta)$ as

$$b(\theta) = \Delta f(x;\theta) - Hf(x;\theta)x,$$

which does not depend on x.

For creating this tape, the values of `pfun$xtape` and `pfun$dyntape` are used.

Value

An `Rcpp_ADFun` object. The independent argument to the function are the dynamic parameters of `pfun`.

See Also

Other tape builders: `avgrange()`, `fixdynamic()`, `fixindependent()`, `keeprange()`, `tape_Hessian()`, `tape_Jacobian()`, `tape_bdryw()`, `tape_logJacdet()`, `tape_smd()`, `tape_swap()`, `tape_uld()`

---

tape_Hessian *Tape the Hessian of a CppAD Tape*

Description

Creates a tape of the Hessian of a function taped by `CppAD`. The taped function represented by `pfun` must be scalar-valued (i.e. a vector of length 1). The `x` vector and `dynparam` are used as the values to conduct the taping.

Usage

```
tape_Hessian(pfun)
```

Arguments

`pfun`	An `Rcpp_ADFun` object.

Details

When the returned tape is evaluated (via say `eval()`), the resultant vector contains the Hessian in long format (see https://cppad.readthedocs.io/latest/Hessian.html): suppose the function represented by `pfun` maps from n-dimensional space to 1-dimensional space, then the first n elements of the vector is the gradient of the partial derivative with respect to the first dimension of the function's domain; the next n elements of the vector is the gradient of the partial derivative of the second dimension of the function's domain. The Hessian as a matrix, can be obtained by using `as.matrix()` with `ncol = n`.

For creating this tape, the values of `pfun$xtape` and `pfun$dyntape` are used.

Value

An Rcpp_ADFun object.

See Also

Other tape builders: avgrange(), fixdynamic(), fixindependent(), keeprange(), tape_Jacobian(), tape_bdryw(), tape_gradoffset(), tape_logJacdet(), tape_smd(), tape_swap(), tape_uld()

---

tape_Jacobian *Tape the Jacobian of CppAD Tape*

Description

Creates a tape of the Jacobian of a function taped by CppAD. When the function returns a real value (as is the case for densities and the score matching objective) the Jacobian is equivalent to the gradient. The x vector is used as the value to conduct the taping.

Usage

```
tape_Jacobian(pfun)
```

Arguments

pfun	An Rcpp_ADFun object.

Details

When the returned tape is evaluated (via say $eval(), the resultant vector contains the Jacobian in long format (see https://cppad.readthedocs.io/latest/Jacobian.html). Suppose the function represented by pfun maps from n-dimensional space to m-dimensional space, then the first n elements of vector is the gradient of the first component of function output. The next n elements of the vector is the gradient of the second component of the function output. The Jacobian as a matrix, could then be obtained by as.matrix() with byrow = TRUE and ncol = n.

For creating this tape, the values of pfun$xtape and pfun$dyntape are used.

Value

An Rcpp_ADFun object.

See Also

Other tape builders: avgrange(), fixdynamic(), fixindependent(), keeprange(), tape_Hessian(), tape_bdryw(), tape_gradoffset(), tape_logJacdet(), tape_smd(), tape_swap(), tape_uld()

tape_logJacdet *Tape the log of Jacobian determinant of a CppAD Tape*

Description

Creates a tape of the log of the Jacobian determinant of a function taped by CppAD. The `x` vector is used as the value to conduct the taping.

For creating this tape, the values of `pfun$xtape` and `pfun$dyntape` are used.

Usage

```
tape_logJacdet(pfun)
```

Arguments

`pfun`	An `Rcpp_ADFun` object.

Value

An `Rcpp_ADFun` object.

See Also

Other tape builders: `avgrange()`, `fixdynamic()`, `fixindependent()`, `keeprange()`, `tape_Hessian()`, `tape_Jacobian()`, `tape_bdryw()`, `tape_gradoffset()`, `tape_smd()`, `tape_swap()`, `tape_uld()`

tape_smd *Build CppAD Tapes for Score Matching*

Description

For a parametric model family, the function `tape_smd()` generates `CppAD` tapes for the unnormalised log-density of the model family and of the score matching discrepancy function $A(z) + B(z) + C(z)$ (defined in `scorematchingtheory`). Three steps are performed by `tape_smd()`: first an object that specifies the manifold and any transformation to another manifold is created; then a tape of the unnormalised log-density is created; finally a tape of $A(z) + B(z) + C(z)$ is created.

Usage

```
tape_smd(
  start,
  tran = "identity",
  end = start,
  ll,
  ytape,
```

```
  usertheta,
  bdryw = "ones",
  acut = 1,
  thetatape_creator = function(n) {
     seq(length.out = n)
 },
  verbose = FALSE
)
```

Arguments

start	The starting manifold. Used for checking that tran and man match.
tran	The name of a transformation. Available transformations are • "sqrt" • "alr" • "clr" • "none" or 'identity'
end	The name of the manifold that tran maps start to. Available manifolds are: • "sph" unit sphere • "Hn111" hyperplane normal to the vector $1, 1, 1, 1, \ldots$ • "sim" simplex • "Euc" Euclidean space
ll	An unnormalised log-density with respect to the metric of the starting manifold. ll must be either an Rcpp_ADFun object created by tape_uld() for a custom unnormalised log-density function or the name of an inbuilt function.
ytape	An example measurement value to use for creating the tapes. In the natural (i.e. start) manifold of the density function. Please ensure that ytape is the interior of the manifold and non-zero.
usertheta	A vector of parameter elements for the likelihood function. NA elements will become *dynamic parameters*. Other elements will be fixed at the provided value. The length of usertheta must be the correct length for the log-density - **no checking is conducted**.
bdryw	The name of the boundary weight function or the tape of a custom boundary weight function. See tape_bdryw(). "ones" for manifolds without boundary. For the simplex and positive orthant of the sphere, "prodsq" and "minsq" are possible - see ppi() for more information on these.
acut	An optional parameter passed to the built-in boundary weight functions. See ppi() for more information.
thetatape_creator	A function that accepts an integer n, and returns a vector of n length. The function is used to fill in the NA elements of usertheta when building the tapes. Please ensure that the values filled by thetatape_creator lead to plausible parameter vectors for the chosen log-density.
verbose	If TRUE more details are printed when taping. These details are for debugging and will likely be comprehensible only to users familiar with the source code of this package.

Details

Only some combinations of `start`, `tran` and `end` are available because `tran` must map between `start` and `end`. These combinations of `start-tran-end` are currently available:

- sim-sqrt-sph
- sim-identity-sim
- sim-alr-Euc
- sim-clr-Hn111
- sph-identity-sph
- Euc-identity-Euc

To build a tape for the score matching discrepancy function, the `scorematchingad` first tapes the map from a point z on the `end` manifold to the value of the unnormalised log-density, where the independent variable is the z, the dynamic parameter is a vector of the parameters to estimate, and the remaining model parameters are fixed and not estimated. This tape is then used to generate a tape for the score matching discrepancy function where the parameters to estimate are the independent variable.

The transforms of the manifold must be implemented in `C++` and selected by name.

Currently available unnormalised log-density functions are:

- dirichlet
- ppi
- vMF
- Bingham
- FB

Value

A list of:

- an `Rcpp_ADFun` object containing a tape of the unnormalised log-density using the metric of the "`end`" manifold (that is the independent variable is on the `end` manifold).
- an `Rcpp_ADFun` object containing a tape of the score matching discrepancy function with the non-fixed parameters of the model as the independent variable, and the measurements on the `end` manifold as the dynamic parameter.
- some information about the tapes

Warning

There is no checking of the inputs `ytape` and `usertheta`.

References

There are no references for Rd macro `\insertAllCites` on this help page.

See Also

Other tape builders: avgrange(), fixdynamic(), fixindependent(), keeprange(), tape_Hessian(), tape_Jacobian(), tape_bdryw(), tape_gradoffset(), tape_logJacdet(), tape_swap(), tape_uld()

Other generic score matching tools: Windham(), cppad_closed(), cppad_search()

Examples

```
p <- 3
u <- rep(1/sqrt(p), p)
ltheta <- p #length of vMF parameter vector
intheta <- rep(NA, length.out = ltheta)
tapes <- tape_smd("sph", "identity", "sph", "vMF",
              ytape = u,
              usertheta = intheta,
              "ones", verbose = FALSE
              )
evaltape(tapes$lltape, u, runif(n = ltheta))
evaltape(tapes$smdtape, runif(n = ltheta), u)

u <- rep(1/3, 3)
tapes <- tape_smd("sim", "sqrt", "sph", "ppi",
              ytape = u,
              usertheta = ppi_paramvec(p = 3),
              bdryw = "minsq", acut = 0.01,
              verbose = FALSE
              )
evaltape(tapes$lltape, u, rppi_egmodel(1)$theta)
evaltape(tapes$smdtape, rppi_egmodel(1)$theta, u)
```

---

tape_swap *Switch Dynamic and Independent Values of a Tape*

Description

Convert an Rcpp_ADFun object so that the independent values become dynamic parameters and the dynamic parameters become independent values

Usage

```
tape_swap(pfun)
```

Arguments

pfun	An Rcpp_ADFun object.

Details

For creating this tape, the values of pfun$xtape and pfun$dyntape are used.

Value

An Rcpp_ADFun object.

See Also

Other tape builders: avgrange(), fixdynamic(), fixindependent(), keeprange(), tape_Hessian(), tape_Jacobian(), tape_bdryw(), tape_gradoffset(), tape_logJacdet(), tape_smd(), tape_uld()

---

tape_uld *Generate a tape of a custom unnormalised log-density*

Description

Generate tapes of unnormalised log-densities. Use tape_ult() to specify a custom unnormalised log-density using C++ code much like TMB::compile(). Use tape_uld_inbuilt() for tapes of inbuilt unnormalised log-densities implemented in this package.

Usage

```
tape_uld_inbuilt(name, x, theta)

tape_uld(fileORcode = "", x, theta, Cppopt = NULL)
```

Arguments

name	Name of an inbuilt function. See details.
x	Value of independent variables for taping.
theta	Value of the dynamic parameter vector for taping.
fileORcode	A character string giving the path name of a file containing the unnormalised log-density definition *OR* code. fileORcode will be treated as a file name if fileORcode contains no new line characters ('\n' or '\r\n') and has a file extension detected by tools::file_ext().
Cppopt	List of named options passed to Rcpp::sourceCpp()

Details

For tape_uld_inbuilt(), currently available unnormalised log-density functions are:

- dirichlet
- ppi
- vMF
- Bingham
- FB

The function tape_uld() uses Rcpp::sourceCpp() to generate a tape of a function defined in C++. (An alternative design, where the function is compiled interactively and then taped using a function internal to scorematchingad, was not compatible with Windows OS).

The result result is NOT safe to save or pass to other CPUs in a parallel operation.

Value

A list of three objects

- fun a function that evaluates the function directly
- tape a tape of the function
- file the temporary file storing the final source code passed to Rcpp::sourceCpp()

Writing the fileORcode Argument

The code (possibly in the file pointed to by fileORcode) must be C++ that uses only CppAD and Eigen, which makes it very similar to the requirements of the input to TMB::compile() (which also uses CppAD and Eigen).

The start of code should always be "a1type fname(const veca1 &x, const veca1 &theta){" where fname is your chosen name of the log-density function, x represents a point in the data space and theta is a vector of parameters for the log-density. This specifies that the function will have two vector arguments (of type veca1) and will return a single numeric value (a1type).

The type a1type is a double with special ability for being taped by CppAD. The veca1 type is a vector of a1type elements, with the vector wrapping supplied by the Eigen C++ package (that is an Eigen matrix with 1 column and dynamic number of rows). In place operations like += on a1type, veca1 and similar have not worked for me (compile errors); fortunately the efficiency of in place operations is irrelevant for taping operations.

The body of the function must use operations from Eigen and/or CppAD, prefixed by Eigen:: and CppAD:: respectively. There are no easy instructions for writing these as it is genuine C++ code, which can be very opaque to those unfamiliar with C++. However, recently ChatGPT and claude.ai have been able to very quickly translating R functions to C++ functions (KLH has been telling these A.I. to use Eigen and CppAD, and giving the definitions of a1type and veca1). I've found the quick reference pages for Eigen useful. Limited unary and binary operations are available directly from CppAD without Eigen. For the purposes of score matching the operations should all be smooth to create a smooth log-density and the normalising constant may be omitted.

See Also

Other tape builders: avgrange(), fixdynamic(), fixindependent(), keeprange(), tape_Hessian(), tape_Jacobian(), tape_bdryw(), tape_gradoffset(), tape_logJacdet(), tape_smd(), tape_swap()

Examples

```
## Not run:
out <- tape_uld(system.file("demo_custom_uld.cpp", package = "scorematchingad"),
                rep(0.2, 5), rep(-0.1, 5))
out$fun(c(0.1, 0.2, 0.2, 0.2, 0.2), c(-0.5, -0.5, -0.1, -0.1, 0))
out$tape$eval(c(0.1, 0.2, 0.2, 0.2, 0.2), c(-0.5, -0.5, -0.1, -0.1, 0))
out$tape$Jac(c(0.1, 0.2, 0.2, 0.2, 0.2), c(-0.5, -0.5, -0.1, -0.1, 0))
out$tape$name

## End(Not run)
```

testquadratic *Test Whether a CppAD Tape is a Quadratic Function*

Description

Uses the CppAD parameter property and derivatives (via tape_Jacobian()) to test whether the tape is quadratic.

Uses the CppAD parameter property and derivatives (via tape_Jacobian()) to test whether the tape is quadratic.

Usage

```
testquadratic(
  tape,
  xmat = matrix(tape$xtape, nrow = 1),
  dynmat = matrix(tape$dyntape, nrow = 1),
  verbose = FALSE
)
```

Arguments

tape	An ADFun object.
xmat	The third-order derivatives at independent variable values of the rows of xmat and dynamic parameters from the rows of dynmat are tested.
dynmat	The third-order derivatives at independent variable values of the rows of xmat and dynamic parameters from the rows of dynmat are tested.
verbose	If TRUE information about the failed tests is printed.

Details

Uses the xtape and dyntape values stored in tape to create new tapes. A tape of the Hessian is obtained by applying tape_Jacobian() twice, and then uses the CppAD parameter property to test whether the Hessian is constant. A function of quadratic form should have constant Hessian.

If xmat and dynmat are non-NULL then testquadratic() also checks the Jacobian of the Hessian at xmat and dynmat values. For quadratic form functions the Jacobian of the Hessian should be zero.

Value

TRUE or FALSE

See Also

Other tape evaluators: evaltape(), quadratictape_parts(), smvalues()

Other tape evaluators: evaltape(), quadratictape_parts(), smvalues()

Examples

```
tapes <- tape_smd(
   "sim", "sqrt", "sph",
   ll = "ppi",
   ytape = c(0.2, 0.3, 0.5),
   usertheta = ppi_paramvec(p = 3),
   bdryw = "minsq",
   acut = 0.1,
   verbose = FALSE)

 testquadratic(tapes$smdtape)
```

---

vMF *Score Matching Estimator for the von-Mises Fisher Distribution*

Description

In general the normalising constant in von Mises Fisher distributions is hard to compute, so Mardia et al. (2016) suggested a hybrid method that uses maximum likelihood to estimate the mean direction and score matching for the concentration. We can also estimate all parameters using score matching (`smfull` method), although this estimator is likely to be less efficient than the hybrid estimator. On the circle the hybrid estimators were often nearly as efficient as maximum likelihood estimators (Mardia et al. 2016). For maximum likelihood estimators of the von Mises Fisher distribution, which all use approximations of the normalising constant, consider `movMF::movMF()`.

Usage

```
vMF(Y, paramvec = NULL, method = "Mardia", w = rep(1, nrow(Y)))
```

Arguments

`Y`	A matrix of multivariate observations in Cartesian coordinates. Each row is a multivariate measurement (i.e. each row corresponds to an individual).
`paramvec`	`smfull` method only: Optional. A vector of same length as the dimension, representing the elements of the $\kappa\mu$ vector.
`method`	Either "Mardia" or "hybrid" for the hybrid score matching estimator from Mardia et al. (2016) or "smfull" for the full score matching estimator.
`w`	An optional vector of weights for each measurement in `Y`

Details

The full score matching estimator (`method = "smfull"`) estimates $\kappa\mu$. The hybrid estimator (`method = "Mardia"`) estimates κ and μ separately. Both use `cppad_closed()` for score matching estimation.

Value

A list of `est`, `SE` and `info`.

- `est` contains the estimates in vector form, `paramvec`, and with user friendly names `k` and `m`.
- `SE` contains estimates of the standard errors if computed. See `cppad_closed()`.
- `info` contains a variety of information about the model fitting procedure and results.

von Mises Fisher Model

The von Mises Fisher density is proportional to

$$\exp(\kappa \mu^T z),$$

where z is on a unit sphere, κ is termed the *concentration*, and μ is the *mean direction unit vector*. The effect of the μ and κ can be decoupled in a sense (p169, Mardia and Jupp 2000), allowing for estimating μ and κ separately.

References

Mardia KV, Jupp PE (2000). *Directional Statistics*, Probability and Statistics. Wiley, Great Britain. ISBN 0-471-95333-4.

Mardia KV, Kent JT, Laha AK (2016). "Score matching estimators for directional distributions." doi:10.48550/arXiv.1604.08470.

See Also

Other directional model estimators: `Bingham()`, `FB()`, `vMF_robust()`

Examples

```
if (requireNamespace("movMF")){
  Y <- movMF::rmovMF(1000, 100 * c(1, 1) / sqrt(2))
  movMF::movMF(Y, 1) #maximum likelihood estimate
} else {
  Y <- matrix(rnorm(1000 * 2, sd = 0.01), ncol = 2)
  Y <- Y / sqrt(rowSums(Y^2))
}
vMF(Y, method = "smfull")
vMF(Y, method = "Mardia")
vMF(Y, method = "hybrid")
```

vMF_robust *Robust Fitting of von Mises Fisher*

Description

Robust estimation for von Mises Fisher distribution using `Windham()`.

Usage

```
vMF_robust(Y, cW, ...)
```

Arguments

`Y`	A matrix of observations in Cartesian coordinates.
`cW`	Tuning constants for each parameter in the vMF parameter vector. If a single number then the constant is the same for each element of the parameter vector.
`...`	Passed to `Windham()` and then passed onto `vMF()`.

See Also

Other directional model estimators: `Bingham()`, `FB()`, `vMF()`

Other Windham robustness functions: `Windham()`, `ppi_robust()`

Examples

```
if (requireNamespace("movMF")){
  Y <- movMF::rmovMF(1000, 100 * c(1, 1) / sqrt(2))
} else {
  Y <- matrix(rnorm(1000 * 2, sd = 0.01), ncol = 2)
  Y <- Y / sqrt(rowSums(Y^2))
}
vMF_robust(Y, cW = c(0.01, 0.01), method = "smfull")
vMF_robust(Y, cW = c(0.01, 0.01), method = "Mardia")
```

Windham *Windham Robustification of Point Estimators for Exponential Family Distributions*

Description

Performs a generalisation of Windham's robustifying method (Windham 1995) for exponential models with natural parameters that are a linear function of the parameters for estimation. Estimators must solve estimating equations of the form

$$\sum_{i=1}^{n} U(z_i; \theta) = 0.$$

The estimate is found iteratively through a fixed point method as suggested by Windham (1995).

Usage

```
Windham(
  Y,
  estimator,
  ldenfun,
  cW,
  ...,
  fpcontrol = list(Method = "Simple", ConvergenceMetricThreshold = 1e-10),
  paramvec_start = NULL,
  alternative_populationinverse = FALSE
)
```

Arguments

Y	A matrix of measurements. Each row is a measurement, each component is a dimension of the measurement.
estimator	A function that estimates parameters from weighted observations. It must have arguments Y that is a matrix of measurements and w that are weights associated with each row of Y. If it accepts arguments paramvec or paramvec_start then these will be used to specify fixed elements of the parameter vector and the starting guess of the parameter vector, respectively. The estimated parameter vector, including any fixed elements, must be the returned object, or the first element of a returned list, or as the paramvec slot within the est slot of the returned object.
ldenfun	A function that returns a vector of values proportional to the log-density for a matrix of observations Y and parameter vector theta.
cW	A vector of robustness tuning constants. When computing the weight for an observation the parameter vector is multiplied element-wise with cW. For the PPI model, generate cW easily using ppi_cW() and ppi_cW_auto().
...	Arguments passed to estimator.
fpcontrol	A named list of control arguments to pass to FixedPoint::FixedPoint() for the fixed point iteration.
paramvec_start	Initially used to check the function estimator. If estimator accepts a paramvec_start, then the current estimate of the parameter vector is passed as paramvec_start to estimator in each iteration.
alternative_populationinverse	The default is to use Windham_populationinverse(). If TRUE an alternative implementation in Windham_populationinverse_alternative() is used. So far we have not seen any difference between the results.

Details

For any family of models with density $f(z;\theta)$, Windham's method finds the parameter set $\hat{\theta}$ such that the estimator applied to observations weighted by $f(z;\hat{\theta})^c$ returns an estimate that matches the theoretical effect of weighting the full population of the model. When f is proportional to $\exp(\eta(\theta)\cdot T(z))$ and $\eta(\theta)$ is linear, these weights are equivalent to $f(z;c\hat{\theta})$ and the theoretical effect of the weighting on the full population is to scale the parameter vector θ by $1+c$.

The function `Windham()` assumes that f is proportional to $\exp(\eta(\theta) \cdot T(z))$ and $\eta(\theta)$ is linear. It allows a generalisation where c is a vector so the weight for an observation z is

$$f(z; c \circ \theta),$$

where θ is the parameter vector, c is a vector of tuning constants, and $\circ$ is the element-wise product (Hadamard product).

The solution is found iteratively (Windham 1995). Given a parameter set θ_n, `Windham()` first computes weights $f(z; c \circ \theta_n)$ for each observation z. Then, a new parameter set $\tilde{\theta}_{n+1}$ is estimated by `estimator` with the computed weights. This new parameter set is element-wise-multiplied by the (element-wise) reciprocal of $1 + c$ to obtain an adjusted parameter set θ_{n+1}. The estimate returned by `Windham()` is the parameter set $\hat{\theta}$ such that $\theta_n \approx \theta_{n+1}$.

Value

A list:

- `paramvec` the estimated parameter vector
- `optim` information about the fixed point iterations and optimisation process. Including a slot `finalweights` for the weights in the final iteration.

See Also

Other generic score matching tools: `cppad_closed()`, `cppad_search()`, `tape_smd()`

Other Windham robustness functions: `ppi_robust()`, `vMF_robust()`

Examples

```
if (requireNamespace("movMF")){
  Y <- movMF::rmovMF(1000, 100 * c(1, 1) / sqrt(2))
} else {
  Y <- matrix(rnorm(1000 * 2, sd = 0.01), ncol = 2)
  Y <- Y / sqrt(rowSums(Y^2))
}
Windham(Y = Y,
   estimator = vMF,
   ldenfun = function(Y, theta){ #here theta is km
     return(drop(Y %*% theta))
   },
   cW = c(0.01, 0.01),
   method = "Mardia")
```

`Windham_populationinverse` *Inverse Transform for the Population Parameters Under Windham Weights*

Description

Returns the matrix which reverses the effect of weights on a population for certain models.

Usage

```
Windham_populationinverse(cW)

Windham_populationinverse_alternative(newtheta, previoustheta, cW, cWav)
```

Arguments

cW	A vector of tuning constants for the Windham robustification method performed by Windham().
newtheta	The parameter vector most recently estimated
previoustheta	The parameter vector estimated in the previous step
cWav	The value of the non-zero elements of cW. That is cW have elements that are zero or equal to cWav.

Details

In the Windham robustification method (Windham()) the effect of weighting a population plays a central role. When the the model density is proportional to $\exp(\eta(\theta) \cdot T(u))$, where $T(u)$ is a vector of sufficient statistics for a measurement u, and η is a *linear* function, Then weights proportional to $\exp(\eta(c \circ \theta) \cdot t(u))$, where c is a vector of tuning constants and $\circ$ is the Hadamard (element-wise) product, have a very simple effect on the population parameter vector θ: the weighted population follows a density of the same form, but with a parameter vector of $(1 + c) \circ \theta$. The inverse of this change to the parameter vector is then a matrix multiplication by a diagonal matrix with elements $1/(1 + c_i)$, with c_i denoting the elements of c.

Value

A diagonal matrix with the same number of columns as cW.

Functions

- Windham_populationinverse(): The matrix with diagonal elements $1/(1 + c_i)$
- Windham_populationinverse_alternative(): The transform implemented as described by Scealy et al. (2024). It is mathematically equivalent to multiplication by the result of Windham_populationinverse() in the situation in Scealy et al. (2024).

Index