Package ‘polykde’

February 17, 2025

Type Package

Title Polyspherical Kernel Density Estimation

Version 1.0.0

Date 2025-02-13

Description Kernel density estimation on the polysphere, hypersphere, and circle. Includes functions for density estimation, regression estimation, ridge estimation, bandwidth selection, kernels, samplers, and homogeneity tests. Companion package to García-Portugués and Meilán-Vila (2024) <doi:10.48550/arXiv.2411.04166> and García-Portugués and Meilán-Vila (2023) <doi:10.1007/978-3-031-32729-2_4>.

License GPL-3

Imports abind, doFuture, foreach, future, gsl, movMF, progressr, Rcpp (>= 1.0.8.3), RcppProgress, rotasym, sphunif

Suggests alphashape3d, DirStats, FixedPoint, ks, manipulate, numDeriv, optimParallel, testthat, viridis, rgl, scatterplot3d, sdetorus, smacof

LinkingTo Rcpp, RcppArmadillo, RcppProgress

URL https://github.com/egarpor/polykde

BugReports https://github.com/egarpor/polykde/issues

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation yes

Author Eduardo García-Portugués [aut, cre] (<https://orcid.org/0000-0002-9224-4111>), Andrea Meilán-Vila [ctb] (<https://orcid.org/0000-0001-8537-9280>)

Maintainer Eduardo García-Portugués <edgarcia@est-econ.uc3m.es>

Repository CRAN

Date/Publication 2025-02-17 11:00:01 UTC

Contents

polykde-package — polykde: *Polyspherical Kernel Density Estimation*

Description

Kernel density estimation on the polysphere, hypersphere, and circle. Includes functions for density estimation, regression estimation, ridge estimation, bandwidth selection, kernels, samplers, and homogeneity tests. Companion package to García-Portugués and Meilán-Vila (2024) <doi:10.48550/arXiv.2411.04166> and García-Portugués and Meilán-Vila (2023) <doi:10.1007/978303132729-2_4>.

Author(s)

Eduardo García-Portugués.

References

García-Portugués, E. and Meilán-Vila, A. (2024). Kernel density estimation with polyspherical data and its applications. *arXiv:2411.04166*. doi:10.48550/arXiv.2411.04166.

García-Portugués, E. and Meilán-Vila, A. (2023). Hippocampus shape analysis via skeletal models and kernel smoothing. In Larriba, Y. (Ed.), *Statistical Methods at the Forefront of Biomedical Advances*, pp. 63–82. Springer, Cham. doi:10.1007/9783031327292_4.

See Also

Useful links:

- `https://github.com/egarpor/polykde`
- Report bugs at `https://github.com/egarpor/polykde/issues`

angles_to_polysph — *Conversion between the angular and Cartesian coordinates of the polysphere*

Description

Obtain the angular coordinates of points on a polysphere $\mathcal{S}^{d_1} \times \cdots \times \mathcal{S}^{d_r}$, and vice versa.

Usage

```
angles_to_polysph(theta, d)

polysph_to_angles(x, d)
```

Arguments

theta	matrix of size `c(n, sum(d))` with the angles.
d	vector with the dimensions of the polysphere.
x	matrix of size `c(n, sum(d + 1))` with the Cartesian coordinates on $\mathcal{S}^{d_1} \times \cdots \times \mathcal{S}^{d_r}$. Assumed to be of unit norm by blocks of coordinates in the rows.

Value

- `angles_to_polysph`: the matrix `x`.
- `polysph_to_angles`: the matrix `theta`.

Examples

```
# Check changes of coordinates
polysph_to_angles(angles_to_polysph(rep(pi / 2, 3), d = 2:1), d = 2:1)
angles_to_polysph(polysph_to_angles(x = c(0, 0, 1, 0, 1), d = 2:1), d = 2:1)
```

`angles_to_sph` *Conversion between the angular and Cartesian coordinates of the hypersphere*

Description

Transforms the angles $(\theta_1, \ldots, \theta_d)$ in $[0, \pi)^{d-1} \times [-\pi, \pi)$ into the Cartesian coordinates

$$(\cos(x_1), \sin(x_1)\cos(x_2), \ldots, \sin(x_1)\cdots\sin(x_{d-1})\cos(x_d), \sin(x_1)\cdots\sin(x_{d-1})\sin(x_d))$$

of the hypersphere $\mathcal{S}^d$, and vice versa.

Usage

```
angles_to_sph(theta)

sph_to_angles(x)
```

Arguments

theta	matrix of size `c(n, d)` with the angles.
x	matrix of size `c(n, d + 1)` with the Cartesian coordinates on $\mathcal{S}^d$. Assumed to be of unit norm by rows.

Value

- `angles_to_sph`: the matrix `x`.
- `sph_to_angles`: the matrix `theta`.

Examples

```
# Check changes of coordinates
sph_to_angles(angles_to_sph(c(pi / 2, 0, pi)))
sph_to_angles(angles_to_sph(rbind(c(pi / 2, 0, pi), c(pi, pi / 2, 0))))
angles_to_sph(sph_to_angles(c(0, sqrt(0.5), sqrt(0.1), sqrt(0.4))))
angles_to_sph(sph_to_angles(rbind(c(0, sqrt(0.5), sqrt(0.1), sqrt(0.4)),
                                  c(0, sqrt(0.5), sqrt(0.5), 0),
                                  c(0, 1, 0, 0),
                                  c(0, 0, 0, -1),
                                  c(0, 0, 1, 0))))
```

---

angles_to_torus — *Conversion between the angular and Cartesian coordinates of the torus*

Description

Transforms the angles $(\theta_1, \ldots, \theta_d)$ in $[-\pi, \pi)^d$ into the Cartesian coordinates

$$(\cos(x_1), \sin(x_1), \ldots, \cos(x_d), \sin(x_d))$$

of the torus $(\mathcal{S}^1)^d$, and vice versa.

Usage

```
angles_to_torus(theta)

torus_to_angles(x)
```

Arguments

`theta`	matrix of size `c(n, d)` with the angles.
`x`	matrix of size `c(n, 2 * d)` with the Cartesian coordinates on $(\mathcal{S}^1)^d$. Assumed to be of unit norm by pairs of coordinates in the rows.

Value

- `angles_to_torus`: the matrix `x`.
- `torus_to_angles`: the matrix `theta`.

Examples

```
# Check changes of coordinates
torus_to_angles(angles_to_torus(c(0, pi / 3, pi / 2)))
torus_to_angles(angles_to_torus(rbind(c(0, pi / 3, pi / 2), c(0, 1, -2))))
angles_to_torus(torus_to_angles(c(0, 1, 1, 0)))
angles_to_torus(torus_to_angles(rbind(c(0, 1, 1, 0), c(0, 1, 0, 1))))
```

bw_cv_kre_polysph *Cross-validation bandwidth selection for polyspherical-on-scalar regression*

Description

Computes least squares cross-validation bandwidths for kernel regression estimation with polyspherical response and scalar predictor. It computes both the bandwidth that minimizes the cross-validation loss and its "one standard error" variant.

Usage

```
bw_cv_kre_polysph(X, Y, d, p = 0, h_grid = bw.nrd(X) * 10^seq(-2, 2, l =
  100), plot_cv = TRUE, fast = TRUE)
```

Arguments

`X`	a vector of size `n` with the predictor sample.
`Y`	a matrix of size `c(n, sum(d) + r)` with the response sample on the polysphere.
`d`	vector of size `r` with dimensions.
`p`	degree of local fit, either `0` or `1`. Defaults to `0`.
`h_grid`	bandwidth grid where to optimize the cross-validation loss. Defaults to `bw.nrd(X) * 10^seq(-1, 1, l = 100)`.
`plot_cv`	plot the cross-validation loss curve? Defaults to `TRUE`.
`fast`	use the faster and equivalent version of the cross-validation loss? Defaults to `TRUE`.

Details

A similar output to `glmnet`'s `cv.glmnet` is returned.

Value

A list with the following fields:

`h_min`	the bandwidth that minimizes the cross-validation loss.
`h_1se`	the largest bandwidth within one standard error of the minimal cross-validation loss.
`cvm`	the mean of the cross-validation loss curve.
`cvse`	the standard error of the cross-validation loss curve.

Examples

```
n <- 50
X <- seq(0, 1, l = n)
Y <- r_path_s2r(n = n, r = 1, sigma = 0.1, spiral = TRUE)[, , 1]
bw_cv_kre_polysph(X = X, Y = Y, d = 2, p = 0)
bw_cv_kre_polysph(X = X, Y = Y, d = 2, p = 1, fast = FALSE)
```

`bw_cv_polysph` *Cross-validation bandwidth selection for polyspherical kernel density estimator*

Description

Likelihood Cross-Validation (LCV) and Least Squares Cross-Validation (LSCV) bandwidth selection for the polyspherical kernel density estimator.

Usage

```
bw_cv_polysph(X, d, kernel = 1, kernel_type = 1, k = 10,
  type = c("LCV", "LSCV")[1], M = 10000, bw0 = NULL, na.rm = FALSE,
  ncores = 1, h_min = 0, upscale = FALSE, deriv = 0, ...)
```

Arguments

`X`	a matrix of size `c(n, sum(d) + r)` with the sample.
`d`	vector of size `r` with dimensions.
`kernel`	kernel employed: `1` for von Mises–Fisher (default); `2` for Epanechnikov; `3` for softplus.
`kernel_type`	type of kernel employed: `1` for product kernel (default); `2` for spherically symmetric kernel.
`k`	softplus kernel parameter. Defaults to `10.0`.
`type`	cross-validation type, either `"LCV"` (default) or `"LSCV"`.
`M`	Monte Carlo samples to use for approximating the integral in the LSCV loss.
`bw0`	initial bandwidth for minimizing the CV loss. If `NULL`, it is computed internally by magnifying the `bw_rot_polysph` bandwidths by 50%.
`na.rm`	remove `NA`s in the objective function? Defaults to `FALSE`.
`ncores`	number of cores used during the optimization. Defaults to `1`.
`h_min`	minimum h enforced (componentwise). Defaults to `0`.
`upscale`	rescale the resulting bandwidths to work for derivative estimation? Defaults to `FALSE`.
`deriv`	derivative order to perform the upscaling. Defaults to `0`.
`...`	further arguments passed to `optim` (if `ncores = 1`) or `optimParallel` (if `ncores > 1`).

Value

A list as optim or optimParallel output. In particular, the optimal bandwidth is stored in par.

Examples

```
n <- 50
d <- 1:2
kappa <- rep(10, 2)
X <- r_vmf_polysph(n = n, d = d, mu = r_unif_polysph(n = 1, d = d),
                   kappa = kappa)
bw_cv_polysph(X = X, d = d, type = "LCV")$par
bw_cv_polysph(X = X, d = d, type = "LSCV")$par
```

---

bw_lcv_min_epa — *Minimum bandwidth allowed in likelihood cross-validation for Epanechnikov kernels*

Description

This function computes the minimum bandwidth allowed in likelihood cross-validation with Epanechnikov kernels, for a given dataset and dimension.

Usage

```
bw_lcv_min_epa(X, d, kernel_type = c("prod", "sph")[1])
```

Arguments

X	a matrix of size c(n, sum(d) + r) with the sample.
d	vector of size r with dimensions.
kernel_type	type of kernel employed: 1 for product kernel (default); 2 for spherically symmetric kernel.

Value

The minimum bandwidth allowed.

Examples

```
n <- 5
d <- 1:3
X <- r_unif_polysph(n = n, d = d)
h_min <- rep(bw_lcv_min_epa(X = X, d = d), length(d))
log_cv_kde_polysph(X = X, d = d, h = h_min - 1e-4, kernel = 2) # Problem
log_cv_kde_polysph(X = X, d = d, h = h_min + 1e-4, kernel = 2) # OK
```

`bw_mrot_polysph` *Marginal rule-of-thumb bandwidth selection for polyspherical kernel density estimator*

Description

Computes marginal (sphere by sphere) rule-of-thumb bandwidths for the polyspherical kernel density estimator using a von Mises–Fisher distribution as reference.

Usage

```
bw_mrot_polysph(X, d, kernel = 1, k = 10, upscale = FALSE, deriv = 0,
  kappa = NULL, ...)
```

Arguments

`X`	a matrix of size `c(n, sum(d) + r)` with the sample.
`d`	vector of size `r` with dimensions.
`kernel`	kernel employed: `1` for von Mises–Fisher (default); `2` for Epanechnikov; `3` for softplus.
`k`	softplus kernel parameter. Defaults to `10.0`.
`upscale`	rescale bandwidths to work on $\mathcal{S}^{d_1} \times \cdots \times \mathcal{S}^{d_r}$ and for derivative estimation? Defaults to `FALSE`. If `upscale = 1`, the order `n` is upscaled. If `upscale = 2`, then also the kernel constant is upscaled.
`deriv`	derivative order to perform the upscaling. Defaults to `0`.
`kappa`	estimate of the concentration parameters. Computed if not provided (default).
`...`	further arguments passed to `nlm`.

Value

A vector of size `r` with the marginal optimal bandwidths.

Examples

```
n <- 100
d <- 1:2
kappa <- rep(10, 2)
X <- r_vmf_polysph(n = n, d = d, mu = r_unif_polysph(n = 1, d = d),
                   kappa = kappa)
bw_rot_polysph(X = X, d = d)$bw
bw_mrot_polysph(X = X, d = d)
```

bw_rot_polysph	*Rule-of-thumb bandwidth selection for polyspherical kernel density estimator*

Description

Computes the rule-of-thumb bandwidth for the polyspherical kernel density estimator using a product of von Mises–Fisher distributions as reference in the Asymptotic Mean Integrated Squared Error (AMISE).

Usage

```
bw_rot_polysph(X, d, kernel = 1, kernel_type = c("prod", "sph")[1],
  bw0 = NULL, upscale = FALSE, deriv = 0, k = 10, kappa = NULL, ...)
```

Arguments

`X`	a matrix of size `c(n, sum(d) + r)` with the sample.
`d`	vector of size `r` with dimensions.
`kernel`	kernel employed: `1` for von Mises–Fisher (default); `2` for Epanechnikov; `3` for softplus.
`kernel_type`	type of kernel employed: `1` for product kernel (default); `2` for spherically symmetric kernel.
`bw0`	initial bandwidth for minimizing the CV loss. If `NULL`, it is computed internally by magnifying the `bw_mrot_polysph` bandwidths by 50%.
`upscale`	rescale bandwidths to work on $\mathcal{S}^{d_1} \times \cdots \times \mathcal{S}^{d_r}$ and for derivative estimation? Defaults to `FALSE`. If `upscale = 1`, the order `n` is upscaled. If `upscale = 2`, then also the kernel constant is upscaled.
`deriv`	derivative order to perform the upscaling. Defaults to `0`.
`k`	softplus kernel parameter. Defaults to `10.0`.
`kappa`	estimate of the concentration parameters. Computed if not provided (default).
`...`	further arguments passed to `nlm`.

Details

The selector assumes that the density curvature matrix $\boldsymbol{R}$ of the unknown density is approximable by that of a product of von Mises–Fisher densities, $\boldsymbol{R}(\boldsymbol{\kappa})$. The estimation of the concentration parameters $\boldsymbol{\kappa}$ is done by maximum likelihood.

Value

A list with entries `bw` (optimal bandwidth) and `opt`, the latter containing the output of `nlm`.

Examples

```
n <- 100
d <- 1:2
kappa <- rep(10, 2)
X <- r_vmf_polysph(n = n, d = d, mu = r_unif_polysph(n = 1, d = d),
                   kappa = kappa)
bw_rot_polysph(X = X, d = d)$bw
```

---

clean_euler_ridge *Clean ridge points coming from spurious fits*

Description

Remove points from the ridge that are spurious. The cleaning is done by removing end points in the Euler algorithm that did not converge, do not have a negative second eigenvalue, or are in low-density regions.

Usage

```
clean_euler_ridge(e, X, p_out = NULL)
```

Arguments

e	outcome from `euler_ridge` or `parallel_euler_ridge`.
X	a matrix of size `c(n, sum(d) + r)` with the sample.
p_out	proportion of outliers to remove. Defaults to `NULL` (no cleaning).

Value

A list with the same structure as that returned by `euler_ridge`, but with the spurious points. The removed points are informed in the `removed` field.

Examples

```
## Test on S^2 with some spurious end points

# Sample
r <- 1
d <- 2
n <- 50
ind_dj <- comp_ind_dj(d = d)
set.seed(987202226)
X <- r_path_s2r(n = n, r = r, spiral = FALSE, Theta = cbind(c(1, 0, 0)),
                sigma = 0.2)[, , 1]
col_X <- rep(gray(0), n)
col_X_alp <- rep(gray(0, alpha = 0.25), n)

# Euler
```

```
h_rid <- 0.5
h_eu <- h_rid^2
N <- 30
eps <- 1e-6
X0 <- r_unif_polysph(n = n, d = d)
Y <- euler_ridge(x = X0, X = X, d = d, h = h_rid, h_euler = h_eu,
                 N = N, eps = eps, keep_paths = TRUE)
Y_removed <- clean_euler_ridge(e = Y, X = X)$removed
col_X[Y_removed] <- 2
col_X_alp[Y_removed] <- 2

# Visualization
i <- N # Between 1 and N
sc3 <- scatterplot3d::scatterplot3d(Y$paths[, , 1], color = col_X_alp,
                                    pch = 19, xlim = c(-1, 1),
                                    ylim = c(-1, 1), zlim = c(-1, 1),
                                    xlab = "x", ylab = "y", zlab = "z")
sc3$points3d(rbind(Y$paths[, , i]), col = col_X, pch = 16, cex = 0.75)
for (k in seq_len(nrow(Y$paths))) {

  sc3$points3d(t(Y$paths[k, , ]), col = col_X_alp[k], type = "l")

}
```

`comp_ind_dj` *Index of hyperspheres on a polysphere*

Description

Given Cartesian coordinates of polyspherical data, computes the `0`-based indexes at which the Cartesian coordinates for each hypersphere start and end.

Usage

```
comp_ind_dj(d)
```

Arguments

`d`	vector of size `r` with dimensions.

Value

A vector of size `sum(d) + 1`.

Examples

```
# Example on (S^1)^3
d <- c(1, 1, 1)
comp_ind_dj(d = d)
comp_ind_dj(d = d) + 1
```

```
# Example on S^1 x S^2
d <- c(1, 2)
comp_ind_dj(d = d)
comp_ind_dj(d = d) + 1
```

`curv_vmf_polysph` *Curvature of a polyspherical von Mises–Fisher density*

Description

Computes the curvature matrix $\boldsymbol{R}(\boldsymbol{\kappa})$ of a product of von Mises–Fisher densities on the polysphere. This curvature is used in the rule-of-thumb selector `bw_rot_polysph`.

Usage

```
curv_vmf_polysph(kappa, d, log = FALSE)
```

Arguments

`kappa`	a vector of size `r` with the von Mises–Fisher concentrations.
`d`	vector of size `r` with dimensions.
`log`	compute the (entrywise) logarithm of the curvature matrix? Defaults to `FALSE`.

Value

A matrix of size `c(length(r), length(r))`.

Examples

```
# Curvature matrix
d <- 2:4
kappa <- 1:3
curv_vmf_polysph(kappa = kappa, d = d)
curv_vmf_polysph(kappa = kappa, d = d, log = TRUE)

# Equivalence on the sphere with DirStats::R_Psi_mixvmf
drop(curv_vmf_polysph(kappa = kappa[1], d = d[1]))
d[1]^2 * DirStats::R_Psi_mixvmf(q = d[1], mu = rbind(c(rep(0, d[1]), 1)),
                                kappa = kappa[1], p = 1)
```

`dist_polysph` *Polyspherical distance*

Description

Computation of the distance between points $\boldsymbol{x}$ and $\boldsymbol{y}$ on the polysphere $\mathcal{S}^{d_1} \times \cdots \times \mathcal{S}^{d_r}$:

$$\sqrt{\sum_{j=1}^{r} d_{\mathcal{S}^{d_j}}(\boldsymbol{x}_j, \boldsymbol{y}_j)^2},$$

where $d_{\mathcal{S}^{d_j}}(\boldsymbol{x}_j, \boldsymbol{y}_j) = \cos^{-1}(\boldsymbol{x}_j'\boldsymbol{y}_j)$.

Usage

```
dist_polysph(x, y, ind_dj, norm_x = FALSE, norm_y = FALSE, std = TRUE)

dist_polysph_cross(x, y, ind_dj, norm_x = FALSE, norm_y = FALSE,
  std = TRUE)

dist_polysph_matrix(x, ind_dj, norm_x = FALSE, norm_y = FALSE,
  std = TRUE)
```

Arguments

`x`	a matrix of size `c(n, sum(d) + r)`.
`y`	either a matrix of size `c(m, sum(d) + r)` or a vector of length `sum(d) + r`.
`ind_dj`	`0`-based index separating the blocks of spheres that is computed with `comp_ind_dj`.
`norm_x, norm_y`	ensure a normalization of the data? Default to `FALSE`.
`std`	standardize distance to $[0, 1]$? Uses that the maximum distance is $\sqrt{r}\pi$. Defaults to `TRUE`.

Value

- `dist_polysph`: a vector of size `n` with the distances between `x` and `y`.
- `dist_polysph_matrix`: a matrix of size `c(n, n)` with the pairwise distances of `x`.
- `dist_polysph_cross`: a matrix of distances of size `c(n, m)` with the cross distances between `x` and `y`.

Examples

```
# Example on S^2 x S^3 x S^1
d <- c(2, 3, 1)
ind_dj <- comp_ind_dj(d)
n <- 3
x <- r_unif_polysph(n = n, d = d)
```

```
y <- r_unif_polysph(n = n, d = d)

# Distances of x to y
dist_polysph(x = x, y = y, ind_dj = ind_dj, std = FALSE)
dist_polysph(x = x, y = y[1, , drop = FALSE], ind_dj = ind_dj, std = FALSE)

# Pairwise distance matrix of x
dist_polysph_matrix(x = x, ind_dj = ind_dj, std = FALSE)

# Cross distances between x and y
dist_polysph_cross(x = x, y = y, ind_dj = ind_dj, std = FALSE)
```

`d_unif_polysph` *Density of the uniform distribution on the polysphere*

Description

Computes the density of the uniform distribution on the polysphere.

Usage

```
d_unif_polysph(x, d, log = FALSE)
```

Arguments

`x`	a matrix of size `c(nx, sum(d) + r)` with the evaluation points.
`d`	vector of size `r` with dimensions.
`log`	compute the logarithm of the density? Defaults to `FALSE`.

Value

A vector of size `nx` with the evaluated density.

Examples

```
# Simple check of integration on S^1 x S^2
d <- c(1, 2)
x <- r_unif_polysph(n = 1e4, d = d)
mean(1 / d_unif_polysph(x = x, d = d)) / prod(rotasym::w_p(p = d + 1))
```

`d_vmf_polysph` *Density of the product of von Mises–Fisher distributions on the polysphere*

Description

Computes the density of the product of von Mises–Fisher densities on the polysphere.

Usage

```
d_vmf_polysph(x, d, mu, kappa, log = FALSE)
```

Arguments

x	a matrix of size `c(nx, sum(d) + r)` with the evaluation points.
d	vector of size `r` with dimensions.
mu	a vector of size `sum(d) + r` with the concatenated von Mises–Fisher means.
kappa	a vector of size `r` with the von Mises–Fisher concentrations.
log	compute the logarithm of the density? Defaults to `FALSE`.

Value

A vector of size `nx` with the evaluated density.

Examples

```
# Simple check of integration on S^1 x S^2
d <- c(1, 2)
mu <- c(0, 1, 0, 1, 0)
kappa <- c(1, 1)
x <- r_vmf_polysph(n = 1e4, d = d, mu = mu, kappa = kappa)
mean(1 / d_vmf_polysph(x = x, d = d, mu = mu, kappa = kappa)) /
  prod(rotasym::w_p(p = d + 1))
```

`eff_kern` *Polyspherical kernel moments and efficiencies*

Description

Computes moments of kernels on $\mathcal{S}^{d_1} \times \cdots \times \mathcal{S}^{d_r}$ and efficiencies of kernels on $(\mathcal{S}^d)^r$.

Usage

```
eff_kern(d, r, k = 10, kernel, kernel_type = c("prod", "sph")[1],
  kernel_ref = "2", kernel_ref_type = c("prod", "sph")[2], ...)

b_d(kernel, d, k = 10, kernel_type = c("prod", "sph")[1], ...)

v_d(kernel, d, k = 10, kernel_type = c("prod", "sph")[1], ...)
```

Arguments

d	a scalar with the common dimension of each hypersphere $\mathcal{S}^d$.
r	a scalar with the number of polyspheres of the same dimension.
k	softplus kernel parameter. Defaults to 10.0.
kernel	kernel employed: 1 for von Mises–Fisher (default); 2 for Epanechnikov; 3 for softplus.
kernel_type	type of kernel. Must be either "prod" (product kernel, default) or "sph" (spherically symmetric kernel).
kernel_ref	reference kernel to which compare the efficiency. Uses the same codification as the kernel. Defaults to "2".
kernel_ref_type	type of the reference kernel. Must be either "prod" (product kernel) or "sph" (spherically symmetric kernel, default).
...	further arguments passed to integrate, such as upper, abs.tol, rel.tol, etc.

Value

- b_d: a vector with the first kernel moment on each hypersphere (common if kernel_type = "sph").
- v_d: a vector with the second kernel moment if kernel_type = "prod", or a scalar if kernel_type = "sph".
- eff_kern: a scalar with the kernel efficiency.

Examples

```
# Kernel moments
b_d(kernel = 2, d = c(2, 3), kernel_type = "prod")
v_d(kernel = 2, d = c(2, 3), kernel_type = "prod")
b_d(kernel = 2, d = c(2, 3), kernel_type = "sph")
v_d(kernel = 2, d = c(2, 3), kernel_type = "sph")

# Kernel efficiencies
eff_kern(d = 2, r = 1, kernel = "1")
eff_kern(d = 2, r = 1, kernel = "2")
eff_kern(d = 2, r = 1, k = 10, kernel = "3")
```

euler_ridge *Euler algorithms for polyspherical density ridge estimation*

Description

Functions to perform density ridge estimation on the polysphere $\mathcal{S}^{d_1} \times \cdots \times \mathcal{S}^{d_r}$ through the Euler algorithm in standard, parallel, or block mode.

Usage

```
euler_ridge(x, X, d, h, h_euler = as.numeric(c()),
  weights = as.numeric(c()), wrt_unif = FALSE, normalized = TRUE,
  norm_x = FALSE, norm_X = FALSE, kernel = 1L, kernel_type = 1L,
  k = 10, N = 1000L, eps = 1e-05, keep_paths = FALSE,
  proj_alt = TRUE, fix_u1 = TRUE, sparse = FALSE, show_prog = TRUE,
  show_prog_j = FALSE)

parallel_euler_ridge(x, X, d, h, h_euler, N = 1000, eps = 1e-05,
  keep_paths = FALSE, cores = 1, ...)

block_euler_ridge(x, X, d, h, h_euler, ind_blocks, N = 1000, eps = 1e-05,
  keep_paths = FALSE, cores = 1, ...)
```

Arguments

`x`	a matrix of size `c(nx, sum(d) + r)` with the starting points for the Euler algorithm.
`X`	a matrix of size `c(n, sum(d) + r)` with the sample.
`d`	vector of size `r` with dimensions.
`h`	vector of size `r` with bandwidths.
`h_euler`	vector of size `r` with the advance steps in the Euler method. Set internally as `h` if not provided.
`weights`	weights for each observation. If provided, a vector of size `n` with the weights for multiplying each kernel. If not provided, set internally to `rep(1 / n, n)`, which gives the standard estimator.
`wrt_unif`	flag to return a density with respect to the uniform measure. If `FALSE` (default), the density is with respect to the Lebesgue measure.
`normalized`	flag to compute the normalizing constant of the kernel and include it in the kernel density estimator. Defaults to `TRUE`.
`norm_x, norm_X`	ensure a normalization of the data? Defaults to `FALSE`.
`kernel`	kernel employed: `1` for von Mises–Fisher (default); `2` for Epanechnikov; `3` for softplus.
`kernel_type`	type of kernel employed: `1` for product kernel (default); `2` for spherically symmetric kernel.

k	softplus kernel parameter. Defaults to 10.0.
N	maximum number of Euler iterations. Defaults to 1e3.
eps	convergence tolerance. Defaults to 1e-5.
keep_paths	keep the Euler paths to the ridge? Defaults to FALSE.
proj_alt	alternative projection. Defaults to TRUE.
fix_u1	ensure the u_1 vector is different from x? Prevents the Euler algorithm to "surf the ridge". Defaults to TRUE.
sparse	use a sparse eigendecomposition of the Hessian? Defaults to FALSE.
show_prog	display a progress bar for x? Defaults to TRUE.
show_prog_j	display a progress bar for N? Defaults to FALSE.
cores	cores to use. Defaults to 1.
...	further arguments passed to euler_ridge.
ind_blocks	indexes of the blocks, a vector or length r.

Details

euler_ridge is the main function to perform density ridge estimation through the Euler algorithm from the starting values x to initiate the ridge path. The function euler_ridge_parallel parallelizes on the starting values x. The function euler_ridge_block runs the Euler algorithm marginally in blocks of hyperspheres, instead of jointly in the whole polysphere. This function requires that all the dimensions are the same.

Value

The three functions return a list with the following fields:

ridge_y	a matrix of size c(nx, sum(d) + r) with the end points of Euler algorithm defining the estimated ridge.
lamb_norm_y	a matrix of size c(nx, sum(d) + r) with the Hessian eigenvalues (largest to smallest) evaluated at end points.
log_dens_y	a column vector of size c(nx, 1) with the logarithm of the density at end points.
paths	an array of size c(nx, sum(d) + r, N + 1) containing the Euler paths.
start_x	a matrix of size c(nx, sum(d) + r) with the starting points for the Euler algorithm.
iter	a column vector of size c(nx, 1) counting the iterations required for each point.
conv	a column vector of size c(nx, 1) with convergence flags.
d	vector d.
h	bandwidth used for the kernel density estimator.
error	a column vector of size c(nx, 1) indicating if errors were found for each path.

Examples

```
## Test on S^2 with a small circle trend

# Sample
r <- 1
d <- 2
n <- 50
ind_dj <- comp_ind_dj(d = d)
set.seed(987204452)
X <- r_path_s2r(n = n, r = r, spiral = FALSE, Theta = cbind(c(1, 0, 0)),
                sigma = 0.35)[, , 1]
col_X_alp <- viridis::viridis(n, alpha = 0.25)
col_X <- viridis::viridis(n)

# Euler
h_rid <- 0.5
h_eu <- h_rid^2
N <- 30
eps <- 1e-6
Y <- euler_ridge(x = X, X = X, d = d, h = h_rid, h_euler = h_eu,
                 N = N, eps = eps, keep_paths = TRUE)
Y

# Visualization
i <- N # Between 1 and N
sc3 <- scatterplot3d::scatterplot3d(Y$paths[, , 1], color = col_X_alp,
                                    pch = 19, xlim = c(-1, 1),
                                    ylim = c(-1, 1), zlim = c(-1, 1),
                                    xlab = "x", ylab = "y", zlab = "z")
sc3$points3d(rbind(Y$paths[, , i]), col = col_X, pch = 16, cex = 0.75)
for (k in seq_len(nrow(Y$paths))) {

  sc3$points3d(t(Y$paths[k, , ]), col = col_X_alp[k], type = "l")

}
```

`grad_hess_kde_polysph` *Gradient and Hessian of the polyspherical kernel density estimator*

Description

Computes the gradient $\mathsf{D}\hat{f}(\boldsymbol{x};\boldsymbol{h})$ and Hessian matrix $\mathsf{H}\hat{f}(\boldsymbol{x};\boldsymbol{h})$ of the kernel density estimator $\hat{f}(\boldsymbol{x};\boldsymbol{h})$ on the polysphere $\mathcal{S}^{d_1} \times \cdots \times \mathcal{S}^{d_r}$.

Usage

```
grad_hess_kde_polysph(x, X, d, h, weights = as.numeric(c()),
  projected = TRUE, proj_alt = TRUE, norm_grad_hess = FALSE,
  log = FALSE, wrt_unif = FALSE, normalized = TRUE, norm_x = FALSE,
  norm_X = FALSE, kernel = 1L, kernel_type = 1L, k = 10)
```

Arguments

- `x` — a matrix of size `c(nx, sum(d) + r)` with the evaluation points.
- `X` — a matrix of size `c(n, sum(d) + r)` with the sample.
- `d` — vector of size `r` with dimensions.
- `h` — vector of size `r` with bandwidths.
- `weights` — weights for each observation. If provided, a vector of size `n` with the weights for multiplying each kernel. If not provided, set internally to `rep(1 / n, n)`, which gives the standard estimator.
- `projected` — compute the *projected* gradient and Hessian that accounts for the radial projection? Defaults to `TRUE`.
- `proj_alt` — alternative projection. Defaults to `TRUE`.
- `norm_grad_hess` — normalize the gradient and Hessian dividing by the kernel density estimator? Defaults to `FALSE`.
- `log` — compute the logarithm of the density? Defaults to `FALSE`.
- `wrt_unif` — flag to return a density with respect to the uniform measure. If `FALSE` (default), the density is with respect to the Lebesgue measure.
- `normalized` — flag to compute the normalizing constant of the kernel and include it in the kernel density estimator. Defaults to `TRUE`.
- `norm_x, norm_X` — ensure a normalization of the data? Defaults to `FALSE`.
- `kernel` — kernel employed: `1` for von Mises–Fisher (default); `2` for Epanechnikov; `3` for softplus.
- `kernel_type` — type of kernel employed: `1` for product kernel (default); `2` for spherically symmetric kernel.
- `k` — softplus kernel parameter. Defaults to `10.0`.

Value

A list with the following components:

- `dens` — a column vector of size `c(nx, 1)` with the kernel density estimator evaluated at `x`.
- `grad` — a matrix of size `c(nx, sum(d) + r)` with the gradient of the kernel density estimator evaluated at `x`.
- `hess` — an array of size `c(nx, sum(d) + r, sum(d) + r)` with the Hessian matrix of the kernel density estimator evaluated at `x`.

Examples

```
# Simple check on (S^1)^2
n <- 3
d <- c(1, 1)
mu <- c(0, 1, 0, 1)
kappa <- c(5, 5)
h <- c(0.2, 0.2)
```

```
X <- r_vmf_polysph(n = n, d = d, mu = mu, kappa = kappa)
grh <- grad_hess_kde_polysph(x = X, X = X, d = d, h = h)
str(grh)
grh
```

`hom_test_polysph` *Homogeneity test for several polyspherical samples*

Description

Permutation tests for the equality of distributions of two or k samples of data on $\mathcal{S}^{d_1} \times \cdots \times \mathcal{S}^{d_r}$. The Jensen–Shannon distance is used to construct a test statistic measuring the discrepancy between the k kernel density estimators. Tests based on the mean and scatter matrices are also available, but for only two samples ($k = 2$).

Usage

```
hom_test_polysph(X, d, labels, type = c("jsd", "mean", "scatter", "hd")[1],
  h = NULL, kernel = 1, kernel_type = 1, k = 10, B = 1000,
  M = 10000, plot_boot = FALSE, seed_jsd = NULL, cv_jsd = TRUE)
```

Arguments

`X`	a matrix of size `c(n, sum(d) + r)` with the sample.
`d`	vector of size `r` with dimensions.
`labels`	vector with `k` different levels indicating the group.
`type`	kind of test to be performed: `"jsd"` (default), a test comparing the kernel density estimators for k groups using the Jensen–Shannon distance; `"mean"`, a simple test for the equality of two means (non-omnibus for testing homogeneity); `"scatter"`, a simple test for the equality of two scatter matrices; `"hd"`, a test comparing the kernel density estimators for two groups using the Hellinger distance.
`h`	vector of size `r` with bandwidths.
`kernel`	kernel employed: `1` for von Mises–Fisher (default); `2` for Epanechnikov; `3` for softplus.
`kernel_type`	type of kernel employed: `1` for product kernel (default); `2` for spherically symmetric kernel.
`k`	softplus kernel parameter. Defaults to `10.0`.
`B`	number of permutations to use. Defaults to `1e3`.
`M`	number of Monte Carlo replicates to use when approximating the Hellinger/Jensen–Shannon distance. Defaults to `1e4`.
`plot_boot`	flag to display a graphical output of the test decision. Defaults to `FALSE`.
`seed_jsd`	seed for the Monte Carlo simulations used to estimate the integrals in the Jensen–Shannon distance.
`cv_jsd`	use cross-validation to approximate the Jensen–Shannon distance? Does not require Monte Carlo. Defaults to `TRUE`.

Details

Only type = "jsd" is able to deal with $k > 2$.

The "jsd" statistic is the Jensen–Shannon divergence. This statistic is bounded in $[0, 1]$. The "mean" statistic measures the maximum (chordal) distance between the estimated group means. This statistic is bounded in $[0, 1]$. The "scatter" statistic measures the maximum affine invariant Riemannian metric between the estimated scatter matrices. The "hd" statistic computes a monotonic transformation of the Hellinger distance, which is the Bhattacharyya divergence (or coefficient).

Value

An object of class "htest" with the following fields:

statistic	the value of the test statistic.
p.value	the p-value of the test.
statistic_perm	the B permuted statistics.
n	a table with the sample sizes per group.
h	bandwidths used.
B	number of permutations.
alternative	a character string describing the alternative hypothesis.
method	the kind of test performed.
data.name	a character string giving the name of the data.

Examples

```
## Two-sample case

# H0 holds
n <- c(50, 100)
X1 <- rotasym::r_vMF(n = n[1], mu = c(0, 0, 1), kappa = 1)
X2 <- rotasym::r_vMF(n = n[2], mu = c(0, 0, 1), kappa = 1)
hom_test_polysph(X = rbind(X1, X2), labels = rep(1:2, times = n),
                 d = 2, type = "jsd", h = 0.5)

# H0 does not hold
X2 <- rotasym::r_vMF(n = n[2], mu = c(0, 1, 0), kappa = 2)
hom_test_polysph(X = rbind(X1, X2), labels = rep(1:2, times = n),
                 d = 2, type = "jsd", h = 0.5)

## k-sample case

# H0 holds
n <- c(50, 100, 50)
X1 <- rotasym::r_vMF(n = n[1], mu = c(0, 0, 1), kappa = 1)
X2 <- rotasym::r_vMF(n = n[2], mu = c(0, 0, 1), kappa = 1)
X3 <- rotasym::r_vMF(n = n[3], mu = c(0, 0, 1), kappa = 1)
hom_test_polysph(X = rbind(X1, X2, X3), labels = rep(1:3, times = n),
                 d = 2, type = "jsd", h = 0.5)
```

```
# H0 does not hold
X3 <- rotasym::r_vMF(n = n[3], mu = c(0, 1, 0), kappa = 2)
hom_test_polysph(X = rbind(X1, X2, X3), labels = rep(1:3, times = n),
                 d = 2, type = "jsd", h = 0.5)
```

`index_ridge` *Index a ridge curve, creating the Smoothed and Indexed Estimated Ridge (SIER)*

Description

Indexing of an unordered collection of points defining the estimated density ridge curve. The indexing is done by a multidimensional scaling map to the real line, while the smoothing is done by local polynomial regression for polyspherical-on-scalar regression.

Usage

```
index_ridge(endpoints, X, d, l_index = 1000, f_index = 2,
  probs_scores = seq(0, 1, l = 101), verbose = FALSE, type_bwd = c("1se",
  "min")[1], p = 0, ...)
```

Arguments

endpoints	a matrix of size c(nx, sum(d) + r) with the end points of the ridge algorithm to be indexed.
X	a matrix of size c(n, sum(d) + r) with the sample.
d	vector of size r with dimensions.
l_index	length of the grid index used for finding projections. Defaults to 1e3.
f_index	factor with the range of the grid for finding ridge projections. Defaults to 2, which is twice the range of MDS indexes.
probs_scores	probabilities for indexing the ridge on the probs_scores-quantiles of the scores. Defaults to seq(0, 1, l = 101).
verbose	show diagnostic plots? Defaults to FALSE.
type_bwd	type of cross-validation bandwidth for Nadaraya–Watson, either "min" (minimizer of the cross-validation loss) or "1se" (the "one standard error rule", smoother). Defaults to "1se".
p	degree of local fit, either 0 or 1. Defaults to 0.
...	further arguments passed to bw_cv_kre_polysph.

Details

Indexing is designed to work with collection of ridge points that admit a linear ordering, so that mapping to the real line by multidimensional scaling is adequate. The indexing will not work properly if the ridge points define a closed curve.

Value

A list with the following fields:

scores_X	a vector of size n with the SIER scores for X.
projs_X	a matrix of size c(n, sum(d) + r) with the projections of X onto the SIER.
ord_X	a vector of size n with the ordering of X induced by the SIER scores.
scores_grid	a vector of size length(probs_scores) with score quantiles associated to the probabilities probs_scores.
ridge_grid	a vector of size length(probs_scores) with the SIER evaluated at scores_grid.
mds_index	a vector of size nx with the multidimensional scaling indexes.
ridge_fun	a function that parametrizes the SIER.
h	bandwidth used for the local polynomial regression.
probs_scores	object probs_scores.

Examples

```
## Test on (S^1)^2

# Sample
set.seed(132121)
r <- 2
d <- rep(1, r)
n <- 200
ind_dj <- comp_ind_dj(d = d)
Th <- matrix(runif(n = n * (r - 1), min = -pi / 2, max = pi / 2),
             nrow = n, ncol = r - 1)
Th[, r - 1] <- sort(Th[, r - 1])
Th <- cbind(Th, sdetorus::toPiInt(
  pi + Th[, r - 1] + runif(n = n, min = -pi / 4, max = pi / 4)))
X <- angles_to_torus(Th)
col_X_alp <- viridis::viridis(n, alpha = 0.25)
col_X <- viridis::viridis(n)

# Euler
h_rid <- rep(0.75, r)
h_eu <- h_rid^2
N <- 200
eps <- 1e-6
Y <- euler_ridge(x = X, X = X, d = d, h = h_rid, h_euler = h_eu,
                 N = N, eps = eps, keep_paths = TRUE)

# Visualization
i <- N # Between 1 and N
plot(rbind(torus_to_angles(Y$paths[, , 1])), col = col_X_alp, pch = 19,
     axes = FALSE, xlim = c(-pi, pi), ylim = c(-pi, pi),
     xlab = "", ylab = "")
points(rbind(torus_to_angles(Y$paths[, , i])), col = col_X, pch = 16,
       cex = 0.75)
sdetorus::torusAxis(1:2)
```

```
for (k in seq_len(nrow(Y$paths))) {

  xy <- torus_to_angles(t(Y$paths[k, , ]))
  sdetorus::linesTorus(x = xy[, 1], y = xy[, 2], col = col_X_alp[k])

}

# SIER
ind_rid <- index_ridge(endpoints = Y$ridge_y, X = X, d = d,
                       probs_scores = seq(0, 1, l = 50))
xy <- torus_to_angles(ind_rid$ridge_grid)
sdetorus::linesTorus(x = xy[, 1], y = xy[, 2], col = 2, lwd = 2)
points(torus_to_angles(ind_rid$ridge_fun(quantile(ind_rid$scores_grid))),
       col = 4, pch = 19)

# Scores
plot(density(ind_rid$scores_X), type = "l", xlab = "Scores",
     ylab = "Density", main = "Scores density")
for (i in 1:n) rug(ind_rid$scores_X[i], col = col_X[i])
```

`interp_polysph` *Interpolation on the polysphere*

Description

Creates a sequence of points on the polysphere linearly interpolating between two points extrinsically.

Usage

```
interp_polysph(x, y, ind_dj, N = 10)
```

Arguments

`x`	a vector of size `sum(d) + r` with the begin point.
`y`	a vector of size `sum(d) + r` with the end point.
`ind_dj`	`0`-based index separating the blocks of spheres that is computed with `comp_ind_dj`.
`N`	number of points in the sequence. Defaults to `10`.

Value

A matrix of size `c(N, sum(d) + r)` with the interpolation.

Examples

```
interp_polysph(x = c(1, 0), y = c(0, 1), ind_dj = comp_ind_dj(d = 1))
```

`kde_polysph` *Polyspherical kernel density estimator*

Description

Computes the kernel density estimator for data on the polysphere $\mathcal{S}^{d_1} \times \cdots \times \mathcal{S}^{d_r}$. Given a sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$, this estimator is

$$\hat{f}(\boldsymbol{x}; \boldsymbol{h}) = \sum_{i=1}^{n} L_{\boldsymbol{h}}(\boldsymbol{x}, \boldsymbol{X}_i)$$

for a kernel L and a vector of bandwidths $\boldsymbol{h}$.

Usage

```
kde_polysph(x, X, d, h, weights = as.numeric(c()), log = FALSE,
  wrt_unif = FALSE, normalized = TRUE, intrinsic = FALSE,
  norm_x = FALSE, norm_X = FALSE, kernel = 1L, kernel_type = 1L,
  k = 10)
```

Arguments

`x`	a matrix of size `c(nx, sum(d) + r)` with the evaluation points.
`X`	a matrix of size `c(n, sum(d) + r)` with the sample.
`d`	vector of size `r` with dimensions.
`h`	vector of size `r` with bandwidths.
`weights`	weights for each observation. If provided, a vector of size `n` with the weights for multiplying each kernel. If not provided, set internally to `rep(1 / n, n)`, which gives the standard estimator.
`log`	compute the logarithm of the density? Defaults to `FALSE`.
`wrt_unif`	flag to return a density with respect to the uniform measure. If `FALSE` (default), the density is with respect to the Lebesgue measure.
`normalized`	flag to compute the normalizing constant of the kernel and include it in the kernel density estimator. Defaults to `TRUE`.
`intrinsic`	use the intrinsic distance, instead of the extrinsic-chordal distance, in the kernel? Defaults to `FALSE`.
`norm_x, norm_X`	ensure a normalization of the data? Defaults to `FALSE`.
`kernel`	kernel employed: `1` for von Mises–Fisher (default); `2` for Epanechnikov; `3` for softplus.
`kernel_type`	type of kernel employed: `1` for product kernel (default); `2` for spherically symmetric kernel.
`k`	softplus kernel parameter. Defaults to `10.0`.

Value

A column vector of size `c(nx, 1)` with the evaluation of kernel density estimator.

Examples

```
# Simple check on S^1 x S^2
n <- 1e3
d <- c(1, 2)
mu <- c(0, 1, 0, 0, 1)
kappa <- c(5, 5)
h <- c(0.2, 0.2)
X <- r_vmf_polysph(n = n, d = d, mu = mu, kappa = kappa)
kde_polysph(x = rbind(mu), X = X, d = d, h = h)
d_vmf_polysph(x = rbind(mu), d = d, mu = mu, kappa = kappa)
```

`kernel` *Kernels on the hypersphere and their derivatives*

Description

An isotropic kernel L on $\mathcal{S}^d$ and its normalizing constant are such that $\int_{\mathcal{S}^d} c(h,d,L) L\left(\frac{1-\boldsymbol{x}'\boldsymbol{y}}{h^2}\right)\,\mathrm{d}\boldsymbol{x} = 1$ (extrinsic-chordal distance) or $\int_{\mathcal{S}^d} c(h,d,L) L\left(\frac{\cos^{-1}(\boldsymbol{x}'\boldsymbol{y})^2}{2h^2}\right)\,\mathrm{d}\boldsymbol{x} = 1$ (intrinsic distance).

Usage

```
L(t, kernel = "1", squared = FALSE, deriv = 0, k = 10,
  inc_sfp = TRUE)

c_kern(h, d, kernel = "1", kernel_type = "1", k = 10, log = FALSE,
  inc_sfp = TRUE, intrinsic = FALSE)

grad_L(x, y, h, kernel = 1, k = 10)

hess_L(x, y, h, kernel = 1, k = 10)
```

Arguments

`t`	vector with the evaluation points.
`kernel`	kernel employed: `1` for von Mises–Fisher (default); `2` for Epanechnikov; `3` for softplus.
`squared`	square the kernel? Only for `deriv = 0`. Defaults to `FALSE`.
`deriv`	kernel derivative. Must be `0`, `1`, or `2`. Defaults to `0`.
`k`	softplus kernel parameter. Defaults to `10.0`.
`inc_sfp`	include `softplus(k)` in the constant? Defaults to `TRUE`.
`h`	vector of size `r` with bandwidths.
`d`	vector of size `r` with dimensions.
`kernel_type`	type of kernel employed: `1` for product kernel (default); `2` for spherically symmetric kernel.

`log`	compute the logarithm of the constant? Defaults to `FALSE`.
`intrinsic`	consider the intrinsic distance? Defaults to `FALSE`.
`x`	a matrix of size `c(nx, sum(d) + r)` with the evaluation points.
`y`	center of the kernel, a vector of size `sum(d) + r`.

Details

The gradient and Hessian are computed for the functions $\boldsymbol{x} \mapsto L\left(\frac{1-\boldsymbol{x}'\boldsymbol{y}}{h^2}\right)$.

Value

- `L`: a vector with the kernel evaluated at `t`.
- `grad_L`: a vector with the gradient evaluated at `x`.
- `hess_L`: a matrix with the Hessian evaluated at `x`.

Examples

```
# Constants in terms of h
h_grid <- seq(0.01, 4, l = 100)
r <- 2
d <- 2
dr <- rep(d, r)
c_vmf <- sapply(h_grid, function(hi)
  log(c_kern(h = rep(hi, r), d = dr, kernel = 1, kernel_type = 2)))
c_epa <- sapply(h_grid, function(hi)
  log(c_kern(h = rep(hi, r), d = dr, kernel = 2, kernel_type = 2)))
c_sfp <- sapply(h_grid, function(hi)
  log(c_kern(h = rep(hi, r), d = dr, kernel = 3, k = 1, kernel_type = 2)))
plot(h_grid, c_epa, type = "l", ylab = "Constant", xlab = "h", col = 2)
lines(h_grid, c_sfp, col = 3)
lines(h_grid, c_vmf, col = 1)
abline(v = sqrt(2), lty = 2, col = 2)

# Kernel and its derivatives
h <- 0.5
x <- c(sqrt(2), -sqrt(2), 0) / 2
y <- c(-sqrt(2), sqrt(3), sqrt(3)) / 3
L(t = (1 - sum(x * y)) / h^2)
grad_L(x = x, y = y, h = h)
hess_L(x = x, y = y, h = h)
```

`kre_polysph` *Local polynomial estimator for polyspherical-on-scalar regression*

Description

Computes a local constant (Nadaraya–Watson) or local linear estimator with polyspherical response and scalar predictor.

Usage

```
kre_polysph(x, X, Y, d, h, p = 0)
```

Arguments

x	a vector of size nx with the evaluation points.
X	a vector of size n with the predictor sample.
Y	a matrix of size c(n, sum(d) + r) with the response sample on the polysphere.
d	vector of size r with dimensions.
h	a positive scalar giving the bandwidth.
p	degree of local fit, either 0 or 1. Defaults to 0.

Value

A vector of size nx with the estimated regression curve evaluated at x.

Examples

```
x_grid <- seq(-0.25, 1.25, l = 200)
n <- 50
X <- seq(0, 1, l = n)
Y <- r_path_s2r(n = n, r = 1, sigma = 0.1, spiral = TRUE)[, , 1]
h0 <- bw_cv_kre_polysph(X = X, Y = Y, d = 2, p = 0, plot_cv = FALSE)$h_1se
sc3 <- scatterplot3d::scatterplot3d(Y, pch = 16, xlim = c(-1, 1),
                                    ylim = c(-1, 1), zlim = c(-1, 1),
                                    xlab = "", ylab = "", zlab = "")
sc3$points3d(kre_polysph(x = x_grid, X = X, Y = Y, d = 2, h = h0, p = 0),
             pch = 16, type = "l", col = 2, lwd = 2)
sc3$points3d(kre_polysph(x = x_grid, X = X, Y = Y, d = 2, h = h0, p = 1),
             pch = 16, type = "l", col = 3, lwd = 2)
```

---

log_cv_kde_polysph — *Cross-validation for the polyspherical kernel density estimator*

Description

Computes the logarithm of the cross-validated kernel density estimator: $\log \hat{f}_{-i}(\boldsymbol{X}_i; \boldsymbol{h}), i = 1, \ldots, n.$

Usage

```
log_cv_kde_polysph(X, d, h, weights = as.numeric(c()), wrt_unif = FALSE,
  normalized = TRUE, intrinsic = FALSE, norm_X = FALSE, kernel = 1L,
  kernel_type = 1L, k = 10)
```

Arguments

- `X` — a matrix of size `c(n, sum(d) + r)` with the sample.
- `d` — vector of size `r` with dimensions.
- `h` — vector of size `r` with bandwidths.
- `weights` — weights for each observation. If provided, a vector of size `n` with the weights for multiplying each kernel. If not provided, set internally to `rep(1 / n, n)`, which gives the standard estimator.
- `wrt_unif` — flag to return a density with respect to the uniform measure. If `FALSE` (default), the density is with respect to the Lebesgue measure.
- `normalized` — flag to compute the normalizing constant of the kernel and include it in the kernel density estimator. Defaults to `TRUE`.
- `intrinsic` — use the intrinsic distance, instead of the extrinsic-chordal distance, in the kernel? Defaults to `FALSE`.
- `norm_X` — ensure a normalization of the data? Defaults to `FALSE`.
- `kernel` — kernel employed: `1` for von Mises–Fisher (default); `2` for Epanechnikov; `3` for softplus.
- `kernel_type` — type of kernel employed: `1` for product kernel (default); `2` for spherically symmetric kernel.
- `k` — softplus kernel parameter. Defaults to `10.0`.

Value

A column vector of size `c(n, 1)` with the evaluation of the logarithm of the cross-validated kernel density estimator.

Examples

```
# Simple check on S^1 x S^2
n <- 5
d <- c(1, 2)
h <- c(0.2, 0.2)
X <- r_unif_polysph(n = n, d = d)
log_cv_kde_polysph(X = X, d = d, h = h)
kde_polysph(x = X[1, , drop = FALSE], X = X[-1, ], d = d, h = h, log = TRUE)
```

`polylog_minus_exp_mu` *Polylogarithm function with negative argument*

Description

Computation of the polylogarithm $\mathrm{Li}_s(-e^{\mu})$.

Usage

```
polylog_minus_exp_mu(mu, s, upper = Inf, ...)
```

Arguments

`mu`	vector with exponents of the negative argument.
`s`	vector with indexes of the polylogarithm.
`upper`	upper limit of integration. Defaults to `Inf`.
`...`	further arguments passed to `integrate`, such as `upper`, `abs.tol`, `rel.tol`, etc.

Details

If `s` is an integer, 1/2, 3/2, or 5/2, then routines from the GSL library to compute Fermi–Dirac integrals are called. Otherwise, numerical integration is used.

Value

A vector of size `length(mu)` or `length(s)` with the values of the polylogarithm.

Examples

```
polylog_minus_exp_mu(mu = 1:5, s = 1)
polylog_minus_exp_mu(mu = 1, s = 1:5)
polylog_minus_exp_mu(mu = 1:5, s = 1:5)
```

`proj_grad_kde_polysph` *Projected gradient of the polyspherical kernel density estimator*

Description

Computes the projected gradient $\mathsf{D}_{(p-1)}\hat{f}(\boldsymbol{x};\boldsymbol{h})$ of the kernel density estimator $\hat{f}(\boldsymbol{x};\boldsymbol{h})$ on the polysphere $\mathcal{S}^{d_1} \times \cdots \times \mathcal{S}^{d_r}$, where $p = \sum_{j=1}^{r} d_j + r$ is the dimension of the ambient space.

Usage

```
proj_grad_kde_polysph(x, X, d, h, weights = as.numeric(c()),
  wrt_unif = FALSE, normalized = TRUE, norm_x = FALSE, norm_X = FALSE,
  kernel = 1L, kernel_type = 1L, k = 10, proj_alt = TRUE,
  fix_u1 = TRUE, sparse = FALSE)
```

Arguments

`x`	a matrix of size `c(nx, sum(d) + r)` with the evaluation points.
`X`	a matrix of size `c(n, sum(d) + r)` with the sample.
`d`	vector of size `r` with dimensions.
`h`	vector of size `r` with bandwidths.
`weights`	weights for each observation. If provided, a vector of size `n` with the weights for multiplying each kernel. If not provided, set internally to `rep(1 / n, n)`, which gives the standard estimator.

`wrt_unif`	flag to return a density with respect to the uniform measure. If `FALSE` (default), the density is with respect to the Lebesgue measure.
`normalized`	flag to compute the normalizing constant of the kernel and include it in the kernel density estimator. Defaults to `TRUE`.
`norm_x, norm_X`	ensure a normalization of the data? Defaults to `FALSE`.
`kernel`	kernel employed: `1` for von Mises–Fisher (default); `2` for Epanechnikov; `3` for softplus.
`kernel_type`	type of kernel employed: `1` for product kernel (default); `2` for spherically symmetric kernel.
`k`	softplus kernel parameter. Defaults to `10.0`.
`proj_alt`	alternative projection. Defaults to `TRUE`.
`fix_u1`	ensure the u_1 vector is different from x? Prevents the Euler algorithm to "surf the ridge". Defaults to `TRUE`.
`sparse`	use a sparse eigendecomposition of the Hessian? Defaults to `FALSE`.

Value

A list with the following components:

`eta`	a matrix of size `c(nx, sum(d) + r)` with the projected gradient evaluated at `x`.
`u1`	a matrix of size `c(nx, sum(d) + r)` with the first non-null Hessian eigenvector evaluated at `x`.
`lamb_norm`	a matrix of size `c(nx, sum(d) + r)` with the Hessian eigenvalues (largest to smallest) evaluated at `x`.

Examples

```
# Simple check on (S^1)^2
n <- 3
d <- c(1, 1)
mu <- c(0, 1, 0, 1)
kappa <- c(5, 5)
h <- c(0.2, 0.2)
X <- r_vmf_polysph(n = n, d = d, mu = mu, kappa = kappa)
proj_grad_kde_polysph(x = X, X = X, d = d, h = h)
```

`proj_polysph` *Projection onto the polysphere*

Description

Projects points on $\mathbb{R}^{d_1+\cdots+d_r+r}$ onto the polysphere $\mathcal{S}^{d_1} \times \cdots \times \mathcal{S}^{d_r}$ by normalizing each block of d_j coordinates.

Usage

```
proj_polysph(x, ind_dj)
```

Arguments

x	a matrix of size `c(n, sum(d) + r)`.
ind_dj	`0`-based index separating the blocks of spheres that is computed with `comp_ind_dj`.

Value

A matrix of size `c(n, sum(d) + r)` with the projected points.

Examples

```
# Example on (S^1)^2
d <- c(1, 1)
x <- rbind(c(2, 0, 1, 1))
proj_polysph(x, ind_dj = comp_ind_dj(d))
```

---

r_g_kern *Sample from the angular kernel density*

Description

Simulation from the angular density function of an isotropic kernel on the hypersphere $\mathcal{S}^d$.

Usage

```
r_g_kern(n, d, h, kernel = "1", k = 10)
```

Arguments

n	sample size.
d	vector of size `r` with dimensions.
h	vector of size `r` with bandwidths.
kernel	kernel employed: `1` for von Mises–Fisher (default); `2` for Epanechnikov; `3` for softplus.
k	softplus kernel parameter. Defaults to `10.0`.

Value

A vector of size `n` with the sample.

Examples

```
hist(r_g_kern(n = 1e3, d = 2, h = 1, kernel = "1"), breaks = 30,
     probability = TRUE, main = "", xlim = c(-1, 1))
hist(r_g_kern(n = 1e3, d = 2, h = 1, kernel = "2"), breaks = 30,
     probability = TRUE, main = "", xlim = c(-1, 1))
hist(r_g_kern(n = 1e3, d = 2, h = 1, kernel = "3"), breaks = 30,
     probability = TRUE, main = "", xlim = c(-1, 1))
```

`r_kde_polysph` *Sample from polyspherical kernel density estimator*

Description

Simulates from the distribution defined by a polyspherical kernel density estimator on $\mathcal{S}^{d_1} \times \ldots \times \mathcal{S}^{d_r}$.

Usage

```
r_kde_polysph(n, X, d, h, kernel = 1, kernel_type = 1, k = 10,
  norm_X = FALSE)
```

Arguments

n	sample size.
X	a matrix of size `c(n, sum(d) + r)` with the sample.
d	vector of size `r` with dimensions.
h	vector of size `r` with bandwidths.
kernel	kernel employed: `1` for von Mises–Fisher (default); `2` for Epanechnikov; `3` for softplus.
kernel_type	type of kernel employed: `1` for product kernel (default); `2` for spherically symmetric kernel.
k	softplus kernel parameter. Defaults to `10.0`.
norm_X	ensure a normalization of the data?

Details

The function uses `r_kern_polysph` to sample from the considered kernel.

Value

A matrix of size `c(n, sum(d) + r)` with the sample.

Examples

```
# Simulated data on (S^1)^2
n <- 50
samp <- r_path_s1r(n = n, r = 2, k = c(1, 2), angles = TRUE)
plot(samp, xlim = c(-pi, pi), ylim = c(-pi, pi), col = rainbow(n),
     axes = FALSE, xlab = "", ylab = "", pch = 16, cex = 0.75)
points(torus_to_angles(r_kde_polysph(n = 10 * n, X = angles_to_torus(samp),
                                     d = c(1, 1), h = c(0.1, 0.1))),
       col = "black", pch = 16, cex = 0.2)
sdetorus::torusAxis()

# Simulated data on S^2
n <- 50
samp <- r_path_s2r(n = n, r = 1, sigma = 0.1, kappa = 5,
                   spiral = TRUE)[, , 1]
sc3d <- scatterplot3d::scatterplot3d(
  samp, xlim = c(-1, 1), ylim = c(-1, 1), zlim = c(-1, 1),
  xlab = "", ylab = "", zlab = "", color = rainbow(n), pch = 16
)
xyz <- r_kde_polysph(n = 10 * n, X = samp, d = 2, h = 0.1)
sc3d$points3d(xyz[, 1], xyz[, 2], xyz[, 3], col = "black", pch = 16,
              cex = 0.2)
```

---

r_kern_polysph *Sample kernel-distributed polyspherical data*

Description

Simulates from the distribution defined by a kernel on the polysphere $\mathcal{S}^{d_1} \times \cdots \times \mathcal{S}^{d_r}$.

Usage

```
r_kern_polysph(n, d, mu, h, kernel = 1, kernel_type = 1, k = 10,
  norm_mu = FALSE)
```

Arguments

n	sample size.
d	vector of size r with dimensions.
mu	a vector of size sum(d) + r with the concatenated means that define the center of the kernel.
h	vector of size r with bandwidths.
kernel	kernel employed: 1 for von Mises–Fisher (default); 2 for Epanechnikov; 3 for softplus.
kernel_type	type of kernel employed: 1 for product kernel (default); 2 for spherically symmetric kernel.
k	softplus kernel parameter. Defaults to 10.0.
norm_mu	ensure a normalization of mu? Defaults to FALSE.

Details

Simulation for non-von Mises–Fisher spherically symmetric kernels is done by acceptance-rejection from a von Mises–Fisher proposal distribution.

Value

A matrix of size `c(n, sum(d) + r)` with the sample.

Examples

```
# Simulate kernels in (S^1)^2
n <- 1e3
h <- c(1, 1)
d <- c(1, 1)
mu <- rep(DirStats::to_cir(pi), 2)
samp_ker <- function(kernel, kernel_type, col, main) {
  data <- r_kern_polysph(n = n, d = d, mu = mu, h = h, kernel = kernel,
                         kernel_type = kernel_type)
  ang <- cbind(DirStats::to_rad(data[, 1:2]),
               DirStats::to_rad(data[, 3:4]))
  plot(ang, xlim = c(0, 2 * pi), ylim = c(0, 2 * pi), pch = 16, cex = 0.25,
       col = col, xlab = expression(Theta[1]), ylab = expression(Theta[2]),
       main = main)
}
old_par <- par(mfcol = c(2, 3))
samp_ker(kernel = 2, kernel_type = 2, col = 1, main = "Epa sph. symmetric")
samp_ker(kernel = 2, kernel_type = 1, col = 2, main = "Epa product")
samp_ker(kernel = 3, kernel_type = 2, col = 1, main = "Sfp sph. symmetric")
samp_ker(kernel = 3, kernel_type = 1, col = 2, main = "Sfp product")
samp_ker(kernel = 1, kernel_type = 2, col = 1, main = "vMF sph. symmetric")
samp_ker(kernel = 1, kernel_type = 1, col = 2, main = "vMF product")
par(old_par)

# Simulate kernels in (S^2)^2
n <- 1e3
h <- c(0.2, 0.6)
d <- c(2, 2)
mu <- c(c(0, 0, 1), c(0, -1, 0))
samp_ker <- function(kernel, kernel_type, main) {
  data <- r_kern_polysph(n = n, d = d, mu = mu, h = h, kernel = kernel,
                         kernel_type = kernel_type)
  scatterplot3d::scatterplot3d(rbind(data[, 1:3], data[, 4:6]),
                               xlim = c(-1, 1), ylim = c(-1, 1),
                               zlim = c(-1, 1), pch = 16, xlab = "",
                               ylab = "", zlab = "", cex.symbols = 0.5,
       color = rep(viridis::viridis(n)[rank(data[, 3])], 2), main = main)
}
old_par <- par(mfcol = c(2, 3))
samp_ker(kernel = 2, kernel_type = 2, main = "Epa sph. symmetric")
samp_ker(kernel = 2, kernel_type = 1, main = "Epa product")
samp_ker(kernel = 3, kernel_type = 2, main = "Sfp sph. symmetric")
samp_ker(kernel = 3, kernel_type = 1, main = "Sfp product")
```

```
samp_ker(kernel = 1, kernel_type = 2, main = "vMF sph. symmetric")
samp_ker(kernel = 1, kernel_type = 1, main = "vMF product")
par(old_par)

# Plot simulated data
n <- 1e3
h <- c(1, 1)
d <- c(2, 2)
samp_ker <- function(kernel, kernel_type, col, main) {
  X <- r_kern_polysph(n = n, d = d, mu = mu, h = h, kernel = kernel,
                      kernel_type = kernel_type)
  S <- cbind((1 - X[, 1:3] %*% mu[1:3]) / h[1]^2,
             (1 - X[, 4:6] %*% mu[4:6]) / h[2]^2)
  plot(S, xlim = c(0, 2 / h[1]^2), ylim = c(0, 2 / h[2]^2), pch = 16,
       cex = 0.25, col = col, xlab = expression(t[1]),
       ylab = expression(t[2]), main = main)
  t_grid <- seq(0, 2 / min(h)^2, l = 100)
  gr <- as.matrix(expand.grid(t_grid, t_grid))
  if (kernel_type == "1") {

    dens <- prod(c_kern(h = h, d = d, kernel = kernel, kernel_type = 1)) *
      L(gr[, 1], kernel = kernel) * L(gr[, 2], kernel = kernel)

  } else if (kernel_type == "2") {

    dens <- c_kern(h = h, d = d, kernel = kernel, kernel_type = 2) *
      L(gr[, 1] + gr[, 2], kernel = kernel)

  }
  dens <- matrix(dens, nrow = length(t_grid), ncol = length(t_grid))
  contour(t_grid, t_grid, dens, add = TRUE, col = col,
          levels = seq(0, 0.2, l = 41))

}
old_par <- par(mfcol = c(2, 3))
samp_ker(kernel = 2, kernel_type = 2, col = 1, main = "Epa sph. symmetric")
samp_ker(kernel = 2, kernel_type = 1, col = 2, main = "Epa product")
samp_ker(kernel = 3, kernel_type = 2, col = 1, main = "Sfp sph. symmetric")
samp_ker(kernel = 3, kernel_type = 1, col = 2, main = "Sfp product")
samp_ker(kernel = 1, kernel_type = 2, col = 1, main = "vMF sph. symmetric")
samp_ker(kernel = 1, kernel_type = 1, col = 2, main = "vMF product")
par(old_par)
```

`r_path_s1r` *Samplers of one-dimensional modes of variation for polyspherical data*

Description

Functions for sampling data on $(\mathcal{S}^d)^r$, for $d = 1, 2$, using one-dimensional modes of variation.

Usage

```
r_path_s1r(n, r, alpha = runif(r, -pi, pi), k = sample(-2:2, size = r,
  replace = TRUE), sigma = 0.25, angles = FALSE)

r_path_s2r(n, r, t = 0, c = 1, Theta = t(rotasym::r_unif_sphere(n = r, p
  = 3)), kappa = 0, sigma = 0.25, spiral = FALSE)
```

Arguments

n	sample size.
r	number of spheres in the polysphere $(\mathcal{S}^d)^r$.
alpha	a vector of size r valued in $[-\pi, \pi)$ with the initial angles for the linear trend. Chosen at random by default.
k	a vector of size r with the **integer** slopes defining the angular linear trend. Chosen at random by default.
sigma	standard deviation of the noise about the one-dimensional mode of variation. Defaults to 0.25.
angles	return angles in $[-\pi, \pi)$? Defaults to FALSE.
t	latitude, with respect to Theta, of the small circle. Defaults to 0 (equator).
c	Clélie curve parameter, changing the spiral wrappings. Defaults to 1.
Theta	a matrix of size c(3, r) giving the north poles for $\mathcal{S}^2$. Useful for rotating the sample. Chosen at random by default.
kappa	concentration von Mises–Fisher parameter for longitudes in small circles. Defaults to 0 (uniform).
spiral	consider a spiral (or, more precisely, a Clélie curve) instead of a small circle? Defaults to FALSE.

Value

An array of size c(n, d, r) with samples on $(\mathcal{S}^d)^r$. If angles = TRUE for r_path_s1r, then a matrix of size c(n ,r) with angles is returned.

Examples

```
# Straight trends on (S^1)^2
n <- 100
samp_1 <- r_path_s1r(n = n, r = 2, k = c(1, 2), angles = TRUE)
plot(samp_1, xlim = c(-pi, pi), ylim = c(-pi, pi), col = rainbow(n),
     axes = FALSE, xlab = "", ylab = "", pch = 16)
sdetorus::torusAxis()

# Straight trends on (S^1)^3
n <- 100
samp_2 <- r_path_s1r(n = n, r = 3, angles = TRUE)
pairs(samp_2, xlim = c(-pi, pi), ylim = c(-pi, pi), col = rainbow(n),
      pch = 16)
sdetorus::torusAxis()
```

```
scatterplot3d::scatterplot3d(
  samp_2, xlim = c(-pi, pi), ylim = c(-pi, pi), zlim = c(-pi, pi),
  xlab = "", ylab = "", zlab = "", color = rainbow(n), pch = 16
)

# Small-circle trends on (S^2)^2
n <- 100
samp_3 <- r_path_s2r(n = n, r = 2, sigma = 0.1, kappa = 5)
old_par <- par(mfrow = c(1, 2))
scatterplot3d::scatterplot3d(
  samp_3[, , 1], xlim = c(-1, 1), ylim = c(-1, 1), zlim = c(-1, 1),
  xlab = "", ylab = "", zlab = "", color = rainbow(n), pch = 16
)
scatterplot3d::scatterplot3d(
  samp_3[, , 2], xlim = c(-1, 1), ylim = c(-1, 1), zlim = c(-1, 1),
  xlab = "", ylab = "", zlab = "", color = rainbow(n), pch = 16
)
par(old_par)

# Spiral trends on (S^2)^2
n <- 100
samp_4 <- r_path_s2r(n = n, r = 2, c = 3, spiral = TRUE, sigma = 0.01)
old_par <- par(mfrow = c(1, 2))
scatterplot3d::scatterplot3d(
  samp_4[, , 1], xlim = c(-1, 1), ylim = c(-1, 1), zlim = c(-1, 1),
  xlab = "", ylab = "", zlab = "", color = rainbow(n), pch = 16
)
scatterplot3d::scatterplot3d(
  samp_4[, , 2], xlim = c(-1, 1), ylim = c(-1, 1), zlim = c(-1, 1),
  xlab = "", ylab = "", zlab = "", color = rainbow(n), pch = 16
)
par(old_par)
```

`r_unif_polysph` *Sample uniform polyspherical data*

Description

Simulates from a uniform distribution on the polysphere $\mathcal{S}^{d_1} \times \cdots \times \mathcal{S}^{d_r}$.

Usage

```
r_unif_polysph(n, d)
```

Arguments

`n`	sample size.
`d`	vector of size `r` with dimensions.

Value

A matrix of size `c(n, sum(d) + r)` with the sample.

Examples

```
# Simulate uniform data on (S^1)^2
r_unif_polysph(n = 10, d = c(1, 1))
```

---

r_vmf_polysph — *Sample von Mises–Fisher distributed polyspherical data*

Description

Simulates from a product of von Mises–Fisher distributions on the polysphere $\mathcal{S}^{d_1} \times \cdots \times \mathcal{S}^{d_r}$.

Usage

```
r_vmf_polysph(n, d, mu, kappa, norm_mu = FALSE)
```

Arguments

`n`	sample size.
`d`	vector of size `r` with dimensions.
`mu`	a vector of size `sum(d) + r` with the concatenated von Mises–Fisher means.
`kappa`	a vector of size `r` with the von Mises–Fisher concentrations.
`norm_mu`	ensure a normalization of `mu`? Defaults to `FALSE`.

Value

A matrix of size `c(n, sum(d) + r)` with the sample.

Examples

```
# Simulate vMF data on (S^1)^2
r_vmf_polysph(n = 10, d = c(1, 1), mu = c(1, 0, 0, 1), kappa = c(1, 1))
```

softplus *Stable computation of the softplus function*

Description

Computes the softplus function $\log(1 + e^t)$ in a numerically stable way for large absolute values of t.

Usage

```
softplus(t)
```

Arguments

t	vector or matrix.

Value

The softplus function evaluated at `t`.

Examples

```
curve(softplus(10 * (1 - (1 - x) / 0.1)), from = -1, to = 1)
```

---

view_srep *s-rep viewer*

Description

Plots a skeletal representation (s-rep) object based on its three-dimensional base, spokes, and boundary.

Usage

```
view_srep(base, dirs, bdry, radii, show_base = TRUE, show_base_pt = TRUE,
  show_bdry = TRUE, show_bdry_pt = TRUE, show_seg = TRUE,
  col_base = "red", col_bndy = "blue", col_seg = "green",
  static = TRUE, texts = NULL, cex_base = ifelse(static, 0.5, 6),
  cex_bdry = ifelse(static, 1, 8), lwd_seg = ifelse(static, 1, 2),
  cex_texts = 1, alpha_base = 0.1, alpha_bdry = 0.15, r_texts = 1.25,
  alpha_ashape3d_base = NULL, alpha_ashape3d_bdry = NULL, lit = FALSE,
  ...)
```

Arguments

- `base` base points, a matrix of size `c(nx, 3)`.
- `dirs` directions of spokes, a matrix of size `c(nx, 3)` with unit vectors.
- `bdry` boundary points, a matrix of size `c(nx, 3)`.
- `radii` radii of spokes, a vector of size `nx`.
- `show_base`, `show_base_pt` show base and base grid? Default to `TRUE`.
- `show_bdry`, `show_bdry_pt` show boundary and boundary grid? Default to `TRUE`.
- `show_seg` show segments? Defaults to `TRUE`.
- `col_base`, `col_bndy`, `col_seg` colors for the base, boundary, and segments. Default to `"red"`, `"blue"`, and `"green"`, respectively.
- `static` use static (`scatterplot3d`) or interactive (`plot3d`) plot? Default to `TRUE`.
- `texts` add text labels? If given, it should be a vector of size `nx` with the labels. Defaults to `NULL`.
- `cex_base`, `cex_bdry` size of the base and boundary points.
- `lwd_seg` width of the segments.
- `cex_texts` size of the text labels. Defaults to `1`.
- `alpha_base`, `alpha_bdry` transparencies for base and boundary. Default to `0.1` and `0.15`, respectively.
- `r_texts` magnification of the radius to separate the text labels. Defaults to `1.25`.
- `alpha_ashape3d_base`, `alpha_ashape3d_bdry` alpha parameters for `ashape3d`. Default to `NULL`.
- `lit` lit parameter passed to `material3d`. Defaults to `FALSE`.
- `...` further arguments to be passed to `plot3d` or `scatterplot3d`.

Value

Creates a static or interactive plot.

Examples

```
base <- r_unif_polysph(n = 50, d = 2)
dirs <- base
radii <- runif(nrow(base), min = 0.5, max = 1)
bdry <- base + radii * dirs
view_srep(base = base, dirs = dirs, bdry = bdry, radii = radii,
          texts = 1:50, xlim = c(-2, 2), ylim = c(-2, 2), zlim = c(-2, 2))
```

Index