Package ‘sphunif’

September 2, 2021

Type Package

Title Uniformity Tests on the Circle, Sphere, and Hypersphere

Version 1.0.1

Date 2021-09-01

Description Implementation of uniformity tests on the circle and (hyper)sphere. The main function of the package is unif_test(), which conveniently collects more than 30 tests for assessing uniformity on S^{p-1}={x in R^p : ||x||=1}, p >= 2. The test statistics are implemented in the unif_stat() function, which allows computing several statistics to several samples within a single call, thus facilitating Monte Carlo experiments. Furthermore, the unif_stat_MC() function allows parallelizing them in a simple way. The asymptotic null distributions of the statistics are available through the function unif_stat_distr(). The core of 'sphunif' is coded in C++ by relying on the 'Rcpp' package. The package also provides several novel datasets and gives the reproducibility for the data application in García-Portugués, Navarro-Esteban and Cuesta-Albertos (2020) <arXiv:2008.09897>.

License GPL-3

LazyData true

Depends R (>= 3.5.0), Rcpp

Imports doFuture, foreach, future, gsl, rotasym

Suggests CompQuadForm, dgof, goftest, knitr, markdown, mvtnorm, numDeriv, progress, progressr, rmarkdown, scatterplot3d, testthat, viridisLite

LinkingTo Rcpp, RcppArmadillo

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

BugReports https://github.com/egarpor/sphunif

Encoding UTF-8

RoxygenNote 7.1.1

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation yes

Author Eduardo García-Portugués [aut, cre]
(<https://orcid.org/0000-0002-9224-4111>),
Thomas Verdebout [aut] (<https://orcid.org/0000-0002-3277-9587>),
Paula Navarro [ctb]

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

Repository CRAN

Date/Publication 2021-09-02 07:40:02 UTC

R topics documented:

sphunif-package   sphunif: *Uniformity Tests on the Circle, Sphere, and Hypersphere*

Description

Implementation of uniformity tests on the circle and (hyper)sphere. The main function of the package is unif_test, which conveniently collects more than 30 tests for assessing uniformity on $S^{p-1} = \{\mathbf{x} \in R^p : ||\mathbf{x}|| = 1\}$, $p \geq 2$. The test statistics are implemented in the unif_stat function, which allows computing several statistics to different samples within a single call, thus facilitating Monte Carlo experiments. Furthermore, the unif_stat_MC function allows parallelizing them in a simple way. The asymptotic null distributions of the statistics are available through the function unif_stat_distr. The core of sphunif-package is coded in C++ by relying on the Rcpp-package. The package also provides several novel datasets and gives the reproducibility for the data application in García-Portugués, Navarro-Esteban and Cuesta-Albertos (2020) <arXiv:2008.09897>.

Author(s)

Eduardo García-Portugués and Thomas Verdebout.

References

García-Portugués, E. and Verdebout, T. (2018) An overview of uniformity tests on the hypersphere. *arXiv:1804.00286*. https://arxiv.org/abs/1804.00286.

García-Portugués, E., Navarro-Esteban, P., Cuesta-Albertos, J. A. (2020) On a projection-based class of uniformity tests on the hypersphere. *arXiv:2008.09897*. https://arxiv.org/abs/2008.09897

García-Portugués, E., Navarro-Esteban, P., and Cuesta-Albertos, J. A. (2021). A Cramér–von Mises test of uniformity on the hypersphere. In Balzano, S., Porzio, G. C., Salvatore, R., Vistocco, D., and Vichi, M. (Eds.), *Statistical Learning and Modeling in Data Analysis*, Studies in Classification, Data Analysis and Knowledge Organization, pp. 107—116. Springer, Cham. doi: 10.1007/9783-030699444_12.

García-Portugués, E., Paindaveine, D., and Verdebout, T. (2021). On the power of Sobolev tests for isotropy under local rotationally symmetric alternatives. *arXiv:2108.09874*. https://arxiv.org/abs/2108.09874

angles_to_sphere   *Conversion between angular and Cartesian coordinates of the (hyper)sphere*

Description

Transforms the angles $(\theta_1, \ldots, \theta_{p-1})'$ in $[0, \pi)^{p-2} \times [-\pi, \pi)$ into the Cartesian coordinates

$$(\cos(x_1), \sin(x_1)\cos(x_2), \ldots, \sin(x_1)\cdots\sin(x_{p-2})\cos(x_{p-1}), \sin(x_1)\cdots\sin(x_{p-2})\sin(x_{p-1}))'$$

of S^{p-1}, and vice versa.

Usage

```
angles_to_sphere(theta)

sphere_to_angles(x)
```

Arguments

theta	matrix of size c(n,p -1) with the angles.
x	matrix of size c(n,p) with the Cartesian coordinates. Assumed to be of unit norm by rows.

Value

For angles_to_sphere, the matrix x. For sphere_to_angles, the matrix theta.

Examples

```
# Check changes of coordinates
sphere_to_angles(angles_to_sphere(c(pi / 2, 0, pi)))
sphere_to_angles(angles_to_sphere(rbind(c(pi / 2, 0, pi), c(pi, pi / 2, 0))))
angles_to_sphere(sphere_to_angles(c(0, sqrt(0.5), sqrt(0.1), sqrt(0.4))))
angles_to_sphere(sphere_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))))

# Circle
sphere_to_angles(angles_to_sphere(0))
sphere_to_angles(angles_to_sphere(cbind(0:3)))
angles_to_sphere(cbind(sphere_to_angles(rbind(c(0, 1), c(1, 0)))))
angles_to_sphere(cbind(sphere_to_angles(rbind(c(0, 1)))))
```

---

avail_tests   *Available circular and (hyper)spherical uniformity tests*

Description

Listing of the tests implemented in the sphunif package.

Usage

```
avail_cir_tests

avail_sph_tests
```

Format

An object of class `character` of length 30.

An object of class `character` of length 14.

Value

A character vector whose elements are valid inputs for the `type` argument in `unif_test`, `unif_stat`, `unif_stat_distr`, and `unif_stat_MC`. `avail_cir_tests` provides the available circular tests and `avail_sph_tests` the (hyper)spherical tests.

Examples

```
# Circular tests
avail_cir_tests

# Spherical tests
avail_sph_tests
```

`A_theta_x` *Surface area of the intersection of two hyperspherical caps*

Description

Computation of

$$A_x(\theta_{ij}) := \frac{1}{\omega_p} \int_{S^{p-1}} 1_{\{\mathbf{X}_i'\boldsymbol{\gamma} \leq x, \mathbf{X}_j'\boldsymbol{\gamma} \leq x\}} \,\mathrm{d}\boldsymbol{\gamma},$$

where $\theta_{ij} := \cos^{-1}(\mathbf{X}_i'\mathbf{X}_j) \in [0, \pi]$, $x \in [-1, 1]$, and ω_p is the surface area of S^{p-1}. $A_x(\theta_{ij})$ is the proportion of surface area of S^{p-1} covered by the intersection of two hyperspherical caps centered at $\mathbf{X}_i$ and $\mathbf{X}_j$ and with common solid angle $\pi - \cos^{-1}(x)$.

Usage

```
A_theta_x(theta, x, p, N = 160L, as_matrix = TRUE)
```

Arguments

`theta`	vectors with values in $[0, \pi]$.
`x`	vector with values in $[-1, 1]$.
`p`	integer giving the dimension of the ambient space R^p that contains S^{p-1}.
`N`	number of points used in the Gauss-Legendre quadrature. Defaults to `160`.
`as_matrix`	return a matrix with the values of $A_x(\theta)$ on the grid formed by `theta` and `x`? If `FALSE`, $A_x(\theta)$ is evaluated on `theta` and `x` if they equal in size. Defaults to `TRUE`.

Details

See García-Portugués et al. (2020) for more details about the $A_x(\theta)$ function.

Value

A matrix of size `c(length(theta),length(x))` containing the evaluation of $A_x(\theta)$ if `as_matrix = TRUE`. Otherwise, a vector of size `c(length(theta)` if `theta` and `x` equal in size.

References

García-Portugués, E., Navarro-Esteban, P., Cuesta-Albertos, J. A. (2020) On a projection-based class of uniformity tests on the hypersphere. *arXiv:2008.09897*. https://arxiv.org/abs/2008.09897

Examples

```
# Plot A_x(theta) for several dimensions and x's
A_lines <- function(x, th = seq(0, pi, l = 200)) {

  plot(th, A_theta_x(theta = th, x = x, p = 2), type = "l",
       col = 1, ylim = c(0, 1.25), main = paste("x =", x),
       ylab = expression(A[x](theta)),
       xlab = expression(theta), axes = FALSE)
  axis(1, at = c(0, pi / 4, pi / 2, 3 * pi / 4, pi),
       labels = expression(0, pi / 4, pi / 2, 3 * pi / 4, pi))
  axis(2); box()
  abline(h = c(0, 1), lty = 2)
  lines(th, A_theta_x(theta = th, x = x, p = 3), col = 2)
  lines(th, A_theta_x(theta = th, x = x, p = 4), col = 3)
  lines(th, A_theta_x(theta = th, x = x, p = 5), col = 4)
  legend("top", lwd = 2, legend = paste("p =", 2:5),
         col = 1:4, cex = 0.75, horiz = TRUE)

}
old_par <- par(mfrow = c(2, 3))
A_lines(x = -0.75)
A_lines(x = -0.25)
A_lines(x = 0)
A_lines(x = 0.25)
A_lines(x = 0.5)
A_lines(x = 0.75)
par(old_par)

# As surface of (theta, x) for several dimensions
A_surf <- function(p, x = seq(-1, 1, l = 201), th = seq(0, pi, l = 201)) {

  col <- c("white", viridisLite::viridis(20))
  breaks <- c(-1, seq(1e-15, 1, l = 21))
  A <- A_theta_x(theta = th, x = x, p = p)
  image(th, x, A, main = paste("p =", p), col = col, breaks = breaks,
        xlab = expression(theta), axes = FALSE)
  axis(1, at = c(0, pi / 4, pi / 2, 3 * pi / 4, pi),
```

```
      labels = expression(0, pi / 4, pi / 2, 3 * pi / 4, pi))
  axis(2); box()
  contour(th, x, A, levels = breaks, add = TRUE)

}
old_par <- par(mfrow = c(2, 2))
A_surf(p = 2)
A_surf(p = 3)
A_surf(p = 4)
A_surf(p = 5)
par(old_par)

# No matrix return
th <- seq(0, pi, l = 5)
x <- seq(-1, 1, l = 5)
diag(A_theta_x(theta = th, x = x, p = 2))
A_theta_x(theta = th, x = x, p = 2, as_matrix = FALSE)
```

`cir_coord_conv` *Transforming between polar and Cartesian coordinates*

Description

Transformation between a matrix `Theta` containing `M` circular samples of size `n` on $[0, 2\pi)$ and an array `X` containing the associated Cartesian coordinates on $S^1 := \{\mathbf{x} \in R^2 : ||\mathbf{x}|| = 1\}$.

Usage

```
Theta_to_X(Theta)

X_to_Theta(X)
```

Arguments

`Theta`	a **matrix** of size `c(n,M)` with `M` samples of size `n` of circular data on $[0, 2\pi)$. Must not contain `NA`'s.
`X`	an **array** of size `c(n,2,M)` containing the Cartesian coordinates of `M` samples of size `n` of directions on S^1. Must not contain `NA`'s.

Value

- `Theta_to_X`: the corresponding `X`.
- `X_to_Theta`: the corresponding `Theta`.

Examples

```
# Sample
Theta <- r_unif_cir(n = 10, M = 2)
X <- r_unif_sph(n = 10, p = 2, M = 2)

# Check equality
sum(abs(X - Theta_to_X(X_to_Theta(X))))
sum(abs(Theta - X_to_Theta(Theta_to_X(Theta))))
```

`cir_gaps` *Circular gaps*

Description

Computation of the circular gaps of an angular sample $\Theta_1, \ldots, \Theta_n$ on $[0, 2\pi)$, defined as

$$\Theta_{(2)} - \Theta_{(1)}, \ldots, \Theta_{(n)} - \Theta_{(n-1)}, 2\pi - \Theta_{(n)} - \Theta_{(1)},$$

where

$$0 \leq \Theta_{(1)} \leq \Theta_{(2)} \leq \ldots \leq \Theta_{(n)} \leq 2\pi.$$

Usage

```
cir_gaps(Theta, sorted = FALSE)
```

Arguments

`Theta`	a **matrix** of size `c(n,M)` with `M` samples of size `n` of circular data on $[0, 2\pi)$. Must not contain `NA`'s.
`sorted`	are the columns of `Theta` sorted increasingly? If `TRUE`, performance is improved. If `FALSE` (default), each column of `Theta` is sorted internally.

Value

A matrix of size `c(n,M)` containing the `n` circular gaps for each of the `M` circular samples.

Warning

Be careful on avoiding the next bad usages of `cir_gaps`, which will produce spurious results:

- The entries of `Theta` are *not* in $[0, 2\pi)$.
- `Theta` is *not* sorted increasingly when `data_sorted = TRUE`.

Examples

```
Theta <- cbind(c(pi, 0, 3 * pi / 2), c(0, 3 * pi / 2, pi), c(5, 3, 1))
cir_gaps(Theta)
```

cir_stat_Kuiper *Statistics for testing circular uniformity*

Description

Low-level implementation of several statistics for assessing circular uniformity on $[0, 2\pi)$ or, equivalently, $S^1 := \{\mathbf{x} \in R^2 : ||\mathbf{x}|| = 1\}$.

Usage

```
cir_stat_Kuiper(Theta, sorted = FALSE, KS = FALSE, Stephens = FALSE)

cir_stat_Watson(Theta, sorted = FALSE, CvM = FALSE, Stephens = FALSE)

cir_stat_Watson_1976(Theta, sorted = FALSE, minus = FALSE)

cir_stat_Range(Theta, sorted = FALSE, gaps_in_Theta = FALSE, max_gap = TRUE)

cir_stat_Rao(Theta, sorted = FALSE, gaps_in_Theta = FALSE)

cir_stat_Greenwood(Theta, sorted = FALSE, gaps_in_Theta = FALSE)

cir_stat_Log_gaps(Theta, sorted = FALSE, gaps_in_Theta = FALSE, abs_val = TRUE)

cir_stat_Vacancy(Theta, a = 2 * pi, sorted = FALSE, gaps_in_Theta = FALSE)

cir_stat_Max_uncover(Theta, a = 2 * pi, sorted = FALSE, gaps_in_Theta = FALSE)

cir_stat_Num_uncover(Theta, a = 2 * pi, sorted = FALSE,
  gaps_in_Theta = FALSE, minus_val = TRUE)

cir_stat_Gini(Theta, sorted = FALSE, gaps_in_Theta = FALSE)

cir_stat_Gini_squared(Theta, sorted = FALSE, gaps_in_Theta = FALSE)

cir_stat_Ajne(Theta, Psi_in_Theta = FALSE)

cir_stat_Rothman(Theta, t = 1/3, Psi_in_Theta = FALSE)

cir_stat_Hodges_Ajne(Theta, asymp_std = FALSE, sorted = FALSE,
  use_Cressie = TRUE)

cir_stat_Cressie(Theta, t = 1/3, sorted = FALSE)

cir_stat_FG01(Theta, sorted = FALSE)

cir_stat_Rayleigh(Theta, m = 1L)
```

```
cir_stat_Bingham(Theta)

cir_stat_Hermans_Rasson(Theta, Psi_in_Theta = FALSE)

cir_stat_Gine_Gn(Theta, Psi_in_Theta = FALSE)

cir_stat_Gine_Fn(Theta, Psi_in_Theta = FALSE)

cir_stat_Pycke(Theta, Psi_in_Theta = FALSE)

cir_stat_Pycke_q(Theta, Psi_in_Theta = FALSE, q = 0.5)

cir_stat_Bakshaev(Theta, Psi_in_Theta = FALSE)

cir_stat_Riesz(Theta, Psi_in_Theta = FALSE, s = 1)

cir_stat_PCvM(Theta, Psi_in_Theta = FALSE)

cir_stat_PRt(Theta, t = 1/3, Psi_in_Theta = FALSE)

cir_stat_PAD(Theta, Psi_in_Theta = FALSE, AD = FALSE, sorted = FALSE)

cir_stat_CCF09(Theta, dirs, K_CCF09 = 25L, original = FALSE)
```

Arguments

`Theta` a **matrix** of size `c(n,M)` with `M` samples of size `n` of circular data on $[0, 2\pi)$. Must not contain `NA`'s.

`sorted` are the columns of `Theta` sorted increasingly? If `TRUE`, performance is improved. If `FALSE` (default), each column of `Theta` is sorted internally.

`KS` compute the Kolmogorov-Smirnov statistic (which is *not* invariant under origin shifts) instead of the Kuiper statistic? Defaults to `FALSE`.

`Stephens` compute Stephens (1970) modification so that the null distribution of the is less dependent on the sample size? The modification does not alter the test decision.

`CvM` compute the Cramér-von Mises statistic (which is *not* invariant under origin shifts) instead of the Watson statistic? Defaults to `FALSE`.

`minus` compute the invariant D_n^- instead of D_n^+? Defaults to `FALSE`.

`gaps_in_Theta` does `Theta` contain the matrix of *circular gaps* that is obtained with `cir_gaps(Theta)`? If `FALSE` (default), the circular gaps are computed internally.

`max_gap` compute the maximum gap for the range statistic? If `TRUE` (default), rejection happens for *large* values of the statistic, which is consistent with the rest of tests. Otherwise, the minimum gap is computed and rejection happens for *low* values.

`abs_val` return the absolute value of the Darling's log gaps statistic? If `TRUE` (default), rejection happens for *large* values of the statistic, which is consistent with the rest of tests. Otherwise, the signed statistic is computed and rejection happens for large *absolute* values.

a
: $a_n = a/n$ parameter used in the length of the arcs of the coverage-based tests. Must be positive. Defaults to `2 * pi`.

minus_val
: return the negative value of the (standardized) number of uncovered spacings? If `TRUE` (default), rejection happens for *large* values of the statistic, which is consistent with the rest of tests. Otherwise, rejection happens for *low* values.

Psi_in_Theta
: does `Theta` contain the shortest angles matrix $\mathbf{\Psi}$ that is obtained with `Psi_mat(array(Theta,dim = c(n,1,M)))`? If `FALSE` (default), $\mathbf{\Psi}$ is computed internally.

t
: t parameter for the Rothman and Cressie tests, a real in $(0, 1)$. Defaults to `1 / 3`.

asymp_std
: normalize the Hodges-Ajne statistic in terms of its asymptotic distribution? Defaults to `FALSE`.

use_Cressie
: compute the Hodges-Ajne statistic as a particular case of the Cressie statistic? Defaults to `TRUE` as it is more efficient. If `FALSE`, the geometric construction in Ajne (1968) is employed.

m
: integer m for the m-modal Rayleigh test. Defaults to `m = 1` (the standard Rayleigh test).

q
: q parameter for the Pycke "q-test", a real in $(0, 1)$. Defaults to `1 / 2`.

s
: s parameter for the s-Riesz test, a real in $(0, 2)$. Defaults to `1`.

AD
: compute the Anderson-Darling statistic (which is *not* invariant under origin shifts) instead of the Projected Anderson-Darling statistic? Defaults to `FALSE`.

dirs
: a matrix of size `c(n_proj,2)` containing `n_proj` random directions (in Cartesian coordinates) on S^1 to perform the CCF09 test.

K_CCF09
: integer giving the truncation of the series present in the asymptotic distribution of the Kolmogorov-Smirnov statistic. Defaults to `25`.

original
: return the CCF09 statistic as originally defined? If `FALSE` (default), a faster and equivalent statistic is computed, and rejection happens for *large* values of the statistic, which is consistent with the rest of tests. Otherwise, rejection happens for *low* values.

Details

Descriptions and references for most of the statistics are available in García-Portugués and Verdebout (2018).

The statistics `cir_stat_PCvM` and `cir_stat_PRt` are provided for the sake of completion, but they equal the more efficiently-implemented statistics `2 * cir_stat_Watson` and `cir_stat_Rothman`, respectively.

Value

A matrix of size `c(M,1)` containing the statistics for each of the `M` samples.

Warning

Be careful on avoiding the next bad usages of the functions, which will produce spurious results:

- The entries of Theta are *not* in $[0, 2\pi)$.
- Theta does *not* contain the circular gaps when gaps_in_Theta = TRUE.
- Theta is *not* sorted increasingly when data_sorted = TRUE.
- Theta does *not* contain Psi_mat(array(Theta,dim = c(n,1,M))) when Psi_in_Theta = TRUE.
- The directions in dirs do *not* have unit norm.

References

García-Portugués, E. and Verdebout, T. (2018) An overview of uniformity tests on the hypersphere. *arXiv:1804.00286*. https://arxiv.org/abs/1804.00286.

Examples

```
## Sample uniform circular data

M <- 2
n <- 100
set.seed(987202226)
Theta <- r_unif_cir(n = n, M = M)

## Tests based on the empirical cumulative distribution function

# Kuiper
cir_stat_Kuiper(Theta)
cir_stat_Kuiper(Theta, Stephens = TRUE)

# Watson
cir_stat_Watson(Theta)
cir_stat_Watson(Theta, Stephens = TRUE)

# Watson (1976)
cir_stat_Watson_1976(Theta)

## Partition-based tests

# Ajne
Theta_array <- Theta
dim(Theta_array) <- c(nrow(Theta), 1, ncol(Theta))
Psi <- Psi_mat(Theta_array)
cir_stat_Ajne(Theta)
cir_stat_Ajne(Psi, Psi_in_Theta = TRUE)

# Rothman
cir_stat_Rothman(Theta, t = 0.5)
cir_stat_Rothman(Theta)
cir_stat_Rothman(Psi, Psi_in_Theta = TRUE)
```

```
# Hodges-Ajne
cir_stat_Hodges_Ajne(Theta)
cir_stat_Hodges_Ajne(Theta, use_Cressie = FALSE)

# Cressie
cir_stat_Cressie(Theta, t = 0.5)
cir_stat_Cressie(Theta)

# FG01
cir_stat_FG01(Theta)

## Spacings-based tests

# Range
cir_stat_Range(Theta)

# Rao
cir_stat_Rao(Theta)

# Greenwood
cir_stat_Greenwood(Theta)

# Log gaps
cir_stat_Log_gaps(Theta)

# Vacancy
cir_stat_Vacancy(Theta)

# Maximum uncovered spacing
cir_stat_Max_uncover(Theta)

# Number of uncovered spacings
cir_stat_Num_uncover(Theta)

# Gini mean difference
cir_stat_Gini(Theta)

# Gini mean squared difference
cir_stat_Gini_squared(Theta)

## Sobolev tests

# Rayleigh
cir_stat_Rayleigh(Theta)
cir_stat_Rayleigh(Theta, m = 2)

# Bingham
cir_stat_Bingham(Theta)

# Hermans-Rasson
cir_stat_Hermans_Rasson(Theta)
cir_stat_Hermans_Rasson(Psi, Psi_in_Theta = TRUE)
```

```
# Gine Fn
cir_stat_Gine_Fn(Theta)
cir_stat_Gine_Fn(Psi, Psi_in_Theta = TRUE)

# Gine Gn
cir_stat_Gine_Gn(Theta)
cir_stat_Gine_Gn(Psi, Psi_in_Theta = TRUE)

# Pycke
cir_stat_Pycke(Theta)
cir_stat_Pycke(Psi, Psi_in_Theta = TRUE)

# Pycke q
cir_stat_Pycke_q(Theta)
cir_stat_Pycke_q(Psi, Psi_in_Theta = TRUE)

# Bakshaev
cir_stat_Bakshaev(Theta)
cir_stat_Bakshaev(Psi, Psi_in_Theta = TRUE)

# Riesz
cir_stat_Riesz(Theta, s = 1)
cir_stat_Riesz(Psi, Psi_in_Theta = TRUE, s = 1)
# Projected Cramér-von Mises
cir_stat_PCvM(Theta)
cir_stat_PCvM(Psi, Psi_in_Theta = TRUE)

# Projected Rothman
cir_stat_PRt(Theta, t = 0.5)
cir_stat_PRt(Theta)
cir_stat_PRt(Psi, Psi_in_Theta = TRUE)

# Projected Anderson-Darling
cir_stat_PAD(Theta)
cir_stat_PAD(Psi, Psi_in_Theta = TRUE)

## Other tests

# CCF09
dirs <- r_unif_sph(n = 3, p = 2, M = 1)[, , 1]
cir_stat_CCF09(Theta, dirs = dirs)

## Connection of Kuiper and Watson statistics with KS and CvM, respectively

# Rotate sample for KS and CvM
alpha <- seq(0, 2 * pi, l = 1e4)
KS_alpha <- sapply(alpha, function(a) {
  cir_stat_Kuiper((Theta[, 2, drop = FALSE] + a) %% (2 * pi), KS = TRUE)
})
CvM_alpha <- sapply(alpha, function(a) {
  cir_stat_Watson((Theta[, 2, drop = FALSE] + a) %% (2 * pi), CvM = TRUE)
})
```

```
AD_alpha <- sapply(alpha, function(a) {
  cir_stat_PAD((Theta[, 2, drop = FALSE] + a) %% (2 * pi), AD = TRUE)
})

# Kuiper is the maximum rotated KS
plot(alpha, KS_alpha, type = "l")
abline(h = cir_stat_Kuiper(Theta[, 2, drop = FALSE]), col = 2)
points(alpha[which.max(KS_alpha)], max(KS_alpha), col = 2, pch = 16)

# Watson is the minimum rotated CvM
plot(alpha, CvM_alpha, type = "l")
abline(h = cir_stat_Watson(Theta[, 2, drop = FALSE]), col = 2)
points(alpha[which.min(CvM_alpha)], min(CvM_alpha), col = 2, pch = 16)

# Anderson-Darling is the average rotated AD?
plot(alpha, AD_alpha, type = "l")
abline(h = cir_stat_PAD(Theta[, 2, drop = FALSE]), col = 2)
abline(h = mean(AD_alpha), col = 3)
```

`comets` *Comet orbits*

Description

Comet orbits data from the JPL Small-Body Database Search Engine. The normal vector of a comet orbit represents is a vector on S^2.

Usage

```
comets
```

Format

A data frame with 3633 rows and 8 variables:

id database ID.

full_name full name/designation following the IUA naming convention.

i inclination; the orbit angle with respect to the ecliptic plane, in radians in $[0, \pi]$.

om longitude of the ascending node; the angle between the normal vector of the orbit and the normal vector of the ecliptic plane, in radians in $[0, 2\pi)$.

per_y sidereal orbital period (in years).

class orbit classification. A factor with levels given below.

diameter diameter from equivalent sphere (in km).

ccf2009 flag indicating if the comet was considered in the data application in Cuesta-Albertos et al. (2009); see details below.

Details

The normal vector to the ecliptic plane of the comet with inclination i and longitude of the ascending node ω is

$$(\sin(i)\sin(\omega), -\sin(i)\cos(\omega), \cos(i))'.$$

A prograde comet has positive $\cos(i)$, negative $\cos(i)$ represents a retrograde comet.

`class` has the following levels:

- `COM`: comet orbit not matching any defined orbit class.
- `CTc`: Chiron-type comet, as defined by Levison and Duncan (T_Jupiter > 3; a > a_Jupiter).
- `ETc`: Encke-type comet, as defined by Levison and Duncan (T_Jupiter > 3; a < a_Jupiter).
- `HTC`: Halley-type comet, classical definition (20y < P < 200y).
- `HYP`: comets on hyperbolic orbits.
- `JFc`: Jupiter-family comet, as defined by Levison and Duncan (2 < T_Jupiter < 3).
- `JFC`: Jupiter-family comet, classical definition (P < 20y).
- `PAR`: comets on parabolic orbits.

Hyperbolic and parabolic comets are not periodic; only elliptical comets are periodic.

The `ccf2009` variable gives the observations considered in Cuesta-Albertos et al. (2009) after fetching in the database in 2007-12-14 for the comets such that `!(class %in% c("HYP","PAR")) & per_y >= 200 & !numbered`. A periodic comet is `numbered` by the IUA only after its second perihelion passage, and then its `id` starts with `c`. Due to the dynamic nature of the data, more comets were added to the database since 2007 and also some observations were updated.

The script performing the data preprocessing is available at `comets.R`. The data was retrieved on 2020-05-07.

Source

https://ssd.jpl.nasa.gov/sbdb_query.cgi

References

Cuesta-Albertos, J. A., Cuevas, A., Fraiman, R. (2009) On projection-based tests for directional and compositional data. *Statistics and Computing*, 19:367–380. doi: 10.1007/s1122200890983

Examples

```
# Load data
data("comets")

# Add normal vectors
comets$normal <- cbind(sin(comets$i) * sin(comets$om),
                       -sin(comets$i) * cos(comets$om),
                       cos(comets$i))

# Add numbered information
comets$numbered <- substr(comets$id, 1, 1) == "c"
```

```
# Tests to be performed
type_tests <- c("PCvM", "PAD", "PRt")

# Excluding the C/1882 R1-X (Great September comet) records with X = B, C, D
comets_ccf2009 <- comets[comets$ccf2009, ][-c(13:15), ]

# Sample size
nrow(comets_ccf2009)

# Tests for the data in Cuesta-Albertos et al. (2009)
tests_ccf2009 <- unif_test(data = comets_ccf2009$normal, type = type_tests,
                           p_value = "asymp")
tests_ccf2009
```

`craters` *Craters named by the IUA*

Description

Named craters of the Solar System by the Gazetteer of Planetary Nomenclature of the International Astronomical Union (IUA).

Usage

```
craters
```

Format

A data frame with 5235 rows and 7 variables:

ID database ID.

name name of the crater.

target name of the celestial body. A factor with 43 levels, such as `"Moon"`, `"Venus"`, or `"Europa"`.

target_type type of celestial body. A factor with 3 levels: `"Planet"`, `"Moon"`, `"Dwarf planet"`, or `"Asteroid"`.

diameter diameter of the crater (in km).

theta longitude angle $\theta \in [0, 2\pi)$ of the crater center.

phi latitude angle $\phi \in [-\pi/2, \pi/2]$ of the crater center.

Details

"Craters" are understood in the Gazetteer of Planetary Nomenclature as roughly circular depressions resulting from impact or volcanic activity (the geological origin is unspecified).

Be aware that the dataset only contains *named* craters by the IUA. Therefore, there is likely a **high uniform bias** on the distribution of craters. Presumably the naming process attempts to cover the planet in a somehow uniform fashion (distant craters are more likely to be named than neighboring craters). Also, there are substantially more craters in the listed bodies than those named by the IUA. See venus and rhea for more detailed and specific crater datasets.

The (θ, ϕ) angles are such their associated planetocentric coordinates are:

$$(\cos(\phi)\cos(\theta), \cos(\phi)\sin(\theta), \sin(\phi))',$$

with $(0, 0, 1)'$ denoting the north pole.

The script performing the data preprocessing is available at craters.R. The data was retrieved on 2020-05-31.

Source

https://planetarynames.wr.usgs.gov/AdvancedSearch

Examples

```
# Load data
data("craters")

# Add Cartesian coordinates
craters$X <- cbind(cos(craters$theta) * cos(craters$phi),
                   sin(craters$theta) * cos(craters$phi),
                   sin(craters$phi))

# Tests to be performed
type_tests <- c("PCvM", "PAD", "PRt")

# Tests for Venus and Rhea
unif_test(data = craters$X[craters$target == "Venus", ], type = type_tests,
          p_value = "asymp")
unif_test(data = craters$X[craters$target == "Rhea", ], type = type_tests,
          p_value = "asymp")
```

F_from_f *Distribution and quantile functions from angular function*

Description

Numerical computation of the distribution function F and the quantile function F^{-1} for an angular function f in a tangent-normal decomposition. $F^{-1}(x)$ results from the inversion of

$$F(x) = \int_{-1}^{x} \omega_{p-1} c_f f(z)(1 - z^2)^{(p-3)/2} \,\mathrm{d}z$$

for $x \in [-1, 1]$, where c_f is a normalizing constant and ω_{p-1} is the surface area of S^{p-2}.

Usage

```
F_from_f(f, p, Gauss = TRUE, N = 320, K = 1000, tol = 1e-06, ...)

F_inv_from_f(f, p, Gauss = TRUE, N = 320, K = 1000, tol = 1e-06, ...)
```

Arguments

f	angular function defined on $[-1, 1]$. Must be vectorized.
p	integer giving the dimension of the ambient space R^p that contains S^{p-1}.
Gauss	use a Gauss–Legendre quadrature rule to integrate f with `N` nodes? Otherwise, rely on `integrate` Defaults to `TRUE`.
N	number of points used in the Gauss–Legendre quadrature. Defaults to `320`.
K	number of equispaced points on $[-1, 1]$ used for evaluating F^{-1} and then interpolating. Defaults to `1e3`.
tol	tolerance passed to `uniroot` for the inversion of F. Also, passed to `integrate`'s `rel.tol` and `abs.tol` if `Gauss = FALSE`. Defaults to `1e-6`.
...	further parameters passed to `f`.

Details

The normalizing constant c_f is such that $F(1) = 1$. It does not need to be part of `f` as it is computed internally.

Interpolation is performed by a monotone cubic spline. `Gauss = TRUE` yields more accurate results, at expenses of a heavier computation.

If `f` yields negative values, these are silently truncated to zero.

Value

A `splinefun` object ready to evaluate F or F^{-1}, as specified.

Examples

```
f <- function(x) rep(1, length(x))
plot(F_from_f(f = f, p = 4, Gauss = TRUE), ylab = "F(x)", xlim = c(-1, 1))
plot(F_from_f(f = f, p = 4, Gauss = FALSE), col = 2, add = TRUE,
     xlim = c(-1, 1))
curve(p_proj_unif(x = x, p = 4), col = 3, add = TRUE, n = 300)
plot(F_inv_from_f(f = f, p = 4, Gauss = TRUE), ylab = "F^{-1}(x)")
plot(F_inv_from_f(f = f, p = 4, Gauss = FALSE), col = 2, add = TRUE)
curve(q_proj_unif(u = x, p = 4), col = 3, add = TRUE, n = 300)
```

`Gauss_Legen` *Gauss–Legendre quadrature*

Description

Convenience for computing the nodes x_k and weights w_k of the *Gauss–Legendre* quadrature formula in (a, b):

$$\int_a^b f(x)w(x)\,\mathrm{d}x \approx \sum_{k=1}^{N} w_k f(x_k).$$

.

Usage

```
Gauss_Legen_nodes(a = -1, b = 1, N = 40L)

Gauss_Legen_weights(a = -1, b = 1, N = 40L)
```

Arguments

`a, b`	scalars giving the interval (a, b). Defaults to $(-1, 1)$.
`N`	number of points used in the Gauss–Legendre quadrature. The following choices are supported: `5`, `10`, `20`, `40`, `80`, `160`, `320`, `640`, `1280`, `2560`, and `5120`. Defaults to `40`.

Details

For C^∞ functions, Gauss–Legendre quadrature can be very efficient. It is exact for polynomials up to degree $2N - 1$.

The nodes and weights up to $N = 80$ were retrieved from NIST and have 10^{-21} precision. For $N = 160$ onwards, the nodes and weights were computed with the `gauss.quad` function from the `statmod` package (Smyth, 1998), and have 10^{-15} precision.

Value

A matrix of size `c(N,1)` with the nodes x_k (`Gauss_Legen_nodes`) or the corresponding weights w_k (`Gauss_Legen_weights`).

References

NIST Digital Library of Mathematical Functions. Release 1.0.20 of 2018-09-15. F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller, and B. V. Saunders, eds. `https://dlmf.nist.gov/`

Smyth, G. K. (1998). Numerical integration. In: *Encyclopedia of Biostatistics*, P. Armitage and T. Colton (eds.), Wiley, London, pp. 3088-3095.

Examples

```
## Integration of a smooth function in (1, 10)

# Weights and nodes for integrating
x_k <- Gauss_Legen_nodes(a = 1, b = 10, N = 40)
w_k <- Gauss_Legen_weights(a = 1, b = 10, N = 40)

# Check quadrature
f <- function(x) sin(x) * x^2 - log(x + 1)
integrate(f, lower = 1, upper = 10, rel.tol = 1e-12)
sum(w_k * f(x_k))

# Exact for polynomials up to degree 2 * N - 1
f <- function(x) (((x + 0.5) / 1e3)^5 - ((x - 0.5)/ 5)^4 +
  ((x - 0.25) / 10)^2 + 1)^20
```

```
sum(w_k * f(x_k))
integrate(f, lower = -1, upper = 1, rel.tol = 1e-12)

## Integration on (0, pi)

# Weights and nodes for integrating
th_k <- Gauss_Legen_nodes(a = 0, b = pi, N = 40)
w_k <- Gauss_Legen_weights(a = 0, b = pi, N = 40)

# Check quadrature
p <- 4
psi <- function(th) -sin(th / 2)
w <- function(th) sin(th)^(p - 2)
integrate(function(th) psi(th) * w(th), lower = 0, upper = pi,
          rel.tol = 1e-12)
sum(w_k * psi(th_k) * w(th_k))

# Integral with Gegenbauer polynomial
k <- 3
C_k <- function(th) drop(Gegen_polyn(theta = th, k = k, p = p))
integrate(function(th) psi(th) * C_k(th) * w(th), lower = 0, upper = pi,
          rel.tol = 1e-12)
th_k <- drop(Gauss_Legen_nodes(a = 0, b = pi, N = 80))
w_k <- drop(Gauss_Legen_weights(a = 0, b = pi, N = 80))
sum(w_k * psi(th_k) * C_k(th_k) * w(th_k))
```

`Gegenbauer` *Gegenbauer polynomials and coefficients*

Description

The Gegenbauer polynomials $\{C_k^{(\lambda)}(x)\}_{k=0}^\infty$ form a family of orthogonal polynomials on the interval $[-1, 1]$ with respect to the weight function $(1 - x^2)^{\lambda - 1/2}$, for $\lambda > -1/2$, $\lambda \neq 0$. They usually appear when dealing with functions defined on $S^{p-1} := \{\mathbf{x} \in R^p : ||\mathbf{x}|| = 1\}$ with index $\lambda = p/2 - 1$.

The Gegenbauer polynomials are somehow simpler to evaluate for $x = \cos(\theta)$, with $\theta \in [0, \pi]$. This simplifies also the connection with the Chebyshev polynomials $\{T_k(x)\}_{k=0}^\infty$, which admit the explicit expression $T_k(\cos(\theta)) = \cos(k\theta)$. The Chebyshev polynomials appear as the limit of the Gegenbauer polynomials (divided by λ) when λ goes to 0, so they can be regarded as the extension by continuity of $\{C_k^{(p/2-1)}(x)\}_{k=0}^\infty$ to the case $p = 2$.

For a reasonably smooth function ψ defined on $[0, \pi]$,

$$\psi(\theta) = \sum_{k=0}^\infty b_{k,p} C_k^{(p/2-1)}(\cos(\theta)),$$

provided that the coefficients

$$b_{k,p} := \frac{1}{c_{k,p}} \int_0^\pi \psi(\theta) C_k^{(p/2-1)}(\cos(\theta))(\sin(\theta))^{p-2}\,\mathrm{d}\theta$$

are finite, where the normalizing constants are

$$c_{k,p} := \int_0^\pi (C_k^{(p/2-1)}(\cos(\theta)))^2 (\sin(\theta))^{p-2}\, \mathrm{d}\theta.$$

The (squared) "Gegenbauer norm" of ψ is

$$\|\psi\|_{G,p}^2 := \int_0^\pi \psi(\theta)^2 C_k^{(p/2-1)}(\cos(\theta))(\sin(\theta))^{p-2}\, \mathrm{d}\theta.$$

The previous expansion can be generalized for a 2-dimensional function ψ defined on $[0,\pi]\times[0,\pi]$:

$$\psi(\theta_1,\theta_2) = \sum_{k=0}^\infty \sum_{m=0}^\infty b_{k,m,p} C_k^{(p/2-1)}(\cos(\theta_1)) C_k^{(p/2-1)}(\cos(\theta_2)),$$

with coefficients

$$b_{k,m,p} := \frac{1}{c_{k,p}c_{m,p}} \int_0^\pi \int_0^\pi \psi(\theta_1,\theta_2) C_k^{(p/2-1)}(\cos(\theta_1)) C_k^{(p/2-1)}(\cos(\theta_2))(\sin(\theta_1))^{p-2}(\sin(\theta_2))^{p-2}\, \mathrm{d}\theta_1\, \mathrm{d}\theta_2.$$

The (squared) "Gegenbauer norm" of ψ is

$$\|\psi\|_{G,p}^2 := \int_0^\pi \int_0^\pi \psi(\theta_1,\theta_2)^2 C_k^{(p/2-1)}(\cos(\theta_1)) C_k^{(p/2-1)}(\cos(\theta_2))(\sin(\theta_1))^{p-2}(\sin(\theta_2))^{p-2}\, \mathrm{d}\theta_1\, \mathrm{d}\theta_2.$$

Usage

```
Gegen_polyn(theta, k, p)

Gegen_coefs(k, p, psi, Gauss = TRUE, N = 320, normalize = TRUE,
  only_const = FALSE, tol = 1e-06, ...)

Gegen_series(theta, coefs, k, p, normalize = TRUE)

Gegen_norm(coefs, k, p, normalize = TRUE, cumulative = FALSE)

Gegen_polyn_2d(theta_1, theta_2, k, m, p)

Gegen_coefs_2d(k, m, p, psi, Gauss = TRUE, N = 320, normalize = TRUE,
  only_const = FALSE, tol = 1e-06, ...)

Gegen_series_2d(theta_1, theta_2, coefs, k, m, p, normalize = TRUE)

Gegen_norm_2d(coefs, k, m, p, normalize = TRUE)
```

Arguments

`theta, theta_1, theta_2`
: vectors with values in $[0,\pi]$.

`k, m`
: vectors with the orders of the Gegenbauer polynomials. Must be integers larger or equal than `0`.

`p`	integer giving the dimension of the ambient space R^p that contains S^{p-1}.
`psi`	function defined in $[0, \pi]$ and whose Gegenbauer coefficients are to be computed. Must be vectorized. For `Gegen_coefs_2d`, it must return a matrix of size `c(length(theta_1),length(theta_2))`.
`Gauss`	use a Gauss–Legendre quadrature rule of `N` nodes in the computation of the Gegenbauer coefficients? Otherwise, call `integrate`. Defaults to `TRUE`.
`N`	number of points used in the Gauss–Legendre quadrature for computing the Gegenbauer coefficients. Defaults to `320`.
`normalize`	consider normalized coefficients (divided by $c_{k,p}$)? Defaults to `TRUE`.
`only_const`	return only the normalizing constants $c_{k,p}$? Defaults to `FALSE`.
`tol`	tolerance passed to `integrate`'s `rel.tol` and `abs.tol` if `Gauss = FALSE`. Defaults to `1e-6`.
`...`	further arguments to be passed to `psi`.
`coefs`	for `Gegen_series` and `Gegen_norm`, a vector of coefficients $b_{k,p}$ with length `length(k)`. For `Gegen_series_2d` and `Gegen_norm_2d`, a matrix of coefficients $b_{k,m,p}$ with size `c(length(k),length(m))`. The order of the coefficients is given by `k` and `m`.
`cumulative`	return the cumulative norm for increasing truncation of the series? Defaults to `FALSE`.

Details

The `Gegen_polyn` function is a wrapper to the functions `gegenpoly_n` and `gegenpoly_array` in the `gsl-package`, which they interface the functions defined in the header file `gsl_sf_gegenbauer.h` (documented here) of the GNU Scientific Library.

Note that the function `Gegen_polyn` computes the regular *unnormalized* Gegenbauer polynomials.

For the case $p = 2$, the Chebyshev polynomials are considered.

Value

- `Gegen_polyn`: a matrix of size `c(length(theta),length(k))` containing the evaluation of the `length(k)` Gegenbauer polynomials at `theta`.
- `Gegen_coefs`: a vector of size `length(k)` containing the coefficients $b_{k,p}$.
- `Gegen_series`: the evaluation of the truncated series expansion, a vector of size `length(theta)`.
- `Gegen_norm`: the Gegenbauer norm of the truncated series, a scalar if `cumulative = FALSE`, otherwise a vector of size `length(k)`.
- `Gegen_polyn_2d`: a 4-dimensional array of size `c(length(theta_1),length(theta_2),length(k),length(m))` containing the evaluation of the `length(k) * length(m)` 2-dimensional Gegenbauer polynomials at the bivariate grid spanned by `theta_1` and `theta_2`.
- `Gegen_coefs_2d`: a matrix of size `c(length(k),length(m))` containing the coefficients $b_{k,m,p}$.
- `Gegen_series_2d`: the evaluation of the truncated series expansion, a matrix of size `c(length(theta_1),length(the`
- `Gegen_norm_2d`: the 2-dimensional Gegenbauer norm of the truncated series, a scalar.

References

Galassi, M., Davies, J., Theiler, J., Gough, B., Jungman, G., Alken, P., Booth, M., and Rossi, F. (2009) *GNU Scientific Library Reference Manual*. Network Theory Ltd. http://www.gnu.org/software/gsl/

NIST Digital Library of Mathematical Functions. Release 1.0.20 of 2018-09-15. F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller, and B. V. Saunders, eds. https://dlmf.nist.gov/

Examples

```
## Representation of Gegenbauer polynomials (Chebyshev polynomials for p = 2)

th <- seq(0, pi, l = 500)
k <- 0:3
old_par <- par(mfrow = c(2, 2))
for (p in 2:5) {
  matplot(th, t(Gegen_polyn(theta = th, k = k, p = p)), lty = 1,
          type = "l", main = substitute(p == d, list(d = p)),
          axes = FALSE, xlab = expression(theta), ylab = "")
  axis(1, at = c(0, pi / 4, pi / 2, 3 * pi / 4, pi),
       labels = expression(0, pi / 4, pi / 2, 3 * pi / 4, pi))
  axis(2); box()
  mtext(text = expression({C[k]^{p/2 - 1}}(cos(theta))), side = 2,
        line = 2, cex = 0.75)
  legend("bottomleft", legend = paste("k =", k), lwd = 2, col = seq_along(k))
}
par(old_par)

## Coefficients and series in p = 2

# Function in [0, pi] to be projected in Chebyshev polynomials
psi <- function(th) -sin(th / 2)

# Coefficients
p <- 2
k <- 0:4
(coefs <- Gegen_coefs(k = k, p = p, psi = psi))

# Series
plot(th, psi(th), type = "l", axes = FALSE, xlab = expression(theta),
      ylab = "", ylim = c(-1.25, 0))
axis(1, at = c(0, pi / 4, pi / 2, 3 * pi / 4, pi),
     labels = expression(0, pi / 4, pi / 2, 3 * pi / 4, pi))
axis(2); box()
col <- viridisLite::viridis(length(coefs))
for (i in seq_along(coefs)) {
  lines(th, Gegen_series(theta = th, coefs = coefs[1:(i + 1)], k = 0:i,
                         p = p), col = col[i])
}
lines(th, psi(th), lwd = 2)
```

```
## Coefficients and series in p = 3

# Function in [0, pi] to be projected in Gegenbauer polynomials
psi <- function(th) tan(th / 3)

# Coefficients
p <- 3
k <- 0:10
(coefs <- Gegen_coefs(k = k, p = p, psi = psi))

# Series
plot(th, psi(th), type = "l", axes = FALSE, xlab = expression(theta),
      ylab = "", ylim = c(0, 2))
axis(1, at = c(0, pi / 4, pi / 2, 3 * pi / 4, pi),
     labels = expression(0, pi / 4, pi / 2, 3 * pi / 4, pi))
axis(2); box()
col <- viridisLite::viridis(length(coefs))
for (i in seq_along(coefs)) {
  lines(th, Gegen_series(theta = th, coefs = coefs[1:(i + 1)], k = 0:i,
                       p = p), col = col[i])
}
lines(th, psi(th), lwd = 2)

## Surface representation

# Surface in [0, pi]^2 to be projected in Gegenbauer polynomials
p <- 3
psi <- function(th_1, th_2) A_theta_x(theta = th_1, x = cos(th_2),
                                      p = p, as_matrix = TRUE)

# Coefficients
k <- 0:20
m <- 0:10
coefs <- Gegen_coefs_2d(k = k, m = m, p = p, psi = psi)

# Series
th <- seq(0, pi, l = 100)
col <- viridisLite::viridis(20)
old_par <- par(mfrow = c(2, 2))
image(th, th, A_theta_x(theta = th, x = cos(th), p = p), axes = FALSE,
      col = col, zlim = c(0, 1), xlab = expression(theta[1]),
      ylab = expression(theta[2]), main = "Original")
axis(1, at = c(0, pi / 4, pi / 2, 3 * pi / 4, pi),
     labels = expression(0, pi / 4, pi / 2, 3 * pi / 4, pi))
axis(2, at = c(0, pi / 4, pi / 2, 3 * pi / 4, pi),
     labels = expression(0, pi / 4, pi / 2, 3 * pi / 4, pi))
box()
for(K in c(5, 10, 20)) {
  A <- Gegen_series_2d(theta_1 = th, theta_2 = th,
                       coefs = coefs[1:(K + 1), ], k = 0:K, m = m, p = p)
  image(th, th, A, axes = FALSE, col = col, zlim = c(0, 1),
        xlab = expression(theta[1]), ylab = expression(theta[2]),
        main = paste(K, "x", m[length(m)], "coefficients"))
```

```
  axis(1, at = c(0, pi / 4, pi / 2, 3 * pi / 4, pi),
       labels = expression(0, pi / 4, pi / 2, 3 * pi / 4, pi))
  axis(2, at = c(0, pi / 4, pi / 2, 3 * pi / 4, pi),
       labels = expression(0, pi / 4, pi / 2, 3 * pi / 4, pi))
  box()
}
par(old_par)
```

---

harmonics *(Hyper)spherical harmonics*

Description

Computation of a certain explicit representation of (hyper)spherical harmonics on $S^{p-1} := \{\mathbf{x} \in R^p : ||\mathbf{x}|| = 1\}$, $p \geq 2$. Details are available in García-Portugués et al. (2021).

Usage

```
g_i_k(x, i = 1, k = 1, m = NULL, show_m = FALSE)
```

Arguments

x	locations in S^{p-1} to evaluate $g_{i,k}$. Either a matrix of size c(nx,p) or a vector of size p. Normalized internally if required (with a warning message).
i, k	alternative indexing to refer to the i-th (hyper)spherical harmonic of order k. i is a positive integer smaller than d_p_k and k is a non-negative integer.
m	(hyper)spherical harmonic index, as used in Proposition 2.1. The index is computed internally from i and k. Defaults to NULL.
show_m	flag to print m if computed internally when m = NULL.

Details

The implementation uses Proposition 2.1 in García-Portugués et al. (2021), which adapts Theorem 1.5.1 in Dai and Xu (2013) with the correction of typos in the normalizing constant h_α and in the definition of the function g_α of the latter theorem.

Value

A vector of size nrow(x).

References

Dai, F. and Xu, Y. (2013). *Approximation Theory and Harmonic Analysis on Spheres and Balls*. Springer, New York. doi: 10.1007/9781461466604

García-Portugués, E., Paindaveine, D., and Verdebout, T. (2021). On the power of Sobolev tests for isotropy under local rotationally symmetric alternatives. *arXiv:2108.09874*. https://arxiv.org/abs/2108.09874

Examples

```
n <- 3e3
old_par <- par(mfrow = c(2, 3))
k <- 2
for (i in 1:d_p_k(p = 3, k = k)) {
  X <- r_unif_sph(n = n, p = 3, M = 1)[, , 1]
  col <- rainbow(n)[rank(g_i_k(x = X, k = k, i = i, show_m = TRUE))]
  scatterplot3d::scatterplot3d(X[, 1], X[, 2], X[, 3], color = col,
                               axis = FALSE, pch = 19)
}
for (k in 0:5) {
  X <- r_unif_sph(n = n, p = 3, M = 1)[, , 1]
  col <- rainbow(n)[rank(g_i_k(x = X, k = k, i = 1, show_m = TRUE))]
  scatterplot3d::scatterplot3d(X[, 1], X[, 2], X[, 3], color = col,
                               axis = FALSE, pch = 19)
}
par(old_par)
```

`int_sph_MC` *Monte Carlo integration of functions on the (hyper)sphere*

Description

Monte Carlo approximation of the integral

$$\int_{S^{p-1}} f(x)\,\mathrm{d}x$$

of a function $f : S^{p-1} \to R$ defined on the (hyper)sphere $S^{p-1} := \{\mathbf{x} \in R^p : ||\mathbf{x}|| = 1\}$, $p \geq 2$.

Usage

```
int_sph_MC(f, p, M = 10000, cores = 1, chunks = ceiling(M/1000),
  seeds = NULL, ...)
```

Arguments

`f`	function to be integrated. Its first argument must be the (hyper)sphere position. Must be vectorized and return a vector of size `nrow(x)` for a matrix input `x`. See examples.
`p`	integer giving the dimension of the ambient space R^p that contains S^{p-1}.
`M`	number of Monte Carlo samples. Defaults to `1e4`.
`cores`	number of cores to perform the integration. Defaults to `1`.
`chunks`	number of chunks to split the `M` Monte Carlo samples. Useful for parallelizing the integration in `chunks` tasks containing `ceiling(M / chunks)` replications. Useful also for avoiding memory bottlenecks when `M` is large. Defaults to `ceiling(M / 1e3)`.

`seeds` if provided, a vector of size `chunks` for fixing the seeds on each of the simulation chunks (useful for reproducing parallel simulations). Specifically, for `k in 1:chunks`, seeds are set as `set.seed(seeds[k],kind = "Mersenne-Twister")` in each chunk. Defaults to `NULL` (no seed setting is done).

`...` optional arguments to be passed to `f` or to `foreach` (for example, `.export` to export global variables or other functions to the `foreach` environment).

Details

It is possible to have a progress bar if `int_sph_MC` is wrapped with `progressr::with_progress` or if `progressr::handlers(global = TRUE)` is invoked (once) by the user. See the examples below. The progress bar is updated with the number of finished chunks.

Value

A scalar with the approximate integral.

Examples

```
## Sequential simulation

# Vectorized functions to be integrated
x1 <- function(x) x[, 1]
quad <- function(x, a = 0) a + rowSums(x^4)

# Approximate \int_{S^{p-1}} x_1 dx = 0
int_sph_MC(f = x1, p = 3, M = 1e4, chunks = 2)

# Approximate \int_{S^{p-1}} (a + \sum_i x_i^4) dx
int_sph_MC(f = quad, p = 2, M = 1e4, a = 0, chunks = 2)

# Compare with Gauss--Legendre integration on S^2
th_k <- Gauss_Legen_nodes(a = 0, b = 2 * pi, N = 40)
w_k <- Gauss_Legen_weights(a = 0, b = 2 * pi, N = 40)
sum(w_k * quad(cbind(cos(th_k), sin(th_k)), a = 1))

## Parallel simulation with a progress bar

# Define a progress bar
require(progress)
require(progressr)
handlers(handler_progress(
  format = ":spin [:bar] :percent Total: :elapsedfull End \u2248 :eta",
  clear = FALSE))

# Call int_sph_MC() within with_progress()
with_progress(int_sph_MC(f = x1, p = 3, cores = 2, M = 1e5, chunks = 100))

# Instead of using with_progress() each time, it is more practical to run
# handlers(global = TRUE)
# once to activate progress bars in your R session
```

`locdev` *Local projected alternatives to uniformity*

Description

Density and random generation for local projected alternatives to uniformity with densities

$$f_{\kappa,\boldsymbol{\mu}}(\mathbf{x}) := \frac{1-\kappa}{\omega_p} + \kappa f(\mathbf{x}'\boldsymbol{\mu})$$

where

$$f(z) = \frac{1}{\omega_p}\left\{1 + \sum_{k=1}^{\infty} u_{k,p} C_k^{p/2-1}(z)\right\}$$

is the *angular function* controlling the local alternative in a Gegenbauer series, $0 \leq \kappa \leq 1$, $\boldsymbol{\mu}$ is a direction on S^{p-1}, and ω_p is the surface area of S^{p-1}. The sequence $\{u_{k,p}\}$ is typically such that $u_{k,p} = \left(1 + \frac{2k}{p-2}\right) b_{k,p}$ for the Gegenbauer coefficients $\{b_{k,p}\}$ of the kernel function of a Sobolev statistic (see the transformation between the coefficients $u_{k,p}$ and $b_{k,p}$).

Also, automatic truncation of the series $\sum_{k=1}^{\infty} u_{k,p} C_k^{p/2-1}(z)$ according to the proportion of "Gegenbauer norm" explained.

Usage

```
f_locdev(z, p, uk)

con_f(f, p, N = 320)

d_locdev(x, mu, f, kappa)

r_locdev(n, mu, f, kappa, F_inv = NULL, ...)

cutoff_locdev(p, K_max = 10000, thre = 0.001, type, Rothman_t = 1/3,
  Pycke_q = 0.5, verbose = FALSE, Gauss = TRUE, N = 320, tol = 1e-06)
```

Arguments

`z`	projected evaluation points for f, a vector with entries on $[-1, 1]$.
`p`	integer giving the dimension of the ambient space R^p that contains S^{p-1}.
`uk`	coefficients $u_{k,p}$ associated to the indexes `1:length(uk)`, a vector.
`f`	angular function defined on $[-1, 1]$. Must be vectorized.
`N`	number of points used in the Gauss–Legendre quadrature for computing the Gegenbauer coefficients. Defaults to `320`.
`x`	locations in S^{p-1} to evaluate the density. Either a matrix of size `c(nx,p)` or a vector of length `p`. Normalized internally if required (with a `warning` message).
`mu`	a unit norm vector of size `p` giving the axis of rotational symmetry.

kappa	the strength of the local alternative, between 0 and 1.
n	sample size, a positive integer.
F_inv	quantile function associated to f. Computed by F_inv_from_f if NULL (default).
...	further parameters passed to F_inv_from_f.
K_max	integer giving the truncation of the series. Defaults to 1e4.
thre	proportion of norm *not* explained by the first terms of the truncated series. Defaults to 1e-3.
type	Sobolev statistic. For $p = 2$, either "Watson", "Rothman", "Pycke_q", or "Hermans_Rasson". For $p \geq 2$, "Ajne", "Gine_Gn", "Gine_Fn", "Bakshaev", "Riesz", "PCvM", "PAD", or "PRt".
Rothman_t	t parameter for the Rothman test, a real in $(0, 1)$. Defaults to 1 / 3.
Pycke_q	q parameter for the Pycke "q-test", a real in $(0, 1)$. Defaults to 1 / 2.
verbose	output information about the truncation (TRUE or 1) and a diagnostic plot (2)? Defaults to FALSE.
Gauss	use a Gauss–Legendre quadrature rule of N nodes in the computation of the Gegenbauer coefficients? Otherwise, call integrate. Defaults to TRUE.
tol	tolerance passed to integrate's rel.tol and abs.tol if Gauss = FALSE. Defaults to 1e-6.

Details

See the definitions of local alternatives in Prentice (1978) and in García-Portugués et al. (2020).

The truncation of $\sum_{k=1}^\infty u_{k,p} C_k^{p/2-1}(z)$ is done to the first K_max terms and then up to the index such that the first terms leave unexplained the proportion thre of the norm of the whole series. Setting thre = 0 truncates to K_max terms exactly. If the series only contains odd or even non-zero terms, then only K_max / 2 addends are *effectively* taken into account in the first truncation.

Value

- f_locdev: angular function evaluated at x, a vector.
- con_f: normalizing constant c_f of f, a scalar.
- d_locdev: density function evaluated at x, a vector.
- r_locdev: a matrix of size c(n,p) containing a random sample from the density $f_{\kappa,\boldsymbol{\mu}}$.
- cutoff_locdev: vector of coefficients $\{u_{k,p}\}$ automatically truncated according to K_max and thre (see details).

References

García-Portugués, E., Navarro-Esteban, P., Cuesta-Albertos, J. A. (2020) On a projection-based class of uniformity tests on the hypersphere. *arXiv:2008.09897*. https://arxiv.org/abs/2008.09897

Prentice, M. J. (1978). On invariant tests of uniformity for directions and orientations. *The Annals of Statistics*, 6(1):169–176. doi: 10.1214/aos/1176344075

Examples

```
## Local alternatives diagnostics

loc_alt_diagnostic  <- function(p, type, thre = 1e-3, K_max = 1e3) {

  # Coefficients of the alternative
  uk <- cutoff_locdev(K_max = K_max, p = p, type = type, thre = thre,
                      N = 640)

  old_par <- par(mfrow = c(2, 2))

  # Construction of f
  z <- seq(-1, 1, l = 1e3)
  f <- function(z) f_locdev(z = z, p = p, uk = uk)
  plot(z, f(z), type = "l", xlab = expression(z), ylab = expression(f(z)),
       main = paste0("Local alternative f, ", type, ", p = ", p), log = "y")

  # Projected density on [-1, 1]
  f_proj <- function(z) rotasym::w_p(p = p - 1) * f(z) *
    (1 - z^2)^((p - 3) / 2)
  plot(z, f_proj(z), type = "l", xlab = expression(z),
       ylab = expression(omega[p - 1] * f(z) * (1 - z^2)^{(p - 3) / 2}),
       main = paste0("Projected density, ", type, ", p = ", p), log = "y",
       sub = paste("Integral:", round(con_f(f = f, p = p), 4)))

  # Quantile function for projected density
  mu <- c(rep(0, p - 1), 1)
  F_inv <- F_inv_from_f(f = f, p = p, K = 5e2)
  plot(F_inv, xlab = expression(x), ylab = expression(F^{-1}*(x)),
       main = paste0("Quantile function, ", type, ", p = ", p))

  # Sample from the alternative and plot the projected sample
  n <- 5e4
  samp <- r_locdev(n = n, mu = mu, f = f, kappa = 1, F_inv = F_inv)
  plot(z, f_proj(z), col = 2, type = "l",
       main = paste0("Simulated projected data, ", type, ", p = ", p),
       ylim = c(0, 1.75))
  hist(samp %*% mu, freq = FALSE, breaks = seq(-1, 1, l = 50), add = TRUE)

  par(old_par)

}

## Local alternatives for the PCvM test

loc_alt_diagnostic(p = 2, type = "PCvM")
loc_alt_diagnostic(p = 3, type = "PCvM")
loc_alt_diagnostic(p = 4, type = "PCvM")
loc_alt_diagnostic(p = 5, type = "PCvM")
loc_alt_diagnostic(p = 11, type = "PCvM")

## Local alternatives for the PAD test
```

```
loc_alt_diagnostic(p = 2, type = "PAD")
loc_alt_diagnostic(p = 3, type = "PAD")
loc_alt_diagnostic(p = 4, type = "PAD")
loc_alt_diagnostic(p = 5, type = "PAD")
loc_alt_diagnostic(p = 11, type = "PAD")

## Local alternatives for the PRt test

loc_alt_diagnostic(p = 2, type = "PRt")
loc_alt_diagnostic(p = 3, type = "PRt")
loc_alt_diagnostic(p = 4, type = "PRt")
loc_alt_diagnostic(p = 5, type = "PRt")
loc_alt_diagnostic(p = 11, type = "PRt")
```

`planets` *Planet orbits*

Description

Planet orbits data from the JPL Keplerian Elements for Approximate Positions of the Major Planets. The normal vector of a planet orbit represents is a vector on S^2.

Usage

```
planets
```

Format

A data frame with 9 rows and 3 variables:

planet names of the planets and Pluto.

i inclination; the orbit angle with respect to the ecliptic plane, in radians in $[0, \pi]$.

om longitude of the ascending node; the angle between the normal vector of the orbit and the normal vector of the ecliptic plane, in radians in $[0, 2\pi)$.

Details

The normal vector to the ecliptic plane of the planet with inclination i and longitude of the ascending node ω is

$$(\sin(i)\sin(\omega), -\sin(i)\cos(\omega), \cos(i))'.$$

The script performing the data preprocessing is available at `planets.R`. The data was retrieved on 2020-05-16.

Source

Table 2b in `https://ssd.jpl.nasa.gov/txt/aprx_pos_planets.pdf`

Examples

```
# Load data
data("planets")

# Add normal vectors
planets$normal <- cbind(sin(planets$i) * sin(planets$om),
                        -sin(planets$i) * cos(planets$om),
                        cos(planets$i))

# Tests to be performed
type_tests <- c("PCvM", "PAD", "PRt")

# Tests with Pluto
unif_test(data = planets$normal, type = type_tests, p_value = "MC")

# Tests without Pluto
unif_test(data = planets$normal[-9, ], type = type_tests, p_value = "MC")
```

Pn *Utilities for projected-ecdf statistics of spherical uniformity*

Description

Computation of the kernels

$$\psi_p^W(\theta) := \int_{-1}^{1} A_x(\theta)\,\mathrm{d}W(F_p(x)),$$

where $A_x(\theta)$ is the proportion of area surface of S^{p-1} covered by the intersection of two hyper-spherical caps with common solid angle $\pi - \cos^{-1}(x)$ and centers separated by an angle $\theta \in [0, \pi]$, F_p is the distribution function of the projected spherical uniform distribution, and W is a measure on $[0, 1]$.

Also, computation of the Gegenbauer coefficients of ψ_p^W:

$$b_{k,p}^W := \frac{1}{c_{k,p}} \int_0^\pi \psi_p^W(\theta) C_k^{p/2-1}(\cos\theta)\,\mathrm{d}\theta.$$

These coefficients can also be computed via

$$b_{k,p}^W = \int_{-1}^{1} a_{k,p}^x\,\mathrm{d}W(F_p(x))$$

for a certain function $x \to a_{k,p}^x$. They serve to define projected alternatives to uniformity.

Usage

```
psi_Pn(theta, q, type, Rothman_t = 1/3, tilde = FALSE, psi_Gauss = TRUE,
  psi_N = 320, tol = 1e-06)

Gegen_coefs_Pn(k, p, type, Rothman_t = 1/3, Gauss = TRUE, N = 320,
  tol = 1e-06, verbose = FALSE)

akx(x, p, k, sqr = FALSE)

f_locdev_Pn(p, type, K = 1000, N = 320, K_max = 10000, thre = 0.001,
  Rothman_t = 1/3, verbose = FALSE)
```

Arguments

`theta`	vectors with values in $[0, \pi]$.
`q`	integer giving the dimension of the sphere S^q.
`type`	type of projected-ecdf test statistic. Must be either `"PCvM"` (Cramér–von Mises), `"PAD"` (Anderson–Darling), or `"PRt"` (Rothman).
`Rothman_t`	t parameter for the Rothman test, a real in $(0, 1)$. Defaults to `1 / 3`.
`tilde`	include the constant and bias term? Defaults to `FALSE`.
`psi_Gauss`	use a Gauss–Legendre quadrature rule with `psi_N` nodes in the computation of the kernel function? Defaults to `TRUE`.
`psi_N`	number of points used in the Gauss–Legendre quadrature for computing the kernel function. Defaults to `320`.
`tol`	tolerance passed to `integrate`'s `rel.tol` and `abs.tol` if `Gauss = FALSE`. Defaults to `1e-6`.
`k`	vectors with the orders of the Gegenbauer polynomials. Must be integers larger or equal than `0`.
`p`	integer giving the dimension of the ambient space R^p that contains S^{p-1}.
`Gauss`	use a Gauss–Legendre quadrature rule of `N` nodes in the computation of the Gegenbauer coefficients? Otherwise, call `integrate`. Defaults to `TRUE`.
`N`	number of points used in the Gauss–Legendre quadrature for computing the Gegenbauer coefficients. Defaults to `320`.
`verbose`	flag to print informative messages. Defaults to `FALSE`.
`x`	evaluation points for $a_{k,p}^x$, a vector with values in $[-1, 1]$.
`sqr`	return the *signed* square root of $a_{k,p}^x$? Defaults to `FALSE`.
`K`	number of equispaced points on $[-1, 1]$ used for evaluating f and then interpolating. Defaults to `1e3`.
`K_max`	integer giving the truncation of the series. Defaults to `1e4`.
`thre`	proportion of norm *not* explained by the first terms of the truncated series. Defaults to `1e-3`.

Details

The evaluation of ψ_p^W and $b_{k,p}^W$ depends on the type of projected-ecdf statistic:

- PCvM: closed-form expressions for ψ_p^W and $b_{k,p}^W$ with $p = 2, 3, 4$, numerical integration required for $p \geq 5$.
- PAD: closed-form expressions for ψ_2^W and $b_{k,3}^W$, numerical integration required for ψ_p^W with $p \geq 3$ and $b_{k,p}^W$ with $p = 2$ and $p \geq 4$.
- PRt: closed-form expressions for ψ_p^W and $b_{k,p}^W$ for any $p \geq 2$.

See García-Portugués et al. (2020) for more details.

Value

- `psi_Pn`: a vector of size `length(theta)` with the evaluation of ψ.
- `Gegen_coefs_Pn`: a vector of size `length(k)` containing the coefficients $b_{k,p}^W$.
- `akx`: a matrix of size `c(length(x),length(k))` containing the coefficients $a_{k,p}^x$.
- `f_locdev_Pn`: the projected alternative f as a function ready to be evaluated.

Author(s)

Eduardo García-Portugués and Paula Navarro-Esteban.

References

García-Portugués, E., Navarro-Esteban, P., Cuesta-Albertos, J. A. (2020) On a projection-based class of uniformity tests on the hypersphere. *arXiv:2008.09897*. https://arxiv.org/abs/2008.09897

Examples

```
# Kernels in the projected-ecdf test statistics
k <- 0:10
coefs <- list()
(coefs$PCvM <- t(sapply(2:5, function(p)
  Gegen_coefs_Pn(k = k, p = p, type = "PCvM"))))
(coefs$PAD <- t(sapply(2:5, function(p)
  Gegen_coefs_Pn(k = k, p = p, type = "PAD"))))
(coefs$PRt <- t(sapply(2:5, function(p)
  Gegen_coefs_Pn(k = k, p = p, type = "PRt"))))

# Gegenbauer expansion
th <- seq(0, pi, length.out = 501)[-501]
old_par <- par(mfrow = c(3, 4))
for (type in c("PCvM", "PAD", "PRt")) {

  for (p in 2:5) {

    plot(th, psi_Pn(theta = th, q = p - 1, type = type), type = "l",
         main = paste0(type, ", p = ", p), xlab = expression(theta),
```

```
       ylab = expression(psi(theta)), axes = FALSE, ylim = c(-1.5, 1))
  axis(1, at = c(0, pi / 4, pi / 2, 3 * pi / 4, pi),
       labels = expression(0, pi / 4, pi / 2, 3 * pi / 4, pi))
  axis(2); box()
  lines(th, Gegen_series(theta = th, coefs = coefs[[type]][p - 1, ],
                         k = k, p = p), col = 2)

 }

}
par(old_par)

# Analytical coefficients vs. numerical integration
test_coef <- function(type, p, k = 0:20) {

  plot(k, log1p(abs(Gegen_coefs_Pn(k = k, p = p, type = type))),
       ylab = "Coefficients", main = paste0(type, ", p = ", p))
  points(k, log1p(abs(Gegen_coefs(k = k, p = p, psi = psi_Pn, type = type,
                                  q = p - 1))), col = 2)
  legend("topright", legend = c("log(1 + Gegen_coefs_Pn))",
                                "log(1 + Gegen_coefs(psi_Pn))"),
         lwd = 2, col = 1:2)

}

# PCvM statistic
old_par <- par(mfrow = c(2, 2))
for (p in 2:5) test_coef(type = "PCvM", p = p)
par(old_par)

# PAD statistic
old_par <- par(mfrow = c(2, 2))
for (p in 2:5) test_coef(type = "PAD", p = p)
par(old_par)

# PRt statistic
old_par <- par(mfrow = c(2, 2))
for (p in 2:5) test_coef(type = "PRt", p = p)
par(old_par)

# akx
akx(x = seq(-1, 1, l = 5), k = 1:4, p = 2)
akx(x = 0, k = 1:4, p = 3)

# PRt alternative to uniformity
z <- seq(-1, 1, l = 1e3)
p <- c(2:5, 10, 15, 17)
col <- viridisLite::viridis(length(p))
plot(z, f_locdev_Pn(p = p[1], type = "PRt")(z), type = "s",
     col = col[1], ylim = c(0, 0.6), ylab = expression(f[Rt](z)))
for (k in 2:length(p)) {
  lines(z, f_locdev_Pn(p = p[k], type = "PRt")(z), type = "s", col = col[k])
}
```

```
legend("topleft", legend = paste("p =", p), col = col, lwd = 2)
```

---

proj_unif *Projection of the spherical uniform distribution*

Description

Density, distribution, and quantile functions of the projection of the spherical uniform random variable on an arbitrary direction, that is, the random variable $\boldsymbol{\gamma}'\mathbf{X}$, where $\mathbf{X}$ is uniformly distributed on the (hyper)sphere $S^{p-1} := \{\mathbf{x} \in R^p : ||\mathbf{x}|| = 1\}$, $p \geq 2$, and $\boldsymbol{\gamma} \in S^{p-1}$ is an *arbitrary* projection direction. Note that the distribution is invariant to the choice of $\boldsymbol{\gamma}$. Also, efficient simulation of $\boldsymbol{\gamma}'\mathbf{X}$.

Usage

```
d_proj_unif(x, p, log = FALSE)

p_proj_unif(x, p, log = FALSE)

q_proj_unif(u, p)

r_proj_unif(n, p)
```

Arguments

x	a vector of size nx or a matrix of size c(nx,1).
p	integer giving the dimension of the ambient space R^p that contains S^{p-1}.
log	compute the logarithm of the density or distribution?
u	vector of probabilities.
n	sample size employed for computing the statistic.

Value

A matrix of size c(nx,1) with the evaluation of the density, distribution, or quantile function at x or u. For r_proj_unif, a random vector of size n.

Author(s)

Eduardo García-Portugués and Paula Navarro-Esteban.

Examples

```
# Density function
curve(d_proj_unif(x, p = 2), from = -2, to = 2, n = 2e2, ylim = c(0, 2))
curve(d_proj_unif(x, p = 3), n = 2e2, col = 2, add = TRUE)
curve(d_proj_unif(x, p = 4), n = 2e2, col = 3, add = TRUE)
curve(d_proj_unif(x, p = 5), n = 2e2, col = 4, add = TRUE)
curve(d_proj_unif(x, p = 6), n = 2e2, col = 5, add = TRUE)

# Distribution function
curve(p_proj_unif(x, p = 2), from = -2, to = 2, n = 2e2, ylim = c(0, 1))
curve(p_proj_unif(x, p = 3), n = 2e2, col = 2, add = TRUE)
curve(p_proj_unif(x, p = 4), n = 2e2, col = 3, add = TRUE)
curve(p_proj_unif(x, p = 5), n = 2e2, col = 4, add = TRUE)
curve(p_proj_unif(x, p = 6), n = 2e2, col = 5, add = TRUE)

# Quantile function
curve(q_proj_unif(u = x, p = 2), from = 0, to = 1, n = 2e2, ylim = c(-1, 1))
curve(q_proj_unif(u = x, p = 3), n = 2e2, col = 2, add = TRUE)
curve(q_proj_unif(u = x, p = 4), n = 2e2, col = 3, add = TRUE)
curve(q_proj_unif(u = x, p = 5), n = 2e2, col = 4, add = TRUE)
curve(q_proj_unif(u = x, p = 6), n = 2e2, col = 5, add = TRUE)

# Sampling
hist(r_proj_unif(n = 1e4, p = 4), freq = FALSE, breaks = 50)
curve(d_proj_unif(x, p = 4), n = 2e2, col = 3, add = TRUE)
```

Psi *Shortest angles matrix*

Description

Efficient computation of the shortest angles matrix $\mathbf{\Psi}$, defined as

$$\Psi_{ij} := \cos^{-1}(\mathbf{X}_i'\mathbf{X}_j), \quad i, j = 1, \ldots, n,$$

for a sample $\mathbf{X}_1, \ldots, \mathbf{X}_n \in S^{p-1} := \{\mathbf{x} \in R^p : ||\mathbf{x}|| = 1\}$, $p \geq 2$.

For a circular sample $\Theta_1, \ldots, \Theta_n \in [0, 2\pi)$, $\mathbf{\Psi}$ can be expressed as

$$\Psi_{ij} = \pi - |\pi - |\Theta_i - \Theta_j||, \quad i, j = 1, \ldots, n.$$

Usage

```
Psi_mat(data, ind_tri = 0L, use_ind_tri = FALSE, scalar_prod = FALSE,
  angles_diff = FALSE)

upper_tri_ind(n)
```

Arguments

`data`	an array of size `c(n,p,M)` containing the Cartesian coordinates of `M` samples of size `n` of directions on S^{p-1}. Alternatively if `p = 2`, an array of size `c(n,1,M)` containing the angles on $[0, 2\pi)$ of the `M` circular samples of size `n` on S^1. Must not contain `NA`'s.
`ind_tri`	if `use_ind_tri = TRUE`, the vector of 0-based indexes provided by `upper_tri_ind(n)`, which allows to extract the upper triangular part of the matrix $\mathbf{\Psi}$. See the examples.
`use_ind_tri`	use the already computed vector index `ind_tri`? If `FALSE` (default), `ind_tri` is computed internally.
`scalar_prod`	return the scalar products $\mathbf{X}_i'\mathbf{X}$ instead of the shortest angles? Only taken into account for data in *Cartesian* form. Defaults to `FALSE`.
`angles_diff`	return the (unwrapped) angles difference $\Theta_i - \Theta_j$ instead of the shortest angles? Only taken into account for data in *angular* form. Defaults to `FALSE`.
`n`	sample size, used to determine the index vector that gives the upper triangular part of $\mathbf{\Psi}$.

Value

- `Psi_mat`: a matrix of size `c(n * (n -1) / 2,M)` containing, for each column, the vector half of $\mathbf{\Psi}$ for each of the `M` samples.
- `upper_tri_ind`: a matrix of size `n * (n -1) / 2` containing the 0-based linear indexes for extracting the upper triangular matrix of a matrix of size `c(n,n)`, diagonal excluded, assuming column-major order.

Warning

Be careful on avoiding the next bad usages of `Psi_mat`, which will produce spurious results:

- The directions in `data` do *not* have unit norm when Cartesian coordinates are employed.
- The entries of `data` are *not* in $[0, 2\pi)$ when polar coordinates are employed.
- `ind_tri` is a vector of size `n * (n -1) / 2` that does *not* contain the indexes produced by `upper_tri_ind(n)`.

Examples

```
# Shortest angles
n <- 5
X <- r_unif_sph(n = n, p = 2, M = 2)
Theta <- X_to_Theta(X)
dim(Theta) <- c(n, 1, 2)
Psi_mat(X)
Psi_mat(Theta)

# Precompute ind_tri
ind_tri <- upper_tri_ind(n)
Psi_mat(X, ind_tri = ind_tri, use_ind_tri = TRUE)
```

```
# Compare with R
A <- acos(tcrossprod(X[, , 1]))
ind <- upper.tri(A)
A[ind]

# Reconstruct matrix
Psi_vec <- Psi_mat(Theta[, , 1, drop = FALSE])
Psi <- matrix(0, nrow = n, ncol = n)
Psi[upper.tri(Psi)] <- Psi_vec
Psi <- Psi + t(Psi)
```

---

p_Kolmogorov *Asymptotic distributions for circular uniformity statistics*

Description

Computation of the asymptotic null distributions of circular uniformity statistics.

Usage

```
p_Kolmogorov(x, K_Kolmogorov = 25L, alternating = TRUE)

d_Kolmogorov(x, K_Kolmogorov = 25L, alternating = TRUE)

p_cir_stat_Ajne(x, K_Ajne = 15L)

d_cir_stat_Ajne(x, K_Ajne = 15L)

p_cir_stat_Bingham(x)

d_cir_stat_Bingham(x)

p_cir_stat_Greenwood(x)

d_cir_stat_Greenwood(x)

p_cir_stat_Gini(x)

d_cir_stat_Gini(x)

p_cir_stat_Gini_squared(x)

d_cir_stat_Gini_squared(x)

p_cir_stat_Hodges_Ajne2(x, n, asymp_std = FALSE)

p_cir_stat_Hodges_Ajne(x, n, exact = TRUE, asymp_std = FALSE)
```

```
d_cir_stat_Hodges_Ajne(x, n, exact = TRUE, asymp_std = FALSE)

p_cir_stat_Kuiper(x, n, K_Kuiper = 12L, second_term = TRUE, Stephens = FALSE)

d_cir_stat_Kuiper(x, n, K_Kuiper = 12L, second_term = TRUE, Stephens = FALSE)

p_cir_stat_Log_gaps(x, abs_val = TRUE)

d_cir_stat_Log_gaps(x, abs_val = TRUE)

p_cir_stat_Max_uncover(x)

d_cir_stat_Max_uncover(x)

p_cir_stat_Num_uncover(x)

d_cir_stat_Num_uncover(x)

p_cir_stat_Pycke(x)

d_cir_stat_Pycke(x)

p_cir_stat_Vacancy(x)

d_cir_stat_Vacancy(x)

p_cir_stat_Watson(x, n = 0L, K_Watson = 25L, Stephens = FALSE)

d_cir_stat_Watson(x, n = 0L, K_Watson = 25L, Stephens = FALSE)

p_cir_stat_Watson_1976(x, K_Watson_1976 = 8L, N = 40L)

d_cir_stat_Watson_1976(x, K_Watson_1976 = 8L)

p_cir_stat_Range(x, n, max_gap = TRUE)

d_cir_stat_Range(x, n, max_gap = TRUE)

p_cir_stat_Rao(x)

d_cir_stat_Rao(x)

p_cir_stat_Rayleigh(x)

d_cir_stat_Rayleigh(x)

p_cir_stat_Bakshaev(x, K_max = 1000, thre = 0, ...)
```

```
d_cir_stat_Bakshaev(x, K_max = 1000, thre = 0, ...)

p_cir_stat_Gine_Fn(x, K_max = 1000, thre = 0, ...)

d_cir_stat_Gine_Fn(x, K_max = 1000, thre = 0, ...)

p_cir_stat_Gine_Gn(x, K_max = 1000, thre = 0, ...)

d_cir_stat_Gine_Gn(x, K_max = 1000, thre = 0, ...)

p_cir_stat_Hermans_Rasson(x, K_max = 1000, thre = 0, ...)

d_cir_stat_Hermans_Rasson(x, K_max = 1000, thre = 0, ...)

p_cir_stat_PAD(x, K_max = 1000, thre = 0, ...)

d_cir_stat_PAD(x, K_max = 1000, thre = 0, ...)

p_cir_stat_PCvM(x, K_max = 1000, thre = 0, ...)

d_cir_stat_PCvM(x, K_max = 1000, thre = 0, ...)

p_cir_stat_PRt(x, t = 1/3, K_max = 1000, thre = 0, ...)

d_cir_stat_PRt(x, t = 1/3, K_max = 1000, thre = 0, ...)

p_cir_stat_Pycke_q(x, q = 0.5, K_max = 1000, thre = 0, ...)

d_cir_stat_Pycke_q(x, q = 0.5, K_max = 1000, thre = 0, ...)

p_cir_stat_Rothman(x, t = 1/3, K_max = 1000, thre = 0, ...)

d_cir_stat_Rothman(x, t = 1/3, K_max = 1000, thre = 0, ...)

p_cir_stat_Riesz(x, s = 1, K_max = 1000, thre = 0, ...)

d_cir_stat_Riesz(x, s = 1, K_max = 1000, thre = 0, ...)
```

Arguments

`x` a vector of size `nx` or a matrix of size `c(nx,1)`.

`K_Kolmogorov, K_Kuiper, K_Watson, K_Watson_1976, K_Ajne`
integer giving the truncation of the series present in the null asymptotic distributions. For the Kolmogorov-Smirnov-related series defaults to `25`; for the others series defaults to a smaller number.

`alternating` use the alternating series expansion for the distribution of the Kolmogorov-Smirnov statistic? Defaults to `TRUE`.

`n`	sample size employed for computing the statistic.
`asymp_std`	compute the distribution associated to the normalized Hodges-Ajne statistic? Defaults to `FALSE`.
`exact`	use the exact distribution for the Hodges-Ajne statistic? Defaults to `TRUE`.
`second_term`	use the second-order series expansion for the distribution of the Kuiper statistic? Defaults to `TRUE`.
`Stephens`	compute Stephens (1970) modification so that the null distribution of the is less dependent on the sample size? The modification does not alter the test decision.
`abs_val`	compute the distribution associated to the absolute value of the Darling's log gaps statistic? Defaults to `TRUE`.
`N`	number of points used in the Gauss-Legendre quadrature. Defaults to `40`.
`max_gap`	compute the distribution associated to the maximum gap for the range statistic? Defaults to `TRUE`.
`K_max`	integer giving the truncation of the series that compute the asymptotic p-value of a Sobolev test. Defaults to `1e3`.
`thre`	error threshold for the tail probability given by the the first terms of the truncated series of a Sobolev test. Defaults to `0` (no further truncation).
`...`	further parameters passed to `p_Sobolev` or `d_Sobolev` (such as `x_tail`).
`t`	t parameter for the Rothman and Cressie tests, a real in $(0, 1)$. Defaults to `1 / 3`.
`q`	q parameter for the Pycke "q-test", a real in $(0, 1)$. Defaults to `1 / 2`.
`s`	s parameter for the s-Riesz test, a real in $(0, 2)$. Defaults to `1`.

Details

Descriptions and references for most of the tests are available in García-Portugués and Verdebout (2018).

Value

A matrix of size `c(nx,1)` with the evaluation of the distribution or density function at `x`.

References

García-Portugués, E. and Verdebout, T. (2018) An overview of uniformity tests on the hypersphere. *arXiv:1804.00286*. https://arxiv.org/abs/1804.00286.

Examples

```
# Ajne
curve(d_cir_stat_Ajne(x), to = 1.5, n = 2e2, ylim = c(0, 4))
curve(p_cir_stat_Ajne(x), n = 2e2, col = 2, add = TRUE)

# Bakshaev
curve(d_cir_stat_Bakshaev(x, method = "HBE"), to = 6, n = 2e2,
      ylim = c(0, 1))
curve(p_cir_stat_Bakshaev(x, method = "HBE"), n = 2e2, add = TRUE, col = 2)
```

```
# Bingham
curve(d_cir_stat_Bingham(x), to = 12, n = 2e2, ylim = c(0, 1))
curve(p_cir_stat_Bingham(x), n = 2e2, col = 2, add = TRUE)
# Greenwood
curve(d_cir_stat_Greenwood(x), from = -6, to = 6, n = 2e2, ylim = c(0, 1))
curve(p_cir_stat_Greenwood(x), n = 2e2, col = 2, add = TRUE)

# Hermans-Rasson
curve(p_cir_stat_Hermans_Rasson(x, method = "HBE"), to = 10, n = 2e2,
      ylim = c(0, 1))
curve(d_cir_stat_Hermans_Rasson(x, method = "HBE"), n = 2e2, add = TRUE,
      col = 2)

# Hodges-Ajne
plot(25:45, d_cir_stat_Hodges_Ajne(cbind(25:45), n = 50), type = "h",
     lwd = 2, ylim = c(0, 1))
lines(25:45, p_cir_stat_Hodges_Ajne(cbind(25:45), n = 50), type = "s",
      col = 2)

# Kolmogorov-Smirnov
curve(d_Kolmogorov(x), to = 3, n = 2e2, ylim = c(0, 2))
curve(p_Kolmogorov(x), n = 2e2, col = 2, add = TRUE)

# Kuiper
curve(d_cir_stat_Kuiper(x, n = 50), to = 3, n = 2e2, ylim = c(0, 2))
curve(p_cir_stat_Kuiper(x, n = 50), n = 2e2, col = 2, add = TRUE)

# Kuiper and Watson with Stephens modification
curve(d_cir_stat_Kuiper(x, n = 8, Stephens = TRUE), to = 2.5, n = 2e2,
      ylim = c(0, 10))
curve(d_cir_stat_Watson(x, n = 8, Stephens = TRUE), n = 2e2, lty = 2,
      add = TRUE)
n <- c(10, 20, 30, 40, 50, 100, 500)
col <- rainbow(length(n))
for (i in seq_along(n)) {
  curve(d_cir_stat_Kuiper(x, n = n[i], Stephens = TRUE), n = 2e2,
        col = col[i], add = TRUE)
  curve(d_cir_stat_Watson(x, n = n[i], Stephens = TRUE), n = 2e2,
        col = col[i], lty = 2, add = TRUE)
}

# Maximum uncovered spacing
curve(d_cir_stat_Max_uncover(x), from = -3, to = 6, n = 2e2, ylim = c(0, 1))
curve(p_cir_stat_Max_uncover(x), n = 2e2, col = 2, add = TRUE)

# Number of uncovered spacing
curve(d_cir_stat_Num_uncover(x), from = -4, to = 4, n = 2e2, ylim = c(0, 1))
curve(p_cir_stat_Num_uncover(x), n = 2e2, col = 2, add = TRUE)

# Log gaps
curve(d_cir_stat_Log_gaps(x), from = -1, to = 4, n = 2e2, ylim = c(0, 1))
curve(p_cir_stat_Log_gaps(x), n = 2e2, col = 2, add = TRUE)
```

```
# Gine Fn
curve(d_cir_stat_Gine_Fn(x, method = "HBE"), to = 2.5, n = 2e2,
      ylim = c(0, 2))
curve(p_cir_stat_Gine_Fn(x, method = "HBE"), n = 2e2, add = TRUE, col = 2)

# Gine Gn
curve(d_cir_stat_Gine_Gn(x, method = "HBE"), to = 2.5, n = 2e2,
      ylim = c(0, 2))
curve(p_cir_stat_Gine_Gn(x, method = "HBE"), n = 2e2, add = TRUE, col = 2)

# Gini mean difference
curve(d_cir_stat_Gini(x), from = -4, to = 4, n = 2e2, ylim = c(0, 1))
curve(p_cir_stat_Gini(x), n = 2e2, col = 2, add = TRUE)

# Gini mean squared difference
curve(d_cir_stat_Gini_squared(x), from = -10, to = 10, n = 2e2,
      ylim = c(0, 1))
curve(p_cir_stat_Gini_squared(x), n = 2e2, col = 2, add = TRUE)

# PAD
curve(d_cir_stat_PAD(x, method = "HBE"), to = 3, n = 2e2, ylim = c(0, 1.5))
curve(p_cir_stat_PAD(x, method = "HBE"), n = 2e2, add = TRUE, col = 2)

# PCvM
curve(d_cir_stat_PCvM(x, method = "HBE"), to = 4, n = 2e2, ylim = c(0, 2))
curve(p_cir_stat_PCvM(x, method = "HBE"), n = 2e2, add = TRUE, col = 2)

# PRt
curve(d_cir_stat_PRt(x, method = "HBE"), n = 2e2, ylim = c(0, 5))
curve(p_cir_stat_PRt(x, method = "HBE"), n = 2e2, add = TRUE, col = 2)

# Pycke
curve(d_cir_stat_Pycke(x), from = -5, to = 10, n = 2e2, ylim = c(0, 1))
curve(p_cir_stat_Pycke(x), n = 2e2, col = 2, add = TRUE)

# Pycke q
curve(d_cir_stat_Pycke_q(x, method = "HBE"), to = 15, n = 2e2,
      ylim = c(0, 1))
curve(p_cir_stat_Pycke_q(x, method = "HBE"), n = 2e2, add = TRUE, col = 2)

# Range
curve(d_cir_stat_Range(x, n = 50), to = 2, n = 2e2, ylim = c(0, 4))
curve(p_cir_stat_Range(x, n = 50), n = 2e2, col = 2, add = TRUE)

# Rao
curve(d_cir_stat_Rao(x), from = -6, to = 6, n = 2e2, ylim = c(0, 1))
curve(p_cir_stat_Rao(x), n = 2e2, col = 2, add = TRUE)

# Rayleigh
curve(d_cir_stat_Rayleigh(x), to = 12, n = 2e2, ylim = c(0, 1))
curve(p_cir_stat_Rayleigh(x), n = 2e2, col = 2, add = TRUE)
```

```
# Riesz
curve(d_cir_stat_Riesz(x, method = "HBE"), to = 6, n = 2e2,
      ylim = c(0, 1))
curve(p_cir_stat_Riesz(x, method = "HBE"), n = 2e2, add = TRUE, col = 2)

# Rothman
curve(d_cir_stat_Rothman(x, method = "HBE"), n = 2e2, ylim = c(0, 5))
curve(p_cir_stat_Rothman(x, method = "HBE"), n = 2e2, add = TRUE, col = 2)

# Vacancy
curve(d_cir_stat_Vacancy(x), from = -4, to = 4, n = 2e2, ylim = c(0, 1))
curve(p_cir_stat_Vacancy(x), n = 2e2, col = 2, add = TRUE)

# Watson
curve(d_cir_stat_Watson(x), to = 0.5, n = 2e2, ylim = c(0, 15))
curve(p_cir_stat_Watson(x), n = 2e2, col = 2, add = TRUE)

# Watson (1976)
curve(d_cir_stat_Watson_1976(x), to = 1.5, n = 2e2, ylim = c(0, 3))
curve(p_cir_stat_Watson_1976(x), n = 2e2, col = 2, add = TRUE)
```

---

p_sph_stat_Bingham *Asymptotic distributions for spherical uniformity statistics*

Description

Computation of the asymptotic null distributions of spherical uniformity statistics.

Usage

```
p_sph_stat_Bingham(x, p)

d_sph_stat_Bingham(x, p)

p_sph_stat_CJ12(x, regime = 1L, beta = 0)

d_sph_stat_CJ12(x, regime = 3L, beta = 0)

p_sph_stat_Rayleigh(x, p)

d_sph_stat_Rayleigh(x, p)

p_sph_stat_Rayleigh_HD(x, p)

d_sph_stat_Rayleigh_HD(x, p)

p_sph_stat_Ajne(x, p, K_max = 1000, thre = 0, ...)

d_sph_stat_Ajne(x, p, K_max = 1000, thre = 0, ...)
```

```
p_sph_stat_Bakshaev(x, p, K_max = 1000, thre = 0, ...)

d_sph_stat_Bakshaev(x, p, K_max = 1000, thre = 0, ...)

p_sph_stat_Gine_Fn(x, p, K_max = 1000, thre = 0, ...)

d_sph_stat_Gine_Fn(x, p, K_max = 1000, thre = 0, ...)

p_sph_stat_Gine_Gn(x, p, K_max = 1000, thre = 0, ...)

d_sph_stat_Gine_Gn(x, p, K_max = 1000, thre = 0, ...)

p_sph_stat_PAD(x, p, K_max = 1000, thre = 0, ...)

d_sph_stat_PAD(x, p, K_max = 1000, thre = 0, ...)

p_sph_stat_PCvM(x, p, K_max = 1000, thre = 0, ...)

d_sph_stat_PCvM(x, p, K_max = 1000, thre = 0, ...)

p_sph_stat_PRt(x, p, t = 1/3, K_max = 1000, thre = 0, ...)

d_sph_stat_PRt(x, p, t = 1/3, K_max = 1000, thre = 0, ...)

p_sph_stat_Riesz(x, p, s = 1, K_max = 1000, thre = 0, ...)

d_sph_stat_Riesz(x, p, s = 1, K_max = 1000, thre = 0, ...)
```

Arguments

`x`	a vector of size `nx` or a matrix of size `c(nx,1)`.
`p`	integer giving the dimension of the ambient space R^p that contains S^{p-1}.
`regime`	type of asymptotic regime for the CJ12 test, either `1` (sub-exponential regime), `2` (exponential), or `3` (super-exponential; default).
`beta`	β parameter in the exponential regime of the CJ12 test, a nonnegative real. Defaults to `0`.
`K_max`	integer giving the truncation of the series that compute the asymptotic p-value of a Sobolev test. Defaults to `1e3`.
`thre`	error threshold for the tail probability given by the the first terms of the truncated series of a Sobolev test. Defaults to `0` (no further truncation).
`...`	further parameters passed to `p_Sobolev` or `d_Sobolev` (such as `x_tail`).
`t`	t parameter for the Rothman and Cressie tests, a real in $(0, 1)$. Defaults to `1 / 3`.
`s`	s parameter for the s-Riesz test, a real in $(0, 2)$. Defaults to `1`.

Details

Descriptions and references on most of the asymptotic distributions are available in García-Portugués and Verdebout (2018).

Value

- r_sph_stat_*: a matrix of size c(n,1) containing the sample.
- p_sph_stat_*, d_sph_stat_*: a matrix of size c(nx,1) with the evaluation of the distribution or density functions at x.

Examples

```
# Ajne
curve(d_sph_stat_Ajne(x, p = 3, method = "HBE"), n = 2e2, ylim = c(0, 4))
curve(p_sph_stat_Ajne(x, p = 3, method = "HBE"), n = 2e2, col = 2,
      add = TRUE)

# Bakshaev
curve(d_sph_stat_Bakshaev(x, p = 3, method = "HBE"), to = 5, n = 2e2,
      ylim = c(0, 2))
curve(p_sph_stat_Bakshaev(x, p = 3, method = "HBE"), n = 2e2, col = 2,
      add = TRUE)

# Bingham
curve(d_sph_stat_Bingham(x, p = 3), to = 20, n = 2e2, ylim = c(0, 1))
curve(p_sph_stat_Bingham(x, p = 3), n = 2e2, col = 2, add = TRUE)

# CJ12
curve(d_sph_stat_CJ12(x, regime = 1), from = -10, to = 10, n = 2e2,
      ylim = c(0, 1))
curve(d_sph_stat_CJ12(x, regime = 2, beta = 0.1), n = 2e2, col = 2,
      add = TRUE)
curve(d_sph_stat_CJ12(x, regime = 3), n = 2e2, col = 3, add = TRUE)
curve(p_sph_stat_CJ12(x, regime = 1), n = 2e2, col = 1, add = TRUE)
curve(p_sph_stat_CJ12(x, regime = 2, beta = 0.1), n = 2e2, col = 2,
      add = TRUE)
curve(p_sph_stat_CJ12(x, regime = 3), col = 3, add = TRUE)

# Gine Fn
curve(d_sph_stat_Gine_Fn(x, p = 3, method = "HBE"), to = 2, n = 2e2,
      ylim = c(0, 2))
curve(p_sph_stat_Gine_Fn(x, p = 3, method = "HBE"), n = 2e2, col = 2,
      add = TRUE)

# Gine Gn
curve(d_sph_stat_Gine_Gn(x, p = 3, method = "HBE"), to = 1.5, n = 2e2,
      ylim = c(0, 2.5))
curve(p_sph_stat_Gine_Gn(x, p = 3, method = "HBE"), n = 2e2, col = 2,
      add = TRUE)

# PAD
```

```
curve(d_sph_stat_PAD(x, p = 3, method = "HBE"), to = 3, n = 2e2,
      ylim = c(0, 1.5))
curve(p_sph_stat_PAD(x, p = 3, method = "HBE"), n = 2e2, col = 2,
      add = TRUE)

# PCvM
curve(d_sph_stat_PCvM(x, p = 3, method = "HBE"), to = 0.6, n = 2e2,
      ylim = c(0, 7))
curve(p_sph_stat_PCvM(x, p = 3, method = "HBE"), n = 2e2, col = 2,
      add = TRUE)

# PRt
curve(d_sph_stat_PRt(x, p = 3, method = "HBE"), n = 2e2, ylim = c(0, 5))
curve(p_sph_stat_PRt(x, p = 3, method = "HBE"), n = 2e2, col = 2, add = TRUE)

# Rayleigh
curve(d_sph_stat_Rayleigh(x, p = 3), to = 15, n = 2e2, ylim = c(0, 1))
curve(p_sph_stat_Rayleigh(x, p = 3), n = 2e2, col = 2, add = TRUE)

# HD-standardized Rayleigh
curve(d_sph_stat_Rayleigh_HD(x, p = 3), from = -4, to = 4, n = 2e2,
      ylim = c(0, 1))
curve(p_sph_stat_Rayleigh_HD(x, p = 3), n = 2e2, col = 2, add = TRUE)

# Riesz
curve(d_sph_stat_Riesz(x, p = 3, method = "HBE"), n = 2e2, from = 0, to = 5,
      ylim = c(0, 2))
curve(p_sph_stat_Riesz(x, p = 3, method = "HBE"), n = 2e2, col = 2,
      add = TRUE)
```

`rhea` *Rhea craters from Hirata (2016)*

Description

Craters on Rhea from Hirata (2016).

Usage

```
rhea
```

Format

A data frame with 3596 rows and 4 variables:

name name of the crater (if named).

diameter diameter of the crater (in km).

theta longitude angle $\theta \in [0, 2\pi)$ of the crater center.

phi latitude angle $\phi \in [-\pi/2, \pi/2]$ of the crater center.

Details

The (θ, ϕ) angles are such their associated planetocentric coordinates are:

$$(\cos(\phi)\cos(\theta), \cos(\phi)\sin(\theta), \sin(\phi))',$$

with $(0, 0, 1)'$ denoting the north pole.

The script performing the data preprocessing is available at `rhea.R`.

Source

`https://agupubs.onlinelibrary.wiley.com/action/downloadSupplement?doi=10.1002%2F2015JE004940& file=jgre20485-sup-0002-TableS1.txt`

References

Hirata, N. (2016) Differential impact cratering of Saturn's satellites by heliocentric impactors. *Journal of Geophysical Research: Planets*, 121:111–117. doi: 10.1002/2015JE004940

Examples

```
# Load data
data("rhea")

# Add Cartesian coordinates
rhea$X <- cbind(cos(rhea$theta) * cos(rhea$phi),
                sin(rhea$theta) * cos(rhea$phi),
                sin(rhea$phi))

# Tests
unif_test(data = rhea$X[rhea$diam > 15 & rhea$diam < 20, ],
          type = c("PCvM", "PAD", "PRt"), p_value = "asymp")
```

`r_alt` *Sample non-uniformly distributed spherical data*

Description

Simple simulation of prespecified non-uniform spherical distributions: von Mises–Fisher (vMF), Mixture of vMF (MvMF), Angular Central Gaussian (ACG), Small Circle (SC), Watson (W), or Cauchy-like (C).

Usage

```
r_alt(n, p, M = 1, alt = "vMF", kappa = 1, nu = 0.5, F_inv = NULL,
  K = 1000, axial_MvMF = TRUE)
```

Arguments

n	sample size.
p	integer giving the dimension of the ambient space R^p that contains S^{p-1}.
M	number of samples of size n. Defaults to 1.
alt	alternative, must be "vMF", "MvMF", "ACG", "SC", "W", or "C". See details below.
kappa	non-negative parameter measuring the strength of the deviation with respect to uniformity (obtained with $\kappa = 0$).
nu	projection along $\mathbf{e}_p$ controlling the modal strip of the small circle distribution. Must be in (-1, 1). Defaults to 0.5.
F_inv	quantile function returned by F_inv_from_f. Used for "SC", "W", and "C". Computed by internally if NULL (default).
K	number of equispaced points on $[-1, 1]$ used for evaluating F^{-1} and then interpolating. Defaults to 1e3.
axial_MvMF	use a mixture of vMF that is axial (i.e., symmetrically distributed about the origin)? Defaults to TRUE.

Details

The parameter kappa is used as κ in the following distributions:

- "vMF": von Mises–Fisher distribution with concentration κ and directional mean $\mathbf{e}_p = (0, 0, \ldots, 1)$.
- "MvMF": equally-weighted mixture of p von Mises–Fisher distributions with common concentration κ and directional means $\pm\mathbf{e}_1, \ldots, \pm\mathbf{e}_p$ if axial_MvMF = TRUE. If axial_MvMF = FALSE, then only means with positive signs are considered.
- "ACG": Angular Central Gaussian distribution with diagonal shape matrix with diagonal given by
$$(1, \ldots, 1, 1 + \kappa)/(p + \kappa).$$
- "SC": Small Circle distribution with axis mean $\mathbf{e}_p = (0, 0, \ldots, 1)$ and concentration κ about the projection along the mean, ν.
- "W": Watson distribution with axis mean $\mathbf{e}_p = (0, 0, \ldots, 1)$ and concentration κ. The Watson distribution is a particular case of the Bingham distribution.
- "C": Cauchy-like distribution with directional mode $\mathbf{e}_p = (0, 0, \ldots, 1)$ and concentration $\kappa = \rho/(1 - \rho^2)$. The circular Wrapped Cauchy distribution is a particular case of this Cauchy-like distribution.

Much faster sampling for "SC", "W", and "C" is achieved providing F_inv, see examples.

Value

An **array** of size c(n,p,M) with M random samples of size n of non-uniformly-generated directions on S^{p-1}.

Examples

```
## Simulation with p = 2

p <- 2
n <- 200
kappa <- 20
nu <- 0.5
rho <- ((2 * kappa + 1) - sqrt(4 * kappa + 1)) / (2 * kappa)
F_inv_SC_2 <- F_inv_from_f(f = function(z) exp(-kappa * (z - nu)^2), p = 2)
F_inv_W_2 <- F_inv_from_f(f = function(z) exp(kappa * z^2), p = 2)
F_inv_C_2 <- F_inv_from_f(f = function(z) (1 - rho^2) /
                            (1 + rho^2 - 2 * rho * z)^(p / 2), p = 2)
x1 <- r_alt(n = n, p = p, alt = "vMF", kappa = kappa)[, , 1]
x2 <- r_alt(n = n, p = p, alt = "MvMF", kappa = kappa)[, , 1]
x3 <- r_alt(n = n, p = p, alt = "ACG", kappa = kappa)[, , 1]
x4 <- r_alt(n = n, p = p, alt = "SC", F_inv = F_inv_SC_2)[, , 1]
x5 <- r_alt(n = n, p = p, alt = "W", F_inv = F_inv_W_2)[, , 1]
x6 <- r_alt(n = n, p = p, alt = "C", F_inv = F_inv_C_2)[, , 1]
r <- runif(n, 0.95, 1.05) # Radius perturbation to improve visualization
plot(r * x1, pch = 16, xlim = c(-1.1, 1.1), ylim = c(-1.1, 1.1), col = 1)
points(r * x2, pch = 16, col = 2)
points(r * x3, pch = 16, col = 3)
points(r * x4, pch = 16, col = 4)
points(r * x5, pch = 16, col = 5)
points(r * x6, pch = 16, col = 6)

## Simulation with p = 3

n <- 200
p <- 3
kappa <- 20
nu <- 0.5
rho <- ((2 * kappa + 1) - sqrt(4 * kappa + 1)) / (2 * kappa)
F_inv_SC_3 <- F_inv_from_f(f = function(z) exp(-kappa * (z - nu)^2), p = 3)
F_inv_W_3 <- F_inv_from_f(f = function(z) exp(kappa * z^2), p = 3)
F_inv_C_3 <- F_inv_from_f(f = function(z) (1 - rho^2) /
                            (1 + rho^2 - 2 * rho * z)^(p / 2), p = 3)
x1 <- r_alt(n = n, p = p, alt = "vMF", kappa = kappa)[, , 1]
x2 <- r_alt(n = n, p = p, alt = "MvMF", kappa = kappa)[, , 1]
x3 <- r_alt(n = n, p = p, alt = "ACG", kappa = kappa)[, , 1]
x4 <- r_alt(n = n, p = p, alt = "SC", F_inv = F_inv_SC_3)[, , 1]
x5 <- r_alt(n = n, p = p, alt = "W", F_inv = F_inv_W_3)[, , 1]
x6 <- r_alt(n = n, p = p, alt = "C", F_inv = F_inv_C_3)[, , 1]
s3d <- scatterplot3d::scatterplot3d(x1, pch = 16, xlim = c(-1.1, 1.1),
                                    ylim = c(-1.1, 1.1), zlim = c(-1.1, 1.1))
s3d$points3d(x2, pch = 16, col = 2)
s3d$points3d(x3, pch = 16, col = 3)
s3d$points3d(x4, pch = 16, col = 4)
s3d$points3d(x5, pch = 16, col = 5)
s3d$points3d(x6, pch = 16, col = 6)
```

r_unif *Sample uniformly distributed circular and spherical data*

Description

Simulation of the uniform distribution on $[0, 2\pi)$ and $S^{p-1} := \{\mathbf{x} \in R^p : ||\mathbf{x}|| = 1\}, p \geq 2$.

Usage

```
r_unif_cir(n, M = 1L, sorted = FALSE)

r_unif_sph(n, p, M = 1L)
```

Arguments

n	sample size.
M	number of samples of size n. Defaults to 1.
sorted	return each circular sample sorted? Defaults to FALSE.
p	integer giving the dimension of the ambient space R^p that contains S^{p-1}.

Value

- r_unif_cir: a **matrix** of size c(n,M) with M random samples of size n of uniformly-generated circular data on $[0, 2\pi)$.
- r_unif_sph: an **array** of size c(n,p,M) with M random samples of size n of uniformly-generated directions on S^{p-1}.

Examples

```
# A sample on [0, 2*pi)
n <- 5
r_unif_cir(n = n)

# A sample on S^1
p <- 2
samp <- r_unif_sph(n = n, p = p)
samp
rowSums(samp^2)

# A sample on S^2
p <- 3
samp <- r_unif_sph(n = n, p = p)
samp
rowSums(samp^2)
```

`Sobolev` *Asymptotic distributions of Sobolev statistics of spherical uniformity*

Description

Approximated density, distribution, and quantile functions for the asymptotic null distributions of Sobolev statistics of uniformity on $S^{p-1} := \{\mathbf{x} \in R^p : ||\mathbf{x}|| = 1\}$. These asymptotic distributions are infinite weighted sums of (central) chi squared random variables:

$$\sum_{k=1}^{\infty} v_k^2 \chi^2_{d_{p,k}},$$

where

$$d_{p,k} := \binom{p+k-3}{p-2} + \binom{p+k-2}{p-2}$$

is the dimension of the space of eigenfunctions of the Laplacian on S^{p-1}, $p \geq 2$, associated to the k-th eigenvalue, $k \geq 1$.

Usage

```
d_p_k(p, k, log = FALSE)

weights_dfs_Sobolev(p, K_max = 1000, thre = 0.001, type, Rothman_t = 1/3,
  Pycke_q = 0.5, Riesz_s = 1, log = FALSE, verbose = TRUE,
  Gauss = TRUE, N = 320, tol = 1e-06, force_positive = TRUE,
  x_tail = NULL)

d_Sobolev(x, p, type, method = c("I", "SW", "HBE")[1], K_max = 1000,
  thre = 0.001, Rothman_t = 1/3, Pycke_q = 0.5, Riesz_s = 1,
  ncps = 0, verbose = TRUE, N = 320, x_tail = NULL, ...)

p_Sobolev(x, p, type, method = c("I", "SW", "HBE", "MC")[1], K_max = 1000,
  thre = 0.001, Rothman_t = 1/3, Pycke_q = 0.5, Riesz_s = 1,
  ncps = 0, verbose = TRUE, N = 320, x_tail = NULL, ...)

q_Sobolev(u, p, type, method = c("I", "SW", "HBE", "MC")[1], K_max = 1000,
  thre = 0.001, Rothman_t = 1/3, Pycke_q = 0.5, Riesz_s = 1,
  ncps = 0, verbose = TRUE, N = 320, x_tail = NULL, ...)
```

Arguments

`p`	integer giving the dimension of the ambient space R^p that contains S^{p-1}.
`k`	sequence of integer indexes.
`log`	compute the logarithm of $d_{p,k}$? Defaults to `FALSE`.
`K_max`	integer giving the truncation of the series that compute the asymptotic p-value of a Sobolev test. Defaults to `1e3`.

thre	error threshold for the tail probability given by the the first terms of the truncated series of a Sobolev test. Defaults to 1e-3.
type	Sobolev statistic. For $p = 2$, either "Watson", "Rothman", "Pycke_q", or "Hermans_Rasson". For $p \geq 2$, "Ajne", "Gine_Gn", "Gine_Fn", "Bakshaev", "Riesz", "PCvM", "PAD", or "PRt".
Rothman_t	t parameter for the Rothman test, a real in $(0, 1)$. Defaults to 1 / 3.
Pycke_q	q parameter for the Pycke "q-test", a real in $(0, 1)$. Defaults to 1 / 2.
Riesz_s	s parameter for the s-Riesz test, a real in $(0, 2)$. Defaults to 1.
verbose	output information about the truncation? Defaults to TRUE.
Gauss	use a Gauss–Legendre quadrature rule of N nodes in the computation of the Gegenbauer coefficients? Otherwise, call integrate. Defaults to TRUE.
N	number of points used in the Gauss–Legendre quadrature for computing the Gegenbauer coefficients. Defaults to 320.
tol	tolerance passed to integrate's rel.tol and abs.tol if Gauss = FALSE. Defaults to 1e-6.
force_positive	set negative
x_tail	scalar evaluation point for determining the upper tail probability. If NULL, set to the 0.90 quantile of the whole series, computed by the "HBE" approximation.
x	vector of quantiles.
method	method for approximating the density, distribution, or quantile function. Must be "I" (Imhof), "SW" (Satterthwaite–Welch), "HBE" (Hall–Buckley–Eagleson), or "MC" (Monte Carlo; only for distribution or quantile functions). Defaults to "I".
ncps	non-centrality parameters. Either 0 (default) or a vector with the same length as weights.
...	further parameters passed to *_wschisq.
u	vector of probabilities.

Details

The truncation of $\sum_{k=1}^\infty v_k^2 \chi^2_{d_{p,k}}$ is done to the first K_max terms and then up to the index such that the first terms explain the tail probability at the x_tail with an absolute error smaller than thre (see details in cutoff_wschisq). This automatic truncation takes place when calling *_Sobolev. Setting thre = 0 truncates to K_max terms exactly. If the series only contains odd or even non-zero terms, then only K_max / 2 addends are *effectively* taken into account in the first truncation.

Value

- d_p_k: a vector of size length(k) with the evaluation of $d_{p,k}$.
- weights_dfs_Sobolev: a list with entries weights and dfs, automatically truncated according to K_max and thre (see details).
- d_Sobolev: density function evaluated at x, a vector.
- p_Sobolev: distribution function evaluated at x, a vector.
- q_Sobolev: quantile function evaluated at u, a vector.

Author(s)

Eduardo García-Portugués and Paula Navarro-Esteban.

Examples

```
# Circular-specific statistics
curve(p_Sobolev(x = x, p = 2, type = "Watson", method = "HBE"),
      n = 2e2, ylab = "Distribution", main = "Watson")
curve(p_Sobolev(x = x, p = 2, type = "Rothman", method = "HBE"),
      n = 2e2, ylab = "Distribution", main = "Rothman")
curve(p_Sobolev(x = x, p = 2, type = "Pycke_q", method = "HBE"), to = 10,
      n = 2e2, ylab = "Distribution", main = "Pycke_q")
curve(p_Sobolev(x = x, p = 2, type = "Hermans_Rasson", method = "HBE"),
      to = 10, n = 2e2, ylab = "Distribution", main = "Hermans_Rasson")

# Statistics for arbitrary dimensions
test_statistic <- function(type, to = 1, pmax = 5, M = 1e3, ...) {

  col <- viridisLite::viridis(pmax - 1)
  curve(p_Sobolev(x = x, p = 2, type = type, method = "MC", M = M,
                  ...), to = to, n = 2e2, col = col[pmax - 1],
                  ylab = "Distribution", main = type, ylim = c(0, 1))
  for (p in 3:pmax) {
    curve(p_Sobolev(x = x, p = p, type = type, method = "MC", M = M,
                    ...), add = TRUE, n = 2e2, col = col[pmax - p + 1])
  }
  legend("bottomright", legend = paste("p =", 2:pmax), col = rev(col),
         lwd = 2)

}

# Ajne
test_statistic(type = "Ajne")

# Gine_Gn
test_statistic(type = "Gine_Gn", to = 1.5)

# Gine_Fn
test_statistic(type = "Gine_Fn", to = 2)

# Bakshaev
test_statistic(type = "Bakshaev", to = 3)

# Riesz
test_statistic(type = "Riesz", Riesz_s = 0.5, to = 3)

# PCvM
test_statistic(type = "PCvM", to = 0.6)

# PAD
test_statistic(type = "PAD", to = 3)
```

```
# PRt
test_statistic(type = "PRt", Rothman_t = 0.5)

# Quantiles
p <- c(2, 3, 4, 11)
t(sapply(p, function(p) q_Sobolev(u = c(0.10, 0.05, 0.01), p = p,
                                  type = "PCvM")))
t(sapply(p, function(p) q_Sobolev(u = c(0.10, 0.05, 0.01), p = p,
                                  type = "PAD")))
t(sapply(p, function(p) q_Sobolev(u = c(0.10, 0.05, 0.01), p = p,
                                  type = "PRt")))

# Series truncation for thre = 1e-5
sapply(p, function(p) length(weights_dfs_Sobolev(p = p, type = "PCvM")$dfs))
sapply(p, function(p) length(weights_dfs_Sobolev(p = p, type = "PRt")$dfs))
sapply(p, function(p) length(weights_dfs_Sobolev(p = p, type = "PAD")$dfs))
```

`Sobolev_coefs` *Transformation between different coefficients in Sobolev statistics*

Description

Given a Sobolev statistic

$$S_{n,p} = \sum_{i,j=1}^{n} \psi(\cos^{-1}(\mathbf{X}_i'\mathbf{X}_j)),$$

for a sample $\mathbf{X}_1, \ldots, \mathbf{X}_n \in S^{p-1} := \{\mathbf{x} \in R^p : ||\mathbf{x}|| = 1\}$, $p \geq 2$, three important sequences are related to $S_{n,p}$.

- Gegenbauer coefficients $\{b_{k,p}\}$ of ψ_p (see, e.g., the projected-ecdf statistics), given by

$$b_{k,p} := \frac{1}{c_{k,p}} \int_0^\pi \psi_p(\theta) C_k^{p/2-1}(\cos\theta) \, \mathrm{d}\theta.$$

- Weights $\{v_{k,p}^2\}$ of the asymptotic distribution of the Sobolev statistic, $\sum_{k=1}^\infty v_k^2 \chi_{d_{p,k}}^2$, given by

$$v_{k,p}^2 = \left(1 + \frac{2k}{p-2}\right)^{-1} b_{k,p}, \quad p \geq 3.$$

- Gegenbauer coefficients $\{u_{k,p}\}$ of the local projected alternative associated to $S_{n,p}$, given by

$$u_{k,p} = \left(1 + \frac{2k}{p-2}\right) v_{k,p}, \quad p \geq 3.$$

For $p = 2$, the factor $(1 + 2k/(p-2))$ is replaced by 2.

Usage

```
bk_to_vk2(bk, p)

bk_to_uk(bk, p, signs = 1)

vk2_to_bk(vk2, p)

vk2_to_uk(vk2, p, signs = 1)

uk_to_vk2(uk, p)

uk_to_bk(uk, p)
```

Arguments

bk	coefficients $b_{k,p}$ associated to the indexes 1:length(bk), a vector.
p	integer giving the dimension of the ambient space R^p that contains S^{p-1}.
signs	signs of the coefficients $u_{k,p}$, a vector of the same size as vk2 or bk, or a scalar. Defaults to 1.
vk2	**squared** coefficients $v_{k,p}^2$ associated to the indexes 1:length(vk2), a vector.
uk	coefficients $u_{k,p}$ associated to the indexes 1:length(uk), a vector.

Details

See more details in Prentice (1978) and García-Portugués et al. (2020). The adequate signs of uk for the "PRt" Rothman test can be retrieved with akx and sqr = TRUE, see the examples.

Value

The corresponding vectors of coefficients vk2, bk, or uk, depending on the call.

References

García-Portugués, E., Navarro-Esteban, P., Cuesta-Albertos, J. A. (2020) On a projection-based class of uniformity tests on the hypersphere. *arXiv:2008.09897*. https://arxiv.org/abs/2008.09897

Prentice, M. J. (1978). On invariant tests of uniformity for directions and orientations. *The Annals of Statistics*, 6(1):169–176. doi: 10.1214/aos/1176344075

Examples

```
# bk, vk2, and uk for the PCvM test in p = 3
(bk <- Gegen_coefs_Pn(k = 1:5, type = "PCvM", p = 3))
(vk2 <- bk_to_vk2(bk = bk, p = 3))
(uk <- bk_to_uk(bk = bk, p = 3))

# vk2 is the same as
weights_dfs_Sobolev(K_max = 10, thre = 0, p = 3, type = "PCvM")$weights
```

```
# bk and uk for the Rothman test in p = 3, with adequate signs
t <- 1 / 3
(bk <- Gegen_coefs_Pn(k = 1:5, type = "PRt", p = 3, Rothman_t = t))
(ak <- akx(x = drop(q_proj_unif(t, p = 3)), p = 3, k = 1:5, sqr = TRUE))
(uk <- bk_to_uk(bk = bk, p = 3, signs = ak))
```

---

sph_stat_Rayleigh *Statistics for testing (hyper)spherical uniformity*

Description

Low-level implementation of several statistics for assessing uniformity on the (hyper)sphere $S^{p-1} := \{\mathbf{x} \in R^p : ||\mathbf{x}|| = 1\}, p \geq 2$.

Usage

```
sph_stat_Rayleigh(X)

sph_stat_Bingham(X)

sph_stat_Ajne(X, Psi_in_X = FALSE)

sph_stat_Gine_Gn(X, Psi_in_X = FALSE, p = 0L)

sph_stat_Gine_Fn(X, Psi_in_X = FALSE, p = 0L)

sph_stat_Pycke(X, Psi_in_X = FALSE, p = 0L)

sph_stat_Bakshaev(X, Psi_in_X = FALSE, p = 0L)

sph_stat_Riesz(X, Psi_in_X = FALSE, p = 0L, s = 1)

sph_stat_PCvM(X, Psi_in_X = FALSE, p = 0L, N = 160L, L = 1000L)

sph_stat_PRt(X, t = 1/3, Psi_in_X = FALSE, p = 0L, N = 160L, L = 1000L)

sph_stat_PAD(X, Psi_in_X = FALSE, p = 0L, N = 160L, L = 1000L)

sph_stat_CCF09(X, dirs, K_CCF09 = 25L, original = FALSE)

sph_stat_Rayleigh_HD(X)

sph_stat_CJ12(X, regime = 3L, Psi_in_X = FALSE, p = 0L)
```

Arguments

X	an **array** of size `c(n,p,M)` containing the Cartesian coordinates of `M` samples of size `n` of directions on S^{p-1}. Must not contain `NA`'s.
Psi_in_X	does `X` contain the shortest angles matrix $\mathbf{\Psi}$ that is obtained with `Psi_mat(X)`? If `FALSE` (default), $\mathbf{\Psi}$ is computed internally.
p	integer giving the dimension of the ambient space R^p that contains S^{p-1}.
s	s parameter for the s-Riesz test, a real in $(0, 2)$. Defaults to `1`.
N	number of points used in the Gauss-Legendre quadrature. Defaults to `160`.
L	number of discretization points to interpolate angular functions that require evaluating an integral. Defaults to `1e3`.
t	t parameter for the Rothman and Cressie tests, a real in $(0, 1)$. Defaults to `1 / 3`.
dirs	a matrix of size `c(n_proj,p)` containing `n_proj` random directions (in Cartesian coordinates) on S^{p-1} to perform the CCF09 test.
K_CCF09	integer giving the truncation of the series present in the asymptotic distribution of the Kolmogorov-Smirnov statistic. Defaults to `5e2`.
original	return the CCF09 statistic as originally defined? If `FALSE` (default), a faster and equivalent statistic is computed, and rejection happens for *large* values of the statistic, which is consistent with the rest of tests. Otherwise, rejection happens for *low* values.
regime	type of asymptotic regime for the CJ12 test, either `1` (sub-exponential regime), `2` (exponential), or `3` (super-exponential; default).

Details

Detailed descriptions and references of the statistics are available in García-Portugués and Verdebout (2018).

The Pycke and CJ12 statistics employ the *scalar products* matrix, rather than the shortest angles matrix, when `Psi_in_X = TRUE`. This matrix is obtained by setting `scalar_prod = TRUE` in `Psi_mat`.

Value

A matrix of size `c(M,1)` containing the statistics for each of the `M` samples.

Warning

Be careful on avoiding the next bad usages of the functions, which will produce spurious results:

- The directions in `X` do *not* have unit norm.
- `X` does *not* contain `Psi_mat(X)` when `X_in_Theta = TRUE`.
- The parameter `p` does *not* match with the dimension of R^p.
- *Not* passing the scalar products matrix to `sph_stat_CJ12` when `Psi_in_X = TRUE`.
- The directions in `dirs` do *not* have unit norm.

References

García-Portugués, E. and Verdebout, T. (2018) An overview of uniformity tests on the hypersphere. *arXiv:1804.00286*. https://arxiv.org/abs/1804.00286.

Examples

```
## Sample uniform spherical data

M <- 2
n <- 100
p <- 3
set.seed(123456789)
X <- r_unif_sph(n = n, p = p, M = M)

## Sobolev tests

# Rayleigh
sph_stat_Rayleigh(X)

# Bingham
sph_stat_Bingham(X)

# Ajne
Psi <- Psi_mat(X)
dim(Psi) <- c(dim(Psi), 1)
sph_stat_Ajne(X)
sph_stat_Ajne(Psi, Psi_in_X = TRUE)

# Gine Gn
sph_stat_Gine_Gn(X)
sph_stat_Gine_Gn(Psi, Psi_in_X = TRUE, p = p)

# Gine Fn
sph_stat_Gine_Fn(X)
sph_stat_Gine_Fn(Psi, Psi_in_X = TRUE, p = p)

# Pycke
sph_stat_Pycke(X)
sph_stat_Pycke(Psi, Psi_in_X = TRUE, p = p)

# Bakshaev
sph_stat_Bakshaev(X)
sph_stat_Bakshaev(Psi, Psi_in_X = TRUE, p = p)

# Riesz
sph_stat_Riesz(X, s = 1)
sph_stat_Riesz(Psi, Psi_in_X = TRUE, p = p, s = 1)

# Projected Cramér-von Mises
sph_stat_PCvM(X)
sph_stat_PCvM(Psi, Psi_in_X = TRUE, p = p)
```

```
# Projected Rothman
sph_stat_PRt(X)
sph_stat_PRt(Psi, Psi_in_X = TRUE, p = p)

# Projected Anderson-Darling
sph_stat_PAD(X)
sph_stat_PAD(Psi, Psi_in_X = TRUE, p = p)

## Other tests

# CCF09
dirs <- r_unif_sph(n = 3, p = p, M = 1)[, , 1]
sph_stat_CCF09(X, dirs = dirs)

## High-dimensional tests

# Rayleigh HD-Standardized
sph_stat_Rayleigh_HD(X)

# CJ12
sph_stat_CJ12(X, regime = 1)
sph_stat_CJ12(Psi, regime = 1, Psi_in_X = TRUE, p = p)
sph_stat_CJ12(X, regime = 2)
sph_stat_CJ12(Psi, regime = 2, Psi_in_X = TRUE, p = p)
sph_stat_CJ12(X, regime = 3)
sph_stat_CJ12(Psi, regime = 3, Psi_in_X = TRUE, p = p)
```

---

unif_stat *Circular and (hyper)spherical uniformity statistics*

Description

Implementation of several statistics for assessing uniformity on the (hyper)sphere $S^{p-1} := \{\mathbf{x} \in R^p : ||\mathbf{x}|| = 1\}$, $p \geq 2$, for a sample $\mathbf{X}_1, \ldots, \mathbf{X}_n \in S^{p-1}$.

unif_stat receives a (several) sample(s) of directions in *Cartesian coordinates*, except for the circular case ($p = 2$) in which the sample(s) can be *angles* $\Theta_1, \ldots, \Theta_n \in [0, 2\pi)$.

unif_stat allows to compute several statistics to several samples within a single call, facilitating thus Monte Carlo experiments.

Usage

```
unif_stat(data, type = "all", data_sorted = FALSE, Rayleigh_m = 1,
  cov_a = 2 * pi, Rothman_t = 1/3, Cressie_t = 1/3, Pycke_q = 0.5,
  Riesz_s = 1, CCF09_dirs = NULL, K_CCF09 = 25, CJ12_reg = 3)
```

Arguments

data	sample to compute the test statistic. An **array** of size c(n,p,M) containing M samples of size n of directions (in Cartesian coordinates) on S^{p-1}. Alternatively, a **matrix** of size c(n,M) with the angles on $[0, 2\pi)$ of the M circular samples of size n on S^1. Other objects accepted are an array of size c(n,1,M) or a vector of size n with angular data. Must not contain NA's.
type	type of test to be applied. A character vector containing any of the following types of tests, depending on the dimension p: • Circular data: any of the names available at object avail_cir_tests. • (Hyper)spherical data: any of the names available at object avail_sph_tests. If type = "all" (default), then type is set as avail_cir_tests or avail_sph_tests, depending on the value of p.
data_sorted	is the circular data sorted? If TRUE, certain statistics are faster to compute. Defaults to FALSE.
Rayleigh_m	integer m for the m-modal Rayleigh test. Defaults to m = 1 (the standard Rayleigh test).
cov_a	$a_n = a/n$ parameter used in the length of the arcs of the coverage-based tests. Must be positive. Defaults to 2 * pi.
Rothman_t	t parameter for the Rothman test, a real in $(0, 1)$. Defaults to 1 / 3.
Cressie_t	t parameter for the Cressie test, a real in $(0, 1)$. Defaults to 1 / 3.
Pycke_q	q parameter for the Pycke "q-test", a real in $(0, 1)$. Defaults to 1 / 2.
Riesz_s	s parameter for the s-Riesz test, a real in $(0, 2)$. Defaults to 1.
CCF09_dirs	a matrix of size c(n_proj,p) containing n_proj random directions (in Cartesian coordinates) on S^{p-1} to perform the CCF09 test. If NULL (default), a sample of size n_proj = 50 directions is computed internally.
K_CCF09	integer giving the truncation of the series present in the asymptotic distribution of the Kolmogorov-Smirnov statistic. Defaults to 5e2.
CJ12_reg	type of asymptotic regime for CJ12 test, either 1 (sub-exponential regime), 2 (exponential), or 3 (super-exponential; default).

Details

Descriptions and references for most of the statistics are available in García-Portugués and Verdebout (2018).

Value

A data frame of size c(M,length(type)), with column names given by type, that contains the values of the test statistics.

References

García-Portugués, E. and Verdebout, T. (2018) An overview of uniformity tests on the hypersphere. *arXiv:1804.00286*. https://arxiv.org/abs/1804.00286.

Examples

```
## Circular data

# Sample
n <- 10
M <- 2
Theta <- r_unif_cir(n = n, M = M)

# Matrix
unif_stat(data = Theta, type = "all")

# Array
unif_stat(data = array(Theta, dim = c(n, 1, M)), type = "all")

# Vector
unif_stat(data = Theta[, 1], type = "all")

## Spherical data

# Circular sample in Cartesian coordinates
n <- 10
M <- 2
X <- array(dim = c(n, 2, M))
for (i in 1:M) X[, , i] <- cbind(cos(Theta[, i]), sin(Theta[, i]))

# Array
unif_stat(data = X, type = "all")

# High-dimensional data
X <- r_unif_sph(n = n, p = 3, M = M)
unif_stat(data = X, type = "all")

## Specific arguments

# Rothman
unif_stat(data = Theta, type = "Rothman", Rothman_t = 0.5)

# CCF09
unif_stat(data = X, type = "CCF09", CCF09_dirs = X[, , 1])
unif_stat(data = X, type = "CCF09", CCF09_dirs = X[, , 1], K_CCF09 = 1)

# CJ12
unif_stat(data = X, type = "CJ12", CJ12_reg = 3)
unif_stat(data = X, type = "CJ12", CJ12_reg = 1)
```

`unif_stat_distr` *Null distributions for circular and (hyper)spherical uniformity statistics*

Description

Approximate computation of the null distributions of several statistics for assessing uniformity on the (hyper)sphere $S^{p-1} := \{\mathbf{x} \in R^p : ||\mathbf{x}|| = 1\}$, $p \geq 2$. The approximation is done either by means of the asymptotic distribution or by Monte Carlo.

Usage

```
unif_stat_distr(x, type, p, n, approx = "asymp", M = 10000,
  stats_MC = NULL, Rothman_t = 1/3, Pycke_q = 0.5, Riesz_s = 1,
  CCF09_dirs = NULL, CJ12_reg = 3, CJ12_beta = 0, Stephens = FALSE,
  K_Kuiper = 25, K_Watson = 25, K_Watson_1976 = 5, K_Ajne = 500,
  K_CCF09 = 25, K_max = 10000, ...)
```

Arguments

`x` evaluation points for the null distribution(s). Either a vector of size `nx`, if the evaluation points are common for the tests in `type`, or a matrix of size `c(nx,length(type))` with columns containing the evaluation points for each test. Must not contain `NA`'s.

`type` type of test to be applied. A character vector containing any of the following types of tests, depending on the dimension p:

- Circular data: any of the names available at object `avail_cir_tests`.
- (Hyper)spherical data: any of the names available at object `avail_sph_tests`.

If `type = "all"` (default), then `type` is set as `avail_cir_tests` or `avail_sph_tests`, depending on the value of p.

`p` integer giving the dimension of the ambient space R^p that contains S^{p-1}.

`n` sample size employed for computing the statistic.

`approx` type of approximation to the null distribution, either `"asymp"` (default) for employing the asymptotic null distribution, if available, or `"MC"`, for employing the Monte Carlo approximation of the exact null distribution.

`M` number of Monte Carlo replications for approximating the null distribution when `approx = "MC"`. Also, number of Monte Carlo samples for approximating the asymptotic distributions based on weighted sums of chi squared random variables. Defaults to `1e4`.

`stats_MC` a data frame of size `c(M,length(type))`, with column names containing the character vector `type`, that results from extracting `$stats_MC` from a call to `unif_stat_MC`. If provided, the computation of Monte Carlo statistics when `approx = "MC"` is skipped. `stats_MC` is checked internally to see if it is sorted. Internally computed if `NULL` (default).

`Rothman_t` t parameter for the Rothman test, a real in $(0, 1)$. Defaults to `1 / 3`.

`Pycke_q` q parameter for the Pycke "q-test", a real in $(0, 1)$. Defaults to `1 / 2`.

`Riesz_s` s parameter for the s-Riesz test, a real in $(0, 2)$. Defaults to `1`.

`CCF09_dirs` a matrix of size `c(n_proj,p)` containing `n_proj` random directions (in Cartesian coordinates) on S^{p-1} to perform the CCF09 test. If `NULL` (default), a sample of size `n_proj = 50` directions is computed internally.

CJ12_reg
: type of asymptotic regime for CJ12 test, either 1 (sub-exponential regime), 2 (exponential), or 3 (super-exponential; default).

CJ12_beta
: β parameter in the exponential regime of CJ12 test, a positive real.

Stephens
: compute Stephens (1970) modification so that the null distribution of the is less dependent on the sample size? The modification does not alter the test decision.

K_Kuiper, K_Watson, K_Watson_1976, K_Ajne
: integer giving the truncation of the series present in the null asymptotic distributions. For the Kolmogorov-Smirnov-related series defaults to 25.

K_CCF09
: integer giving the truncation of the series present in the asymptotic distribution of the Kolmogorov-Smirnov statistic. Defaults to 5e2.

K_max
: integer giving the truncation of the series that compute the asymptotic p-value of a Sobolev test. Defaults to 1e4.

...
: if approx = "MC", optional performance parameters to be passed to unif_stat_MC: chunks, cores, and seed.

Details

When approx = "asymp", statistics that do not have an implemented or known asymptotic are omitted, and a warning is generated.

For Sobolev tests, K_max = 1e4 produces probabilities uniformly accurate with three digits for the "PCvM", "PAD", and "PRt" tests, for dimensions $p \leq 11$. With K_max = 5e4, these probabilities are uniformly accurate in the fourth digit. With K_max = 1e3, only two-digit uniform accuracy is obtained. Uniform accuracy deteriorates when p increases, e.g., a digit accuracy is lost when $p = 51$.

Descriptions and references on most of the asymptotic distributions are available in García-Portugués and Verdebout (2018).

Value

A data frame of size c(nx,length(type)), with column names given by type, that contains the values of the null distributions of the statistics evaluated at x.

References

García-Portugués, E. and Verdebout, T. (2018) An overview of uniformity tests on the hypersphere. *arXiv:1804.00286*. https://arxiv.org/abs/1804.00286.

Examples

```
## Asymptotic distribution

# Circular statistics
x <- seq(0, 1, l = 5)
unif_stat_distr(x = x, type = "Kuiper", p = 2, n = 10)
unif_stat_distr(x = x, type = c("Ajne", "Kuiper"), p = 2, n = 10)
unif_stat_distr(x = x, type = c("Ajne", "Kuiper"), p = 2, n = 10, K_Ajne = 5)

# All circular statistics
```

```
unif_stat_distr(x = x, type = avail_cir_tests, p = 2, n = 10, K_max = 1e3)

# Spherical statistics
unif_stat_distr(x = cbind(x, x + 1), type = c("Rayleigh", "Bingham"),
                p = 3, n = 10)
unif_stat_distr(x = cbind(x, x + 1), type = c("Rayleigh", "Bingham"),
                p = 3, n = 10, M = 100)

# All spherical statistics
unif_stat_distr(x = x, type = avail_sph_tests, p = 3, n = 10, K_max = 1e3)

## Monte Carlo distribution

# Circular statistics
x <- seq(0, 5, l = 10)
unif_stat_distr(x = x, type = avail_cir_tests, p = 2, n = 10, approx = "MC")
unif_stat_distr(x = x, type = "Kuiper", p = 2, n = 10, approx = "MC")
unif_stat_distr(x = x, type = c("Ajne", "Kuiper"), p = 2, n = 10,
                approx = "MC")

# Spherical statistics
unif_stat_distr(x = x, type = avail_sph_tests, p = 3, n = 10,
                approx = "MC")
unif_stat_distr(x = cbind(x, x + 1), type = c("Rayleigh", "Bingham"),
                p = 3, n = 10, approx = "MC")
unif_stat_distr(x = cbind(x, x + 1), type = c("Rayleigh", "Bingham"),
                p = 3, n = 10, approx = "MC")

## Specific arguments

# Rothman
unif_stat_distr(x = x, type = "Rothman", p = 2, n = 10, Rothman_t = 0.5,
                approx = "MC")

# CCF09
dirs <- r_unif_sph(n = 5, p = 3, M = 1)[, , 1]
x <- seq(0, 1, l = 10)
unif_stat_distr(x = x, type = "CCF09", p = 3, n = 10, approx = "MC",
                CCF09_dirs = dirs)
unif_stat_distr(x = x, type = "CCF09", p = 3, n = 10, approx = "MC")

# CJ12
unif_stat_distr(x = x, type = "CJ12", p = 3, n = 100, CJ12_reg = 3)
unif_stat_distr(x = x, type = "CJ12", p = 3, n = 100, CJ12_reg = 2,
               CJ12_beta = 0.01)
unif_stat_distr(x = x, type = "CJ12", p = 3, n = 100, CJ12_reg = 1)
```

`unif_stat_MC` *Monte Carlo simulation of circular and (hyper)spherical uniformity statistics*

Description

Utility for performing Monte Carlo simulation of several statistics for assessing uniformity on the (hyper)sphere $S^{p-1} := \{\mathbf{x} \in R^p : ||\mathbf{x}|| = 1\}, p \geq 2$.

unif_stat_MC provides a convenient wrapper for parallel evaluation of unif_stat, the estimation of critical values under the null distribution, and the computation of empirical powers under the alternative.

Usage

```
unif_stat_MC(n, type = "all", p, M = 10000, r_H1 = NULL,
  crit_val = NULL, alpha = c(0.1, 0.05, 0.01), return_stats = TRUE,
  stats_sorted = FALSE, chunks = ceiling((n * M)/1e+05), cores = 1,
  seeds = NULL, Rayleigh_m = 1, cov_a = 2 * pi, Rothman_t = 1/3,
  Cressie_t = 1/3, Pycke_q = 0.5, Riesz_s = 1, CCF09_dirs = NULL,
  K_CCF09 = 25, CJ12_reg = 3, ...)
```

Arguments

n	sample size.
type	type of test to be applied. A character vector containing any of the following types of tests, depending on the dimension p: • Circular data: any of the names available at object avail_cir_tests. • (Hyper)spherical data: any of the names available at object avail_sph_tests. If type = "all" (default), then type is set as avail_cir_tests or avail_sph_tests, depending on the value of p.
p	integer giving the dimension of the ambient space R^p that contains S^{p-1}.
M	number of Monte Carlo replications. Defaults to 1e4.
r_H1	if provided, the computation of empirical powers is carried out for the alternative hypothesis sampled with r_H1. This must be a function with the same arguments and value as r_unif_sph (see examples). Defaults to NULL, indicating that the critical values are estimated from samples of r_unif_sph.
crit_val	if provided, must be the critical values as returned by $stats_MC in a call to unif_stat_MC. They are used for computing the empirical powers of the tests present in type. Defaults to NULL, which means that no power computation is done.
alpha	vector with significance levels. Defaults to c(0.10,0.05,0.01).
return_stats	return the Monte Carlo statistics? If only the critical values or powers are desired, FALSE saves memory in the returned object. Defaults to TRUE.
stats_sorted	sort the returned Monte Carlo statistics? If TRUE, this is useful for evaluating faster the empirical cumulative distribution function when approximating the distribution in unif_stat_distr. Defaults to FALSE.
chunks	number of chunks to split the M Monte Carlo replications. Useful for parallelizing the simulation study in chunks tasks containing ceiling(M / chunks) replications. Useful also for avoiding memory bottlenecks when M is large. Defaults to ceiling((n * M) / 1e5).

cores	number of cores to perform the simulation. Defaults to 1.
seeds	if provided, a vector of size chunks for fixing the seeds on each of the simulation chunks (useful for reproducing parallel simulations). Specifically, for k in 1:chunks, seeds are set as set.seed(seeds[k], kind = "Mersenne-Twister") in each chunk. Defaults to NULL (no seed setting is done).
Rayleigh_m	integer m for the m-modal Rayleigh test. Defaults to m = 1 (the standard Rayleigh test).
cov_a	$a_n = a/n$ parameter used in the length of the arcs of the coverage-based tests. Must be positive. Defaults to 2 * pi.
Rothman_t	t parameter for the Rothman test, a real in $(0, 1)$. Defaults to 1 / 3.
Cressie_t	t parameter for the Cressie test, a real in $(0, 1)$. Defaults to 1 / 3.
Pycke_q	q parameter for the Pycke "q-test", a real in $(0, 1)$. Defaults to 1 / 2.
Riesz_s	s parameter for the s-Riesz test, a real in $(0, 2)$. Defaults to 1.
CCF09_dirs	a matrix of size c(n_proj, p) containing n_proj random directions (in Cartesian coordinates) on S^{p-1} to perform the CCF09 test. If NULL (default), a sample of size n_proj = 50 directions is computed internally.
K_CCF09	integer giving the truncation of the series present in the asymptotic distribution of the Kolmogorov-Smirnov statistic. Defaults to 5e2.
CJ12_reg	type of asymptotic regime for CJ12 test, either 1 (sub-exponential regime), 2 (exponential), or 3 (super-exponential; default).
...	optional arguments to be passed to the r_H1 sampler or to foreach (for example, .export to export global variables or other functions to the foreach environment).

Details

It is possible to have a progress bar if unif_stat_MC is wrapped with progressr::with_progress or if progressr::handlers(global = TRUE) is invoked (once) by the user. See the examples below. The progress bar is updated with the number of finished chunks.

All the tests reject for large values of the test statistic (max_gap = TRUE is assumed for the Range test), so the critical values for the significance levels alpha correspond to the alpha-upper quantiles of the null distribution of the test statistic.

The Monte Carlo simulation for the CCF09 test is made conditionally on the choice of CCF09_dirs. That is, all the Monte Carlo statistics share the same random directions.

Value

A list with the following entries:

- crit_val_MC: a data frame of size c(length(alpha), length(type)), with column names given by type and rows corresponding to the significance levels alpha, that contains the estimated critical values of the tests.
- power_MC: a data frame of size c(nrow(crit_val), length(type)), with column names given by type and rows corresponding to the significance levels of crit_val, that contains the empirical powers of the tests. NA if crit_val = NULL.
- stats_MC: a data frame of size c(M, length(type)), with column names given by type, that contains the Monte Carlo statistics.

Examples

```
## Critical values

# Single statistic, specific alpha
cir <- unif_stat_MC(n = 10, M = 1e2, type = "Ajne", p = 2, alpha = 0.15)
summary(cir$stats_MC)
cir$crit_val_MC

# All circular statistics
cir <- unif_stat_MC(n = 10, M = 1e2, p = 2)
head(cir$stats_MC)
cir$crit_val_MC

# All spherical statistics
sph <- unif_stat_MC(n = 10, M = 1e2, p = 3)
head(sph$stats_MC)
sph$crit_val_MC

## Using a progress bar

# Define a progress bar
require(progress)
require(progressr)
handlers(handler_progress(
  format = ":spin [:bar] :percent Total: :elapsedfull End \u2248 :eta",
  clear = FALSE))

# Call unif_stat_MC() within with_progress()
with_progress(unif_stat_MC(n = 10, M = 1e2, p = 3, chunks = 10))

# With several cores
with_progress(unif_stat_MC(n = 10, M = 1e2, p = 3, chunks = 10, cores = 2))

# Instead of using with_progress() each time, it is more practical to run
# handlers(global = TRUE)
# once to activate progress bars in your R session

## Power computation

# Single statistic
cir_pow <- unif_stat_MC(n = 10, M = 1e2, type = "Ajne", p = 2,
                        crit_val = cir$crit_val_MC)
cir_pow$crit_val_MC
cir_pow$power_MC

# All circular statistics
cir_pow <- unif_stat_MC(n = 10, M = 1e2, p = 2, crit_val = cir$crit_val_MC)
cir_pow$crit_val_MC
cir_pow$power_MC

# All spherical statistics
sph_pow <- unif_stat_MC(n = 10, M = 1e2, p = 3, crit_val = sph$crit_val_MC)
```

```
sph_pow$crit_val_MC
sph_pow$power_MC

## Custom r_H1

# Circular
r_H1 <- function(n, p, M, l = 0.05) {

  stopifnot(p == 2)
  Theta_to_X(matrix(runif(n * M, 0, (2 - l) * pi), n, M))

}
dirs <- r_unif_sph(n = 5, p = 2, M = 1)[, , 1]
cir <- unif_stat_MC(n = 50, M = 1e2, p = 2, CCF09_dirs = dirs)
cir_pow <- unif_stat_MC(n = 50, M = 1e2, p = 2, r_H1 = r_H1, l = 0.10,
                        crit_val = cir$crit_val_MC, CCF09_dirs = dirs)
cir_pow$crit_val_MC
cir_pow$power_MC

# Spherical
r_H1 <- function(n, p, M, l = 0.5) {

  samp <- array(dim = c(n, p, M))
  for (j in 1:M) {

    samp[, , j] <- mvtnorm::rmvnorm(n = n, mean = c(l, rep(0, p - 1)),
                                    sigma = diag(rep(1, p)))
    samp[, , j] <- samp[, , j] / sqrt(rowSums(samp[, , j]^2))

  }
  return(samp)

}
dirs <- r_unif_sph(n = 5, p = 3, M = 1)[, , 1]
sph <- unif_stat_MC(n = 50, M = 1e2, p = 3, CCF09_dirs = dirs)
sph_pow <- unif_stat_MC(n = 50, M = 1e2, p = 3, r_H1 = r_H1, l = 0.5,
                       crit_val = sph$crit_val_MC, CCF09_dirs = dirs)
sph_pow$power_MC

## Pre-built r_H1

# Circular
dirs <- r_unif_sph(n = 5, p = 2, M = 1)[, , 1]
cir_pow <- unif_stat_MC(n = 50, M = 1e2, p = 2, r_H1 = r_alt, alt = "vMF",
                        kappa = 1, crit_val = cir$crit_val_MC,
                        CCF09_dirs = dirs)
cir_pow$power_MC

# Spherical
dirs <- r_unif_sph(n = 5, p = 3, M = 1)[, , 1]
sph_pow <- unif_stat_MC(n = 50, M = 1e2, p = 3, r_H1 = r_alt, alt = "vMF",
                        kappa = 1, crit_val = sph$crit_val_MC,
                        CCF09_dirs = dirs)
```

```
sph_pow$power_MC
```

unif_test *Circular and (hyper)spherical uniformity tests*

Description

Implementation of several uniformity tests on the (hyper)sphere $S^{p-1} := \{\mathbf{x} \in R^p : ||\mathbf{x}|| = 1\}$, $p \geq 2$, with calibration either in terms of their asymptotic/exact distributions, if available, or Monte Carlo.

`unif_test` receives a sample of directions $\mathbf{X}_1, \ldots, \mathbf{X}_n \in S^{p-1}$ in *Cartesian coordinates*, except for the circular case ($p = 2$) in which the sample can be represented in terms of *angles* $\Theta_1, \ldots, \Theta_n \in [0, 2\pi)$.

`unif_test` allows to perform several tests within a single call, facilitating thus the exploration of a dataset by applying several tests.

Usage

```
unif_test(data, type = "all", p_value = "asymp", alpha = c(0.1, 0.05,
  0.01), M = 10000, stats_MC = NULL, crit_val = NULL,
  data_sorted = FALSE, Rayleigh_m = 1, cov_a = 2 * pi, Rothman_t = 1/3,
  Cressie_t = 1/3, Pycke_q = 0.5, Riesz_s = 1, CCF09_dirs = NULL,
  K_CCF09 = 25, CJ12_reg = 3, CJ12_beta = 0, K_max = 10000, ...)
```

Arguments

`data` sample to perform the test. A matrix of size `c(n,p)` containing a sample of size `n` of directions (in Cartesian coordinates) on S^{p-1}. Alternatively if `p = 2`, a matrix of size `c(n,1)` containing the `n` angles on $[0, 2\pi)$ of the circular sample on S^1. Other objects accepted are an array of size `c(n,p,1)` with directions (in Cartesian coordinates), or a vector of size `n` or an array of size `c(n,1,1)` with angular data. Must not contain `NA`'s.

`type` type of test to be applied. A character vector containing any of the following types of tests, depending on the dimension p:

- Circular data: any of the names available at object `avail_cir_tests`.
- (Hyper)spherical data: any of the names available at object `avail_sph_tests`.

If `type = "all"` (default), then `type` is set as `avail_cir_tests` or `avail_sph_tests`, depending on the value of p.

`p_value` type of p-value computation. Either `"MC"` for employing the approximation by Monte Carlo of the exact null distribution, `"asymp"` (default) for the use of the asymptotic/exact null distribution (if available), or `"crit_val"` for approximation by means of the table of critical values `crit_val`.

`alpha` vector with significance levels. Defaults to `c(0.10,0.05,0.01)`.

M number of Monte Carlo replications for approximating the null distribution when approx = "MC". Also, number of Monte Carlo samples for approximating the asymptotic distributions based on weighted sums of chi squared random variables. Defaults to 1e4.

stats_MC a data frame of size c(M,length(type)), with column names containing the character vector type, that results from extracting $stats_MC from a call to unif_stat_MC. If provided, the computation of Monte Carlo statistics when approx = "MC" is skipped. stats_MC is checked internally to see if it is sorted. Internally computed if NULL (default).

crit_val table with critical values for the tests, to be used if p_value = "crit_val". A data frame, with column names containing the character vector type and rows corresponding to the significance levels alpha, that results from extracting $crit_val_MC from a call to unif_stat_MC. Internally computed if NULL (default).

data_sorted is the circular data sorted? If TRUE, certain statistics are faster to compute. Defaults to FALSE.

Rayleigh_m integer m for the m-modal Rayleigh test. Defaults to m = 1 (the standard Rayleigh test).

cov_a $a_n = a/n$ parameter used in the length of the arcs of the coverage-based tests. Must be positive. Defaults to 2 * pi.

Rothman_t t parameter for the Rothman test, a real in $(0, 1)$. Defaults to 1 / 3.

Cressie_t t parameter for the Cressie test, a real in $(0, 1)$. Defaults to 1 / 3.

Pycke_q q parameter for the Pycke "q-test", a real in $(0, 1)$. Defaults to 1 / 2.

Riesz_s s parameter for the s-Riesz test, a real in $(0, 2)$. Defaults to 1.

CCF09_dirs a matrix of size c(n_proj,p) containing n_proj random directions (in Cartesian coordinates) on S^{p-1} to perform the CCF09 test. If NULL (default), a sample of size n_proj = 50 directions is computed internally.

K_CCF09 integer giving the truncation of the series present in the asymptotic distribution of the Kolmogorov-Smirnov statistic. Defaults to 5e2.

CJ12_reg type of asymptotic regime for CJ12 test, either 1 (sub-exponential regime), 2 (exponential), or 3 (super-exponential; default).

CJ12_beta β parameter in the exponential regime of CJ12 test, a positive real.

K_max integer giving the truncation of the series that compute the asymptotic p-value of a Sobolev test. Defaults to 1e4.

... If p_value = "MC" or p_value = "crit_val", optional performance parameters to be passed to unif_stat_MC: chunks, cores, and seed.

Details

All the tests reject for large values of the test statistic, so the critical values for the significance levels alpha correspond to the alpha-upper quantiles of the null distribution of the test statistic.

When p_value = "asymp", tests that do not have an implemented or known asymptotic are omitted, and a warning is generated.

When p_value = "MC", it is possible to have a progress bar indicating the Monte Carlo simulation progress if unif_test is wrapped with progressr::with_progress or if progressr::handlers(global = TRUE) is invoked (once) by the user. See the examples below. The progress bar is updated with the number of finished chunks.

All the statistics are continuous random variables except the Hodges–Ajne statistic ("Hodges_Ajne"), the Cressie statistic ("Cressie"), and the number of (different) uncovered spacings ("Num_uncover"). These three statistics are discrete random variables.

The Monte Carlo calibration for the CCF09 test is made conditionally on the choice of CCF09_dirs. That is, all the Monte Carlo statistics share the same random directions.

Descriptions and references for most of the tests are available in García-Portugués and Verdebout (2018).

Value

If only a **single test** is performed, a list with class htest containing the following components:

- statistic: the value of the test statistic.
- p.value: the p-value of the test. If p_value = "crit_val", an NA.
- alternative: a character string describing the alternative hypothesis.
- method: a character string indicating what type of test was performed.
- data.name: a character string giving the name of the data.
- reject: the rejection decision for the levels of significance alpha.
- crit_val: a vector with the critical values for the significance levels alpha used with p_value = "MC" or p_value = "asymp".

If **several tests** are performed, a type-named list with entries for each test given by the above list.

References

García-Portugués, E. and Verdebout, T. (2018) An overview of uniformity tests on the hypersphere. *arXiv:1804.00286*. https://arxiv.org/abs/1804.00286.

Examples

```
## Asymptotic distribution

# Circular data
n <- 10
samp_cir <- r_unif_cir(n = n)

# Matrix
unif_test(data = samp_cir, type = "Ajne", p_value = "asymp")

# Vector
unif_test(data = samp_cir[, 1], type = "Ajne", p_value = "asymp")

# Array
unif_test(data = array(samp_cir, dim = c(n, 1, 1)), type = "Ajne",
```

```
          p_value = "asymp")

# Several tests
unif_test(data = samp_cir, type = avail_cir_tests, p_value = "asymp")

# Spherical data
n <- 10
samp_sph <- r_unif_sph(n = n, p = 3)

# Array
unif_test(data = samp_sph, type = "Bingham", p_value = "asymp")

# Matrix
unif_test(data = samp_sph[, , 1], type = "Bingham", p_value = "asymp")

# Several tests
unif_test(data = samp_sph, type = avail_sph_tests, p_value = "asymp")

## Monte Carlo

# Circular data
unif_test(data = samp_cir, type = "Ajne", p_value = "MC")
unif_test(data = samp_cir, type = avail_cir_tests, p_value = "MC")

# Spherical data
unif_test(data = samp_sph, type = "Bingham", p_value = "MC")
unif_test(data = samp_sph, type = avail_sph_tests, p_value = "MC")

# Caching stats_MC
stats_MC_cir <- unif_stat_MC(n = nrow(samp_cir), p = 2)$stats_MC
stats_MC_sph <- unif_stat_MC(n = nrow(samp_sph), p = 3)$stats_MC
unif_test(data = samp_cir, type = avail_cir_tests,
          p_value = "MC", stats_MC = stats_MC_cir)
unif_test(data = samp_sph, type = avail_sph_tests, p_value = "MC",
          stats_MC = stats_MC_sph)

## Critical values

# Circular data
unif_test(data = samp_cir, type = avail_cir_tests, p_value = "crit_val")

# Spherical data
unif_test(data = samp_sph, type = avail_sph_tests, p_value = "crit_val")

# Caching crit_val
crit_val_cir <- unif_stat_MC(n = n, p = 2)$crit_val_MC
crit_val_sph <- unif_stat_MC(n = n, p = 3)$crit_val_MC
unif_test(data = samp_cir, type = avail_cir_tests,
          p_value = "crit_val", crit_val = crit_val_cir)
unif_test(data = samp_sph, type = avail_sph_tests, p_value = "crit_val",
          crit_val = crit_val_sph)

## Specific arguments
```

```
# Rothman
unif_test(data = samp_cir, type = "Rothman", Rothman_t = 0.5)

# CCF09
unif_test(data = samp_sph, type = "CCF09", p_value = "MC",
          CCF09_dirs = samp_sph[1:2, , 1])
unif_test(data = samp_sph, type = "CCF09", p_value = "MC",
          CCF09_dirs = samp_sph[3:4, , 1])

## Using a progress bar when p_value = "MC"

# Define a progress bar
require(progress)
require(progressr)
handlers(handler_progress(
  format = ":spin [:bar] :percent Total: :elapsedfull End \u2248 :eta",
  clear = FALSE))

# Call unif_test() within with_progress()
with_progress(
  unif_test(data = samp_sph, type = avail_sph_tests, p_value = "MC",
            chunks = 10, M = 1e3)
)

# With several cores
with_progress(
  unif_test(data = samp_sph, type = avail_sph_tests, p_value = "MC",
            cores = 2, chunks = 10, M = 1e3)
)

# Instead of using with_progress() each time, it is more practical to run
# handlers(global = TRUE)
# once to activate progress bars in your R session
```

---

venus *Venus craters*

Description

Craters on Venus from the USGS Astrogeology Science Center.

Usage

```
venus
```

Format

A data frame with 967 rows and 4 variables:

name name of the crater (if named).

diameter diameter of the crater (in km).

theta longitude angle $\theta \in [0, 2\pi)$ of the crater center.

phi latitude angle $\phi \in [-\pi/2, \pi/2]$ of the crater center.

Details

The (θ, ϕ) angles are such their associated planetocentric coordinates are:

$$(\cos(\phi)\cos(\theta), \cos(\phi)\sin(\theta), \sin(\phi))',$$

with $(0, 0, 1)'$ denoting the north pole.

The script performing the data preprocessing is available at `venus.R`.

Source

https://astrogeology.usgs.gov/search/map/Venus/venuscraters

Examples

```
# Load data
data("venus")

# Add Cartesian coordinates
venus$X <- cbind(cos(venus$theta) * cos(venus$phi),
                 sin(venus$theta) * cos(venus$phi),
                 sin(venus$phi))

# Tests
unif_test(data = venus$X, type = c("PCvM", "PAD", "PRt"), p_value = "asymp")
```

`wschisq` *Weighted sums of non-central chi squared random variables*

Description

Approximated density, distribution, and quantile functions for weighted sums of non-central chi squared random variables:

$$Q_K = \sum_{i=1}^{K} w_i \chi^2_{d_i}(\lambda_i),$$

where $w_1, \ldots, w_n$ are positive weights, $d_1, \ldots, d_n$ are positive degrees of freedom, and $\lambda_1, \ldots, \lambda_n$ are non-negative non-centrality parameters. Also, simulation of Q_K.

Usage

```
d_wschisq(x, weights, dfs, ncps = 0, method = c("I", "SW", "HBE")[1],
  exact_chisq = TRUE, imhof_epsabs = 1e-06, imhof_epsrel = 1e-06,
  imhof_limit = 10000, grad_method = "simple",
  grad_method.args = list(eps = 1e-07))

p_wschisq(x, weights, dfs, ncps = 0, method = c("I", "SW", "HBE", "MC")[1],
  exact_chisq = TRUE, imhof_epsabs = 1e-06, imhof_epsrel = 1e-06,
  imhof_limit = 10000, M = 10000, MC_sample = NULL)

q_wschisq(u, weights, dfs, ncps = 0, method = c("I", "SW", "HBE", "MC")[1],
  exact_chisq = TRUE, imhof_epsabs = 1e-06, imhof_epsrel = 1e-06,
  imhof_limit = 10000, nlm_gradtol = 1e-06, nlm_iterlim = 1000,
  M = 10000, MC_sample = NULL)

r_wschisq(n, weights, dfs, ncps = 0)

cutoff_wschisq(thre = 1e-04, weights, dfs, ncps = 0, log = FALSE,
  x_tail = NULL)
```

Arguments

`x` vector of quantiles.

`weights` vector with the positive weights of the sum. Must have the same length as `dfs`.

`dfs` vector with the positive degrees of freedom of the chi squared random variables. Must have the same length as `weights`.

`ncps` non-centrality parameters. Either `0` (default) or a vector with the same length as `weights`.

`method` method for approximating the density, distribution, or quantile function. Must be `"I"` (Imhof), `"SW"` (Satterthwaite–Welch), `"HBE"` (Hall–Buckley–Eagleson), or `"MC"` (Monte Carlo; only for distribution or quantile functions). Defaults to `"I"`.

`exact_chisq` if `weights` and `dfs` have length one, shall the `Chisquare` functions be called? Otherwise, the approximations are computed for this exact case. Defaults to `TRUE`.

`imhof_epsabs, imhof_epsrel, imhof_limit`
precision parameters passed to `imhof`'s `epsabs`, `epsrel`, and `limit`, respectively. They default to `1e-6`, `1e-6`, and `1e4`.

`grad_method, grad_method.args`
numerical differentiation parameters passed to `grad`'s `method` and `method.args`, respectively. They default to `"simple"`, and `list(eps = 1e-7)` (better precision than `imhof_epsabs` to avoid numerical artifacts).

`M` number of Monte Carlo samples for approximating the distribution if `method = "MC"`. Defaults to `1e4`.

`MC_sample` if provided, it is employed when `method = "MC"`. If not, it is computed internally.

`u` vector of probabilities.

`nlm_gradtol, nlm_iterlim`
: convergence control parameters passed to `nlm`'s `gradtol` and `iterlim`, respectively. They default to `1e-6` and `1e3`.

`n`
: sample size.

`thre`
: vector with the error thresholds of the tail probability and mean/variance explained by the first terms of the series. Defaults to `1e-4`. See details.

`log`
: are `weights` and `dfs` given in log-scale? Defaults to `FALSE`.

`x_tail`
: scalar evaluation point for determining the upper tail probability. If `NULL`, set to the `0.90` quantile of the whole series, computed by the `"HBE"` approximation.

Details

Four methods are implemented for approximating the distribution of a weighted sum of chi squared random variables:

- `"I"`: Imhof's approximation (Imhof, 1961) for the evaluation of the distribution function. If this method is selected, the function is simply a wrapper to `imhof` from the `CompQuadForm` package (Duchesne and Lafaye De Micheaux, 2010).
- `"SW"`: Satterthwaite–Welch (Satterthwaite, 1946; Welch, 1938) approximation, consisting in matching the first *two* moments of Q_K with a gamma distribution.
- `"HBE"`: Hall–Buckley–Eagleson (Hall, 1983; Buckley and Eagleson, 1988) approximation, consisting in matching the first *three* moments of Q_K with a gamma distribution.
- `"MC"`: Monte Carlo approximation using the empirical cumulative distribution function with `M` simulated samples.

The Imhof method is exact up to the prescribed numerical accuracy. It is also the most time-consuming method. The density and quantile functions for this approximation are obtained by numerical differentiation and inversion, respectively, of the approximated distribution.

For the methods based on gamma matching, the `GammaDist` density, distribution, and quantile functions are invoked. The Hall–Buckley–Eagleson approximation tends to overperform the Satterthwaite–Welch approximation.

The Monte Carlo method is relatively inaccurate and slow, but serves as an unbiased reference of the true distribution function. The inversion of the empirical cumulative distribution is done by `quantile`.

An empirical comparison of these and other approximation methods is given in Bodenham and Adams (2016).

`cutoff_wschisq` removes `NAs/NaNs` in `weights` or `dfs` with a message. The threshold `thre` ensures that the tail probability of the truncated and whole series differ less than `thre` at `x_tail`, or that `thre` is the proportion of the mean/variance of the whole series that is *not* retained. The (upper) tail probabilities for evaluating truncation are computed using the Hall–Buckley–Eagleson approximation at `x_tail`.

Value

- `d_wschisq`: density function evaluated at `x`, a vector.
- `p_wschisq`: distribution function evaluated at `x`, a vector.

- `q_wschisq`: quantile function evaluated at `u`, a vector.
- `r_wschisq`: a vector of size `n` containing a random sample.
- `cutoff_wschisq`: a data frame with the indexes up to which the truncated series explains the tail probability with absolute error `thre`, or the proportion of the mean/variance of the whole series that is *not* explained by the truncated series.

Author(s)

Eduardo García-Portugués and Paula Navarro-Esteban.

References

Bodenham, D. A. and Adams, N. M. (2016). A comparison of efficient approximations for a weighted sum of chi-squared random variables. *Statistics and Computing*, 26(4):917–928. doi: 10.1007/s1122201595834

Buckley, M. J. and Eagleson, G. K. (1988). An approximation to the distribution of quadratic forms in normal random variables. *Australian Journal of Statistics*, 30(1):150–159. doi: 10.1111/j.1467-842X.1988.tb00471.x

Duchesne, P. and Lafaye De Micheaux, P. (2010) Computing the distribution of quadratic forms: Further comparisons between the Liu–Tang–Zhang approximation and exact methods. *Computational Statistics and Data Analysis*, 54(4):858–862. doi: 10.1016/j.csda.2009.11.025

Hall, P. (1983). Chi squared approximations to the distribution of a sum of independent random variables. *Annals of Probability*, 11(4):1028–1036. doi: 10.1214/aop/1176993451

Imhof, J. P. (1961). Computing the distribution of quadratic forms in normal variables. *Biometrika*, 48(3/4):419–426. doi: 10.2307/2332763

Satterthwaite, F. E. (1946). An approximate distribution of estimates of variance components. *Biometrics Bulletin*, 2(6):110–114. doi: 10.2307/3002019

Welch, B. L. (1938). The significance of the difference between two means when the population variances are unequal. *Biometrika*, 29(3/4):350–362. doi: 10.2307/2332010

Examples

```
# Plotting functions for the examples
add_approx_dens <- function(x, dfs, weights, ncps) {

  lines(x, d_wschisq(x, weights = weights, dfs = dfs, ncps = ncps,
                     method = "SW", exact_chisq = FALSE), col = 3)
  lines(x, d_wschisq(x, weights = weights, dfs = dfs, ncps = ncps,
                     method = "HBE", exact_chisq = FALSE), col = 4)
  lines(x, d_wschisq(x, weights = weights, dfs = dfs, ncps = ncps,
                     method = "I", exact_chisq = TRUE), col = 2)
  legend("topright", legend = c("True", "SW", "HBE", "I"), lwd = 2,
         col = c(1, 3:4, 2))

}
add_approx_distr <- function(x, dfs, weights, ncps, ...) {

  lines(x, p_wschisq(x, weights = weights, dfs = dfs, ncps = ncps,
```

```
                     method = "SW", exact_chisq = FALSE), col = 3)
  lines(x, p_wschisq(x, weights = weights, dfs = dfs, ncps = ncps,
                     method = "HBE", exact_chisq = FALSE), col = 4)
  lines(x, p_wschisq(x, weights = weights, dfs = dfs, ncps = ncps,
                     method = "MC", exact_chisq = FALSE), col = 5,
                     type = "s")
  lines(x, p_wschisq(x, weights = weights, dfs = dfs, ncps = ncps,
                     method = "I", exact_chisq = TRUE), col = 2)
  legend("bottomright", legend = c("True", "SW", "HBE", "MC", "I"), lwd = 2,
         col = c(1, 3:5, 2))

}
add_approx_quant <- function(u, dfs, weights, ncps, ...) {

  lines(u, q_wschisq(u, weights = weights, dfs = dfs, ncps = ncps,
                     method = "SW", exact_chisq = FALSE), col = 3)
  lines(u, q_wschisq(u, weights = weights, dfs = dfs, ncps = ncps,
                     method = "HBE", exact_chisq = FALSE), col = 4)
  lines(u, q_wschisq(u, weights = weights, dfs = dfs, ncps = ncps,
                     method = "MC", exact_chisq = FALSE), col = 5,
                     type = "s")
  lines(u, q_wschisq(u, weights = weights, dfs = dfs, ncps = ncps,
                     method = "I", exact_chisq = TRUE), col = 2)
  legend("topleft", legend = c("True", "SW", "HBE", "MC", "I"), lwd = 2,
         col = c(1, 3:5, 2))

}

# Validation plots for density, distribution, and quantile functions
u <- seq(0.01, 0.99, l = 100)
old_par <- par(mfrow = c(1, 3))

# Case 1: 1 * ChiSq_3(0) + 1 * ChiSq_3(0) = ChiSq_6(0)
weights <- c(1, 1)
dfs <- c(3, 3)
ncps <- 0
x <- seq(-1, 30, l = 100)
main <- expression(1 * chi[3]^2 * (0) + 1 * chi[3]^2 * (0))
plot(x, dchisq(x, df = 6), type = "l", main = main, ylab = "Density")
add_approx_dens(x = x, weights = weights, dfs = dfs, ncps = ncps)
plot(x, pchisq(x, df = 6), type = "l", main = main, ylab = "Distribution")
add_approx_distr(x = x, weights = weights, dfs = dfs, ncps = ncps)
plot(u, qchisq(u, df = 6), type = "l", main = main, ylab = "Quantile")
add_approx_quant(u = u, weights = weights, dfs = dfs, ncps = ncps)

# Case 2: 2 * ChiSq_3(1) + 1 * ChiSq_6(0.5) + 0.5 * ChiSq_12(0.25)
weights <- c(2, 1, 0.5)
dfs <- c(3, 6, 12)
ncps <- c(1, 0.5, 0.25)
x <- seq(0, 70, l = 100)
main <- expression(2 * chi[3]^2 * (1)+ 1 * chi[6]^2 * (0.5) +
                   0.5 * chi[12]^2 * (0.25))
samp <- r_wschisq(n = 1e4, weights = weights, dfs = dfs, ncps = ncps)
```

```
hist(samp, breaks = 50, freq = FALSE, main = main, ylab = "Density",
     xlim = range(x), xlab = "x"); box()
add_approx_dens(x = x, weights = weights, dfs = dfs, ncps = ncps)
plot(x, ecdf(samp)(x), main = main, ylab = "Distribution", type = "s")
add_approx_distr(x = x, weights = weights, dfs = dfs, ncps = ncps)
plot(u, quantile(samp, probs = u), type = "s", main = main,
     ylab = "Quantile")
add_approx_quant(u = u, weights = weights, dfs = dfs, ncps = ncps)

# Case 3: \sum_{k = 1}^K k^(-3) * ChiSq_{5k}(1 / k^2)
K <- 1e2
weights<- 1 / (1:K)^3
dfs <- 5 * 1:K
ncps <- 1 / (1:K)^2
x <- seq(0, 25, l = 100)
main <- substitute(sum(k^(-3) * chi[5 * k]^2 * (1 / k^2), k == 1, K),
                   list(K = K))
samp <- r_wschisq(n = 1e4, weights = weights, dfs = dfs, ncps = ncps)
hist(samp, breaks = 50, freq = FALSE, main = main, ylab = "Density",
     xlim = range(x), xlab = "x"); box()
add_approx_dens(x = x, weights = weights, dfs = dfs, ncps = ncps)
plot(x, ecdf(samp)(x), main = main, ylab = "Distribution", type = "s")
add_approx_distr(x = x, weights = weights, dfs = dfs, ncps = ncps)
plot(u, quantile(samp, probs = u), type = "s", main = main,
     ylab = "Quantile")
add_approx_quant(u = u, weights = weights, dfs = dfs, ncps = ncps)
par(old_par)

# Cutoffs for infinite series of the last example
K <- 1e7
log_weights<- -3 * log(1:K)
log_dfs <- log(5) + log(1:K)
(cutoff <- cutoff_wschisq(thre = 10^(-(1:4)), weights = log_weights,
                          dfs = log_dfs, log = TRUE))

# Approximation
x <- seq(0, 25, l = 100)
l <- length(cutoff$mean)
main <- expression(sum(k^(-3) * chi[5 * k]^2, k == 1, K))
col <- viridisLite::viridis(l)
plot(x, d_wschisq(x, weights = exp(log_weights[1:cutoff$mean[l]]),
                  dfs = exp(log_dfs[1:cutoff$mean[l]])), type = "l",
     ylab = "Density", col = col[l], lwd = 3)
for(i in rev(seq_along(cutoff$mean)[-l])) {
  lines(x, d_wschisq(x, weights = exp(log_weights[1:cutoff$mean[i]]),
                     dfs = exp(log_dfs[1:cutoff$mean[i]])), col = col[i])
}
legend("topright", legend = paste0(rownames(cutoff), " (", cutoff$mean, ")"),
       lwd = 2, col = col)
```

Index