Package 'pdSpecEst'

December 10, 2017

Type Package

Title An Analysis Toolbox for Hermitian Positive Definite Matrices

Version 1.2.1

Description An implementation of data analysis tools for samples of symmetric or Hermitian positive definite matrices, such as collections of covariance matrices or spectral density matrices. The tools in this package can be used to perform (i) intrinsic manifold wavelet regression and clustering for curves (1D) or surfaces (2D) of Hermitian positive definite matrices, and (ii) exploratory data analysis and inference for samples of (curves of) Hermitian positive definite matrices by means of intrinsic manifold data depth and manifold rank-based hypothesis tests.

URL https://github.com/JorisChau/pdSpecEst, https://jchau.shinyapps.io/pdSpecEst/

Depends R (>= 3.3.1)

License GPL-2

LazyData TRUE

Imports multitaper, Rcpp, ddalpha

RoxygenNote 6.0.1

LinkingTo Rcpp, RcppArmadillo (>= 0.7.500.0.0)

SystemRequirements GNU make, C++11

Suggests knitr, rmarkdown, testthat, grid, ggplot2, reshape2, viridis, ggthemes

VignetteBuilder knitr

NeedsCompilation yes

Author Joris Chau [aut, cre]

Maintainer Joris Chau <j.chau@uclouvain.be>

Repository CRAN

Date/Publication 2017-12-10 15:10:00 UTC

R topics documented:

Expm . . . 2
H.coeff . . . 3
InvWavTransf1D . . . 4
InvWavTransf2D . . . 6
Logm . . . 8
Mid . . . 8
ParTrans . . . 9
pdCART . . . 10
pdConfInt1D . . . 12
pdDepth . . . 15
pdDist . . . 16
pdMean . . . 17
pdNeville . . . 18
pdPgram . . . 20
pdPgram2D . . . 22
pdPolynomial . . . 24
pdRankTests . . . 25
pdSpecClust1D . . . 27
pdSpecClust2D . . . 30
pdSpecEst . . . 32
pdSpecEst1D . . . 33
pdSpecEst2D . . . 35
pdSplineReg . . . 37
rARMA . . . 38
rExamples . . . 40
rExamples2D . . . 41
WavTransf1D . . . 42
WavTransf2D . . . 44

Index **46**

`Expm` *Exponential map*

Description

`Expm(P, H)` computes the projection of a Hermitian matrix `H` from the tangent space at a Hermitian PD matrix `P` to the Riemannian manifold of Hermitian PD matrices via the exponential map as in (Pennec, 2006). This is the unique inverse of the logarithmic map `Logm`.

Usage

```
Expm(P, H)
```

Arguments

P	a Hermitian positive definite matrix.
H	a Hermitian matrix (of equal dimension as P).

References

Pennec, X. (2006). Intrinsic statistics on Riemannian manifolds: Basic tools for geometric measurements. *Journal of Mathematical Imaging and Vision* 25(1), 127-154.

See Also

Logm, ParTrans

Examples

```
H <- matrix(complex(real = rnorm(9), imaginary = rnorm(9)), nrow = 3)
diag(H) <- rnorm(3)
H[lower.tri(H)] <- t(Conj(H))[lower.tri(H)]
p <- matrix(complex(real = rnorm(9), imaginary = rnorm(9)), nrow = 3)
P <- t(Conj(p)) %*% p
Expm(P, H)
```

H.coeff *Orthonormal basis expansion of a Hermitian matrix*

Description

H.coeff expands a (d, d)-dimensional Hermitian matrix H with respect to an orthonormal (in terms of the Frobenius inner product) basis of the space of Hermitian matrices, i.e. it transforms H into a numeric vector of d^2 real-valued basis coefficients. This is possible as the space of Hermitian matrices is a real vector space. Let E_{nm} be a (d, d)-dimensional zero matrix with a 1 at location (n, m). The orthonormal basis contains the following matrix elements: let $1 \leq n \leq d$ and $1 \leq m \leq d$,

If n == m the real matrix element E_{nn}

If n < m the complex matrix element $2i/\sqrt{2}E_{nm}$

If n > m the real matrix element $2/\sqrt{2}E_{nm}$

The orthonormal basis coefficients are ordered by scanning through the matrix H in a row-by-row fashion.

Usage

```
H.coeff(H, inverse = F)
```

Arguments

H	if !isTRUE(inverse), a (d, d)-dimensional Hermitian matrix; if isTRUE(inverse), a numeric vector of length d^2 with d an integer.
inverse	a logical value that determines whether the forward basis transform (inverse = FALSE) or the inverse basis transform (inverse = TRUE) should be applied.

Value

If inverse = FALSE takes as input a (d, d)-dimensional Hermitian matrix and outputs a numeric vector of length d^2 containing the real-valued basis coefficients. If inverse = TRUE takes as input a d^2-dimensional numeric vector of basis coefficients and outputs the corresponding (d, d)-dimensional Hermitian matrix.

Examples

```
## random Hermitian matrix
H <- matrix(complex(real = rnorm(9), imaginary = rnorm(9)), nrow = 3)
diag(H) <- rnorm(3)
H[lower.tri(H)] <- t(Conj(H))[lower.tri(H)]

## orthonormal basis expansion
h <- H.coeff(H)
H1 <- H.coeff(h, inverse = TRUE) ## reconstructed Hermitian matrix
all.equal(H, H1)
```

InvWavTransf1D *Inverse average-interpolation 1D wavelet transform*

Description

InvWavTransf1D computes the inverse intrinsic average-interpolation (AI) wavelet transform of an array of coarsest-scale HPD midpoints combined with a pyramid of Hermitian wavelet coefficients as explained in (Chau and von Sachs, 2017). This is the inverse operation of the function WavTransf1D.

Usage

```
InvWavTransf1D(D, M0, order = 5, jmax, periodic = F,
  metric = "Riemannian", progress = F, ...)
```

Arguments

D	a list of arrays containing the pyramid of wavelet coefficients, where each array contains the (d, d)-dimensional wavelet coefficients from the finest wavelet scale j = jmax up to the coarsest wavelet scale j = 0. This is the same format as the $D component given as output by WavTransf1D.

`M0`	a numeric array containing the midpoint(s) at the coarsest scale `j = 0` in the midpoint pyramid. This is the same format as the `$M0` component given as output by `WavTransf1D`.
`order`	an odd integer larger or equal to 1 corresponding to the order of the intrinsic AI refinement scheme, defaults to `order = 5`. Note that if `order > 9`, the computational cost significantly increases as the wavelet transform no longer uses a fast wavelet refinement scheme based on pre-determined weights.
`jmax`	the maximum scale (resolution) up to which the HPD midpoints (i.e. scaling coefficients) are reconstructed. If `jmax` is not specified it is set equal to the resolution in the finest wavelet scale `jmax = length(D)`.
`periodic`	a logical value determining whether the curve of HPD matrices can be reflected at the boundary for improved wavelet refinement schemes near the boundaries of the domain. This is useful for spectral matrix estimation, where the spectral matrix is a symmetric and periodic curve in the frequency domain. Defaults to `periodic = F`.
`metric`	the metric that the space of HPD matrices is equipped with. The default choice is `"Riemannian"`, but this can also be one of: `"logEuclidean"`, `"Cholesky"`, `"rootEuclidean"` or `"Euclidean"`. The inverse intrinsic AI wavelet transform fundamentally relies on the chosen metric.
`progress`	should a console progress bar be displayed? Defaults to `progress = F`.
`...`	additional arguments for internal use.

Details

The input list of arrays `D` and array `M0` correspond to a pyramid of wavelet coefficients and the coarsest-scale HPD midpoints respectively, both are structured in the same way as in the output of `WavTransf1D`. As in the forward AI wavelet transform, if the refinement order is an odd integer smaller or equal to 9, the function computes the inverse wavelet transform using a fast wavelet refinement scheme based on weighted geometric averages with pre-determined weights as explained in (Chau and von Sachs, 2017a). If the refinement order is an odd integer larger than 9, the wavelet refinement scheme is based on intrinsic polynomial prediction using Neville's algorithm on the Riemannian manifold. The function computes the inverse intrinsic AI wavelet transform equipping the space of HPD matrices with one of the following metrics: (i) Riemannian metric (default) as in (Bhatia, 2009, Chapter 6), (ii) log-Euclidean metric, the Euclidean inner product between matrix logarithms, (iii) Cholesky metric, the Euclidean inner product between Cholesky decompositions, (iv) Euclidean metric and (v) root-Euclidean metric. The default choice (Riemannian) has several appealing properties not shared by the other metrics, see (Chau and von Sachs, 2017a) for more details.

Value

Returns a (d, d, m)-dimensional array corresponding to a curve of length m of (d, d)-dimensional HPD matrices.

References

Chau, J. and von Sachs, R. (2017) *Positive definite multivariate spectral estimation: a geometric wavelet approach*. Available at http://arxiv.org/abs/1701.03314.

See Also

WavTransf1D, pdSpecEst1D, pdNeville

Examples

```
P <- rExamples(2^8, example = "bumps")
P.wt <- WavTransf1D(P$f) ## forward transform
P.f <- InvWavTransf1D(P.wt$D, P.wt$M0) ## backward transform
all.equal(P.f, P$f)
```

InvWavTransf2D *Inverse average-interpolation 2D wavelet transform*

Description

InvWavTransf2D computes the inverse intrinsic average-interpolation (AI) wavelet transform of an array of coarsest-scale HPD midpoints combined with a 2D pyramid of Hermitian wavelet coefficients. This is the inverse operation of the function WavTransf2D.

Usage

```
InvWavTransf2D(D, M0, order = c(3, 3), jmax, metric = "Riemannian",
  progress = T, ...)
```

Arguments

D	a list of arrays containing the 2D pyramid of wavelet coefficients, where each array contains the (d, d)-dimensional wavelet coefficients from the finest wavelet scale `j = jmax` up to the coarsest wavelet scale `j = 0`. This is the same format as the `$D` component given as output by WavTransf2D.
M0	a numeric array containing the midpoint(s) at the coarsest scale `j = 0` in the 2D midpoint pyramid. This is the same format as the `$M0` component given as output by WavTransf2D.
order	a 2-dimensional numeric vector of odd integers larger or equal to 1 corresponding to the marginal orders of the intrinsic 2D AI refinement scheme, defaults to `order = c(5, 5)`. Note that if `max(order) > 9`, the computational cost significantly increases as the wavelet transform no longer uses a fast wavelet refinement scheme based on pre-determined weights.
jmax	the maximum scale (resolution) up to which the 2D surface of HPD midpoints (i.e. scaling coefficients) are reconstructed. If `jmax` is not specified it is set equal to the resolution in the finest wavelet scale `jmax = length(D)`.
metric	the metric that the space of HPD matrices is equipped with. The default choice is `"Riemannian"`, but this can also be one of: `"logEuclidean"`, `"Cholesky"`, `"rootEuclidean"` or `"Euclidean"`. The inverse intrinsic 2D AI wavelet transform fundamentally relies on the chosen metric.

progress	should a console progress bar be displayed? Defaults to progress = T.
...	additional arguments for internal use.

Details

The input list of arrays D and array M0 correspond to a 2D pyramid of wavelet coefficients and the coarsest-scale HPD midpoints respectively, both are structured in the same way as in the output of WavTransf2D. As in the forward AI wavelet transform, if both marginal refinement orders are smaller or equal to 9, the function computes the inverse wavelet transform using a fast wavelet refinement scheme based on weighted geometric averages with pre-determined weights. If one of the marginal refinement order is an odd integer larger than 9, the wavelet refinement scheme is based on intrinsic polynomial surface prediction using Neville's algorithm on the Riemannian manifold (pdNeville). By default InvWavTransf2D computes the inverse intrinsic 2D AI wavelet transform equipping the space of HPD matrices with (i) the Riemannian metric. Instead, the space of HPD matrices can also be equipped with one of the following metrics; (ii) log-Euclidean metric, the Euclidean inner product between matrix logarithms, (iii) Cholesky metric, the Euclidean inner product between Cholesky decompositions, (iv) Euclidean metric and (v) root-Euclidean metric. The default choice (Riemannian) has several appealing properties not shared by the other metrics, see (Chau and von Sachs, 2017a) for more details.

Value

Returns a (d, d, n_1, n_2)-dimensional array corresponding to a rectangular surface of size n_1 by n_2 of (d, d)-dimensional HPD matrices.

References

Chau, J. and von Sachs, R. (2017) *Positive definite multivariate spectral estimation: a geometric wavelet approach*. Available at http://arxiv.org/abs/1701.03314.

See Also

WavTransf2D, pdSpecEst2D, pdNeville

Examples

```
P <- rExamples2D(c(2^4, 2^4), 2, example = "tvar")
P.wt <- WavTransf2D(P$f) ## forward transform
P.f <- InvWavTransf2D(P.wt$D, P.wt$M0) ## backward transform
all.equal(P.f, P$f)
```

Logm *Logarithmic map*

Description

`Logm(P, Q)` computes the projection of a Hermitian PD matrix `Q` on the Riemannian manifold to the tangent space attached at the Hermitian PD matrix `P` via the logarithmic map as in (Pennec, 2006). This is the unique inverse of the exponential map `Expm`.

Usage

```
Logm(P, Q)
```

Arguments

`P`	a Hermitian positive definite matrix.
`Q`	a Hermitian positive definite matrix (of equal dimension as `P`).

References

Pennec, X. (2006). Intrinsic statistics on Riemannian manifolds: Basic tools for geometric measurements. *Journal of Mathematical Imaging and Vision* 25(1), 127-154.

See Also

`Expm`, `ParTrans`

Examples

```
q <- matrix(complex(real = rnorm(9), imaginary = rnorm(9)), nrow = 3)
Q <- t(Conj(q)) %*% q
p <- matrix(complex(real = rnorm(9), imaginary = rnorm(9)), nrow = 3)
P <- t(Conj(p)) %*% p
Logm(P, Q)
```

Mid *Geodesic midpoint between HPD-matrices*

Description

`Mid` calculates the geodesic midpoint between two Hermitian PD matrices as in (Bhatia, 2009, Chapter 6).

Usage

```
Mid(A, B)
```

Arguments

A, B	Hermitian positive definite matrices (of equal dimension).

References

Bhatia, R. (2009). *Positive Definite Matrices*. New Jersey: Princeton University Press.

See Also

pdMean

Examples

```
a <- matrix(complex(real = rnorm(9), imaginary = rnorm(9)), nrow = 3)
A <- t(Conj(a)) %*% a
b <- matrix(complex(real = rnorm(9), imaginary = rnorm(9)), nrow = 3)
B <- t(Conj(b)) %*% b
Mid(A, B)
```

ParTrans *Parallel transport*

Description

ParTrans() computes the parallel transport on the manifold of HPD matrices equipped with the Riemannian metric as described in e.g. (Chau and von Sachs, 2017a). That is, the function computes the parallel transport of a vector (Hermitian matrix) W in the tangent space at the point (HPD matrix) P along a geodesic curve in the direction of the vector V in the tangent space at P for a unit time step.

Usage

```
ParTrans(P, V, W)
```

Arguments

P	a (d, d)-dimensional HPD matrix.
V	a (d, d)-dimensional Hermitian matrix corresponding to a vector in the tangent space of P.
W	a (d, d)-dimensional Hermitian matrix corresponding to a vector in the tangent space of P.

Value

a (d, d)-dimensional Hermitian matrix corresponding to the parallel transportation of W in the direction of V along a geodesic curve for a unit time step.

References

Chau, J. and von Sachs, R. (2017a). *Positive definite multivariate spectral estimation: a geometric wavelet approach*. Available at http://arxiv.org/abs/1701.03314.

See Also

Expm, Logm

Examples

```
## Transport the vector W to the tangent space at the identity
W <- matrix(complex(real = rnorm(9), imaginary = rnorm(9)), nrow = 3)
diag(W) <- rnorm(3)
W[lower.tri(W)] <- t(Conj(W))[lower.tri(W)]
p <- matrix(complex(real = rnorm(9), imaginary = rnorm(9)), nrow = 3)
P <- t(Conj(p)) %*% p

ParTrans(P, Logm(P, diag(3)), W) ## whitening transport
```

---

pdCART *Tree-structured trace thresholding of wavelet coefficients*

Description

pdCart() performs hard tree-structured thresholding of the wavelet coefficients obtained with WavTransf1D or WavTransf2D based on the trace of the whitened wavelet coefficients, see e.g. (Chau and von Sachs, 2017).

Usage

```
pdCART(D, D.white, order, alpha = 1, tree = T, ...)
```

Arguments

D	a list of wavelet coefficients as obtained from WavTransf1D or WavTransf2D.
D.white	a list of whitened wavelet coefficients as obtained from WavTransf1D or WavTransf2D.
order	the order(s) of the intrinsic 1D or 2D AI refinement scheme as in WavTransf1D and WavTransf2D.
alpha	tuning parameter specifying the sparsity parameter as alpha times the universal threshold.
tree	logical value, if tree = T performs tree-structured thresholding, otherwise performs non-tree-structured hard thresholding of the coefficients.
...	additional parameters for internal use.

Details

Depending on the structure of the input list of arrays `D` the function performs 1D or 2D tree-structured thresholding of wavelet coefficients. The optimal tree of wavelet coefficients is found by minimization of the *complexity penalized residual sum of squares* (CPRESS) criterion in (Donoho, 1997), via a fast tree-pruning algorithm. By default, the sparsity parameter is set equal to `alpha` times the universal threshold $\sigma_w\sqrt{(2\log(n))}$, where σ_w^2 is the noise variance of the traces of the whitened wavelet coefficients determined from the finest wavelet scale and n is the total number of coefficients. By default, `alpha = 1`, with `alpha = 0`, the sparsity parameter is zero and we do not threshold any coefficients.

Value

Returns a list with two components:

`w`	a list of logical values specifying which coefficients to keep, with each list component corresponding to an individual wavelet scale.
`D_w`	the list of thresholded wavelet coefficients, with each list component corresponding to an individual wavelet scale.

Note

For thresholding of 1D wavelet coefficients, the noise variance of the traces of the whitened wavelet coefficients is constant across scales as shown in (Chau and von Sachs, 2017a). For thresholding of 2D wavelet coefficients, there is a discrepancy between the constant noise variance of the traces of the whitened wavelet coefficients of the first `abs(J1 - J2)` scales and the remaining scales, where $J_1 = \log_2(n_1)$ and $J_2 = \log_2(n_2)$ with n_1 and n_2 the dyadic number of observations in each marginal direction of the 2D rectangular tensor grid. The reason is that the variances of the traces of the whitened coefficients are not homogeneous between: (i) scales at which the 1D wavelet refinement scheme is applied and (ii) scales at which the 2D wavelet refinement scheme is applied. To correct for this discrepancy, the variances of the coefficients at the 2D wavelet scales are normalized by the noise variance determined from the finest wavelet scale. The variances of the coefficients at the 1D wavelet scales are normalized using the analytic noise variance of the traces of the whitened coefficients for a grid of complex random Wishart matrices, which corresponds to the distributional behavior of the pseudo HPD periodogram matrices, or the asymptotic distributional behavior of the actual HPD periodogram matrices. Note that if the 2D time-frequency grid of is a square grid, i.e. $n_1 = n_2$, the variances of the traces of the whitened coefficients are again homogeneous across all wavelet scales.

References

Chau, J. and von Sachs, R. (2017a). *Positive definite multivariate spectral estimation: a geometric wavelet approach*. Available at http://arxiv.org/abs/1701.03314.

Donoho, D.L. (1997). *CART and best-ortho-basis: a connection*. Annals of Statistics, 25(5), 1870-1911.

See Also

`WavTransf1D`, `InvWavTransf1D`, `WavTransf2D`, `InvWavTransf2D`

Examples

```
## 1D
X <- rExamples(256, example = "bumps")
Coeffs <- WavTransf1D(X$per)
pdCART(Coeffs$D, Coeffs$D.white, order = 5)$w ## logical tree of non-zero coefficients

## Not run:
## 2D
P <- rExamples2D(c(2^7, 2^7), 3, example = "tvar")$per
Coeffs <- WavTransf2D(P, jmax = 5)
pdCART(Coeffs$D, Coeffs$D.white, order = c(3, 3))$w

## End(Not run)
```

---

pdConfInt1D *Intrinsic depth-based bootstrap confidence regions*

Description

pdConfInt1D constructs depth-based parametric bootstrap confidence regions as explained in (Chau and von Sachs, 2017a) for a wavelet-based spectral matrix estimator obtained with pdSpecEst1D based on intrinsic manifold data depths as in pdDepth.

Usage

```
pdConfInt1D(f, alpha = 0.05, ci.region, boot.samples = 1000,
  metric = "logEuclidean", depth = "spatial", return.f = F, f.0 = NULL,
  ...)
```

Arguments

f	a (d, d, m)-dimensional array corresponding to a wavelet-denoised HPD (d, d)-dimensional spectral estimate at m different frequencies given as output with pdSpecEst1D.
alpha	a numerical vector of quantiles (between 0 and 1) determining the $100(1-\alpha)\%$-confidence levels, defaults to alpha = 0.05.
ci.region	a 2-dimensional numeric vector c(min.ci, max.ci) specifying the domain of the simultaneous confidence region. The frequency domain $[0, \pi]$ is normalized to a unit interval, e.g. ci.region = c(0.5, 1) constructs a simultaneous confidence region over the second half of the frequency domain. ci.region can also be a $(2, L)$-dimensional matrix, where each column of the matrix specificies the domain of an individual simultaneous confidence region. If ci.region is not specified, it defaults to the unit interval, i.e. the entire frequency domain.
boot.samples	number of bootstrap spectral estimates, defaults to boot.samples = 1e3.

`metric`	the metric that the space of HPD matrices is equipped with. The default choice is `"logEuclidean"`, but this can also be one of: `"Riemannian"`, `"Cholesky"`, `"rootEuclidean"` or `"Euclidean"`. This argument is passed on to the spectral estimator in `pdSpecEst1D` and the depth calculation in `pdDepth`. The default choice is the Log-Euclidean metric as the computational effort is significantly reduced in comparison to e.g. the Riemannian metric.
`depth`	the data depth measure, one of `'zonoid'`, `'gdd'`, or `'spatial'` corresponding to the manifold zonoid depth, geodesic distance depth, and manifold spatial depth respectively.
`return.f`	a logical value, if `return.f = TRUE` the function also returns the $100(1-\alpha)\%$-most central bootstrapped spectral estimates. By default `return.f = FALSE` to save memory use of the returned object.
`f.0`	optional target spectrum in the same format as `f`. The function computes the depth of the target spectrum with respect to the cloud of bootstrap spectral estimates and checks whether the target `f.0` is covered by the constructed depth-based confidence region.
`...`	additional arguments passed on to the functions `pdPgram` and `pdSpecEst1D` used internally.

Details

The parametric bootstrap procedure exploits the data generating process of a stationary time series via its Cramer representation, and is equivalent to e.g. (Fiecas and Ombao, 2016) among others. Given a consistent wavelet-based spectral estimator `f` obtained with `pdSpecEst1D` corresponding to a sequence of m HPD matrices along the frequency range $(0, \pi]$, where w.l.o.g. m is assumed to be even, the bootstrap confidence regions are constructed as:

1. Generate a bootstrapped time series trace `X_b` of length n via its Cramer representation using complex normal random variates and transfer functions given by the Hermitian square root of the estimate `f`.
2. Compute a bootstrapped wavelet-based spectral estimate `f_b` based on `X_b`.
3. Repeat 1. and 2. many times. Construct $100(1-\alpha)\%$-pointwise or -simultaneous confidence sets by taking the $100(1-\alpha)\%$-most central depth-based region in the cloud of bootstrapped spectral estimates based on one of the intrinsic manifold data depths in `pdDepth`.

The depth-based confidence balls (maximum/minimum depths and radii) are given by the component `depth.CI`. In particular, a given curve of HPD matrices is covered by the confidence ball if its manifold data depth with respect to the cloud of bootstrapped spectral estimates is above the minimum depth in the given confidence ball.
If `return.f = TRUE`, the function also returns the $100(1-\alpha)\%$-most central bootstrapped spectral estimates, by default `return.f = FALSE` to save memory use of the returned object.
If, in addition, we supply a target spectrum `f.0` (i.e. a numerical array with the same dimensions as `f`), the function checks whether the target `f.0` is covered by the constructed confidence regions or not by computing the depth of `f.0` with respect to the cloud of bootstrapped spectral estimates.

Value

The function returns a list with the following components:

`depth.CI` a matrix or a list of matrices, with each matrix giving the depth-based confidence ball (maximum depth, minimum depth and radius) at an individual region in the frequency domain as specified by the argument `ci.region` and at the various $100(1-\alpha)\%$-confidence levels specified by the argument `alpha`.

`f.CI` If `return.f = TRUE` returns the $100(1-\alpha)\%$-most central bootstrapped spectral estimates at each of the individual regions in the frequency domain specified by the argument `ci.region` at the level corresponding to the first argument in the input vector α. If `return.f = FALSE` returns `NULL`.

If a target spectrum `f.0` is supplied, i.e. `!is.null(f.0)`, the returned list includes the additional components:

`cover.f` a list of logical vectors checking whether the constructed confidence balls cover the target spectrum `f.0`. Each vector corresponds to an individual region in the frequency domain as specified by the argument `ci.region` at the various $100(1-\alpha)\%$-confidence levels specified by the argument `alpha`.

`depth.f` a numeric vector of depth values of the target spectrum `f.0` with respect to the cloud of bootstrapped spectral estimates at each of the individual regions in the frequency domain as specified by the argument `ci.region`.

References

Chau, J. and von Sachs, R. (2017a). *Positive definite multivariate spectral estimation: a geometric wavelet approach*. Available at http://arxiv.org/abs/1701.03314.

Chau, J., Ombao, H., and von Sachs, R. (2017b). *Data depth and rank-based tests for covariance and spectral density matrices*. Available at http://arxiv.org/abs/1706.08289.

Fiecas, M., Ombao, H. (2016). *Modeling the evolution of dynamic brain processes during an associative learning experiment*. Journal of the American Statistical Association, 111(516), 1440-1453.

Examples

```
## Not run:
set.seed(123)
example <- rExamples(2^8, example = "gaussian")
f.hat <- pdSpecEst1D(example$per, metric = "logEuclidean")
boot.ci <- pdConfInt1D(f.hat$f, alpha = c(0.1, 0.05, 0.01), ci.region = c(0.45, 0.55),
                       boot.samples = 1E3, f.0 = example$f)
boot.ci

## End(Not run)
```

pdDepth *Data depth for HPD matrices*

Description

pdDepth calculates the data depth of a Hermitian PD matrix with respect to a given data cloud (i.e. a sample or collection) of Hermitian PD matrices, or the integrated data depth of a sequence (curve) of Hermitian PD matrices with respect to a given data cloud of sequences (curves) of Hermitian PD matrices.

Usage

```
pdDepth(y = NULL, X, method = c("zonoid", "gdd", "spatial"),
  metric = "Riemannian")
```

Arguments

y	either a (d, d)-dimensional array corresponding to a $(d \times d)$-dimensional Hermitian PD matrix, or a (d, d, n)-dimensional array corresponding to a sequence or curve of Hermitian PD matrices. Defaults to NULL, in which case the data depth of each individual object in X with respect to the data cloud X itself is calculated.
X	depending on the input y either a (d, d, S)-dimensional array corresponding to a data cloud of S individual Hermitian PD matrices, or a (d, d, n, S)-dimensional array corresponding to a data cloud of S sequences or curves of n individual Hermitian PD matrices.
method	the data depth measure, one of 'zonoid', 'gdd', or 'spatial' corresponding to the manifold zonoid depth, geodesic distance depth, and manifold spatial depth respectively.
metric	the metric that the space of HPD matrices is equipped with. The default choice is "Riemannian", but this can also be one of: "logEuclidean", "Cholesky", "rootEuclidean" or "Euclidean".

Details

Available pointwise or integrated manifold data depth functions for samples of Hermitian PD matrices are (i) manifold zonoid depth, (ii) geodesic distance depth and (iii) manifold spatial depth. The various data depth measures and their theoretical properties are described in (Chau, Ombao, and von Sachs, 2017b). If y is a $(d \times d)$-dimensional Hermitian PD matrix (i.e. (d, d)-dimensional array), X should be a (d, d, S)-dimensional array of Hermitian PD matrices and the pointwise data depth values are computed. If y is a sequence of $(d \times d)$-dimensional Hermitian PD matrices of length n (i.e. (d, d, n)-dimensional array), X should be a (d, d, n, S)-dimensional array of sequences of Hermitian PD matrices and the integrated data depth values according to (Chau, Ombao, and von Sachs, 2017b) are computed. If is.null(y), the data depth of each individual object (i.e. a Hermitian PD matrix or a sequence of Hermitian PD matrices) in X is computed with respect to the data cloud X.

The function computes the intrinsic data depths based on the metric space of HPD matrices equipped

with one of the following metrics: (i) Riemannian metric (default) as in (Bhatia, 2009, Chapter 6), (ii) log-Euclidean metric, the Euclidean inner product between matrix logarithms, (iii) Cholesky metric, the Euclidean inner product between Cholesky decompositions, (iv) Euclidean metric and (v) root-Euclidean metric. The default choice (Riemannian) has several appealing properties not shared by the other metrics, see (Chau, Ombao and von Sachs, 2017b) for more details.

Value

If `!is.null(y)`, `pdDepth` returns the numeric depth value of `y` with respect to `X`. If `is.null(y)`, `pdDepth` returns a numeric vector of length `S` corresponding to the depth values of each individual object in `X` with respect to `X` itself.

Note

Note that the data depth computations in the metric space equipped with the Riemannian metric may be significantly slower than the depth computations based on one of the alternative metrics.

References

Chau, J., Ombao, H., and von Sachs, R. (2017b). *Data depth and rank-based tests for covariance and spectral density matrices*. Available at http://arxiv.org/abs/1706.08289.

See Also

`pdDist`, `pdRankTests`

Examples

```
## Pointwise depth
X1 <- replicate(50, Expm(diag(2), H.coeff(rnorm(4), inverse = TRUE)))
pdDepth(y = diag(2), X1, method = "gdd") ## depth of one point
pdDepth(X = X1, method = "gdd") ## depth of each point in the data cloud

## Integrated depth
X2 <- replicate(50, replicate(5, Expm(diag(2), H.coeff(rnorm(4), inverse = TRUE))))
pdDepth(y = replicate(5, diag(2)), X2, method = "zonoid", metric = "logEuclidean")
pdDepth(X = X2, method = "zonoid", metric = "logEuclidean")
```

---

pdDist *Compute distance between two HPD matrices*

Description

`pdDist` calculates a distance between two Hermitian PD matrices.

Usage

```
pdDist(A, B, method = "Riemannian")
```

Arguments

A, B	Hermitian positive definite matrices (of equal dimension).
method	the distance measure, one of 'Riemannian', 'logEuclidean', 'Cholesky', 'Euclidean', 'squareRoot' or 'Procrustes'. Defaults to 'Riemannian'.

Details

Available distance measures between two Hermitian PD matrices are (i) Riemannian distance (default) as in (Bhatia, 2009, Chapter 6), (ii) log-Euclidean distance, the Euclidean distance between matrix logarithms, (iii) Cholesky distance, the Euclidean distance between Cholesky decompositions, (iv) Euclidean distance, (v) root-Euclidean distance and (vi) Procrustes distance as in (Dryden et al., 2009). In particular, pdDist generalizes the function shapes::distcov to compute the distance between two symmetric positive definite matrices to the distance between two Hermitian positive definite matrices.

References

Bhatia, R. (2009). *Positive Definite Matrices*. New Jersey: Princeton University Press.

Dryden, I.L., Koloydenko, A., Zhou, D. (2009). Non-Euclidean statistics for covariance matrices, with applications to diffusion tensor imaging. *Annals of Applied Statistics*, 3(3), 1102-1123.

Examples

```
a <- matrix(complex(real = rnorm(9), imaginary = rnorm(9)), nrow = 3)
A <- t(Conj(a)) %*% a
b <- matrix(complex(real = rnorm(9), imaginary = rnorm(9)), nrow = 3)
B <- t(Conj(b)) %*% b
pdDist(A, B) ## Riemannian distance
```

---

pdMean *Weighted geometric mean of HPD matrices*

Description

pdMean calculates an (approximate) weighted geometric mean of S different $(d \times d)$-dimensional Hermitian PD matrices based on the Riemannian metric by the fast recursive algorithm in (Chau and von Sachs, 2017) or the slower but more accurate gradient descent algorithm in (Pennec, 2006). By default, the unweighted geometric mean is computed.

Usage

```
pdMean(M, w, grad_desc = F, max_iter = 1000, tol, ...)
```

Arguments

M	a (d, d, S)-dimensional array of Hermitian PD matrices.
w	an S-dimensional nonnegative weight vector, such that sum(w) = 1.
grad_desc	a logical value deciding if the gradient descent algorithm be used, defaults to FALSE.
max_iter	maximum number of iterations in gradient descent algorithm, only used if isTRUE(grad_desc).
tol	optional tolerance parameter in gradient descent algorithm, only used if isTRUE(grad_desc), defaults to .Machine$double.eps.
...	additional arguments for internal usage.

References

Chau, J. and von Sachs, R. (2017). *Positive definite multivariate spectral estimation: a geometric wavelet approach*. Available at http://arxiv.org/abs/1701.03314.

Pennec, X. (2006). Intrinsic statistics on Riemannian manifolds: Basic tools for geometric measurements. *Journal of Mathematical Imaging and Vision* 25(1), 127-154.

See Also

Mid

Examples

```
m <- function(){
 X <- matrix(complex(real=rnorm(9), imaginary=rnorm(9)), nrow=3)
 t(Conj(X)) %*% X
}
M <- replicate(100, m())
z <- rnorm(100)
w <- abs(z)/sum(abs(z))
Ave <- pdMean(M, w)
```

---

pdNeville — *Polynomial interpolation of curves (1D) or surfaces (2D) of HPD matrices*

Description

pdNeville() performs intrinsic polynomial interpolation of curves or surfaces on the manifold of HPD matrices equipped with the Riemannian metric via iterative geodesic interpolation, see e.g. (Chau and von Sachs, 2017).

Usage

```
pdNeville(P, X, x, metric = "Riemannian")
```

Arguments

P	for polynomial curve interpolation, a (d, d, N)-dimensional array corresponding to a sequence of (d, d)-dimensional HPD matrices, i.e. control points, through which the interpolated polynomial curve passes. For polynomial surface interpolation, a (d, d, N_1, N_2)-dimensional array corresponding to a tensor product grid of (d, d)-dimensional matrices, i.e. control points, through which the interpolated polynomial surface passes.
X	for polynomial curve interpolation, a numeric vector of length N specifying the time points the interpolated polynomial passes through the control points `P`. For polynomial surface interpolation, a list with as elements two numeric vectors `x` and `y` of length N_1 and N_2 respectively. The numeric vectors specify the time points on the tensor product grid `expand.grid(X$x, X$y)` the interpolated polynomial passes trough the control points `P`.
x	for polynomial curve interpolation, a numeric vector specifying the time grid (resolution) at which the interpolated polynomial is estimated. For polynomial surface interpolation, a list with as elements two numeric vectors `x` and `y` specifying the time tensor product grid (resolution) `expand.grid(x$x, x$y)` at which the interpolated polynomial surface is estimated.
metric	the metric the space of HPD gets equipped with, by default `metric = "Riemannian"`, but instead this can also be set to `metric = "Euclidean"` to perform (standard) Euclidean polynomial interpolation of curves or surfaces in the space of HPD matrices.

Details

For polynomial curve interpolation, given N control points (i.e. HPD matrices), the degree of the interpolated polynomial is $N - 1$. For polynomial surface interpolation, given $N_1 \times N_2$ control points (i.e. HPD matrices) on a tensor product grid, the interpolated polynomial has bi-degree $(N_1 - 1, N_2 - 1)$. The function `pdNeville()` determines whether polynomial curve or polynomial surface interpolation has to be performed based on the function input.

Value

For polynomial curve interpolation, a `(d, d, length(x))`-dimensional array containing the interpolated polynomial of degree $N - 1$ at the time grid `x` passing through the control points `P` at times `X`. For polynomial surface interpolation, a `(d, d, length(x$x), length(x$y))`-dimensional array containing the interpolated polynomial of bi-degree $N_1 - 1, N_2 - 1$ at the time grid `expand.grid(x$x, x$y)` passing through the control points `P` at times `expand.grid(X$x, X$y)`.

References

Chau, J. and von Sachs, R. (2017a). *Positive definite multivariate spectral estimation: a geometric wavelet approach*. Available at http://arxiv.org/abs/1701.03314.

See Also

`pdPolynomial`

Examples

```
### Polynomial curve interpolation
P <- rExamples(100, example = 'gaussian')$f[, , 10*(1:5)]
P.poly <- pdNeville(P, (1:5)/5, (1:50)/50)
## Examine matrix-component (1,1)
plot((1:50)/50, Re(P.poly[1, 1, ]), type = "l") ## interpolated polynomial
lines((1:5)/5, Re(P[1, 1, ]), col = 2) ## control points

### Polynomial surface interpolation
P.surf <- array(P[, , 1:4], dim = c(2,2,2,2)) ## control points
P.poly <- pdNeville(P.surf, list(x = c(0, 1), y = c(0, 1)), list(x = (0:10)/10, y = (0:10)/10))
```

---

pdPgram *Tapered HPD periodogram matrix*

Description

Given a multivariate time series, pdPgram calculates a tapered HPD periodogram matrix based on averaging raw HPSD periodogram matrices of tapered multivariate time series segments.

Usage

```
pdPgram(X, B, method = c("multitaper", "bartlett"), bias.corr = F,
  nw = pi)
```

Arguments

X	an (n, d)-dimensional matrix corresponding to a multivariate time series, with the d columns corresponding to the components of the time series.
B	depending on method, either the number of orthogonal Slepian tapers, or the number of non-overlapping segments to compute Bartlett's averaged periodogram. By default, B = d, such that the averaged periodogram is guaranteed to be positive definite.
method	the tapering method, either "multitaper" or "bartlett" explained in the Details section below. Defaults to "multitaper".
bias.corr	should the Riemannian manifold bias-correction be applied to the HPD periodogram matrix? Defaults to FALSE.
nw	a positive numeric value corresponding to the time-bandwidth parameter of the Slepian tapering functions, see also dpss, defaults to nw = pi.

Details

If `method == "multitaper"`, `pdPgram` calculates a (d, d)-dimensional multitaper periodogram matrix based on B Discrete Prolate Spheroidal (i.e. Slepian) orthogonal tapering functions as in `dpss` applied to the d-dimensional time series `X`. If `method == "bartlett"`, `pdPgram` computes a Bartlett spectral estimator by averaging the periodogram matrices of `B` non-overlapping segments of the d-dimensional time series `X`. Note that Bartlett's spectral estimator is a specific (trivial) case of a multitaper spectral estimator with uniform orthogonal tapering windows.
If we perform additional periodogram matrix denoising in the space of HPD matrices equipped with the Riemannian metric or the Log-Euclidean metric, we should set `bias.corr = T`, which corrects for the asymptotic bias of the periodogram matrix on the manifold of HPD matrices equipped with the Riemannian or Log-Euclidean metric as described in (Chau and von Sachs, 2017). The pre-smoothed HPD periodogram matrix (i.e. an initial noisy HPD spectral estimator) can be given as input to the intrinsic wavelet HPD spectral estimation procedure in `pdSpecEst1D`. In this case, we set `bias.corr = F` (the default) as the appropriate bias-correction based on the chosen metric is applied by the function `pdSpecEst1D`.

Value

A list containing two components:

`freq`	vector of of $n/2$ frequencies in $[0, 0.5)$ at which the periodogram is computed.
`P`	a $(d, d, n/2)$-dimensional array containing the (d, d)-dimensional tapered periodogram matrices at frequencies corresponding to `freq`.

References

Bartlett, M.S. (1950). *Periodogram analysis and continuous spectra*. Biometrika 37 (1-2): 1-16.

Chau, J. and von Sachs, R. (2017). *Positive definite multivariate spectral estimation: a geometric wavelet approach*. Available at http://arxiv.org/abs/1701.03314.

Brockwell, P.J. and Davis, R.A. (1991). *Time series: Theory and Methods*. New York: Springer.

See Also

`pdPgram2D`, `dpss`

Examples

```
## ARMA(1,1) process: Example 11.4.1 in (Brockwell and Davis, 1991)

Phi <- array(c(0.7, 0, 0, 0.6, rep(0, 4)), dim = c(2, 2, 2))
Theta <- array(c(0.5, -0.7, 0.6, 0.8, rep(0, 4)), dim = c(2, 2, 2))
Sigma <- matrix(c(1, 0.71, 0.71, 2), nrow = 2)
ts.sim <- rARMA(200, 2, Phi, Theta, Sigma)
ts.plot(ts.sim$X) # plot generated time series traces
pgram <- pdPgram(ts.sim$X)
```

pdPgram2D *Tapered HPD time-varying periodogram matrix*

Description

Given a multivariate time series, pdPgram2D calculates a tapered HPD time-varying periodogram matrix based on averaging raw HPSD time-varying periodogram matrices of tapered multivariate time series segments.

Usage

```
pdPgram2D(X, B, tf.grid, method = c("dpss", "hermite"), nw = pi,
  bias.corr = F)
```

Arguments

X	an (n, d)-dimensional matrix corresponding to a multivariate time series, with the d columns corresponding to the components of the time series.
B	depending on method, either the number of orthogonal Slepian or Hermite tapering functions. By default, B = d, such that the multitaper periodogram is guaranteed to be positive definite.
tf.grid	a list with two components tf.grid$time and tf.grid$frequency specifying the rectangular grid of time-frequency points at which the multitaper periodogram is estimated. tf.grid$time should be a numeric vector of rescaled time points in (0,1). tf.grid$frequency should be a numeric vector of frequency points in (0,0.5), with 0.5 corresponding to the Nyquist frequency.
method	the tapering method, either "dpss" or "hermite" explained in the Details section below. Defaults to method = "dpss".
nw	a positive numeric value corresponding to the time-bandwidth parameter of the tapering functions, see also dpss, defaults to nw = pi. Both the Slepian and Hermite tapers are rescaled with the same time-bandwidth parameter.
bias.corr	should the Riemannian manifold bias-correction be applied to the HPD periodogram matrix? Defaults to FALSE.

Details

If method == "dpss", pdPgram2D calculates a (d, d)-dimensional multitaper time-varying periodogram matrix based on sliding B Discrete Prolate Spheroidal (i.e. Slepian) orthogonal tapering functions as in dpss applied to the d-dimensional time series X. If $B \geq d$, the multitaper time-varying periodogram matrix is guaranteed to be positive definite at each time-frequency point in the grid expand.grid(tf.grid$time, tf.grid$frequency). Essentially, the function computes a multitaper periodogram matrix (as in pdPgram) in each of a number of non-overlapping time series segments of X, with the time series segments centered around the (rescaled) time points in tf.grid$time. If method == "hermite", the function calculates a multitaper time-varying periodogram matrix replacing the Slepian tapers by orthogonal Hermite tapering functions.
If we perform additional periodogram matrix denoising in the space of HPD matrices equipped with

the Riemannian metric or the Log-Euclidean metric, we should set `bias.corr = T`, which corrects for the asymptotic bias of the periodogram matrix on the manifold of HPD matrices equipped with the Riemannian or Log-Euclidean metric as described in (Chau and von Sachs, 2017). The pre-smoothed HPD periodogram matrix (i.e. an initial noisy HPD spectral estimator) can be given as input to the intrinsic wavelet HPD spectral estimation procedure in `pdSpecEst1D`. In this case, we set `bias.corr = F` (the default) as the appropriate bias-correction based on the chosen metric is applied by the function `pdSpecEst1D`.

Value

A list containing two components:

`tf.grid`	a list with two components corresponding to the rectangular grid of time-frequency points at which the multitaper periodogram is returned.
`P`	a (d, d, m_1, m_2)-dimensional array with `m_1 = length(tf.grid$time)` and `m_2 = length(tf.grid$frequency)` containing the (d, d)-dimensional tapered periodogram matrices at the time-frequency points corresponding to `tf.grid`.

References

Bartlett, M.S. (1950). *Periodogram analysis and continuous spectra*. Biometrika 37 (1-2): 1-16.

Chau, J. and von Sachs, R. (2017). *Positive definite multivariate spectral estimation: a geometric wavelet approach*. Available at http://arxiv.org/abs/1701.03314.

Brockwell, P.J. and Davis, R.A. (1991). *Time series: Theory and Methods*. New York: Springer.

See Also

`pdPgram`, `dpss`

Examples

```
## Coefficient matrices
Phi1 <- array(c(0.4, 0, 0, 0.8, rep(0, 4)), dim = c(2, 2, 2))
Phi2 <- array(c(0.8, 0, 0, 0.4, rep(0, 4)), dim = c(2, 2, 2))
Theta <- array(c(0.5, -0.7, 0.6, 0.8, rep(0, 4)), dim = c(2, 2, 2))
Sigma <- matrix(c(1, 0.71, 0.71, 2), nrow = 2)

## Generate piecewise stationary time series
ts.Phi <- function(Phi) rARMA(2^9, 2, Phi, Theta, Sigma)$X
ts <- rbind(ts.Phi(Phi1), ts.Phi(Phi2))

pgram <- pdPgram2D(ts)
```

pdPolynomial *Generate intrinsic HPD polynomial curves*

Description

pdPolynomial() generates intrinsic polynomial curves on the manifold of HPD matrices equipped with the Riemannian metric according to the numerical integration procedure described in (Hinkle et al., 2014). Given an initial starting point p0 (i.e. HPD matrix) on the Riemannian manifold and the covariant derivatives up to order $k-1$ at p0, pdPolynomial() approximates the uniquely existing intrinsic polynomial curve of degree k passing through p0 with the given covariant derivatives up to order $k-1$ and vanishing higher order covariant derivatives.

Usage

```
pdPolynomial(p0, v0, delta.t = 0.01, steps = 100)
```

Arguments

p0	a (d, d)-dimensional HPD matrix specifying the starting point of the polynomial curve.
v0	a (d, d, k)-dimensional array corresponding to a sequence of covariant derivatives of order zero up to order $k-1$ at the starting point p0.
delta.t	a numeric value determining the incrementing step size in the numerical integration procedure. A smaller step size results in a higher resolution and therefore a more accurate approximation of the polynomial curve, defaults to delta.t = 0.01.
steps	number of incrementing steps in the numerical integration procedure, defaults to steps = 100.

Value

A (d, d, length(steps))-dimensional array containing the approximated intrinsic polynomial curve of degree k passing through p0 with the given covariant derivatives up to order $k-1$ and vanishing higher order covariant derivatives.

References

Hinkle, J., Fletcher, P. and Joshi, S. (2014). Intrinsic polynomials for regression on Riemannian manifolds. *Journal of Mathematical Imaging and Vision* 50, 32-52.

See Also

pdNeville

Examples

```
## First-order polynomial
p0 <- diag(3) ## HPD starting point
v0 <- array(H.coeff(rnorm(9), inverse = TRUE), dim = c(3, 3, 1)) ## zero-th order cov. derivative
P.poly <- pdPolynomial(p0, v0)

## First-order polynomials coincide with geodesic curves
geo <- function(A, B, t) Expm(A, t * Logm(A, B))
P.geo <- sapply(seq(0, 1, length = 100), function(t) geo(p0, P.poly[, , 100], t),
                simplify = "array")
all.equal(P.poly, P.geo)
```

---

pdRankTests *Rank-based hypothesis tests for HPD matrices*

Description

pdRankTests performs generalized rank-based hypothesis testing in the metric space of HPD matrices equipped with the Riemannian or Log-Euclidean metric for samples of Hermitian PD matrices or samples of sequences (curves) of Hermitian PD matrices as described in (Chau, Ombao, and von Sachs, 2017b).

Usage

```
pdRankTests(data, sample.sizes, test = c("rank.sum", "krusk.wall",
  "signed.rank", "bartels"), depth = c("gdd", "zonoid", "spatial"),
  metric = c("Riemannian", "logEuclidean"))
```

Arguments

data	either a (d, d, S)-dimensional array corresponding to an array of pooled individual samples of Hermitian PD matrices, or a (d, d, n, S)-dimensional array corresponding to an array of pooled individual samples of sequences of Hermitian PD matrices.
sample.sizes	a numeric vector corresponding to the individual sample sizes in the argument data, such that sum(sample.sizes) is equal to S. Not required for tests "signed-rank" and "bartels", as the sample sizes are automatically determined from data.
test	rank-based hypothesis testing procedure, one of "rank.sum", "krusk.wall", "signed.rank", "bartels" explained in the Details section below.
depth	data depth measure used in the rank-based tests, one of "gdd", "zonoid", or "spatial" corresponding to the geodesic distance depth, manifold zonoid depth and manifold spatial depth respectively. Defaults to "gdd". Not required for test "signed.rank".
metric	the metric that the space of HPD matrices is equipped with, either "Riemannian" or "logEuclidean". Defaults to "Riemannian".

Details

For samples of $(d \times d)$-dimensional Hermitian PD matrices with pooled sample size S, the argument `data` is a (d, d, S)-dimensional array of Hermitian PD matrices, where the individual samples are combined along the third array dimension. For samples of sequences of $(d \times d)$-dimensional Hermitian PD matrices with pooled sample size S, the argument `data` is a (d, d, n, S)-dimensional array of sequences of Hermitian PD matrices, where the individual samples are combined along the fourth array dimension. The argument `sample.sizes` specifies the sizes of the individual samples so that `sum(sample.sizes)` is equal to `S`.
The available generalized rank-based testing procedures (specified by the argument `test`) are:

`"rank.sum"` Manifold Wilcoxon rank-sum test to test for homogeneity of distributions of two independent samples of Hermitian PD matrices or samples of sequences of Hermitian PD matrices. The usual univariate ranks are replaced by data depth induced ranks via `pdDepth`.

`"krusk.wall"` Manifold Kruskal-Wallis test to test for homogeneity of distributions of more than two independent samples of Hermitian PD matrices or samples of sequences of Hermitian PD matrices. The usual univariate ranks are replaced by data depth induced ranks via `pdDepth`.

`"signed.rank"` Manifold signed-rank test to test for homogeneity of distributions of independent paired or matched samples of Hermitian PD matrices. The manifold signed-rank test is *not* based on data depth induced ranks, but on a specific difference score on the Riemannian manifold of Hermitian PD matrices.

`"bartels"` Manifold Bartels-von Neumann test to test for randomness (i.e. exchangeability) within a single independent sample of Hermitian PD matrices or a sample of sequences of Hermitian PD matrices. The usual univariate ranks are replaced by data depth induced ranks via `pdDepth`.

The function computes the generalized rank-based test statistics in the *complete* metric space of HPD matrices equipped with one of the following metrics: (i) Riemannian metric (default) as in (Bhatia, 2009, Chapter 6), or (ii) Log-Euclidean metric, the Euclidean inner product between matrix logarithms. The default Riemannian metric is invariant under congruence transformation by any invertible matrix, whereas the Log-Euclidean metric is only invariant under congruence transformation by unitary matrices, see (Chau, Ombao and von Sachs 2017b) for more details.

Value

The function returns a list with five components:

`test`	name of the rank-based test
`p.value`	p-value of the test
`statistic`	computed test statistic
`null.distr`	distribution of the test statistic under the null hypothesis
`depth.values`	computed data depth values (if available)

Note

The manifold signed-rank test also provides a valid test for equivalence of spectral matrices of two multivariate stationary time series based on the Hermitian PD periodogram matrices obtained via `pdPgram`, see (Chau, Ombao, and von Sachs, 2017b) for the details.

References

Chau, J., Ombao, H., and von Sachs, R. (2017b). *Data depth and rank-based tests for covariance and spectral density matrices*. Available at http://arxiv.org/abs/1706.08289.

Brockwell, P.J. and Davis, R.A. (1991). *Time series: Theory and Methods*. New York: Springer.

See Also

pdDepth, pdPgram

Examples

```
## null hypothesis is true
data <- replicate(100, Expm(diag(2), H.coeff(rnorm(4), inverse = TRUE)))
pdRankTests(data, sample.sizes = c(50, 50), test = "rank.sum") ## homogeneity 2 samples
pdRankTests(data, sample.sizes = rep(25, 4), test = "krusk.wall") ## homogeneity 4 samples
pdRankTests(data, test = "bartels") ## randomness

## null hypothesis is false
data1 <- array(c(data, replicate(50, Expm(diag(2), H.coeff(0.5 * rnorm(4), inverse = TRUE)))),
                dim = c(2,2,150))
pdRankTests(data1, sample.sizes = c(100, 50), test = "rank.sum")
pdRankTests(data1, sample.sizes = rep(50, 3), test = "krusk.wall")
pdRankTests(data1, test = "bartels")

## signed-rank test for equivalence of spectra
## ARMA(1,1) process: Example 11.4.1 in (Brockwell and Davis, 1991)
Phi <- array(c(0.7, 0, 0, 0.6, rep(0, 4)), dim = c(2, 2, 2))
Theta <- array(c(0.5, -0.7, 0.6, 0.8, rep(0, 4)), dim = c(2, 2, 2))
Sigma <- matrix(c(1, 0.71, 0.71, 2), nrow = 2)
pgram <- function(Sigma) pdPgram(rARMA(2^9, 2, Phi, Theta, Sigma)$X)$P

## null is true
pdRankTests(array(c(pgram(Sigma), pgram(Sigma)), dim = c(2,2,2^8)), test = "signed.rank")
## null is false
pdRankTests(array(c(pgram(Sigma), pgram(0.5 * Sigma)), dim = c(2,2,2^8)), test = "signed.rank")
```

---

pdSpecClust1D *Intrinsic 1D wavelet-based clustering of multivariate spectra.*

Description

pdSpecClust1D performs clustering of multivariate spectral matrices via a two-step fuzzy clustering algorithm in the intrinsic manifold wavelet domain of curves in the space of HPD matrices equipped with a metric, e.g. the Riemannian metric, specified by the user.

Usage

```
pdSpecClust1D(P, K, jmax, metric = "Riemannian", m = 2, d.jmax = 0.1,
  eps = c(1e-04, 1e-04), tau = 0.5, max.iter = 50, return.centers = F,
  ...)
```

Arguments

P	a (d, d, n, S)-dimensional array of n-dimensional sequences of HPD matrices for S different subjects, with n a dyadic number.
K	the number of clusters, should be a integer larger than 1.
jmax	an upper bound on the maximum wavelet scale to be considered in the clustering procedure. If jmax is not specified, it is set equal to the largest (i.e. finest) possible wavelet scale.
metric	the metric that the space of HPD matrices is equipped with. The default choice is "Riemannian", but this can also be one of: "logEuclidean", "Cholesky", "rootEuclidean" or "Euclidean". The intrinsic distance measures in the clustering algorithm fundamentally rely on the chosen metric.
m	the fuzziness parameter for both the fuzzy c-medoids and the weighted fuzzy c-means algorithm. m should be larger or equal to 1. If $m = 1$ the cluster assignments are no longer fuzzy (i.e. the procedure performs hard clustering).
d.jmax	a proportion that is used to determine the maximum wavelet scale to be considered in the clustering procedure. A larger value d.jmax leads to less wavelet coefficients being taken into account, and therefore lower computational effort in the procedure. If d.jmax is not specified, by default d.jmax = 0.1.
eps	an optional vector with two elements determining the stopping criterion. The fuzzy c-medoids algorithm (i.e. first clustering step) terminates if the (integrated) intrinsic distance between cluster centers is smaller than eps[1]. The weighted fuzzy c-means (i.e. second clustering step) terminates if the (integrated) distance between cluster centers is smaller than eps[2]. If eps is not specified, by default eps = c(1E-4, 1E-4).
tau	an optional argument tuning the weight given to the cluster assignments obtained in the first step of the clustering algorithm. If tau is not specified, by default tau = 0.5.
max.iter	an optional argument tuning the maximum number of iterations in both the first and second step of the clustering algorithm, defaults to max.iter = 50.
return.centers	should the cluster centers transformed back the space of HPD matrices also be returned? Defaults to return.centers = F.
...	additional arguments for internal use.

Details

The input array P contains initial noisy HPD spectral estimates of the (d, d)-dimensional spectral matrices at n different frequencies for S different subjects, where n is a dyadic number. The initial spectral estimates can be e.g. the tapered HPD periodograms given as output by pdPgram.
For each subject s, thresholded wavelet coefficients in the intrinsic manifold wavelet domain are

calculated by pdSpecEst1D.
The maximum wavelet scale taken into account in the clustering procedure is determined by the arguments jmax and d.jmax. The maximum scale is set to the minimum of jmax and the wavelet scale j for which the proportion of nonzero thresholded wavelet coefficients (averaged across subjects) is smaller than d.jmax.
The S subjects are assigned to K different clusters in a probabilistic fashion according to a two-step procedure:

1. In the first step, an intrinsic fuzzy c-medoids algorithm, with fuzziness parameter m is applied to the S coarsest midpoints at scale j = 0 in the subject-specific midpoints pyramids. Note that the intrinsic c-medoids algorithm crucially relies on the metric that the space of HPD matrices gets equipped with.
2. In the second step, a weighted fuzzy c-means algorithm based on the Euclidean distance function, also with fuzziness parameter m, is applied to the nonzero thresholded wavelet coefficients of the S different subjects. The tuning parameter tau controls the weight given to the cluster assignments obtained in the first step of the clustering algorithm.

If return.centers = T, the function also returns the K HPD spectral curves corresponding to the cluster centers based on the given metric by applying the intrinsic inverse 1D wavelet transform (InvWavTransf1D) to the cluster centers in the wavelet domain.

Value

The function returns a list with 6 components:

cl.prob an (S, K)-dimensional matrix, where the value at position (s, k) in the matrix corresponds to the probabilistic cluster membership assignment of subject s with respect to cluster k.

cl.centers.D a list of K wavelet coefficient pyramids, where each pyramid of wavelet coefficients is associated to a cluster center.

cl.centers.M0 a list K arrays of coarse-scale midpoints at scale j = 0, where each array is associated to a cluster center.

cl.centers.f if return.centers = T returns a list of K (d, d, n)-dimensional arrays, where each array corresponds to a discretized curve of HPD matrices associated to a cluster center. If return.centers = F, cl.centers.f returns NULL.

cl.jmax the maximum wavelet scale taken into account in the clustering procedure determined by the input arguments jmax and d.jmax.

iters the number of iterations in respectively the first and second step of the clustering procedure.

References

Chau, J. and von Sachs, R. (2017). *Positive definite multivariate spectral estimation: a geometric wavelet approach*. Available at http://arxiv.org/abs/1701.03314.

Brockwell, P.J. and Davis, R.A. (1991). *Time series: Theory and Methods*. New York: Springer.

See Also

pdSpecEst1D, WavTransf1D, pdDist, pdPgram

Examples

```
## ARMA(1,1) process: Example 11.4.1 in (Brockwell and Davis, 1991)

Phi1 <- array(c(0.5, 0, 0, 0.6, rep(0, 4)), dim = c(2, 2, 2))
Phi2 <- array(c(0.7, 0, 0, 0.4, rep(0, 4)), dim = c(2, 2, 2))
Theta <- array(c(0.5, -0.7, 0.6, 0.8, rep(0, 4)), dim = c(2, 2, 2))
Sigma <- matrix(c(1, 0.71, 0.71, 2), nrow = 2)

## Generate periodogram data for 10 subjects
pgram <- function(Phi) pdPgram(rARMA(2^9, 2, Phi, Theta, Sigma)$X)$P
P <- array(c(replicate(5, pgram(Phi1)), replicate(5, pgram(Phi2))), dim=c(2,2,2^8,10))

cl <- pdSpecClust1D(P, K = 2, metric = "logEuclidean")
```

`pdSpecClust2D` *Intrinsic 2D wavelet-based clustering of multivariate time-varying spectra.*

Description

`pdSpecClust2D` performs clustering of multivariate time-varying spectral matrices via a two-step fuzzy clustering algorithm in the intrinsic manifold wavelet domain of surface in the space of HPD matrices equipped with a metric, e.g. the Riemannian metric, specified by the user. This function extends `pdSpecClust2D` for clustering surfaces instead of curves of HPD matrices.

Usage

```
pdSpecClust2D(P, K, jmax, metric = "Riemannian", m = 2, d.jmax = 0.1,
  eps = c(1e-04, 1e-04), tau = 0.5, max.iter = 50, return.centers = F,
  ...)
```

Arguments

`P`	a (d, d, n_1, n_2, S)-dimensional array corresponding to discretized surfaces of HPD matrices for `S` different subjects at $n_1 \times n_2$ different time-frequency points (on a rectangular tensor grid), with n_1 and n_2 dyadic numbers.
`K`	the number of clusters, should be a integer larger than 1.
`jmax`	an upper bound on the maximum wavelet scale to be considered in the clustering procedure. If `jmax` is not specified, it is set equal to the largest (i.e. finest) possible wavelet scale.
`metric`	the metric that the space of HPD matrices is equipped with. The default choice is `"Riemannian"`, but this can also be one of: `"logEuclidean"`, `"Cholesky"`, `"rootEuclidean"` or `"Euclidean"`. The intrinsic distance measures in the clustering algorithm fundamentally rely on the chosen metric.

`m`	the fuzziness parameter for both the fuzzy c-medoids and the weighted fuzzy c-means algorithm. `m` should be larger or equal to 1. If $m = 1$ the cluster assignments are no longer fuzzy (i.e. the procedure performs hard clustering).
`d.jmax`	a proportion that is used to determine the maximum wavelet scale to be considered in the clustering procedure. A larger value `d.jmax` leads to less wavelet coefficients being taken into account, and therefore lower computational effort in the procedure. If `d.jmax` is not specified, by default `d.jmax = 0.1`.
`eps`	an optional vector with two elements determining the stopping criterion. The fuzzy c-medoids algorithm (i.e. first clustering step) terminates if the (integrated) intrinsic distance between cluster centers is smaller than `eps[1]`. The weighted fuzzy c-means (i.e. second clustering step) terminates if the (integrated) distance between cluster centers is smaller than `eps[2]`. If `eps` is not specified, by default `eps = c(1E-4, 1E-4)`.
`tau`	an optional argument tuning the weight given to the cluster assignments obtained in the first step of the clustering algorithm. If `tau` is not specified, by default `tau = 0.5`.
`max.iter`	an optional argument tuning the maximum number of iterations in both the first and second step of the clustering algorithm, defaults to `max.iter = 50`.
`return.centers`	should the cluster centers transformed back the space of HPD matrices also be returned? Defaults to `return.centers = F`.
`...`	additional arguments for internal use.

Details

The input array `P` corresponds to initial noisy HPD time-varying spectral estimates of the (d, d)-dimensional spectral matrices at $m_1 \times m_2$ different time-frequency points for S different subjects, with m_1, m_2 dyadic numbers. The initial spectral estimates can be e.g. the tapered HPD time-varying periodograms given as output by `pdPgram2D`.
For each subject s, thresholded wavelet coefficients in the intrinsic 2D manifold wavelet domain are calculated by `pdSpecEst2D`.
The maximum wavelet scale taken into account in the clustering procedure is determined by the arguments `jmax` and `d.jmax`. The maximum scale is set to the minimum of `jmax` and the wavelet scale j for which the proportion of nonzero thresholded wavelet coefficients (averaged across subjects) is smaller than `d.jmax`.
The S subjects are assigned to K different clusters in a probabilistic fashion according to a two-step procedure:

1. In the first step, an intrinsic fuzzy c-medoids algorithm, with fuzziness parameter m is applied to the S coarsest midpoints at scale `j = 0` in the subject-specific 2D midpoints pyramids. Note that the intrinsic c-medoids algorithm crucially relies on the metric that the space of HPD matrices gets equipped with.
2. In the second step, a weighted fuzzy c-means algorithm based on the Euclidean distance function, also with fuzziness parameter m, is applied to the nonzero thresholded wavelet coefficients of the S different subjects. The tuning parameter `tau` controls the weight given to the cluster assignments obtained in the first step of the clustering algorithm.

If `return.centers = T`, the function also returns the `K` HPD time-varying spectral matrices corresponding to the cluster centers based on the given metric by applying the intrinsic inverse 2D wavelet transform (`InvWavTransf2D`) to the cluster centers in the wavelet domain.

Value

The function returns a list with 6 components:

cl.prob an (S, K)-dimensional matrix, where the value at position (s, k) in the matrix corresponds to the probabilistic cluster membership assignment of subject s with respect to cluster k.

cl.centers.D a list of `K` wavelet coefficient pyramids, where each 2D pyramid of wavelet coefficients is associated to a cluster center.

cl.centers.M0 an array `K` (d, d)-dimensional coarse-scale midpoints at scale `j = 0`, where each midpoint is associated to a cluster center.

cl.centers.f if `return.centers = T` returns a list of `K` (d, d, n_1, n_2)-dimensional arrays, where each array corresponds to a discretized surface of HPD matrices associated to a cluster center. If `return.centers = F`, `cl.centers.f` returns `NULL`.

cl.jmax the maximum wavelet scale taken into account in the clustering procedure determined by the input arguments `jmax` and `d.jmax`.

iters the number of iterations in respectively the first and second step of the clustering procedure.

See Also

`pdSpecEst2D`, `WavTransf2D`, `pdDist`, `pdPgram2D`

Examples

```
## Generate periodogram date for 4 subjects
pgram <- function(seed) rExamples2D(c(2^5, 2^5), 2, example = "smiley", seed = seed)$per
P <- array(c(replicate(2, pgram(1)), replicate(2, pgram(2))), dim=c(2,2,2^5,2^5,4))

cl <- pdSpecClust2D(P, K = 2, metric = "logEuclidean")
```

`pdSpecEst` *pdSpecEst: An Analysis Toolbox for Hermitian Positive Definite Matrices*

Description

The `pdSpecEst` (**p**ositive **d**efinite **Spec**tral **Est**imation) package provides data analysis tools for samples of symmetric or Hermitian positive definite matrices, such as collections of (non-degenerate) covariance matrices or spectral density matrices.

Details

The tools in this package can be used to perform:

- *Intrinsic manifold wavelet regression* and *clustering* for curves (1D) and surfaces (2D) of Hermitian positive definite matrices. These implementations are based in part on the paper (Chau and von Sachs, 2017a).
- *Exploratory data analysis* and *inference* for samples of Hermitian positive definite matrices by means of intrinsic manifold data depth and rank-based hypothesis tests. These implementations are based on the paper (Chau, Ombao and von Sachs, 2017b).

For more details and examples on how to use the package see the accompanying vignettes in the vignettes folder. A demo Shiny app to test out the implemented functions in the package is available here.

Author and maintainer: **Joris Chau** (<j.chau@uclouvain.be>).

Install the current development version via `devtools::install_github("JorisChau/pdSpecEst")`.

References

Chau, J. and von Sachs, R. (2017a). *Positive definite multivariate spectral estimation: a geometric wavelet approach*. Available at http://arxiv.org/abs/1701.03314.

Chau, J., Ombao, H., and von Sachs, R. (2017b). *Data depth and rank-based tests for covariance and spectral density matrices*. Available at http://arxiv.org/abs/1706.08289.

`pdSpecEst1D` *Intrinsic 1D wavelet-based spectral matrix estimation*

Description

`pdSpecEst1D` calculates a (d, d)-dimensional HPD wavelet-denoised spectral matrix estimator by: (i) applying an intrinsic 1D AI wavelet transform (`WavTransf1D`) to an initial noisy HPD spectral estimate, (ii) (tree-structured) thresholding of the wavelet coefficients (`pdCART`) and (iii) applying an intrinsic inverse 1D AI wavelet transform (`InvWavTransf1D`). The complete estimation procedure is described in detail in (Chau and von Sachs, 2017a).

Usage

```
pdSpecEst1D(P, order = 5, policy = "universal", metric = "Riemannian",
  alpha = 1, return = "f", ...)
```

Arguments

`P`	a (d, d, m)-dimensional array of HPD matrices, with m a dyadic number.
`order`	an odd integer larger or equal to 1 corresponding to the order of the intrinsic AI refinement scheme, defaults to `order = 5`. Note that if `order > 9`, the computational cost significantly increases as the wavelet transform no longer uses a fast wavelet refinement scheme based on pre-determined weights.

`policy`	a character, one of `"universal"` or `"cv"`, defaults to `policy = "universal"`.
`metric`	the metric that the space of HPD matrices is equipped with. The default choice is `"Riemannian"`, but this can also be one of: `"logEuclidean"`, `"Cholesky"`, `"rootEuclidean"` or `"Euclidean"`. The intrinsic AI wavelet transform fundamentally relies on the chosen metric.
`alpha`	an optional tuning parameter in the wavelet the thresholding procedure. If `policy = "universal"`, the sparsity parameter in the tree-structured wavelet thresholding procedure is set to `alpha` times the universal threshold, defaults to `alpha = 1`.
`return`	an optional argument that specifies whether the denoised spectral estimator is returned or not.
`...`	additional arguments for internal use.

Details

The input array `P` corresponds to an initial noisy HPD spectral estimate of the (d, d)-dimensional spectral matrix at `m` different frequencies, with $m = 2^J$ for some $J > 0$. This can be e.g. a multitaper HPD periodogram given as output by the function `pdPgram`.
`P` is transformed to the wavelet domain by the function `WavTransf1D`, which applies an intrinsic 1D AI wavelet transform based on e.g. the Riemannian metric. The noise is removed by tree-structured thresholding of the wavelet coefficients based on the trace of the whitened coefficients as in `pdCART` by minimization of a *complexity penalized residual sum of squares* (CPRESS) criterion in (Donoho, 1997), via a fast tree-pruning algorithm. As in `pdCART`, the sparsity parameter is set equal to `alpha` times the universal threshold where the noise variance of the traces of the whitened wavelet coefficients determined from the finest wavelet scale. If the thresholding policy is set to `policy = "universal"`, the sparsity parameter is set equal to the universal threshold. If the thresholding policy is set to `policy = "cv"`, a data-adaptive sparsity parameter is computed via two-fold cross-validation as in (Nason, 1996) based on the chosen metric.
If `return == 'f'` the thresholded wavelet coefficients are transformed back to the frequency domain by the inverse intrinsic 1D AI wavelet transform via `InvWavTransf1D` giving the wavelet-denoised HPD spectral estimate.

Value

The function returns a list with four components:

`f`	a (d, d, m)-dimensional array corresponding to the wavelet-denoised HPD (d, d)-dimensional spectral estimate at the `m` different frequencies. If `!(return == 'f')`, the inverse wavelet transform of the thresholded wavelet coefficients is not computed and `f` is set equal to `NULL`.
`D`	the pyramid of threshold wavelet coefficients. This is a list of arrays, where each array contains the (d, d)-dimensional thresholded wavelet coefficients from the finest wavelet scale `j = jmax` up to the coarsest wavelet scale `j = 0`.
`M0`	a numeric array containing the midpoint(s) at the coarsest scale `j = 0` in the midpoint pyramid.
`tree.weights`	a list of logical values specifying which coefficients to keep, with each list component corresponding to an individual wavelet scale.

`alpha.opt` the wavelet thresholding tuning parameter equal to the input argument `alpha` if `policy = "universal"`; or determined data-adaptively via two-fold cross-validation if `policy = "cv"`.

References

Chau, J. and von Sachs, R. (2017a). *Positive definite multivariate spectral estimation: a geometric wavelet approach*. Available at http://arxiv.org/abs/1701.03314.

Nason, G.P. (1996). *Wavelet shrinkage using cross-validation*. Journal of the Royal Statistical Society: Series B, 58, 463-479.

See Also

pdPgram, WavTransf1D, InvWavTransf1D, pdCART

Examples

```
P <- rExamples(2^8, example = "bumps")$per
f <- pdSpecEst1D(P)
```

---

pdSpecEst2D *Intrinsic 2D wavelet-based time-varying spectral matrix estimation*

Description

`pdSpecEst2D` calculates a (d, d)-dimensional HPD wavelet-denoised time-varying spectral matrix estimator by: (i) applying an intrinsic 2D AI wavelet transform (`WavTransf2D`) to an initial noisy HPD spectral estimate, (ii) (tree-structured) thresholding of the wavelet coefficients (`pdCART`) and (iii) applying an intrinsic inverse 2D AI wavelet transform (`InvWavTransf2D`).

Usage

```
pdSpecEst2D(P, order = c(3, 3), metric = "Riemannian", alpha = 1,
  return = "f", ...)
```

Arguments

`P` a (d, d, m_1, m_2)-dimensional array of HPD matrices, with m_1, m_2 both dyadic numbers.

`order` a 2-dimensional numeric vector of odd integers larger or equal to 1 corresponding to the marginal orders of the intrinsic 2D AI refinement scheme, defaults to `order = c(3, 3)`. Note that the computational cost significantly increases if `max(order) > 9` as the wavelet transform no longer uses a fast wavelet refinement scheme based on pre-determined weights.

`metric`	the metric that the space of HPD matrices is equipped with. The default choice is `"Riemannian"`, but this can also be one of: `"logEuclidean"`, `"Cholesky"`, `"rootEuclidean"` or `"Euclidean"`. The intrinsic AI wavelet transform fundamentally relies on the chosen metric.
`alpha`	an optional tuning parameter in the wavelet the thresholding procedure. If `policy = "universal"`, the sparsity parameter in the tree-structured wavelet thresholding procedure is set to `alpha` times the universal threshold, defaults to `alpha = 1`.
`return`	an optional argument that specifies whether the denoised spectral estimator is returned or not.
`...`	additional arguments for internal use.

Details

The input array `P` corresponds to an initial noisy HPD time-varying spectral estimate of the (d, d)-dimensional spectral matrix at $m_1 \times m_2$ different time-frequency points, with m_1, m_2 dyadic numbers. This can be e.g. a multitaper HPD time-varying periodogram given as output by the function `pdPgram2D`.
`P` is transformed to the wavelet domain by the function `WavTransf2D`, which applies an intrinsic 2D AI wavelet transform based on e.g. the Riemannian metric. The noise is removed by tree-structured thresholding of the wavelet coefficients based on the trace of the whitened coefficients as in `pdCART` by minimization of a *complexity penalized residual sum of squares* (CPRESS) criterion in (Donoho, 1997), via a fast tree-pruning algorithm. As in `pdCART`, the sparsity parameter is set equal to `alpha` times the universal threshold where the noise variance of the traces of the whitened wavelet coefficients determined from the finest wavelet scale.
If `return == 'f'` the thresholded wavelet coefficients are transformed back to the frequency domain by the inverse intrinsic 2D AI wavelet transform via `InvWavTransf2D` giving the wavelet-denoised HPD time-varying spectral estimate.

Value

The function returns a list with four components:

`f`	a (d, d, m_1, m_2)-dimensional array corresponding to the wavelet-denoised HPD (d, d)-dimensional nonstationary spectral estimate at the $m1 \times m2$ different time-frequency points. If `!(return == 'f')`, the inverse wavelet transform of the thresholded wavelet coefficients is not computed and `f` is set equal to `NULL`.
`D`	the 2D pyramid of threshold wavelet coefficients. This is a list of arrays, where each array contains the 2D grid of (d, d)-dimensional thresholded wavelet coefficients from the finest wavelet scale `j = jmax` up to the coarsest wavelet scale `j = 0`.
`M0`	a numeric array containing the midpoint(s) at the coarsest scale `j = 0` in the 2D midpoint pyramid.
`tree.weights`	a list of logical values specifying which coefficients to keep, with each list component corresponding to an individual wavelet scale.
`D.raw`	the 2D pyramid of non-thresholded wavelet coefficients in the same format as the component `$D`.

References

Chau, J. and von Sachs, R. (2017a). *Positive definite multivariate spectral estimation: a geometric wavelet approach*. Available at http://arxiv.org/abs/1701.03314.

See Also

pdPgram2D, WavTransf2D, InvWavTransf2D, pdCART

Examples

```
## Not run:
 P <- rExamples2D(c(2^7, 2^7), 3, example = "tvar")$per
 f <- pdSpecEst2D(P)

## End(Not run)
```

---

pdSplineReg *Cubic smoothing spline regression for HPD matrices*

Description

`pdSplineReg()` performs cubic smoothing spline regression on the Riemannian manifold of HPD matrices equipped with the Riemannian metric through minimization of a penalized regression objective function using a geometric conjugate gradient descent method as outlined in (Boumal and Absil, 2011). In contrast to (Boumal and Absil, 2011), we set the term in the objective function based on the first-order finite geometric differences to zero, such that the solutions of the regression problem are approximating cubic splines.

Usage

```
pdSplineReg(P, f0, lam = 1, Nd, ini.step = 1, max.iter = 100,
  eps = 0.001, ...)
```

Arguments

`P`	a (d, d, n)-dimensional array corresponding to a sequence of observed noisy HPD matrices.
`f0`	a (d, d, n)-dimensional array corresponding to an initial HPD curve estimator of the smooth target curve of HPD matrices.
`lam`	a smoothness penalty parameter, defaults to `lam = 1`. If `lam = 0`, the penalized regression estimator coincides with geodesic interpolation of the data points. If `lam` increases to ∞, the penalized regression estimator is approximately a fitted geodesic curve.
`Nd`	a numeric value (`Nd <= n`) determining a lower resolution of the cubic spline regression estimator to speed up computations, defaults to the number of data observations n.

ini.step	initial candidate step size in a backtracking line search based on the Armijo-Goldstein condition, defaults to ini.step = 1.
max.iter	maximum number of gradient descent iterations, defaults to max.iter = 100.
eps	stopping criterion. The gradient descent procedure exits if the absolute difference of the of the evaluated objective function is smaller than eps, defaults to eps = 1E-3.
...	additional arguments for internal usage.

Value

A list with three components:

f	a (d, d, N_d)-dimensional array containing the HPD cubic smoothing spline regression estimator.
cost	a numeric vector containing the costs of the objective function after each gradient descent iteration.
total.iter	total number of gradient descent iterations.

References

Boumal, N. and Absil, P-A. (2011). A discrete regression method on manifolds and its applications to data on SO(n). *IFAC Proceedings Volumes*, 44, 2284-2289.

Examples

```
set.seed(2)
P <- rExamples(100, example = 'gaussian')
P.spline <- pdSplineReg(P$per, P$per, lam = 0.5, Nd = 25)
## Examine matrix-component (1,1)
plot((1:50)/50, Re(P$per[1, 1, ]), type = "l", lty = 2) ## noisy observations
lines((1:25)/25, Re(P.spline$f[1, 1, ])) ## penalized regression estimator
lines((1:50)/50, Re(P$f[1, 1, ]), col = 2, lty = 2) ## smooth target
```

---

rARMA *Simulate vARMA(2,2) time series observations*

Description

rARMA generates d-dimensional time series observations from a vector ARMA(2,2) (autoregressive moving average) process based on Gaussian white noise for testing and simulation purposes.

Usage

```
rARMA(n, d, Phi, Theta, Sigma, burn = 100, freq = NULL)
```

Arguments

n	number of time series observations to be generated.
d	dimension of the multivariate time series.
Phi	a $(d, d, 2)$-dimensional array, with Phi[, , 1] and Phi[, , 2] the autoregressive parameter matrices.
Theta	a $(d, d, 2)$-dimensional array, with Theta[, , 1] and Theta[, , 2] the moving-average parameter matrices.
Sigma	the covariance matrix of the Gaussian white noise component.
burn	a burn-in period when generating the time series observations, by default burn = 100.
freq	an optional vector of frequencies, if !is.null(freq) the function also returns the underlying spectral matrix of the stationary generating process at the frequencies corresponding to freq.

Value

The function returns a list with two components:

X	generated time series observations, the d columns correspond to the components of the multivariate time series.
f	if !is.null(freq), f is a (d, d, length(freq))-dimensional array containing the underlying spectral matrix of the process at frequencies corresponding to freq. If is.null(freq), f is set to NULL.

References

Brockwell, P.J. and Davis, R.A. (1991). *Time series: Theory and Methods*. New York: Springer.

Examples

```
## ARMA(1,1) process: Example 11.4.1 in (Brockwell and Davis, 1991)

freq <- seq(from = pi / 100, to = pi, length = 100)
Phi <- array(c(0.7, 0, 0, 0.6, rep(0, 4)), dim = c(2, 2, 2))
Theta <- array(c(0.5, -0.7, 0.6, 0.8, rep(0, 4)), dim = c(2, 2, 2))
Sigma <- matrix(c(1, 0.71, 0.71, 2), nrow = 2)
ts.sim <- rARMA(200, 2, Phi, Theta, Sigma, freq=freq)
ts.plot(ts.sim$X) # plot generated time series traces.
```

`rExamples` *Several example spectral matrices*

Description

`rExamples()` generates stationary time series observations from several example HPD spectral matrices for testing and simulation purposes. For more details, we refer to the simulation studies in (Chau and von Sachs, 2017).

Usage

```
rExamples(n, example = c("heaviSine", "bumps", "two-cats", "gaussian"), ...)
```

Arguments

`n`	number of time series observations to be generated.
`example`	the example spectral matrix, one of `'heaviSine'`, `'bumps'`, `'two-cats'` or `'gaussian'`.
`...`	additional arguments passed on to `pdPgram`.

Details

The examples include: (i) a (3×3) heaviSine HPD spectral matrix consisting of smooth sinosoids with a break, (ii) a (3×3) bumps HPD spectral matrix containing peaks and bumps of various smoothness degrees, (iii) a (3×3) two-cats HPD spectral matrix visualizing the contour of two side-by-side cats, with inhomogeneous smoothness across frequency, and (iv) a (2×2) Gaussian HPD spectral matrix consisting of smooth random Gaussian functions. The time series observations are generated via the Cram\'er representation based on the transfer function of the example spectral matrix and complex normal random variates.

Value

Returns a list with four components:

`f`	example spectral matrix, `f` is a `(d, d, length(freq))`-dimensional array, corresponding to a $(d \times d)$-dimensional spectral matrix at the Fourier frequencies `freq`.
`freq`	vector of Fourier frequencies from zero to π.
`per`	multitaper HPD periodogram of the generated time series, by default pre-smoothed using `B = d` DPSS tapering functions, see `pdPgram` for details.
`ts`	generated time series observations, the `d` columns correspond to the components of the multivariate time series.

References

Chau, J. and von Sachs, R. (2017). *Positive definite multivariate spectral estimation: a geometric wavelet approach*. Available at http://arxiv.org/abs/1701.03314.

See Also

pdPgram

Examples

```
example <- rExamples(100, example = "bumps")
ts.plot(Re(example$ts)) # plot generated time series.
```

---

rExamples2D *Several example time-varying spectral matrices*

Description

rExamples2D() generates several example HPD time-varying spectral matrices for testing and simulation purposes.

Usage

```
rExamples2D(n, d = 3, B, example = c("smiley", "tvar", "peak", "facets"),
  bias.corr = F, ...)
```

Arguments

n	numeric vector with two components corresponding to the size of the discretized time-frequency grid at which the spectral matrix is generated.
d	row- (resp. column-)dimension of the generated spectral matrix.
B	complex Wishart distribution degrees of freedom, defaults to B = d, such that the generated pseudo periodogram matrix observations are positive definite. Note that B coincides with the number of tapering functions in the (time-varying) multitaper periodogram in pdPgram2D.
example	the example spectral matrix, one of 'smiley', 'tvar', 'peak' or 'facets'.
bias.corr	should the Riemannian manifold bias-correction be applied to the HPD periodogram matrix? Defaults to FALSE.
...	additional arguments for internal use.

Details

The examples include: (i) a $(d \times d)$ smiley HPD spectral matrix consisting of constant surfaces of random HPD matrices in the shape of a smiley face, (ii) a $(d \times d)$ tvar HPD spectral matrix generated from a time-varying vector auto- regressive process of order 1 with a random time-varying coefficient matrix (Φ), (iii) a $(d \times d)$ generally smooth HPD spectral matrix containing a pronounced peak in the center of the discretized time-frequency grid and (iv) a $(d \times d)$ facets HPD spectral matrix consisting of several facets generated from random geodesic surfaces. In addition to the HPD spectral matrices, the function also generates pseudo HPD periodogram observations as independent complex random HPD Wishart matrices centered around the generating HPD spectral matrix with

B degrees of freedom. Informally, such random matrix behavior corresponds to the asymptotic distribution of actual HPD periodogram observations (as obtained via pdPgram2D) of a multivariate time series with the given generating HPD spectral matrix.

Value

Returns a list with two components:

`f`	example spectral matrix, `f` is a `(d, d, n[1], n[2])`-dimensional array, corresponding to a $(d \times d)$-dimensional spectral matrix at a (n_1, n_2)-dimensional grid of time-frequency points.
`tf.grid`	a list with two components `tf.grid$time` and `tf.grid$frequency` specifying the rectangular grid of time-frequency points at which the spectral matrix is generated. `tf.grid$time` is a numeric vector of rescaled time points in `(0,1]`. `tf.grid$frequency` is a numeric vector of frequency points in `(0,0.5]`, with 0.5 corresponding to the Nyquist frequency.
`per`	pseudo HPD periodogram observations generated as random complex Wishart matrices centered around `f` with `B` degrees of freedom at (n_1, n_2)-dimensional grid of time-frequency points.

References

Chau, J. and von Sachs, R. (2017). *Positive definite multivariate spectral estimation: a geometric wavelet approach*. Available at http://arxiv.org/abs/1701.03314.

See Also

pdPgram2D

Examples

```
example <- rExamples2D(n = c(2^5, 2^5), example = "smiley")
```

WavTransf1D *Forward average-interpolation 1D wavelet transform*

Description

`WavTransf1D` computes the forward intrinsic average-interpolation (AI) wavelet transform of a curve in the manifold of HPD matrices equipped with a metric specified by the user (e.g. the Riemannian metric) as described in (Chau and von Sachs, 2017).

Usage

```
WavTransf1D(P, order = 5, jmax, periodic = F, metric = "Riemannian",
  progress = F, ...)
```

Arguments

P	a (d, d, m)-dimensional array of HPD matrices, with $m = 2^J$ for some $J > 0$.
order	an odd integer larger or equal to 1 corresponding to the order of the intrinsic AI refinement scheme, defaults to `order = 5`. Note that if `order > 9`, the computational cost significantly increases as the wavelet transform no longer uses a fast wavelet refinement scheme based on pre-determined weights.
jmax	the maximum scale up to which the wavelet coefficients are computed. If `jmax` is not specified it is set equal to the maximum possible scale `jmax = J-1`.
periodic	a logical value determining whether the curve of HPD matrices can be reflected at the boundary for improved wavelet refinement schemes near the boundaries of the domain. This is useful for spectral matrix estimation, where the spectral matrix is a symmetric and periodic curve in the frequency domain. Defaults to `periodic = F`.
metric	the metric that the space of HPD matrices is equipped with. The default choice is `"Riemannian"`, but this can also be one of: `"logEuclidean"`, `"Cholesky"`, `"rootEuclidean"` or `"Euclidean"`. The intrinsic AI wavelet transform fundamentally relies on the chosen metric.
progress	should a console progress bar be displayed? Defaults to `progress = F`.
...	additional arguments for internal use.

Details

The input array `P` corresponds to a discretized curve of (d, d)-dimensional HPD matrices of dyadic length. `WavTransf1D` then computes the intrinsic AI wavelet transform of `P` based on the given refinement order and the chosen metric. If the refinement order is an odd integer smaller or equal to 9, the function computes the wavelet transform using a fast wavelet refinement scheme based on weighted geometric averages with pre-determined weights as explained in (Chau and von Sachs, 2017a). If the refinement order is an odd integer larger than 9, the wavelet refinement scheme is based on intrinsic polynomial prediction using Neville's algorithm on the Riemannian manifold. The function computes the intrinsic AI wavelet transform equipping the space of HPD matrices with one of the following metrics: (i) Riemannian metric (default) as in (Bhatia, 2009, Chapter 6), (ii) log-Euclidean metric, the Euclidean inner product between matrix logarithms, (iii) Cholesky metric, the Euclidean inner product between Cholesky decompositions, (iv) Euclidean metric and (v) root-Euclidean metric. The default choice (Riemannian) has several appealing properties not shared by the other metrics, see (Chau and von Sachs, 2017a) for more details.

Value

The function returns a list with three components:

D	the pyramid of wavelet coefficients. This is a list of arrays, where each array contains the (d, d)-dimensional wavelet coefficients from the finest wavelet scale `j = jmax` up to the coarsest wavelet scale `j = 0`.
D.white	the pyramid of whitened wavelet coefficients. The structure of `D.white` is the same as `D`, but with the wavelet coefficients replaced by their whitened counterparts as explained in (Chau and von Sachs, 2017).

M0 — a numeric array containing the midpoint(s) at the coarsest scale `j = 0` in the midpoint pyramid.

References

Chau, J. and von Sachs, R. (2017a). *Positive definite multivariate spectral estimation: a geometric wavelet approach*. Available at http://arxiv.org/abs/1701.03314.

See Also

InvWavTransf1D, pdSpecEst1D, pdNeville

Examples

```
P <- rExamples(2^8, example = "bumps")
P.wt <- WavTransf1D(P$f, periodic = FALSE)
```

WavTransf2D — *Forward average-interpolation 2D wavelet transform*

Description

`WavTransf2D` computes the forward intrinsic average-interpolation (AI) wavelet transform of a rectangular surface in the manifold of HPD matrices equipped with a metric specified by the user (e.g. the Riemannian metric).

Usage

```
WavTransf2D(P, order = c(3, 3), jmax, metric = "Riemannian", progress = T,
  ...)
```

Arguments

P	a $(d, d, n1, n2)$-dimensional array of Hermitian PD matrices, with $n_1 = 2^{J_1}$ and $n_2 = 2^{J_2}$ for some $J_1, J_2 > 0$.
order	a 2-dimensional numeric vector of odd integers larger or equal to 1 corresponding to the marginal orders of the intrinsic 2D AI refinement scheme, defaults to `order = c(3, 3)`. Note that the computational cost significantly increases if `max(order) > 9` as the wavelet transform no longer uses a fast wavelet refinement scheme based on pre-determined weights.
jmax	the maximum scale up to which the wavelet coefficients are computed. If `jmax` is not specified it is set equal to the maximum possible scale `jmax = max(J1, J2) - 1`.
metric	the metric that the space of HPD matrices is equipped with. The default choice is `"Riemannian"`, but this can be one of: `"Riemannian"`, `"logEuclidean"`, `"Cholesky"`, `"rootEuclidean"` or `"Euclidean"`. The intrinsic AI wavelet transform fundamentally relies on the chosen metric.
progress	should a console progress bar be displayed? Defaults to `progress = T`.
...	additional arguments for internal use.

Details

The 4-dimensional array P corresponds to a discretized rectangular surface of (d, d)-dimensional HPD matrices. The rectangular surface is of size n_1 by n_2, where both n_1 and n_2 are supposed to be dyadic numbers. WavTransf2D then computes the intrinsic AI wavelet transform of P based on the given refinement orders and the chosen metric. If both marginal refinement orders are smaller or equal to 9, the function computes the wavelet transform using a fast wavelet refinement scheme based on weighted geometric averages with pre-determined weights. If one of the marginal refinement order is an odd integer larger than 9, the wavelet refinement scheme is based on intrinsic polynomial surface prediction using Neville's algorithm on the Riemannian manifold (pdNeville). By default WavTransf2D computes the intrinsic 2D AI wavelet transform equipping the space of HPD matrices with (i) the Riemannian metric. Instead, the space of HPD matrices can also be equipped with one of the following metrics; (ii) log-Euclidean metric, the Euclidean inner product between matrix logarithms, (iii) Cholesky metric, the Euclidean inner product between Cholesky decompositions, (iv) Euclidean metric and (v) root-Euclidean metric. The default choice (Riemannian) has several appealing properties not shared by the other metrics, see (Chau and von Sachs, 2017a) for more details.

Value

The function returns a list with three components:

D	the 2D pyramid of wavelet coefficients. This is a list of arrays, where each 4-dimensional array contains the (d, d)-dimensional wavelet coefficients in a 2D grid of locations from the finest wavelet scale j = jmax up to the coarsest wavelet scale j = 0.
D.white	the 2D pyramid of whitened wavelet coefficients. The structure of D.white is the same as D, but with the wavelet coefficients replaced by their whitened counterparts as explained in (Chau and von Sachs, 2017).
M0	a numeric array containing the midpoint(s) at the coarsest scale j = 0 in the 2D midpoint pyramid.

References

Chau, J. and von Sachs, R. (2017a). *Positive definite multivariate spectral estimation: a geometric wavelet approach*. Available at http://arxiv.org/abs/1701.03314.

See Also

InvWavTransf2D, pdSpecEst2D, pdNeville

Examples

```
P <- rExamples2D(c(2^4, 2^4), 2, example = "tvar")
P.wt <- WavTransf2D(P$f)
```

Index

dpss, *20–23*

Expm, 2, *8*, *10*

H.coeff, 3

InvWavTransf1D, 4, *11*, *29*, *33–35*, *44*
InvWavTransf2D, 6, *11*, *32*, *35–37*, *45*

Logm, *2, 3*, 8, *10*

Mid, 8, *18*

ParTrans, *3*, *8*, 9
pdCART, 10, *33–37*
pdConfInt1D, 12
pdDepth, *12, 13*, 15, *26, 27*
pdDist, *16*, 16, *29*, *32*
pdMean, *9*, 17
pdNeville, *6, 7*, 18, *24*, *44, 45*
pdPgram, *13*, 20, *22, 23*, *26–29*, *34, 35*, *40, 41*
pdPgram2D, *21*, 22, *31, 32*, *36, 37*, *41, 42*
pdPolynomial, *19*, 24
pdRankTests, *16*, 25
pdSpecClust1D, 27
pdSpecClust2D, 30
pdSpecEst, 32
pdSpecEst-package *(*pdSpecEst*)*, 32
pdSpecEst1D, *6*, *12, 13*, *21*, *23*, *29*, 33, *44*
pdSpecEst2D, *7*, *31, 32*, 35, *45*
pdSplineReg, 37

rARMA, 38
rExamples, 40
rExamples2D, 41

WavTransf1D, *4–6*, *10, 11*, *29*, *33–35*, 42
WavTransf2D, *6, 7*, *10, 11*, *32*, *35–37*, 44