Package 'pdSpecEst'

October 14, 2022

Type Package

Title An Analysis Toolbox for Hermitian Positive Definite Matrices

Version 1.2.4

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 wavelet transforms for curves (1D) or surfaces (2D) of Hermitian positive definite matrices with applications to dimension reduction, denoising and clustering in the space of Hermitian positive definite matrices; and (ii) exploratory data analysis and inference for samples of positive definite matrices by means of intrinsic data depth functions and rank-based hypothesis tests in the space of Hermitian positive definite matrices.

URL https://github.com/JorisChau/pdSpecEst

Depends R (>= 3.4.0)

License GPL-2

LazyData TRUE

Imports multitaper, Rcpp, ddalpha, Rdpack

RdMacros Rdpack

Encoding UTF-8

RoxygenNote 6.1.1

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 <joris.chau@openanalytics.eu>

Repository CRAN

Date/Publication 2020-01-08 09:10:07 UTC

R topics documented:

`Expm` *Riemannian HPD 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 manifold of Hermitian PD matrices equipped with the affine-invariant Riemannian metric via the exponential map as in e.g., (Pennec et al. 2006). This is the unique inverse of the Riemannian 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, Fillard P, Ayache N (2006). "A Riemannian framework for tensor computing." *International Journal of Computer Vision*, **66**(1), 41–66.

See Also

Logm, pdParTrans

Examples

```
## Generate 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)]
## Generate random HPD matrix
p <- matrix(complex(real = rnorm(9), imaginary = rnorm(9)), nrow = 3)
P <- t(Conj(p)) %*% p
## Compute exponential map
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. That is, H.coeff transforms H into a numeric vector of d^2 real-valued basis coefficients, which 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 $(1, 1) \leq (n, m) \leq (d, d)$. 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 = FALSE)
```

Arguments

H	if inverse = FALSE, a (d, d)-dimensional Hermitian matrix; if inverse = TRUE, 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 AI wavelet transform for curve of HPD matrices*

Description

InvWavTransf1D computes an inverse intrinsic average-interpolation (AI) wavelet transform mapping an array of coarsest-scale HPD midpoints combined with a pyramid of Hermitian wavelet coefficients to a curve in the manifold of HPD matrices equipped with a metric specified by the user, as described in (Chau and von Sachs 2019) and Chapter 3 of (Chau 2018). This is the inverse operation of the function WavTransf1D.

Usage

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

Arguments

D	a list of arrays containing the pyramid of wavelet coefficients, where each array contains the (d, d)-dimensional wavelet coefficients from the coarsest wavelet scale j = 0 up to the finest wavelet scale j = jmax. 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 = FALSE`.

`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"`, `"Euclidean"` or `"Riemannian-Rahman"`. See also the Details section below.

`...` 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 intrinsic averages with pre-determined weights as explained in (Chau and von Sachs 2019) and Chapter 3 of (Chau 2018). If the refinement order is an odd integer larger than 9, the wavelet refinement scheme uses intrinsic polynomial prediction based on Neville's algorithm in the Riemannian manifold (via `pdNeville`).

The function computes the inverse intrinsic AI wavelet transform in the space of HPD matrices equipped with one of the following metrics: (i) the affine-invariant Riemannian metric (default) as detailed in e.g., (Bhatia 2009)[Chapter 6] or (Pennec et al. 2006); (ii) the log-Euclidean metric, the Euclidean inner product between matrix logarithms; (iii) the Cholesky metric, the Euclidean inner product between Cholesky decompositions; (iv) the Euclidean metric; or (v) the root-Euclidean metric. The default choice of metric (affine-invariant Riemannian) satisfies several useful properties not shared by the other metrics, see (Chau and von Sachs 2019) or (Chau 2018) for more details. Note that this comes at the cost of increased computation time in comparison to one of the other metrics.

Value

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

References

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

Chau J (2018). *Advances in Spectral Analysis for Multivariate, Nonstationary and Replicated Time Series*. phdthesis, Universite catholique de Louvain.

Chau J, von Sachs R (2019). "Intrinsic wavelet regression for curves of Hermitian positive definite matrices." *Journal of the American Statistical Association*. doi: 10.1080/01621459.2019.1700129.

Pennec X, Fillard P, Ayache N (2006). "A Riemannian framework for tensor computing." *International Journal of Computer Vision*, **66**(1), 41–66.

See Also

WavTransf1D, pdSpecEst1D, pdNeville

Examples

```
P <- rExamples1D(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 AI wavelet transform for surface of HPD matrices*

Description

InvWavTransf2D computes the inverse intrinsic average-interpolation (AI) wavelet transform mapping an array of coarsest-scale HPD midpoints combined with a 2D pyramid of Hermitian wavelet coefficients to a surface in the manifold of HPD matrices equipped with a metric specified by the user, as described in Chapter 5 of (Chau 2018). This is the inverse operation of the function WavTransf2D.

Usage

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

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 coarsest wavelet scale j = 0 up to the finest wavelet scale j = jmax. 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 $(1,1) \leq$ `order` $\leq (9,9)$ corresponding to the marginal orders of the intrinsic 2D AI refinement scheme, defaults to `order = c(3, 3)`.
`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"`. See also the Details section below.
`...`	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, the marginal refinement orders should be smaller or equal to 9, and the function computes the wavelet transform using a fast wavelet refinement scheme based on weighted intrinsic averages with pre-determined weights as explained in Chapter 5 of (Chau 2018). By default `WavTransf2D` computes the inverse intrinsic 2D AI wavelet transform equipping the space of HPD matrices with (i) the affine-invariant Riemannian metric as detailed in e.g., (Bhatia 2009)[Chapter 6] or (Pennec et al. 2006). Instead, the space of HPD matrices can also be equipped with one of the following metrics; (ii) the Log-Euclidean metric, the Euclidean inner product between matrix logarithms; (iii) the Cholesky metric, the Euclidean inner product between Cholesky decompositions; (iv) the Euclidean metric and (v) the root-Euclidean metric. The default choice of metric (affine-invariant Riemannian) satisfies several useful properties not shared by the other metrics, see (Chau 2018) for more details. Note that this comes at the cost of increased computation time in comparison to one of the other metrics.

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

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

Chau J (2018). *Advances in Spectral Analysis for Multivariate, Nonstationary and Replicated Time Series*. phdthesis, Universite catholique de Louvain.

Pennec X, Fillard P, Ayache N (2006). "A Riemannian framework for tensor computing." *International Journal of Computer Vision*, **66**(1), 41–66.

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 *Riemannian HPD logarithmic map*

Description

`Logm(P, Q)` computes the projection of a Hermitian PD matrix `Q` in the manifold of HPD matrices equipped with the affine-invariant Riemannian metric to the tangent space attached at the Hermitian PD matrix `P` via the logarithmic map as in e.g., (Pennec et al. 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`, `pdParTrans`

Examples

```
## Generate two random HPD matrices
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
## Compute logarithmic map
Logm(P, Q)
```

Mid *Geodesic midpoint between HPD matrices*

Description

Mid calculates the geodesic midpoint between two HPD matrices under the affine-invariant Riemannian metric 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*. Princeton University Press, New Jersey.

See Also

pdMean

Examples

```
## Generate two random HPD matrices
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
## Compute midpoint
Mid(A, B)
## Midpoint coincides with two-point intrinsic Karcher mean
all.equal(pdMean(array(c(A, B), dim = c(3, 3, 2))), Mid(A, B))
```

pdCART *Tree-structured trace thresholding of wavelet coefficients*

Description

pdCART performs hard tree-structured thresholding of the Hermitian matrix-valued wavelet coefficients obtained with WavTransf1D or WavTransf2D based on the trace of the whitened wavelet coefficients, as explained in (Chau and von Sachs 2019) or (Chau 2018). This function is primarily written for internal use in other functions and is typically not used as a stand-alone function.

Usage

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

Arguments

D	a list of wavelet coefficients as obtained from the $D component of WavTransf1D or WavTransf2D .
D.white	a list of whitened wavelet coefficients as obtained from the $D.white component of 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 penalty/sparsity parameter as alpha times the universal threshold.
tree	logical value, if tree = TRUE performs tree-structured thresholding, otherwise performs non-tree-structured hard thresholding of the coefficients.
...	additional arguments 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 penalty parameter in the optimization procedure 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, if alpha = 0, the penalty parameter is zero and the coefficients remain untouched.

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 starting from the coarsest wavelet scale j = 0.
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 seen in (Chau and von Sachs 2019). For thresholding of 2D wavelet coefficients, there is a discrepancy between the constant noise variance of the traces of the whitened wavelet coefficients at the first abs(J1 - J2) scales and the remaining scales, as discussed in Chapter 5 of (Chau 2018), 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 asymptotic distributional behavior of the HPD periodogram matrices obtained with e.g., pdPgram2D. Note that if the time-frequency grid is square, 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 (2018). *Advances in Spectral Analysis for Multivariate, Nonstationary and Replicated Time Series*. phdthesis, Universite catholique de Louvain.

Chau J, von Sachs R (2019). "Intrinsic wavelet regression for curves of Hermitian positive definite matrices." *Journal of the American Statistical Association*. doi: 10.1080/01621459.2019.1700129.

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

See Also

WavTransf1D, InvWavTransf1D, WavTransf2D, InvWavTransf2D

Examples

```
## 1D tree-structured trace thresholding
P <- rExamples1D(2^8, example = "bumps")$P
Coeffs <- WavTransf1D(P)
pdCART(Coeffs$D, Coeffs$D.white, order = 5)$w ## logical tree of non-zero coefficients

## Not run:
## 2D tree-structured trace thresholding
P <- rExamples2D(c(2^6, 2^6), 2, example = "tvar")$P
Coeffs <- WavTransf2D(P)
pdCART(Coeffs$D, Coeffs$D.white, order = c(3, 3))$w

## End(Not run)
```

---

pdDepth *Data depth for HPD matrices*

Description

pdDepth calculates the data depth of a HPD matrix with respect to a given data cloud (i.e., a sample or collection) of HPD matrices, or the integrated data depth of a sequence (curve) of HPD matrices with respect to a given data cloud of sequences (curves) of HPD matrices as detailed in (Chau et al. 2019).

Usage

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

Arguments

y	either a (d, d)-dimensional HPD matrix, or a (d, d, n)-dimensional array corresponding to a sequence or curve of HPD 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, X is either a (d, d, S)-dimensional array corresponding to a data cloud of S individual HPD 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 'gdd', 'zonoid' or 'spatial' corresponding to the geodesic distance depth, intrinsic zonoid depth, and intrinsic 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". See also the Details section below.

Details

Available pointwise or integrated intrinsic data depth functions for samples of HPD matrices are: (i) geodesic distance depth, (ii) intrinsic zonoid depth and (iii) intrinsic spatial depth. The various data depth measures and their theoretical properties are described in (Chau et al. 2019). If y is a (d, d)-dimensional HPD matrix, X should be a (d, d, S)-dimensional array corresponding to a length S sequence of (d, d)-dimensional HPD matrices and the pointwise data depth values are computed. If y is a sequence of (d, d)-dimensional HPD matrices of length n (i.e., (d, d, n)-dimensional array), X should be a (d, d, n, S)-dimensional array of replicated sequences of HPD matrices and the integrated data depth values according to (Chau et al. 2019) are computed. If is.null(y), the data depth of each individual object (i.e., a HPD matrix or a sequence of HPD matrices) in X is computed with respect to the data cloud X.

The function computes the intrinsic data depth values based on the metric space of HPD matrices equipped with one of the following metrics: (i) Riemannian metric (default) as detailed in e.g., (Bhatia 2009)[Chapter 6] or (Pennec et al. 2006), (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 properties not shared by the other metrics, see (Chau et al. 2019) 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 vector of depth values for each individual object in X with respect to X itself.

Note

The function does not check for positive definiteness of the input matrices, and may fail if matrices are close to being singular.

The data depth computations under the Riemannian metric are more involved than under the other metrics, and may therefore result in (significantly) higher computation times.

References

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

Chau J, Ombao H, von Sachs R (2019). "Intrinsic data depth for Hermitian positive definite matrices." *Journal of Computational and Graphical Statistics*, **28**(2), 427–439. doi: 10.1080/10618600.2018.1537926.

Pennec X, Fillard P, Ayache N (2006). "A Riemannian framework for tensor computing." *International Journal of Computer Vision*, **66**(1), 41–66.

See Also

pdDist, pdRankTests

Examples

```
## Pointwise depth
X1 <- replicate(50, Expm(diag(2), H.coeff(rnorm(4), inverse = TRUE)))
pdDepth(y = diag(2), X = X1) ## depth of one point
pdDepth(X = X1) ## 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, metric = "Riemannian")
```

Arguments

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

Details

Available distance measures between two HPD matrices are: (i) the affine-invariant Riemannian distance (default) as in e.g., (Bhatia 2009)[Chapter 6] or (Pennec et al. 2006); (ii) the Log-Euclidean distance, the Euclidean distance between matrix logarithms; (iii) the Cholesky distance, the Euclidean distance between Cholesky decompositions; (iv) the Euclidean distance; (v) the root-Euclidean distance; and (vi) the 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, in order to compute the distance between two Hermitian positive definite matrices.

References

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

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

Pennec X, Fillard P, Ayache N (2006). "A Riemannian framework for tensor computing." *International Journal of Computer Vision*, **66**(1), 41–66.

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
```

---

pdkMeans *K-means clustering for HPD matrices*

Description

pdkMeans performs (fuzzy) k-means clustering for collections of HPD matrices, such as covariance or spectral density matrices, based on a number of different metrics in the space of HPD matrices.

Usage

```
pdkMeans(X, K, metric = "Riemannian", m = 1, eps = 1e-05,
  max_iter = 100, centroids)
```

Arguments

`X`	a (d, d, S)-dimensional array of (d, d)-dimensional HPD matrices for S different subjects. Also accepts a (d, d, n, S)-dimensional array, which is understood to be an array of n-dimensional sequences of (d, d)-dimensional HPD matrices for S different subjects.
`K`	the number of clusters, a positive integer larger than 1.
`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"`. Additional details are given below.
`m`	a fuzziness parameter larger or equal to 1. If $m = 1$ the cluster assignments are no longer fuzzy, i.e., the procedure performs hard clustering. Defaults to `m = 1`.
`eps`	an optional tolerance parameter determining the stopping criterion. The k-means algorithm terminates if the intrinsic distance between cluster centers is smaller than `eps`, defaults to `eps = 1e-05`.
`max_iter`	an optional parameter tuning the maximum number of iterations in the k-means algorithm, defaults to `max_iter = 100`.
`centroids`	an optional (d, d, K)- or (d, d, n, K)-dimensional array depending on the input array `X` specifying the initial cluster centroids. If not specified, `K` initial cluster centroids are randomly sampled without replacement from the input array `X`.

Details

The input array `X` corresponds to a collection of (d, d)-dimensional HPD matrices for S different subjects. If the fuzziness parameter satisfies `m > 1`, the S subjects are assigned to K different clusters in a probabilistic fashion according to a fuzzy k-means algorithm as detailed in classical texts, such as (Bezdek 1981). If `m = 1`, the S subjects are assigned to the K clusters in a non-probabilistic fashion according to a standard (hard) k-means algorithm. If not specified by the user, the K cluster centers are initialized by random sampling without replacement from the input array of HPD matrices `X`. The distance measure in the (fuzzy) k-means algorithm is induced by the metric on the space of HPD matrices specified by the user. By default, the space of HPD matrices is equipped with (i) the affine-invariant Riemannian metric (`metric = 'Riemannian'`) as detailed in e.g., (Bhatia 2009)[Chapter 6] or (Pennec et al. 2006). Instead, this can also be one of: (ii) the log-Euclidean metric (`metric = 'logEuclidean'`), the Euclidean inner product between matrix logarithms; (iii) the Cholesky metric (`metric = 'Cholesky'`), the Euclidean inner product between Cholesky decompositions; (iv) the Euclidean metric (`metric = 'Euclidean'`); or (v) the root-Euclidean metric (`metric = 'rootEuclidean'`). The default choice of metric (affine-invariant Riemannian) satisfies several useful properties not shared by the other metrics, see e.g., *C18*pdSpecEst for more details. Note that this comes at the cost of increased computation time in comparison to one of the other metrics.

Value

Returns a list with two components:

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

cl.centroids either a (d, d, K)- or (d, d, n, K)-dimensional array depending on the input array `X` corresponding respectively to the `K` (d, d)- or (d, d, n)-dimensional final cluster centroids.

References

Bezdek J (1981). *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, New York.

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

Pennec X, Fillard P, Ayache N (2006). "A Riemannian framework for tensor computing." *International Journal of Computer Vision*, **66**(1), 41–66.

See Also

`pdDist`, `pdSpecClust1D`, `pdSpecClust2D`

Examples

```
## Generate 20 random HPD matrices in 2 groups
m <- function(rescale){
 x <- matrix(complex(real = rescale * rnorm(9), imaginary = rescale * rnorm(9)), nrow = 3)
 t(Conj(x)) %*% x
}
X <- array(c(replicate(10, m(0.25)), replicate(10, m(1))), dim = c(3, 3, 20))

## Compute fuzzy k-means cluster assignments
cl <- pdkMeans(X, K = 2, m = 2)$cl.assignments
```

---

pdMean *Weighted Karcher mean of HPD matrices*

Description

`pdMean` calculates an (approximate) weighted Karcher or Frechet mean of a sample of (d, d)-dimensional HPD matrices intrinsic to a user-specified metric. In the case of the affine-invariant Riemannian metric as detailed in e.g., (Bhatia 2009)[Chapter 6] or (Pennec et al. 2006), the weighted Karcher mean is either approximated via the fast recursive algorithm in (Ho et al. 2013) or computed via the slower, but more accurate, gradient descent algorithm in (Pennec 2006). By default, the unweighted Karcher mean is computed.

Usage

```
pdMean(M, w, metric = "Riemannian", grad_desc = FALSE, maxit = 1000,
  reltol)
```

Arguments

M	a (d, d, S)-dimensional array corresponding to a sample of (d, d)-dimensional HPD matrices of size S.
w	an S-dimensional nonnegative weight vector, such that sum(w) = 1.
metric	the distance measure, one of 'Riemannian', 'logEuclidean', 'Cholesky', 'Euclidean' or 'rootEuclidean'. Defaults to 'Riemannian'.
grad_desc	if metric = "Riemannian", a logical value indicating if the gradient descent algorithm in (Pennec 2006) should be used, defaults to FALSE.
maxit	maximum number of iterations in gradient descent algorithm, only used if grad_desc = TRUE and metric = "Riemannian". Defaults to 1000
reltol	optional tolerance parameter in gradient descent algorithm, only used if grad_desc = TRUE and metric = "Riemannian". Defaults to 1E-10.

Note

The function does not check for positive definiteness of the input matrices, and (depending on the specified metric) may fail if matrices are close to being singular.

References

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

Ho J, Cheng G, Salehian H, Vemuri B (2013). "Recursive Karcher expectation estimators and recursive law of large numbers." *Artificial Intelligence and Statistics*, 325–332.

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

Pennec X, Fillard P, Ayache N (2006). "A Riemannian framework for tensor computing." *International Journal of Computer Vision*, **66**(1), 41–66.

See Also

Mid, pdMedian

Examples

```
## Generate random sample of HPD matrices
m <- function(){
 X <- matrix(complex(real=rnorm(9), imaginary=rnorm(9)), nrow=3)
 t(Conj(X)) %*% X
}
M <- replicate(100, m())
z <- rnorm(100)
## Generate random weight vector
w <- abs(z)/sum(abs(z))
## Compute weighted (Riemannian) Karcher mean
pdMean(M, w)
```

pdMedian *Weighted intrinsic median of HPD matrices*

Description

pdMedian calculates a weighted intrinsic median of a sample of (d, d)-dimensional HPD matrices based on a Weiszfeld algorithm intrinsic to the chosen metric.In the case of the affine-invariant Riemannian metric as detailed in e.g., (Bhatia 2009)[Chapter 6] or (Pennec et al. 2006), the intrinsic Weiszfeld algorithm in (Fletcher et al. 2009) is used. By default, the unweighted intrinsic median is computed.

Usage

```
pdMedian(M, w, metric = "Riemannian", maxit = 1000, reltol)
```

Arguments

M	a (d, d, S)-dimensional array corresponding to a sample of (d, d)-dimensional HPD matrices of size S.
w	an S-dimensional nonnegative weight vector, such that sum(w) = 1.
metric	the distance measure, one of 'Riemannian', 'logEuclidean', 'Cholesky', 'Euclidean' or 'rootEuclidean'. Defaults to 'Riemannian'.
maxit	maximum number of iterations in gradient descent algorithm. Defaults to 1000
reltol	optional tolerance parameter in gradient descent algorithm. Defaults to 1E-10.

Note

The function does not check for positive definiteness of the input matrices, and (depending on the specified metric) may fail if matrices are close to being singular.

References

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

Fletcher P, Venkatasubramanian S, Joshi S (2009). "The geometric median on Riemannian manifolds with application to robust atlas estimation." *NeuroImage*, **45**(1), S143–S152.

Pennec X, Fillard P, Ayache N (2006). "A Riemannian framework for tensor computing." *International Journal of Computer Vision*, **66**(1), 41–66.

See Also

pdMean

Examples

```
## Generate random sample of HPD matrices
m <- function(){
 X <- matrix(complex(real=rnorm(9), imaginary=rnorm(9)), nrow=3)
 t(Conj(X)) %*% X
}
M <- replicate(100, m())
## Generate random weight vector
z <- rnorm(100)
w <- abs(z)/sum(abs(z))
## Compute weighted intrinsic (Riemannian) median
pdMedian(M, w)
```

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

Description

`pdNeville` performs intrinsic polynomial interpolation of curves or surfaces of HPD matrices in the metric space of HPD matrices equipped with the affine-invariant Riemannian metric (see (Bhatia 2009)[Chapter 6] or (Pennec et al. 2006)) via Neville's algorithm based on iterative geodesic interpolation detailed in (Chau and von Sachs 2019) and in Chapter 3 and 5 of (Chau 2018).

Usage

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

Arguments

`P`	for polynomial curve interpolation, a (d, d, N)-dimensional array corresponding to a length N sequence of (d, d)-dimensional HPD matrices (control points) through which the interpolating 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 HPD matrices (control points) through which the interpolating polynomial surface passes.
`X`	for polynomial curve interpolation, a numeric vector of length N specifying the time points at which the interpolating 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)` at which the interpolating polynomial passes trough the control points `P`.
`x`	for polynomial curve interpolation, a numeric vector specifying the time points (locations) at which the interpolating polynomial is evaluated. For polynomial surface interpolation, a list with as elements two numeric vectors `x` and `y` specifying the time points (locations) on the tensor product grid `expand.grid(x$x, x$y)` at which the interpolating polynomial surface is evaluated.

`metric` the metric on the space of HPD matrices, by default `metric = "Riemannian"`, but instead of the Riemannian metric 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 surface is of bi-degree $(N_1 - 1, N_2 - 1)$. Depending on the input array `P`, the function decides whether polynomial curve or polynomial surface interpolation is performed.

Value

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

Note

If `metric = "Euclidean"`, the interpolating curve or surface may not be positive definite everywhere as the space of HPD matrices equipped with the Euclidean metric has its boundary at a finite distance.

The function does not check for positive definiteness of the input matrices, and may fail if `metric = "Riemannian"` and the input matrices are close to being singular.

References

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

Chau J (2018). *Advances in Spectral Analysis for Multivariate, Nonstationary and Replicated Time Series*. phdthesis, Universite catholique de Louvain.

Chau J, von Sachs R (2019). "Intrinsic wavelet regression for curves of Hermitian positive definite matrices." *Journal of the American Statistical Association*. doi: 10.1080/01621459.2019.1700129.

Pennec X, Fillard P, Ayache N (2006). "A Riemannian framework for tensor computing." *International Journal of Computer Vision*, **66**(1), 41–66.

See Also

`pdPolynomial`

Examples

```
### Polynomial curve interpolation
P <- rExamples1D(50, 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))
```

---

pdParTrans *Riemannian HPD parallel transport*

Description

pdParTrans computes the parallel transport on the manifold of HPD matrices equipped with the affine-invariant Riemannian metric as described in e.g., Chapter 2 of (Chau 2018). That is, the function computes the parallel transport of a Hermitian matrix W in the tangent space at the HPD matrix P along a geodesic curve in the direction of the Hermitian matrix V in the tangent space at P for a unit time step.

Usage

```
pdParTrans(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 (2018). *Advances in Spectral Analysis for Multivariate, Nonstationary and Replicated Time Series*. phdthesis, Universite catholique de Louvain.

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

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

---

pdPgram *Multitaper HPD periodogram matrix*

Description

Given a multivariate time series, pdPgram computes a multitapered HPD periodogram matrix based on averaging raw Hermitian PSD periodogram matrices of tapered multivariate time series segments.

Usage

```
pdPgram(X, B, method = c("multitaper", "bartlett"), bias.corr = F,
  nw = 3)
```

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 the argument method, either the number of orthogonal DPSS 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 an asymptotic bias-correction under the affine-invariant Riemannian metric be applied to the HPD periodogram matrix? Defaults to FALSE.
nw	a positive numeric value corresponding to the time-bandwidth parameter of the DPSS tapering functions, see also dpss, defaults to nw = 3.

Details

If `method = "multitaper"`, `pdPgram` calculates a (d, d)-dimensional multitaper periodogram matrix based on B DPSS (Discrete Prolate Spheroidal Sequence or 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.
In the case of subsequent periodogram matrix denoising in the space of HPD matrices equipped with the affine-invariant Riemannian metric, one should set `bias.corr = T`, thereby correcting for the asymptotic bias of the periodogram matrix in the manifold of HPD matrices equipped with the affine-invariant metric as explained in (Chau and von Sachs 2019) and Chapter 3 of (Chau 2018). The pre-smoothed HPD periodogram matrix (i.e., an initial noisy HPD spectral estimator) can be given as input to the function `pdSpecEst1D` to perform intrinsic wavelet-based spectral matrix estimation. In this case, set `bias.corr = F` (the default) as the appropriate bias-corrections are applied internally by the function `pdSpecEst1D`.

Value

A list containing two components:

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

References

Chau J (2018). *Advances in Spectral Analysis for Multivariate, Nonstationary and Replicated Time Series*. phdthesis, Universite catholique de Louvain.

Chau J, von Sachs R (2019). "Intrinsic wavelet regression for curves of Hermitian positive definite matrices." *Journal of the American Statistical Association*. doi: 10.1080/01621459.2019.1700129.

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 *Multitaper HPD time-varying periodogram matrix*

Description

Given a multivariate time series, pdPgram2D computes a multitapered HPD time-varying periodogram matrix based on averaging raw Hermitian PSD time-varying periodogram matrices of tapered multivariate time series segments.

Usage

```
pdPgram2D(X, B, tf.grid, method = c("dpss", "hermite"), nw = 3,
  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 the argument method, either the number of orthogonal DPSS 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 evaluated. tf.grid$time should be a numeric vector of rescaled time points in the range (0,1). tf.grid$frequency should be a numeric vector of frequency points in the range (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 = 3. Both the DPSS and Hermite tapers are rescaled with the same time-bandwidth parameter.
bias.corr	should an asymptotic bias-correction under the affine-invariant Riemannian metric 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 DPSS (Discrete Prolate Spheroidal Sequence or 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). In short, the function pdPgram2D 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 DPSS tapers by orthogonal Hermite tapering functions as in e.g.,

(Bayram and Baraniuk 1996).
In the case of subsequent periodogram matrix denoising in the space of HPD matrices equipped with the affine-invariant Riemannian metric, one should set `bias.corr = T`, thereby correcting for the asymptotic bias of the periodogram matrix in the manifold of HPD matrices equipped with the affine-invariant metric as explained in (Chau and von Sachs 2019) and Chapter 3 and 5 of (Chau 2018). The pre-smoothed HPD periodogram matrix (i.e., an initial noisy HPD spectral estimator) can be given as input to the function `pdSpecEst2D` to perform intrinsic wavelet-based time-varying spectral matrix estimation. In this case, set `bias.corr = F` (the default) as the appropriate bias-corrections are applied internally by the function `pdSpecEst2D`.

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 evaluated.
`P`	a (d, d, m_1, m_2)-dimensional array with `m_1 = length(tf.grid$time)` and `m_2 = length(tf.grid$frequency)` corresponding to the (d, d)-dimensional tapered periodogram matrices evaluated at the time-frequency points in `tf.grid`.

References

Bayram M, Baraniuk R (1996). "Multiple window time-frequency analysis." In *Proceedings of the IEEE-SP International Symposium on Time-Frequency and Time-Scale Analysis*, 173–176.

Chau J (2018). *Advances in Spectral Analysis for Multivariate, Nonstationary and Replicated Time Series*. phdthesis, Universite catholique de Louvain.

Chau J, von Sachs R (2019). "Intrinsic wavelet regression for curves of Hermitian positive definite matrices." *Journal of the American Statistical Association*. doi: 10.1080/01621459.2019.1700129.

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 in the manifold of HPD matrices equipped with the affine-invariant Riemannian metric (see (Bhatia 2009)[Chapter 6] or (Pennec et al. 2006)) according to the numerical integration procedure in (Hinkle et al. 2014). Given an initial starting point `p0` (i.e., a HPD matrix) in the Riemannian manifold and 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 (d, d)-dimensional Hermitian matrix-valued covariant derivatives from 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 corresponding to a generated (approximate) intrinsic polynomial curve in the space of (d, d)-dimensional HPD matrices of degree k passing through `p0` with the given covariant derivatives `v0` up to order $k-1$ and vanishing higher order covariant derivatives.

References

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

Hinkle J, Fletcher P, Joshi S (2014). "Intrinsic polynomials for regression on Riemannian manifolds." *Journal of Mathematical Imaging and Vision*, **50**(1-2), 32–52.

Pennec X, Fillard P, Ayache N (2006). "A Riemannian framework for tensor computing." *International Journal of Computer Vision*, **66**(1), 41–66.

See Also

pdNeville, pdParTrans

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
P.geo <- sapply(seq(0, 1, length = 100), function(t) Expm(p0, t * Logm(p0, P.poly[, , 100])),
                simplify = "array")
all.equal(P.poly, P.geo)
```

---

pdRankTests *Rank-based hypothesis tests for HPD matrices*

Description

pdRankTests performs a number of generalized rank-based hypothesis tests in the metric space of HPD matrices equipped with the affine-invariant Riemannian metric or Log-Euclidean metric for samples of HPD matrices or samples of sequences (curves) of HPD matrices as described in Chapter 4 of (Chau 2018).

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 (d, d)-dimensional HPD matrices, or a (d, d, n, S)-dimensional array corresponding to an array of pooled individual samples of length n sequences of (d, d)-dimensional HPD matrices.
sample_sizes	a numeric vector specifying the individual sample sizes in the pooled sample 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 the input array 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, intrinsic zonoid depth and intrinsic spatial depth respectively. Defaults to `"gdd"`. Not required for test `"signed.rank"`. See the documentation of the function `pdDepth` for additional details about the different depth measures.

`metric` the metric that the space of HPD matrices is equipped with, either `"Riemannian"` or `"logEuclidean"`. Defaults to `"Riemannian"`.

Details

For samples of (d, d)-dimensional HPD matrices with pooled sample size S, the argument `data` is a (d, d, S)-dimensional array of (d, d)-dimensional HPD matrices, where the individual samples are combined along the third array dimension. For samples of sequences of (d, d)-dimensional HPD matrices with pooled sample size S, the argument `data` is a (d, d, n, S)-dimensional array of length n sequences of (d, d)-dimensional HPD 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"` Intrinsic Wilcoxon rank-sum test to test for homogeneity of distributions of two independent samples of HPD matrices or samples of sequences of HPD matrices. The usual univariate ranks are replaced by data depth induced ranks obtained with `pdDepth`.

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

`"signed.rank"` Intrinsic signed-rank test to test for homogeneity of distributions of independent paired or matched samples of HPD matrices. The intrinsic signed-rank test is *not* based on data depth induced ranks, but on a specific difference score in the Riemannian manifold of HPD matrices equipped with either the affine-invariant Riemannian or Log-Euclidean metric.

`"bartels"` Intrinsic Bartels-von Neumann test to test for randomness (i.e., exchangeability) within a single independent sample of HPD matrices or a sample of sequences of HPD matrices. The usual univariate ranks are replaced by data depth induced ranks obtained with `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) the Riemannian metric (default) as detailed in e.g., (Bhatia 2009)[Chapter 6] or (Pennec et al. 2006); or (ii) the 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 2018)[Chapter 4] 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 intrinsic signed-rank test also provides a valid test for equivalence of spectral matrices of two multivariate stationary time series based on the HPD periodogram matrices obtained via `pdPgram`, see (Chau 2018)[Chapter 4] for the details.

The function does not check for positive definiteness of the input matrices, and may fail if matrices are close to being singular.

The data depth computations under the Riemannian metric are more involved than under the Log-Euclidean metric, and may therefore result in (significantly) higher computation times.

References

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

Chau J (2018). *Advances in Spectral Analysis for Multivariate, Nonstationary and Replicated Time Series*. phdthesis, Universite catholique de Louvain.

Pennec X, Fillard P, Ayache N (2006). "A Riemannian framework for tensor computing." *International Journal of Computer Vision*, **66**(1), 41–66.

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")

## Not run:
## signed-rank test for equivalence of spectra of multivariate time series
## 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(2^8, 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")

## End(Not run)
```

---

pdSpecClust1D *Intrinsic wavelet HPD spectral matrix clustering*

Description

pdSpecClust1D performs clustering of HPD spectral matrices corrupted by noise (e.g. HPD periodograms) by combining wavelet thresholding and fuzzy clustering in the intrinsic wavelet coefficient domain according to the following steps:

1. Transform a collection of noisy HPD spectral matrices to the intrinsic wavelet domain and denoise the HPD matrix curves by (tree-structured) thresholding of wavelet coefficients with pdSpecEst1D.
2. Apply an intrinsic fuzzy c-means algorithm to the coarsest midpoints at scale j = 0 across subjects.
3. Taking into account the fuzzy cluster assignments in the previous step, apply a weighted fuzzy c-means algorithm to the nonzero thresholded wavelet coefficients across subjects from scale j = 1 up to j = jmax.

More details can be found in Chapter 3 of (Chau 2018) and the accompanying vignettes.

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 = FALSE, ...)
```

Arguments

P	a (d, d, n, S)-dimensional array of HPD matrices, corresponding to a collection of sequences of (d, d)-dimensional HPD matrices of length n, with $n = 2^J$ for some $J > 0$, for S different subjects.
K	the number of clusters, a positive 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 maximum (i.e., finest) wavelet scale minus 2.
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". Additional details are given below.

m	the fuzziness parameter for both fuzzy c-means algorithms. 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 components determining the stopping criterion. The first step in the cluster procedure terminates if the (integrated) intrinsic distance between cluster centers is smaller than eps[1]. The second step in the cluster procedure terminates if the (integrated) Euclidean distance between cluster centers is smaller than eps[2]. By default eps = c(1e-04, 1e-04).
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 = FALSE.
...	additional arguments passed on to pdSpecEst1D.

Details

The input array P corresponds to a collection of initial noisy HPD spectral estimates of the (d, d)-dimensional spectral matrix at n different frequencies, with $n = 2^J$ for some $J > 0$, for S different subjects. These can be e.g., multitaper HPD periodograms given as output by the function pdPgram. First, for each subject $s = 1, \ldots, S$, thresholded wavelet coefficients in the intrinsic wavelet domain are calculated by pdSpecEst1D, see the function documentation for additional details on the wavelet thresholding procedure.
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-means algorithm, with fuzziness parameter m is applied to the S coarsest midpoints at scale j = 0 in the subject-specific midpoint pyramids. Note that the distance function in the intrinsic c-means algorithm relies on the chosen metric on the space of HPD matrices.
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.

The function computes the forward and inverse intrinsic AI wavelet transform in the space of HPD matrices equipped with one of the following metrics: (i) the affine-invariant Riemannian metric

(default) as detailed in e.g., (Bhatia 2009)[Chapter 6] or (Pennec et al. 2006); (ii) the log-Euclidean metric, the Euclidean inner product between matrix logarithms; (iii) the Cholesky metric, the Euclidean inner product between Cholesky decompositions; (iv) the Euclidean metric; or (v) the root-Euclidean metric. The default choice of metric (affine-invariant Riemannian) satisfies several useful properties not shared by the other metrics, see (Chau and von Sachs 2019) or (Chau 2018) for more details. Note that this comes at the cost of increased computation time in comparison to one of the other metrics.
If `return.centers = TRUE`, the function also returns the `K` HPD spectral matrix curves corresponding to the cluster centers based on the given metric by applying the intrinsic inverse AI wavelet transform (`InvWavTransf1D`) to the cluster centers in the wavelet domain.

Value

Depending on the input the function returns a list with five or six 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 of `K` arrays of coarse-scale midpoints at scale `j = 0`, where each array is associated to a cluster center.

cl.centers.f only available if `return.centers = TRUE`, returning a list of `K` (d, d, n)-dimensional arrays, where each array corresponds to a length n curve of (d, d)-dimensional HPD matrices associated to a cluster center.

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

References

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

Chau J (2018). *Advances in Spectral Analysis for Multivariate, Nonstationary and Replicated Time Series*. phdthesis, Universite catholique de Louvain.

Chau J, von Sachs R (2019). "Intrinsic wavelet regression for curves of Hermitian positive definite matrices." *Journal of the American Statistical Association*. doi: 10.1080/01621459.2019.1700129.

Pennec X, Fillard P, Ayache N (2006). "A Riemannian framework for tensor computing." *International Journal of Computer Vision*, **66**(1), 41–66.

See Also

`pdSpecEst1D`, `pdSpecClust2D`, `pdkMeans`

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 in 2 groups
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 wavelet HPD time-varying spectral clustering*

Description

`pdSpecClust2D` performs clustering of HPD time-varying spectral matrices corrupted by noise (e.g. HPD time-varying periodograms) by combining wavelet thresholding and fuzzy clustering in the intrinsic wavelet coefficient domain according to the following steps:

1. Transform a collection of noisy HPD time-varying spectral matrices to the intrinsic wavelet domain and denoise the HPD matrix surfaces by (tree-structured) thresholding of wavelet coefficients with `pdSpecEst2D`.
2. Apply an intrinsic fuzzy c-means algorithm to the coarsest midpoints at scale `j = 0` across subjects.
3. Taking into account the fuzzy cluster assignments in the previous step, apply a weighted fuzzy c-means algorithm to the nonzero thresholded wavelet coefficients across subjects from scale `j = 1` up to `j = jmax`.

More details can be found in Chapter 3 of (Chau 2018) and the accompanying vignettes.

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 = FALSE, ...)
```

Arguments

`P`	a `(d,d,n[1],n[2],S)`-dimensional array of HPD matrices, corresponding to a collection of surfaces of (d, d)-dimensional HPD matrices of size $n_1 \times n_2$, with $n_1 = 2^{J_1}$ and $n_2 = 2^{J_2}$ for some $J_1, J_2 > 0$, for S different subjects.
`K`	the number of clusters, a positive 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 maximum (i.e., finest) wavelet scale minus 2.
`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"`. Additional details are given below.

m	the fuzziness parameter for both fuzzy c-means algorithms. 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 components determining the stopping criterion. The first step in the cluster procedure terminates if the (integrated) intrinsic distance between cluster centers is smaller than eps[1]. The second step in the cluster procedure terminates if the (integrated) Euclidean distance between cluster centers is smaller than eps[2]. By default eps = c(1e-04, 1e-04).
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 = FALSE.
...	additional arguments passed on to pdSpecEst2D.

Details

The input array P corresponds to a collection of initial noisy HPD time-varying spectral estimates of the (d, d)-dimensional time-varying spectral matrix at $n_1 \times n_2$ time-frequency points, with n_1, n_2 dyadic numbers, for S different subjects. These can be e.g., multitaper HPD time-varying periodograms given as output by the function pdPgram2D.
First, for each subject $s = 1, \ldots, S$, thresholded wavelet coefficients in the intrinsic wavelet domain are calculated by pdSpecEst2D, see the function documentation for additional details on the wavelet thresholding procedure.
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-means algorithm, with fuzziness parameter m is applied to the S coarsest midpoints at scale j = 0 in the subject-specific 2D midpoint pyramids. Note that the distance function in the intrinsic c-means algorithm relies on the chosen metric on the space of HPD matrices.
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.

The function computes the forward and inverse intrinsic 2D AI wavelet transform in the space of HPD matrices equipped with one of the following metrics: (i) the affine-invariant Riemannian

metric (default) as detailed in e.g., (Bhatia 2009)[Chapter 6] or (Pennec et al. 2006); (ii) the log-Euclidean metric, the Euclidean inner product between matrix logarithms; (iii) the Cholesky metric, the Euclidean inner product between Cholesky decompositions; (iv) the Euclidean metric; or (v) the root-Euclidean metric. The default choice of metric (affine-invariant Riemannian) satisfies several useful properties not shared by the other metrics, see (Chau 2018) for more details. Note that this comes at the cost of increased computation time in comparison to one of the other metrics.
If `return.centers = TRUE`, 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 AI wavelet transform (`InvWavTransf2D`) to the cluster centers in the wavelet domain.

Value

Depending on the input the function returns a list with five or six 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 a list of `K` arrays of coarse-scale midpoints at scale `j = 0`, where each array is associated to a cluster center.

cl.centers.f only available if `return.centers = TRUE`, returning a list of `K (d,d,n[1],n[2])`-dimensional arrays, where each array corresponds to an$n_1 \times n_2$-sized surface of (d, d)-dimensional HPD matrices associated to a cluster center.

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

References

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

Chau J (2018). *Advances in Spectral Analysis for Multivariate, Nonstationary and Replicated Time Series*. phdthesis, Universite catholique de Louvain.

Pennec X, Fillard P, Ayache N (2006). "A Riemannian framework for tensor computing." *International Journal of Computer Vision*, **66**(1), 41–66.

See Also

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

Examples

```
## Not run:
## Generate noisy HPD surfaces for 6 subjects in 2 groups
n <- c(2^5, 2^5)
P <- array(c(rExamples2D(n, example = "tvar", replicates = 3)$P,
             rExamples2D(n, example = "tvar", replicates = 3)$P), dim = c(2, 2, n, 6))
cl <- pdSpecClust2D(P, K = 2, metric = "logEuclidean")
```

```
## End(Not run)
```

---

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 positive definite covariance matrices or spectral density matrices.

Details

The tools in this package can be used to perform:

- *Intrinsic wavelet transforms* for curves (1D) and surfaces (2D) of Hermitian positive definite matrices, with applications to for instance: dimension reduction, denoising and clustering for curves or surfaces of Hermitian positive definite matrices, such as (time-varying) Fourier spectral density matrices. These implementations are based in part on the paper (Chau and von Sachs 2019) and Chapters 3 and 5 of (Chau 2018).
- Exploratory data analysis and inference for samples of Hermitian positive definite matrices by means of *intrinsic data depth* and *depth rank-based hypothesis tests*. These implementations are based on the paper (Chau et al. 2019) and Chapter 4 of (Chau 2018).

For more details and examples on how to use the package see the accompanying vignettes in the vignettes folder. An R-Shiny app to demonstrate and test the implemented functionality 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 (2018). *Advances in Spectral Analysis for Multivariate, Nonstationary and Replicated Time Series*. phdthesis, Universite catholique de Louvain.

Chau J, Ombao H, von Sachs R (2019). "Intrinsic data depth for Hermitian positive definite matrices." *Journal of Computational and Graphical Statistics*, **28**(2), 427–439. doi: 10.1080/10618600.2018.1537926.

Chau J, von Sachs R (2019). "Intrinsic wavelet regression for curves of Hermitian positive definite matrices." *Journal of the American Statistical Association*. doi: 10.1080/01621459.2019.1700129.

pdSpecEst1D *Intrinsic wavelet HPD spectral estimation*

Description

pdSpecEst1D calculates a (d, d)-dimensional HPD wavelet-denoised spectral matrix estimator by applying the following steps to an initial noisy HPD spectral estimate (obtained with e.g., pdPgram):

1. a forward intrinsic AI wavelet transform, with WavTransf1D,
2. (tree-structured) thresholding of the wavelet coefficients, with pdCART,
3. an inverse intrinsic AI wavelet transform, with InvWavTransf1D.

The complete estimation procedure is described in more detail in (Chau and von Sachs 2019) or Chapter 3 of (Chau 2018).

Usage

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

Arguments

P	a (d, d, m)-dimensional array of HPD matrices, corresponding to a sequence of (d, d)-dimensional HPD matrices of length m, 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.
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", "Euclidean" or "Riemannian-Rahman". See also the Details section below.
alpha	an optional tuning parameter in the wavelet thresholding procedure. The penalty (or sparsity) parameter in the tree-structured wavelet thresholding procedure in pdCART is set to alpha times the estimated universal threshold, defaults to alpha = 1.
return_val	an optional argument that specifies whether the denoised spectral estimator is returned or not. See the Details section below.
...	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 a metric specified by the user. The noise is removed by tree-structured thresholding of the wavelet coefficients based on the trace of the whitened coefficients with pdCART by minimization of a *complexity penalized residual sum of squares* (CPRESS) criterion via the fast tree-pruning algorithm in (Donoho 1997). The penalty or sparsity parameter in the optimization procedure is set equal to `alpha` times the universal threshold, where the noise variance of the traces of the whitened wavelet coefficients are determined from the finest wavelet scale. See (Chau and von Sachs 2019) and Chapter 3 of (Chau 2018) for further details.

The function computes the forward and inverse intrinsic AI wavelet transform in the space of HPD matrices equipped with one of the following metrics: (i) the affine-invariant Riemannian metric (default) as detailed in e.g., (Bhatia 2009)[Chapter 6] or (Pennec et al. 2006); (ii) the log-Euclidean metric, the Euclidean inner product between matrix logarithms; (iii) the Cholesky metric, the Euclidean inner product between Cholesky decompositions; (iv) the Euclidean metric; or (v) the root-Euclidean metric. The default choice of metric (affine-invariant Riemannian) satisfies several useful properties not shared by the other metrics, see (Chau and von Sachs 2019) or (Chau 2018) for more details. Note that this comes at the cost of increased computation time in comparison to one of the other metrics.

If `return_val = 'f'` the thresholded wavelet coefficients are transformed back to the frequency domain by the inverse intrinsic 1D AI wavelet transform via InvWavTransf1D, returning the wavelet-denoised HPD spectral estimate.

Value

The function returns a list with the following five components:

`f`	a (d, d, m)-dimensional array of HPD matrices, corresponding to the HPD wavelet-denoised estimate of the same resolution as the input array `P`. If `return_val != '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 coarsest wavelet scale `j = 0` up to the finest wavelet scale `j = jmax`.
`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 starting from the coarsest wavelet scale `j = 0`.
`D.raw`	the pyramid of non-thresholded wavelet coefficients in the same format as the component `$D`.

Note

The function does not check for positive definiteness of the input matrices, and (depending on the specified metric) may fail if matrices are close to being singular.

References

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

Chau J (2018). *Advances in Spectral Analysis for Multivariate, Nonstationary and Replicated Time Series*. phdthesis, Universite catholique de Louvain.

Chau J, von Sachs R (2019). "Intrinsic wavelet regression for curves of Hermitian positive definite matrices." *Journal of the American Statistical Association*. doi: 10.1080/01621459.2019.1700129.

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

Pennec X, Fillard P, Ayache N (2006). "A Riemannian framework for tensor computing." *International Journal of Computer Vision*, **66**(1), 41–66.

See Also

pdPgram, WavTransf1D, InvWavTransf1D, pdCART

Examples

```
P <- rExamples1D(2^8, example = "bumps")$P
f <- pdSpecEst1D(P)
```

---

pdSpecEst2D *Intrinsic wavelet HPD time-varying spectral estimation*

Description

pdSpecEst2D calculates a (d, d)-dimensional HPD wavelet-denoised time-varying spectral matrix estimator by applying the following steps to an initial noisy HPD time-varying spectral estimate (obtained with e.g., pdPgram2D):

1. a forward intrinsic AI wavelet transform, with WavTransf2D,
2. (tree-structured) thresholding of the wavelet coefficients, with pdCART,
3. an inverse intrinsic AI wavelet transform, with InvWavTransf2D.

The complete estimation procedure is described in more detail in Chapter 5 of (Chau 2018).

Usage

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

Arguments

P	a $(d, d, n1, n2)$-dimensional array of HPD matrices corresponding to a rectangular surface of (d, d)-dimensional HPD matrices of size $n_1 \times n_2$, 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 $(1, 1) \leq$ order $\leq (9, 9)$ corresponding to the marginal orders of the intrinsic 2D AI refinement scheme, defaults to order = c(3, 3).
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". See also the Details section below.
alpha	an optional tuning parameter in the wavelet thresholding procedure. The penalty (or sparsity) parameter in the tree-structured wavelet thresholding procedure in pdCART is set to alpha times the estimated universal threshold, defaults to alpha = 1.
return_val	an optional argument that specifies whether the denoised spectral estimator is returned or not. See the Details section below.
...	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 a time-frequency grid of size $m_1 \times m_2$, 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 a metric specified by the user. The noise is removed by tree-structured thresholding of the wavelet coefficients based on the trace of the whitened coefficients with pdCART by minimization of a *complexity penalized residual sum of squares* (CPRESS) criterion via the fast tree-pruning algorithm in (Donoho 1997). The penalty (i.e., sparsity) parameter in the optimization procedure is set equal to alpha times the universal threshold, where the noise variance of the traces of the whitened wavelet coefficients are determined from the finest wavelet scale. See Chapter 5 of (Chau 2018) for further details.

The function computes the forward and inverse intrinsic 2D AI wavelet transform in the space of HPD matrices equipped with one of the following metrics: (i) the affine-invariant Riemannian metric (default) as detailed in e.g., (Bhatia 2009)[Chapter 6] or (Pennec et al. 2006); (ii) the log-Euclidean metric, the Euclidean inner product between matrix logarithms; (iii) the Cholesky metric, the Euclidean inner product between Cholesky decompositions; (iv) the Euclidean metric; or (v) the root-Euclidean metric. The default choice of metric (affine-invariant Riemannian) satisfies several useful properties not shared by the other metrics, see (Chau and von Sachs 2019) or (Chau 2018) for more details. Note that this comes at the cost of increased computation time in comparison to one of the other metrics.

If return_val = 'f' the thresholded wavelet coefficients are transformed back to the time-frequency domain by the inverse intrinsic 2D AI wavelet transform via InvWavTransf2D, returning the wavelet-denoised HPD time-varying spectral estimate.

Value

The function returns a list with the following five components:

`f`	a $(d, d, m1, m2)$-dimensional array of HPD matrices, corresponding to the HPD wavelet-denoised estimate on the same resolution grid of size $m_1 \times m_2$ as specified by the input array `P`. If `return_val != '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 rectangular grid (d, d)-dimensional thresholded wavelet coefficients from the coarsest wavelet scale `j = 0` up to the finest wavelet scale `j = jmax`.
`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 starting from the coarsest wavelet scale `j = 0`.
`D.raw`	the 2D pyramid of non-thresholded wavelet coefficients in the same format as the component `$D`.

References

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

Chau J (2018). *Advances in Spectral Analysis for Multivariate, Nonstationary and Replicated Time Series*. phdthesis, Universite catholique de Louvain.

Chau J, von Sachs R (2019). "Intrinsic wavelet regression for curves of Hermitian positive definite matrices." *Journal of the American Statistical Association*. doi: 10.1080/01621459.2019.1700129.

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

Pennec X, Fillard P, Ayache N (2006). "A Riemannian framework for tensor computing." *International Journal of Computer Vision*, **66**(1), 41–66.

See Also

`pdPgram2D`, `WavTransf2D`, `InvWavTransf2D`, `pdCART`

Examples

```
## Not run:
P <- rExamples2D(c(2^6, 2^6), 2, example = "tvar")$P
f <- pdSpecEst2D(P)

## End(Not run)
```

pdSplineReg *Cubic smoothing spline regression for HPD matrices*

Description

pdSplineReg() performs cubic smoothing spline regression in the space of HPD matrices equipped with the affine-invariant Riemannian metric through minimization of a penalized regression objective function using a geometric conjugate gradient descent method as outlined in (Boumal and Absil 2011a) and (Boumal and Absil 2011b). This is a specific implementation of the more general algorithm in (Boumal and Absil 2011a) and (Boumal and Absil 2011b), setting the part 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 length n sequence of (d, d)-dimensional noisy HPD matrices.
f0	a (d, d, n)-dimensional array corresponding to an initial estimate of the smooth target curve of (d, d)-dimensional HPD matrices.
lam	a smoothness penalty, defaults to lam = 1. If lam = 0, the penalized curve estimate coincides with geodesic interpolation of the data points with respect to the Riemannian metric. 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 computation time, defaults to 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	optional tolerance parameter in gradient descent algorithm. The gradient descent procedure exits if the absolute difference between consecutive evaluations of the objective function is smaller than eps, defaults to eps = 1E-3.
...	additional arguments for internal use.

Value

A list with three components:

f	a (d, d, N_d)-dimensional array corresponding to a length Nd estimated cubic smoothing spline curve of (d, d)-dimensional HPD matrices.
cost	a numeric vector containing the costs of the objective function at each gradient descent iteration.
total_iter	total number of gradient descent iterations.

Note

This function does not check for positive definiteness of the matrices given as input, and may fail if matrices are close to being singular.

References

Boumal N, Absil P (2011a). "Discrete regression methods on the cone of positive-definite matrices." In *IEEE ICASSP, 2011*, 4232–4235.

Boumal N, Absil P (2011b). "A discrete regression method on manifolds and its application to data on SO(n)." *IFAC Proceedings Volumes*, **44**(1), 2284–2289.

Examples

```
## Not run:
set.seed(2)
P <- rExamples1D(50, example = 'gaussian', noise.level = 0.1)
P.spline <- pdSplineReg(P$P, P$P, lam = 0.5, Nd = 25)

## Examine matrix-component (1,1)
plot((1:50)/50, Re(P$P[1, 1, ]), type = "l", lty = 2) ## noisy observations
lines((1:25)/25, Re(P.spline$f[1, 1, ])) ## estimate
lines((1:50)/50, Re(P$f[1, 1, ]), col = 2, lty = 2) ## smooth target

## End(Not run)
```

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

Description

`rARMA` generates d-dimensional time series observations from a vARMA(2,2) (vector-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 (AR) coefficient matrices.
`Theta`	a $(d, d, 2)$-dimensional array, with `Theta[, , 1]` and `Theta[, , 2]` the moving-average (MA) coefficient 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 Fourier spectral matrix of the stationary generating process evaluated at the frequencies in 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 corresponding to the underlying Fourier spectral matrix curve of (d, d)-dimensional HPD matrices evaluated at the frequencies in freq. If is.null(freq), f is set to NULL.

References

Brockwell P, Davis R (2006). *Time Series: Theory and Methods*. Springer, New York.

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.
```

---

rExamples1D *Several example curves of HPD matrices*

Description

rExamples1D() generates several example (locally) smooth target *curves* of HPD matrices corrupted by noise in a manifold of HPD matrices for testing and simulation purposes. For more details, see also Chapter 2 and 3 in (Chau 2018).

Usage

```
rExamples1D(n, d = 3, example = c("bumps", "two-cats", "heaviSine",
  "gaussian", "mix-gaussian", "arma", "peaks", "blocks"), user.f = NULL,
  return.ts = FALSE, replicates = 1, noise = "riem-gaussian",
  noise.level = 1, df.wishart = NULL, nblocks = 10)
```

Arguments

n	number of sampled matrices to be generated.
d	row- (resp. column-)dimension of the generated matrices. Defaults to d = 3.
example	the example target HPD matrix curve, one of 'bumps', 'two-cats', 'heaviSine', 'gaussian', 'mix-gaussian', 'arma', 'peaks' or 'blocks'.
user.f	user-specified target HPD matrix curve, should be a (d, d, n)-dimensional array, corresponding to a length n curve of (d, d)-dimensional HPD matrices.
return.ts	a logical value, if return.ts = TRUE the function also returns time series observations generated via the Cramer representation based on the transfer function of the example HPD spectral matrix and complex normal random variates. Defaults to return.ts = FALSE.
replicates	a positive integer specifying the number of replications of noisy HPD matrix curves to be generated based on the target curve of HPD matrices. Defaults to replicates = 1
noise	noise distribution for the generated noisy curves of HPD matrices, one of 'riem-gaussian', 'log-gaussian', 'wishart', 'log-wishart' or 'periodogram', defaults to 'riem-gaussian'. Additional details are given below.
noise.level	parameter to tune the signal-to-noise ratio for the generated noisy HPD matrix observations, only used if noise != 'periodogram'. If noise.level = 0, the noise distributions are degenerate and the noisy HPD matrix observations coincide with the target HPD matrices. Defaults to noise.level = 1.
df.wishart	optional parameter to specify the degrees of freedom in the case of a Wishart noise distribution (noise = 'wishart' or noise = 'log-wishart'); or the number of DPSS tapers in the case of generated periodogram matrices if noise = 'periodogram'. By default df.wishart is equal to the dimension d to guarantee positive definiteness of the generated noisy matrices.
nblocks	optional parameter to specify the number of constant segments in the 'blocks' HPD matrix curve. Only used if example = 'blocks'.

Details

The examples include: (i) a $(3, 3)$-dimensional 'bumps' HPD matrix curve containing peaks and bumps of various smoothness degrees; (ii) a $(3, 3)$-dimensional 'two-cats' HPD matrix curve visualizing the contour of two side-by-side cats, with inhomogeneous smoothness across the domain; (iii) a $(3, 3)$-dimensional 'heaviSine' HPD matrix curve consisting of smooth sinosoids with a break; (iv) a $(2, 2)$-dimensional 'gaussian' HPD matrix curve consisting of smooth Gaussian functions; (v) a (d, d)-dimensional 'mix-gaussian' HPD matrix curve consisting of a weighted linear combination of smooth Gaussian functions; (vi) a $(2, 2)$-dimensional 'arma' HPD matrix curve generated from the smooth spectral matrix of a 2-dimensional stationary ARMA(1,1)-process; (vii) a (d, d)- dimensional 'peaks' HPD matrix curve containing several sharp peaks across the domain; and (viii) a (d, d)-'blocks' HPD matrix curve generated from locally constant segments of HPD matrices.

In addition to the smooth target curve of HPD matrices, the function also returns a noisy version of the target curve of HPD matrices, corrupted by a user-specified noise distribution. By default, the noisy HPD matrix observations follow an intrinsic signal plus i.i.d. noise model with respect

to the affine-invariant Riemannian metric, with a matrix log-Gaussian noise distribution (`noise = 'riem-gaussian'`), such that the Riemannian Karcher means of the observations coincide with the target curve of HPD matrices. Additional details can be found in Chapters 2, 3, and 5 of (Chau 2018). Other available signal-noise models include: (ii) a Log-Euclidean signal plus i.i.d. noise model, with a matrix log-Gaussian noise distribution (`noise = 'log-gaussian'`); (iii) a Riemannian signal plus i.i.d. noise model, with a complex Wishart noise distribution (`noise = 'wishart'`); (iv) a Log-Euclidean signal plus i.i.d. noise model, with a complex Wishart noise distribution (`noise = 'log-wishart'`); and (v) noisy periodogram observations obtained with `pdPgram` from a stationary time series generated via the Cramer representation based on the transfer function of the target HPD spectral matrix curve and complex normal random variates (`noise = 'periodogram'`). If `return.ts = TRUE`, the function also returns the generated time series observations, which are not generated by default if `noise != 'periodogram'`.

Value

Depending on the input arguments returns a list with two or three components:

`f`	a (d, d, n)-dimensional array, corresponding to the length n example target curve of (d, d)-dimensional HPD matrices.
`P`	a (d, d, n)-dimensional array, corresponding to a length n curve of noisy (d, d)-dimensional HPD matrices centered around the smooth target HPD matrix curve `f`. If `replicates > 1`, `P` is a `(d,d,n,length(replicates))`-dimensional array, corresponding to a collection of replicated length n curves of noisy (d, d)-dimensional HPD matrices centered around the smooth target HPD matrix curve `f`.
`ts`	generated d-dimensional time series observations, only available if `return.ts = TRUE`.

Note

If `noise = 'wishart'`, the generated noisy HPD matrix observations are independent complex Wishart matrices, which can be interpreted informally as pseudo-periodogram matrix observations, as the periodogram matrices based on strictly stationary time series observations obtained with `noise = 'periodogram'` are asymptotically independent and asymptotically complex Wishart distributed, see e.g., (Brillinger 1981).

References

Brillinger D (1981). *Time Series: Data Analysis and Theory*. Holden-Day, San Francisco.

Chau J (2018). *Advances in Spectral Analysis for Multivariate, Nonstationary and Replicated Time Series*. phdthesis, Universite catholique de Louvain.

See Also

`rExamples2D`, `pdPgram`, `rARMA`

Examples

```
example <- rExamples1D(100, example = "bumps", return.ts = TRUE)
plot.ts(Re(example$ts), main = "3-d time series") # plot generated time series
```

`rExamples2D` *Several example surfaces of HPD matrices*

Description

`rExamples2D()` generates several example (locally) smooth target *surfaces* of HPD matrices corrupted by noise in a manifold of HPD matrices for testing and simulation purposes. For more details, see also Chapter 2 and 5 in (Chau 2018).

Usage

```
rExamples2D(n, d = 2, example = c("smiley", "tvar", "facets", "peak"),
  replicates = 1, noise = "riem-gaussian", noise.level = 1,
  df.wishart = NULL)
```

Arguments

`n`	integer vector `c(n1, n2)` specifying the number of sampled matrices to be generated on a rectangular surface.
`d`	row- (resp. column-)dimension of the generated matrices. Defaults to `d = 2`.
`example`	the example target HPD matrix surface, one of `'smiley'`, `'tvar'`, `'facets'` or `'peak'`.
`replicates`	a positive integer specifying the number of replications of noisy HPD matrix surfaces to be generated based on the target surface of HPD matrices. Defaults to `replicates = 1`
`noise`	noise distribution for the generated noisy surfaces of HPD matrices, one of `'riem-gaussian'`, `'log-gaussian'`, `'wishart'`, `'log-wishart'` or `'periodogram'`, defaults to `'riem-gaussian'`. Additional details are given below.
`noise.level`	parameter to tune the signal-to-noise ratio for the generated noisy HPD matrix observations. If `noise.level = 0`, the noise distributions are degenerate and the noisy HPD matrix observations coincide with the target HPD matrices. Defaults to `noise.level = 1`.
`df.wishart`	optional parameter to specify the degrees of freedom in the case of a Wishart noise distribution (`noise = 'wishart'` or `noise = 'log-wishart'`). By default `df.wishart` is equal to the dimension `d` to guarantee positive definiteness of the generated noisy matrices.

Details

The examples include: (i) a (d, d)-dimensional `'smiley'` HPD matrix surface consisting of constant surfaces of random HPD matrices in the shape of a smiley face; (ii) a (d, d)-dimensional `'tvar'` HPD matrix surface generated from a time-varying vector-auto- regressive process of order 1 with random time-varying coefficient matrix (Φ); (iii) a (d, d)-dimensional `'facets'` HPD matrix surface consisting of several facets generated from random geodesic surfaces; and (iv) a (d, d)-dimensional `'peak'` HPD matrix surface containing a pronounced peak in the center of its 2-d (e.g., time-frequency) domain.

In addition to the (locally) smooth target surface of HPD matrices, the function also returns a noisy version of the target surface of HPD matrices, corrupted by a user-specified noise distribution. By default, the noisy HPD matrix observations follow an intrinsic signal plus i.i.d. noise model with respect to the affine-invariant Riemannian metric, with a matrix log-Gaussian noise distribution (`noise = 'riem-gaussian'`), such that the Riemannian Karcher means of the observations coincide with the target surface of HPD matrices. Additional details can be found in Chapters 2, 3, and 5 of (Chau 2018). Other available signal-noise models include: (ii) a Log-Euclidean signal plus i.i.d. noise model, with a matrix log-Gaussian noise distribution (`noise = 'log-gaussian'`); (iii) a Riemannian signal plus i.i.d. noise model, with a complex Wishart noise distribution (`noise = 'wishart'`); (iv) a Log-Euclidean signal plus i.i.d. noise model, with a complex Wishart noise distribution (`noise = 'log-wishart'`).

Value

Returns a list with two components:

`f` — a `(d,d,n[1],n[2])`-dimensional array, corresponding to the $(n_1 \times n_2)$-sized example target surface of (d, d)-dimensional HPD matrices.

`P` — a `(d,d,n[1],n[2])`-dimensional array, corresponding to the $(n_1 \times n_2)$-sized surface of noisy (d, d)-dimensional HPD matrices centered around the smooth target HPD matrix surface `f`. If `replicates > 1`, `P` is a `(d,d,n[1],n[2],length(replicates))`-dimensional array, corresponding to a collection of replicated $(n_1 \times n_2)$-sized surfaces of noisy (d, d)-dimensional HPD matrices centered around the smooth target HPD matrix surface `f`.

References

Chau J (2018). *Advances in Spectral Analysis for Multivariate, Nonstationary and Replicated Time Series*. phdthesis, Universite catholique de Louvain.

See Also

`rExamples1D`, `pdPgram2D`

Examples

```
example <- rExamples2D(n = c(32, 32), example = "smiley")
```

`WavTransf1D` *Forward AI wavelet transform for curve of HPD matrices*

Description

`WavTransf1D` computes a forward intrinsic average-interpolating (AI) wavelet transform for a curve in the manifold of HPD matrices equipped with a metric specified by the user, such as the affine-invariant Riemannian metric, as described in (Chau and von Sachs 2019) and Chapter 3 of (Chau 2018).

Usage

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

Arguments

`P`	a (d, d, m)-dimensional array of HPD matrices, corresponding to a sequence of (d, d)-dimensional HPD matrices of length m, 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`, where `J = log2(m)`.
`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, in which case the spectral matrix is a symmetric and periodic curve in the frequency domain. Defaults to `periodic = FALSE`.
`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"`, `"Euclidean"` or `"Riemannian-Rahman"`. See also the Details section below.
`...`	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 intrinsic averages with pre-determined weights as explained in (Chau and von Sachs 2019) and Chapter 3 of (Chau 2018). If the refinement order is an odd integer larger than 9, the wavelet refinement scheme uses intrinsic polynomial prediction based on Neville's algorithm in the

Riemannian manifold (via pdNeville).
The function computes the intrinsic AI wavelet transform in the space of HPD matrices equipped with one of the following metrics: (i) the affine-invariant Riemannian metric (default) as detailed in e.g., (Bhatia 2009)[Chapter 6] or (Pennec et al. 2006); (ii) the log-Euclidean metric, the Euclidean inner product between matrix logarithms; (iii) the Cholesky metric, the Euclidean inner product between Cholesky decompositions; (iv) the Euclidean metric; or (v) the root-Euclidean metric. The default choice of metric (affine-invariant Riemannian) satisfies several useful properties not shared by the other metrics, see (Chau and von Sachs 2019) or (Chau 2018) for more details. Note that this comes at the cost of increased computation time in comparison to one of the other metrics.

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 Hermitian wavelet coefficients from the coarsest wavelet scale `j = 0` up to the finest wavelet scale `j = jmax`
.	
`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 2019).
`M0`	a numeric array containing the midpoint(s) at the coarsest scale `j = 0` in the midpoint pyramid.

Note

The function does not check for positive definiteness of the input matrices, and (depending on the specified metric) may fail if matrices are close to being singular.

References

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

Chau J (2018). *Advances in Spectral Analysis for Multivariate, Nonstationary and Replicated Time Series*. phdthesis, Universite catholique de Louvain.

Chau J, von Sachs R (2019). "Intrinsic wavelet regression for curves of Hermitian positive definite matrices." *Journal of the American Statistical Association*. doi: 10.1080/01621459.2019.1700129.

Pennec X, Fillard P, Ayache N (2006). "A Riemannian framework for tensor computing." *International Journal of Computer Vision*, **66**(1), 41–66.

See Also

InvWavTransf1D, pdSpecEst1D, pdNeville

Examples

```
P <- rExamples1D(2^8, example = "bumps")
P.wt <- WavTransf1D(P$f, periodic = FALSE)
```

`WavTransf2D` *Forward AI wavelet transform for surface of HPD matrices*

Description

`WavTransf2D` computes a forward intrinsic average-interpolation (AI) wavelet transform for a rectangular surface in the manifold of HPD matrices equipped with a metric specified by the user, such as the affine-invariant Riemannian metric, as described in Chapter 5 of (Chau 2018).

Usage

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

Arguments

`P`	a $(d, d, n1, n2)$-dimensional array of HPD matrices corresponding to a rectangular surface of (d, d)-dimensional HPD matrices of size $n_1 \times n_2$, 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 $(1, 1) \leq$ `order` $\leq (9, 9)$ corresponding to the marginal orders of the intrinsic 2D AI refinement scheme, defaults to `order = c(3, 3)`.
`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"`. See also the Details section below.
`...`	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. The marginal refinement orders should be smaller or equal to 9, and the function computes the wavelet transform using a fast wavelet refinement scheme based on weighted intrinsic averages with pre-determined weights as explained in Chapter 5 of (Chau 2018). By default `WavTransf2D` computes the intrinsic 2D AI wavelet transform equipping the space of HPD matrices with (i) the affine-invariant Riemannian metric as detailed in e.g., (Bhatia 2009)[Chapter 6] or (Pennec et al. 2006). Instead, the space of HPD matrices can

also be equipped with one of the following metrics; (ii) the Log-Euclidean metric, the Euclidean inner product between matrix logarithms; (iii) the Cholesky metric, the Euclidean inner product between Cholesky decompositions; (iv) the Euclidean metric and (v) the root-Euclidean metric. The default choice of metric (affine-invariant Riemannian) satisfies several useful properties not shared by the other metrics, see (Chau 2018) for more details. Note that this comes at the cost of increased computation time in comparison to one of the other metrics.

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 coarsest wavelet scale `j = 0` up to the finest wavelet scale `j = jmax`.
`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 Chapter 5 of (Chau 2018).
`M0`	a numeric array containing the midpoint(s) at the coarsest scale `j = 0` in the 2D midpoint pyramid.

Note

The function does not check for positive definiteness of the input matrices, and (depending on the specified metric) may fail if matrices are close to being singular.

References

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

Chau J (2018). *Advances in Spectral Analysis for Multivariate, Nonstationary and Replicated Time Series*. phdthesis, Universite catholique de Louvain.

Pennec X, Fillard P, Ayache N (2006). "A Riemannian framework for tensor computing." *International Journal of Computer Vision*, **66**(1), 41–66.

See Also

`InvWavTransf2D`, `pdSpecEst2D`, `pdNeville`

Examples

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

Index