Package: CovEsts (via r-universe)

May 19, 2026

Title Nonparametric Estimators for Covariance Functions

Version 1.1.0

Description Several nonparametric estimators of autocovariance functions. Procedures for constructing their confidence regions by using bootstrap techniques. Methods to correct autocovariance estimators and several tools for analysing and comparing them. Supplementary functions, including kernel computations and discrete cosine Fourier transforms. For more details see Bilchouris and Olenko (2025) <doi:10.17713/ajs.v54i1.1975>.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0)

Imports graphics, grDevices, parallel, stats, utils

Config/testthat/edition 3

URL https://github.com/AdamBilchouris/CovEsts

BugReports https://github.com/AdamBilchouris/CovEsts/issues

Repository https://adambilchouris.r-universe.dev

Date/Publication 2026-04-19 09:30:18 UTC

RemoteUrl https://github.com/adambilchouris/covests

RemoteRef HEAD

RemoteSha 3428c0a47e3444cfd33197fe6dafead9873c5eb3

Contents

`adjusted_est` *Compute the Kernel Regression Estimator.*

Description

This function computes the kernel regression estimator of the autocovariance function.

Usage

```
adjusted_est(
  X,
  x,
  t,
  b,
  kernel_name = c("gaussian", "wave", "rational_quadratic", "bessel_j"),
  kernel_params = c(),
  pd = TRUE,
  type = c("autocovariance", "autocorrelation"),
  meanX = mean(X),
  custom_kernel = FALSE,
  parallel = FALSE,
  ncores = parallel::detectCores() - 1,
  cl_export = NULL,
  cl = NULL
)
```

Arguments

`X`	A vector representing observed values of the time series.
`x`	A vector of lags.
`t`	The arguments at which the autocovariance function is calculated at.
`b`	Bandwidth parameter, greater than 0.
`kernel_name`	The name of the symmetric kernel (see kernel_symm_ec) function to be used. Possible values are: gaussian, wave, rational_quadratic, and bessel_j. Alternatively, a custom kernel function can be provided, see the examples.
`kernel_params`	A vector of parameters of the kernel function. See kernel_symm_ec for parameters.
`pd`	Whether a positive-definite estimate should be used. Defaults to `TRUE`.
`type`	Compute either the 'autocovariance' or 'autocorrelation'. Defaults to 'autocovariance'.
`meanX`	The average value of `X`. Defaults to `mean(X)`.
`custom_kernel`	If a custom kernel is to be used or not. Defaults to `FALSE`.
`parallel`	Whether or not the computations should be done in parallel or not. Defaults to `FALSE`.

ncores	The number of cores to be used in the parallel computations. Defaults to the number cores - 1 (threads if hyperthreading is available), calculated from `parallel::detectCores() - 1`.
cl_export	A vector of any additional functions or variables to export for parallel computations. This may be required if `estimator` is not within the package. Defaults to `NULL`.
cl	An optional cluster object created by `parallel::makeCluster`. Defaults to `NULL`, which creates a temporary PSOCK cluster.

Details

The kernel regression estimator of an autocovariance function is defined as

$$\hat{\rho}(t) = \left(\sum_{i=1}^{N} \sum_{j=1}^{N} \check{X}_{ij} K((t - (t_i - t_j))/b) \right) \left(\sum_{i=1}^{N} \sum_{j=1}^{N} K((t - (t_i - t_j))/b) \right)^{-1},$$

where $\check{X}_{ij} = (X(t_i) - \bar{X})(X(t_j) - \bar{X})$.

If `pd` is `TRUE`, the estimator will be made positive-definite through the following procedure

1. Take the discrete Fourier cosine transform, $\widehat{\mathcal{F}}(\theta)$, of the estimated autocovariance function
2. Compute a modified spectrum $\widetilde{\mathcal{F}}(\theta) = \max(\widehat{\mathcal{F}}(\theta), 0)$ for all sample frequencies.
3. Perform the Fourier inversion to obtain a new estimator.

Value

A `CovEsts` S3 object (list) with the following values

`acf` A numeric vector containing the autocovariance/autocorrelation estimates.

`lags` A numeric vector containing the lag indices used to compute the estimates on.

`est_type` The type of estimate, namely 'autocorrelation' or 'autocovariance', this depends on the `type` parameter.

`est_used` The estimator function used, in this case, 'adjusted_est'.

References

Hall, P. & Patil, P. (1994). Properties of nonparametric estimators of autocovariance for stationary random fields. Probability Theory and Related Fields 99(3), 399-424. https://doi.org/10.1007/bf01199899

Hall, P., Fisher, N. I., & Hoffmann, B. (1994). On the nonparametric estimation of covariance functions. The Annals of Statistics 22(4), 2115-2134. https://doi.org/10.1214/aos/1176325774

Examples

```
X <- c(1, 2, 3, 4)
adjusted_est(X, 1:4, 1:3, 0.1, "gaussian")
my_kernel <- function(x, params) {
  stopifnot(params[1] > 0, length(x) >= 1)
 return(exp(-((abs(x) / params[1])^params[2])) * (2 * params[1] * gamma(1 + 1/params[2])))
```

```
}
adjusted_est(X, 1:4, 1:3, 0.1, my_kernel, c(0.25), custom_kernel = TRUE)
## Not run:
library(parallel)
X <- c(1, 2, 3, 4)
my_cl <- makePSOCKcluster(2)
adjusted_est(X, 1:4, 1:3, 0.1, "gaussian", parallel = TRUE, cl = my_cl)
stopCluster(my_cl)

## End(Not run)
```

`adjusted_spline` *Compute Adjusted Splines.*

Description

A helper function that is an implementation of the formula from Choi, Li & Wang (2013, p. 616),

$$f_j^{(l)}(x) = \frac{m+1}{l}\left(f_j^{(l-1)}(x+1) - \tau_{j-p} f_j^{(l-1)}(x) + \tau_{j-p+l+1} f_{j+1}^{(l-1)}(x) - f_{j+1}^{(l-1)}(x+1)\right),$$

where m is the number of nonboundary knots, p is the order of the spline, l is the order of the adjusted spline (the function $f_j^{(l)}(\cdot)$) and $j = 1, 2, \ldots, m+p$.

Usage

```
adjusted_spline(x, j, l, p, m, taus)
```

Arguments

`x`	Argument of the function.
`j`	Index of basis function of order l.
`l`	Order of function.
`p`	The order of the splines.
`m`	The number of nonboundary knots.
`taus`	Vector of τs, see get_taus.

Value

A numeric value of the adjusted spline $f_j^{(l)}(x)$.

References

Choi, I., Li, B. & Wang, X. (2013). Nonparametric Estimation of Spatial and Space-Time Covariance Function. JABES 18, 611-630. https://doi.org/10.1007/s13253-013-0152-z

Examples

```
## Not run:
taus <- get_taus(3, 2)
adjusted_spline(1, 2, 1, 3, 2, taus)

## End(Not run)
```

`area_between` *Area Between Estimated Autocovariance Functions.*

Description

This function estimates the area between two estimated autocovariance functions.

Usage

```
area_between(est1, est2, lags = c(), plot = FALSE)

## S3 method for class 'CovEsts'
area_between(est1, est2, lags = c(), plot = FALSE)

## Default S3 method:
area_between(est1, est2, lags = c(), plot = FALSE)
```

Arguments

`est1`	A numeric vector representing the first estimated autocovariance function.
`est2`	A numeric vector of the same length as `est1` representing the second estimated autocovariance function
`lags`	An optional vector of lags starting from 0 up until some other lag. If empty, a vector of lags is created starting from 0 until `len(est1) - 1`, by 1.
`plot`	A boolean determining whether a plot should be created. By default, no plot is created.

Details

This function estimates the area between two estimated autocovariance functions over a set of lags, from 0 up to h_n defined by

$$\int_0^{h_n} \left| \hat{C}_1(h) - \hat{C}_2(h) \right| dh,$$

where $\hat{C}_1(\cdot)$ and $\hat{C}_2(\cdot)$ are estimated autocovariance functions.

To approximate this integral the trapezoidal rule is used.

If `lags` is empty, a uniform time grid with a step of 1 will be used which may result in a different area than if lags is specified.

Value

A numeric value representing the estimated area between two estimated autocovariance functions.

Methods (by class)

- `area_between(CovEsts)`: Method for `CovEsts` objects.
- `area_between(default)`: Method for numeric vectors.

Examples

```
x <- seq(0, 5, by=0.1)
estCov1 <- exp(-x^2)
estCov2 <- exp(-x^2.1)
area_between(estCov1, estCov2, lags=x)
area_between(estCov1, estCov2, lags=x, plot = TRUE)
```

`as.double.CovEsts` *as.double Method for CovEsts Objects*

Description

This function provides a method for as.double for CovEsts objects, allowing for as.numeric to be called. It extracts the acf values from the object.

Usage

```
## S3 method for class 'CovEsts'
as.double(x, ...)
```

Arguments

`x`	A CovEsts S3 object.
`...`	Refer to base::double.

Value

The acf values from the CovEsts object.

Examples

```
as.numeric(standard_est(c(1, 2, 3)))
```

block_bootstrap *Block Bootstrap*

Description

This function performs block bootstrap (moving or circular) to obtain a $(1-\alpha)\%$ confidence-interval for the autocovariance function. It will also compute average autocovariance function across all bootstrapped estimates.

Usage

```
block_bootstrap(
  X,
  maxLag,
  x = 0:length(X),
  n_bootstrap = 100,
  l = ceiling(length(X)^(1/3)),
  estimator = standard_est,
  type = c("autocovariance", "autocorrelation"),
  alpha = 0.05,
  boot_type = c("moving", "circular"),
  parallel = FALSE,
  ncores = parallel::detectCores() - 1,
  cl_export = NULL,
  boot_seed = NULL,
  cl = NULL,
  ...
)
```

Arguments

X	A vector representing observed values of the time series.
maxLag	The maximum lag to compute the estimated autocovariance function at.
x	A vector of lag indices. Defaults to the sequence `0:length(X)`. Must be at least as large as `maxLag + 1`.
n_bootstrap	The number of times to run moving block bootstrap. Defaults to 100.
l	The block length considered for bootstrap. Defaults to $\lceil N \rceil^{1/3}$, where N is the length of the observation window.
estimator	The function name of the estimator to use. Defaults to `standard\_est`.
type	Compute either the 'autocovariance' or 'autocorrelation'. Defaults to 'autocovariance'.
alpha	The quantile used to compute the $(1-\alpha)\%$ confidence interval. Defaults to 0.05.
boot_type	What type of block bootstrap should be used, either 'moving' for moving block bootstrap or 'circular' for circular block bootstrap.

parallel	Whether or not the bootstrap computations should be done in parallel or not. Defaults to FALSE.
ncores	The number of cores to be used in the parallel bootstrap computations. Defaults to the number cores - 1 (threads if hyperthreading is available), calculated from parallel::detectCores() - 1.
cl_export	A vector of any additional functions or variables to export for parallel computations. This may be required if estimator is not within the package. Defaults to NULL.
boot_seed	An integer seed for reproducibility. This is used for
cl	An optional cluster object created by parallel::makeCluster. Defaults to NULL, which creates a temporary PSOCK cluster.
...	Optional named arguments to the chosen estimator. See the examples.

Details

This function performs block bootstrap to obtain a $(1-\alpha)\%$ confidence-interval for the autocovariance function. It will also compute average autocovariance function across all bootstrapped estimates.

Moving block bootstrap can be described as follows. For some times series $X(1), X(2), \ldots, X(n)$, construct $k \in N$ overlapping blocks of the form $B_i = (X(i), \ldots, X(i+\ell-1))$, where $\ell \in \{1, \ldots, n\}$ is the block length. Randomly sample, with replacement, from the discrete uniform distribution with on $\{1, \ldots, n-\ell+1\}$ to obtain a set of random starting locations $I_1, \ldots, I_k$. Construct a bootstrapped time series $B_1^*, B_2^*, \ldots, B_k^*$, where $B_i^* = B_{I_i}$. The bootstrapped time series is truncated to have length n, and will be of the form $X^*(1), \ldots, X^*(n)$. The autocovariance function is then computed for the bootstrapped time series.

An alternative to moving block bootstrap is circular block bootstrap. Circular block bootstrap uses the time series like a circle, that is, the observation at $n+i$ is the same as the observation at location i. For example, for the block $B_{n-\ell+2}$, we obtain $(X(n-\ell+2), \ldots, X(n), X(n+1))$ is the same as $(X(n-\ell+2), \ldots, X(n), X(1))$. When performing random sampling to obtain starting locations, the set $\{1, \ldots, n\}$ is now considered. The procedure for constructing the bootstrap time series after constructing the blocks and selecting the starting indices is the same as moving block bootstrap.

This process is repeated n_bootstrap times to obtain n_boostrap estimates of the autocovariance function using the bootstrapped time series, for which the average autocovariance function and the $(1-\alpha)\%$ confidence intervals are constructed pointwise for each lag.

The choice of the block length, ℓ, depends on the degree of dependence present in the time series. If the time series has a high degree of dependence, a larger block size should be chosen to ensure the dependency structure is maintained within the block.

Any estimator of the autocovariance function can be used in this function, including a custom estimator not in the package, see the examples.

When parallel = TRUE, L'Ecuyer-CMRG is used to ensure independent random numbers across workers. The serial version uses the Mersenne-Twister algorithm.

Value

A BootEsts S3 object (list) with the following values

`acf_avg` A numeric vector containing the average autocovariance/autocorrelation bootstrap estimate.

`lags` A numeric vector containing the lag indices used to compute the estimates on.

`acf_orig` A numeric vector containing the nonbootstrapped autocovariance/autocorrelation estimate.

`acf_mat` A numeric matrix of the a matrix of all of the bootstrap estimates.

`conf_lower` A numeric vector containing the lower bounds for the estimated pointwise confidence interval.

`conf_upper` A numeric vector containing the upper bounds for the estimated pointwise confidence interval.

`est_type` The type of estimate, namely 'autocorrelation' or 'autocovariance', this depends on the argument `type`.

`est_used` The estimator function used, this depends on the argument `estimator`.

`boot_type` The type of estimate, namely 'moving' or 'circular', this depends on the argument `boot_type`.

`alpha` A numeric value, which is the α value used to compute the confidence intervals, this depends on the argument `alpha`.

References

Chapters 2.5 and 2.7 in Lahiri, S. N. (2003). Resampling Methods for Dependent Data. Springer. https://doi.org/10.1007/978-1-4757-3803-2

Künsch, H. R. (1989). The Jackknife and the Bootstrap for General Stationary Observations. The Annals of Statistics 17(3), 1217-1241. https://doi.org/10.1214/aos/1176347265

Politis, D. N. & Romano, J. P. (1991). A Circular Block-Resampling Procedure for Stationary Data. In R. LePage & L. Billard, eds, Exploring the Limits of Bootstrap, Wiley, 263-270.

Examples

```
X <- c(1, 2, 3, 3, 2, 1)
block_bootstrap(X, 4, n_bootstrap = 3, l = 2, type = 'autocorrelation')
block_bootstrap(X, 4, n_bootstrap = 3, l = 2, type = 'autocovariance')
block_bootstrap(X, 4, n_bootstrap = 3, l = 2, estimator=tapered_est,
    rho = 0.5, window_name = 'blackman', window_params = c(0.16),
    type='autocorrelation')
my_cov_est <- function(X, maxLag) {
  n <- length(X)
  covVals <- rep(0, maxLag + 1)
  for(h in 0:maxLag) {
    covVals_t <- (X[1:(n-h)] - mean(X)) * (X[(1+h):n] - mean(X))
    covVals[h] <- sum(covVals_t) / (n - h)
  }
  return(covVals)
}
block_bootstrap(X, 4, n_bootstrap = 3, l = 2, estimator=my_cov_est)

plot(LakeHuron, main="Lake Huron Levels", ylab="Feet")
```

```
X <- as.vector(LakeHuron)
block_bootstrap(X, 20, n_bootstrap = 100, l = 40, type = 'autocorrelation')
block_bootstrap(X, 20, n_bootstrap = 100, l = 40, type = 'autocorrelation')
block_bootstrap(X, 20, n_bootstrap = 100, l = 40, estimator=tapered_est,
    rho = 0.5, window_name = 'blackman', window_params = c(0.16),
    type='autocorrelation')

my_cov_est <- function(X, maxLag) {
  n <- length(X)
  covVals <- rep(0, maxLag + 1)
  for(h in 0:maxLag) {
    covVals_t <- (X[1:(n-h)] - mean(X)) * (X[(1+h):n] - mean(X))
    covVals[h] <- sum(covVals_t) / (n - h)
  }
  return(covVals)
}
block_bootstrap(X, 20, n_bootstrap = 100, l = 40, estimator = my_cov_est,
    type = 'autocorrelation')
## Not run:
library(parallel)
X <- c(1, 2, 3, 3, 2, 1)
my_cl <- makePSOCKcluster(2)
block_bootstrap(X, 4, n_bootstrap = 1000, l = 3, parallel = TRUE, cl = my_cl)
stopCluster(my_cl)

## End(Not run)
```

`bootstrap_sample` *Block Bootstrap Sample*

Description

This function generates block bootstrap samples for either moving block bootstrap or circular bootstrap.

Usage

```
bootstrap_sample(X, l, k, boot_type = c("moving", "circular"))
```

Arguments

`X`	A vector representing observed values of the time series.
`l`	The block length considered for bootstrap.
`k`	The number of blocks considered for bootstrap.
`boot_type`	What type of block bootstrap should be used, either 'moving' for moving block bootstrap or 'circular' for circular block bootstrap.

Details

This function generates a block bootstrap sample for a time series X. For the moving block bootstrap and circular bootstrap procedures see block_bootstrap and the included references.

Value

A vector of length `length(X)` whose values are a bootstrapped time series.

References

Chapters 2.5 and 2.7 in Lahiri, S. N. (2003). Resampling Methods for Dependent Data. Springer. https://doi.org/10.1007/978-1-4757-3803-2

Künsch, H. R. (1989). The Jackknife and the Bootstrap for General Stationary Observations. The Annals of Statistics 17(3), 1217-1241. https://doi.org/10.1214/aos/1176347265

Politis, D. N. & Romano, J. P. (1991). A Circular Block-Resampling Procedure for Stationary Data. In R. LePage & L. Billard, eds, Exploring the Limits of Bootstrap, Wiley, 263-270.

Examples

```
X <- c(1, 2, 3, 3, 2, 1)
bootstrap_sample(X, 2, 3)
```

---

check_pd *Check if an Autocovariance Function Estimate is Positive-Definite or Not.*

Description

This function checks if an autocovariance function estimate is positive-definite or not by determining if the eigenvalues of the corresponding matrix (see the Details section) are all positive.

Usage

```
check_pd(est)

## S3 method for class 'CovEsts'
check_pd(est)

## Default S3 method:
check_pd(est)
```

Arguments

`est`	A numeric vector, corresponding cyclic matrix representing an estimated autocovariance function, or a `CovEsts` S3 object.

Details

For an autocovariance function estimate $\hat{C}(\cdot)$ over a set of lags separated by a constant difference $\{h_0, h_1, h_2, \ldots, h_n\}$, construct the symmetric matrix

$$\begin{bmatrix} \hat{C}(h_0) & \hat{C}(h_1) & \cdots & \hat{C}(h_{n-1}) & \hat{C}(h_n) \\ \hat{C}(h_1) & \hat{C}(h_0) & \cdots & \hat{C}(h_{n-2}) & \hat{C}(h_{n-1}) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \hat{C}(h_{n-1}) & \hat{C}(h_{n-2}) & \cdots & \hat{C}(h_0) & \hat{C}(h_1) \\ \hat{C}(h_n) & \hat{C}(h_{n-1}) & \cdots & \hat{C}(h_1) & \hat{C}(h_0) \end{bmatrix}.$$

The eigendecomposition of this matrix is computed to determine if all eigenvalues are positive. If so, the estimated autocovariance function is assumed to be positive-definite.

Value

A boolean where `TRUE` denotes a positive-definite autocovariance function estimate and `FALSE` for an estimate that is not positive-definite.

Methods (by class)

- `check_pd(CovEsts)`: Method for `CovEsts` objects.
- `check_pd(default)`: Method for numeric vectors.

Examples

```
x <- seq(0, 5, by=0.1)
estCov <- exp(-x^2)
check_pd(estCov)
```

---

corrected_est *Kernel Correction of the Standard Estimator.*

Description

This function computes the standard autocovariance estimator and applies kernel correction to it,

$$\widehat{C}_T^{(a)}(h) = \widehat{C}(h) a_T(h),$$

where $a_T(h) := a(h/N_T)$. It uses a kernel $a(\cdot)$ which decays or vanishes to zero (depending on the type of kernel) where $a(0) = 1$. The rate or value at which the kernel vanishes is N_T, which is recommended to be of order $0.1N$, where N is the length of the observation window, however, one may need to play with this value.

Usage

```
corrected_est(
  X,
 kernel_name = c("gaussian", "exponential", "wave", "rational_quadratic", "spherical",
    "circular", "bessel_j", "matern", "cauchy"),
  kernel_params = c(),
  N_T = 0.1 * length(X),
  pd = TRUE,
  maxLag = length(X) - 1,
  x = 0:length(X),
  type = c("autocovariance", "autocorrelation"),
  meanX = mean(X),
  custom_kernel = FALSE
)
```

Arguments

X	A vector representing observed values of the time series.
kernel_name	The name of the kernel_ec function to be used. Possible values are: gaussian, exponential, wave, rational_quadratic, spherical, circular, bessel_j, matern, cauchy.
kernel_params	A vector of parameters of the kernel function. See kernel_ec for parameters. In the case of gaussian, wave, rational_quadratic, spherical and circular, N_T takes the place of θ. For kernels that require parameters other than θ, such as the Matern kernel, those parameters are passed.
N_T	The range at which the kernel function vanishes at. Recommended to be $0.1N$ when considering all lags. This parameter may be large for a lag small estimation lag.
pd	Whether a positive-definite estimate should be used. Defaults to TRUE.
maxLag	An optional parameter that determines the maximum lag to compute the estimated autocovariance function at. Defaults to length(X) - 1.
x	A vector of lag indices. Defaults to the sequence 0:length(X). Must be at least as large as maxLag + 1.
type	Compute either the 'autocovariance' or 'autocorrelation'. Defaults to 'autocovariance'.
meanX	The average value of X. Defaults to mean(X).
custom_kernel	If a custom kernel is to be used or not. Defaults to FALSE. See examples.

Details

The aim of this estimator is gradually bring the estimated values to zero through the use of a kernel multiplier. This can be useful when estimating an autocovariance function that is short-range dependent as estimators can have large fluctuations as the lag increases, or to deal with the wave artefacts for large lags, see Bilchouris and Olenko (2025). This estimator can be positive-definite depending on whether the choice of $\widehat{C}(\cdot)$ and a are chosen to be positive-definite or not.

Value

A `CovEsts` S3 object (list) with the following values

`acf` A numeric vector containing the autocovariance/autocorrelation estimates.

`lags` A numeric vector containing the lag indices used to compute the estimates on.

`est_type` The type of estimate, namely 'autocorrelation' or 'autocovariance', this depends on the `type` parameter.

`est_used` The estimator function used, in this case, 'corrected_est'.

References

Yaglom, AM (1987). Correlation Theory of Stationary and Related Random Functions. Volume I: Basic Results. Springer New York. https://doi.org/10.1007/978-1-4612-4628-2

Bilchouris, A. & Olenko, A (2025). On Nonparametric Estimation of Covariogram. Austrian Statistical Society (Vol. 54, Issue 1). https://doi.org/10.17713/ajs.v54i1.1975

Examples

```
X <- c(1, 2, 3)
corrected_est(X, "gaussian")

X <- rnorm(1000)
Y <- c(X[1], X[2])
for(i in 3:length(X)) { Y[i] <- X[i] - 0.3*X[i - 1] - 0.6*X[i - 2] }
plot(Y)
plot(corrected_est(Y, "bessel_j",
     kernel_params=c(0, 1), N_T=0.2*length(Y)))

# Custom kernel
my_kernel <- function(x, theta, params) {
  stopifnot(theta > 0, length(x) >= 1, all(x >= 0))
  return(sapply(x, function(t) ifelse(t == 0, 1,
         ifelse(t == Inf, 0,
      (sin((t^params[1]) / theta) / ((t^params[1]) / theta)) * cos((t^params[2]) / theta)))))
}
plot(corrected_est(Y,
     my_kernel, kernel_params=c(2, 0.25), custom_kernel = TRUE))
```

`cyclic_matrix` *Create a Cyclic Matrix for a Given Vector.*

Description

This helper function creates a symmetric matrix from a given vector v.

Usage

```
cyclic_matrix(v)
```

Arguments

v	A numeric vector.

Details

This function creates a symmetric matrix for a given vector v. If $v = \{v_0, v_1, \ldots, v_{N-1}, v_N\}$, then the symmetric matrix will has the form

$$\begin{bmatrix} v_0 & v_1 & \cdots & v_{N-1} & v_N \\ v_1 & v_0 & \cdots & v_{N-2} & v_{N-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ v_{N-1} & v_{N-2} & \cdots & v_0 & v_1 \\ v_N & v_{N-1} & \cdots & v_1 & v_0 \end{bmatrix}$$

Value

A symmetric matrix.

Examples

```
## Not run:
v <- c(1, 2, 3)
cyclic_matrix(v)

## End(Not run)
```

---

dct_1d *Compute 1D Discrete Cosine Transform*

Description

This function computes the Type-II discrete cosine transform.

Usage

```
dct_1d(X)
```

Arguments

X	A vector of values for which the discrete cosine transform is being computed.

Details

The Type-II discrete cosine transform is obtained using stats::fft. If X is of length N, construct a new signal Y of length $4N$, with values $Y_{2n} = 0, Y_{2n+1} = x_n$ for $0 \leq n < N$, and $Y_{2N} = 0, Y_{4N-n} = y_n$ for $0 < n < 2N$. After this, the Type-II discrete cosine transform is computed by `0.5 * Re(stats::fft(Y))[1:(length(Y) / 4)]`.

Value

A vector of discrete cosine transform values.

References

Ochoa-Dominguez, H. & Rao, K.R. (2019). Discrete Cosine Transform, Second Edition. CRC Press. https://doi.org/10.1201/9780203729854

Makhoul, J. (1980). A Fast Cosine Transform in One and Two Dimensions. IEEE Transactions on Acoustics, Speech, and Signal Processing 28(1), 27-34. https://doi.org/10.1109/TASSP.1980.1163351

Stasiński, R. (2002). DCT Computation Using Real-Valued DFT Algorithms. Proceedings of the 11th European Signal Processing Conference.

Examples

```
X <- c(1, 2, 3)
dct_1d(X)
```

---

generate_knots *Generate Spline Knots.*

Description

A helper function that generates $m + 2$ spline knots of the form:

$$\kappa_0 = 0, \kappa_1 = 1/(m+1), \ldots, \kappa_m = m/(m+1), \kappa_{m+1} = 1.$$

The knots are equally spaced with boundary knots $\kappa_0 = 0$ and $\kappa_{m+1} = 1$.

Usage

```
generate_knots(m)
```

Arguments

m	The number of nonboundary knots.

Value

A numeric vector representing the knots, including the boundary knots.

References

Choi, I., Li, B. & Wang, X. (2013). Nonparametric Estimation of Spatial and Space-Time Covariance Function. JABES 18, 611-630. https://doi.org/10.1007/s13253-013-0152-z

Examples

```
## Not run:
generate_knots(3)

## End(Not run)
```

`get_taus` *Get all τ.*

Description

A helper function to obtain all $\tau_i, i = -p, \ldots, m + p + 1$, where each τ_i is as follows. For $i = -p, -p + 1, \ldots, -2, -1, m + 2, m + 3, \ldots, m + p, m + p + 1$, it is equal to $\tau_i = i/(m + 1)$, and for $i = 0, \ldots, m + 1$, it is $\tau_i = \kappa_i$. See Choi, Li & Wang (2013, p. 615) for details.

Usage

```
get_taus(p, m)
```

Arguments

`p`	The order of the splines.
`m`	The number of nonboundary knots.

Value

A numeric vector of all $\tau_i, i = -p, \ldots, m + p + 1$.

References

Choi, I., Li, B. & Wang, X. (2013). Nonparametric Estimation of Spatial and Space-Time Covariance Function. JABES 18, 611-630. https://doi.org/10.1007/s13253-013-0152-z

Examples

```
## Not run:
get_taus(3, 2)

## End(Not run)
```

H2n *Compute Normalisation Factor*

Description

This helper function is used in the computation of the normalisation factor the function tapered_est,

$$H_{2,n}(0) = \sum_{s=1}^{n} a((s-1/2)/n;\rho)^2,$$

where $a(\cdot;\cdot)$ is a window function.

Usage

```
H2n(
  n,
  rho,
  window_name = c("tukey", "triangular", "sine", "power_sine", "blackman",
    "hann_poisson", "welch"),
  window_params = c(1),
  custom_window = FALSE
)
```

Arguments

n	The sample size.
rho	A scale parameter in $(0, 1]$.
window_name	The name of the window_ec function to be used. Possible values are: tukey, triangular, power_sine, blackman_window, hann_poisson, welch. Alternatively, a custom window function can be provided, see the example for taper.
window_params	A vector of parameters of the window function.
custom_window	If a custom window is to be used or not. Defaults to FALSE.

Value

A single value being $H_{2,n}(0)$.

Examples

```
## Not run:
H2n(3, 0.6, "tukey")

## End(Not run)
```

`hilbert_schmidt` *Hilbert-Schmidt Norm Between Estimated Autocovariance Functions.*

Description

This function computes the Hilbert-Schidmt norm between two estimated autocovariance functions.

Usage

```
hilbert_schmidt(est1, est2)

## S3 method for class 'CovEsts'
hilbert_schmidt(est1, est2)

## Default S3 method:
hilbert_schmidt(est1, est2)
```

Arguments

`est1`	A numeric vector representing the first estimated autocovariance function.
`est2`	A numeric vector of the same length as `est1` representing the second estimated autocovariance function

Details

This function computes the Hilbert-Schidmt norm between two estimated autocovariance functions. The Hilbert-Schmidt norm of a matrix

$$D = [(d_{i,j})_{1\leq i,j\leq n}] = \begin{bmatrix} D(h_0) & D(h_1) & \cdots & D(h_{n-1}) & D(h_n) \\ D(h_1) & D(h_0) & \cdots & D(h_{n-2}) & D(h_{n-1}) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ D(h_{n-1}) & D(h_{n-2}) & \cdots & D(h_0) & D(h_1) \\ D(h_n) & D(h_{n-1}) & \cdots & D(h_1) & D(h_0) \end{bmatrix},$$

over a set of lags $\{h_0, h_1, \ldots, h_N\}$, where $D(h) = \hat{C}_1(h) - \hat{C}_2(h)$, is defined as

$$\|D\|_{HS} = \sqrt{\sum_{i,j} d_{i,j}^2}.$$

Value

A numeric value representing the estimated Hilbert-Schmidt norm between two estimated autocovariance functions.

Methods (by class)

- `hilbert_schmidt(CovEsts)`: Method for `CovEsts` objects.
- `hilbert_schmidt(default)`: Method for numeric vectors.

Examples

```
x <- seq(0, 5, by=0.1)
estCov1 <- exp(-x^2)
estCov2 <- exp(-x^2.1)
hilbert_schmidt(estCov1, estCov2)
```

`idct_1d` *Compute 1D Inverse Discrete Cosine Transform*

Description

This function computes the inverse of the Type-II discrete cosine transform.

Usage

```
idct_1d(X)
```

Arguments

`X`	A vector of values for which the discrete cosine transform is being computed.

Details

The Type-II inverse discrete cosine transform is computed using stats::fft. For autocovariance function estimation, the spectrum is given in the input `X` and then an inverse FFT is applied.

The original spectrum, `dct_full`, from `X` is obtained as follows. First, the original sample Fourier spectrum is reconstructed using `dct_full <- c(X, 0, -X[-1], 0, rev(X[-1]))`. After this, an inverse FFT is applied, `idct <- Re(stats::fft(dct_full, inverse = TRUE)) * (2 / length(dct_full))`, which gives the original function with additional zero-values at even indices. The zeroes are dropped, which gives the untransformed X.

Value

A vector with the inverse transform values.

References

Ochoa-Dominguez, H. & Rao, K.R. (2019). Discrete Cosine Transform, Second Edition. CRC Press. https://doi.org/10.1201/9780203729854

endolith (2013). Fast Cosine Transform via FFT. Signal Processing Stack Exchange. https://dsp.stackexchange.com/a/10606

Examples

```
X <- dct_1d(c(1, 2, 3))
idct_1d(X)
```

kernel_ec *1D Isotropic Kernels.*

Description

This function computes one of the isotropic kernels listed below. Note that `kernel()` is **deprecated**, please use `kernel_ec()` instead. Unlike kernel_symm_ec, these kernels are only defined when $x \geq 0$. They are used as kernel multipliers in estimators corrected_est and kernel_est.

Usage

```
kernel_ec(
  x,
  name = c("gaussian", "exponential", "wave", "rational_quadratic", "spherical",
    "circular", "bessel_j", "matern", "cauchy"),
  params = c(1)
)
```

Arguments

`x`	A vector or matrix of arguments of at least length 1 for which the kernel is computed at.
`name`	The name of the kernel. Options are: gaussian, exponential, wave, rational_quadratic, spherical, circular, bessel_j, matern, and cauchy.
`params`	A vector of parameters for the kernel. See the documentation below for the position of the parameters. All kernels have a scale parameter as the first value in the vector.

Details

Gaussian Kernel. The isotropic Gaussian kernel, which is positive-definite for $R^d, d \in N$, is defined as

$$a(x;\theta) = \exp(-x^2/\theta), \theta > 0.$$

The `params` argument is of the form `c(`θ`)`.

Exponential Kernel. The isotropic exponential kernel, which is positive-definite for $R^d, d \in N$, is defined as

$$a(x;\theta) = \exp(-x/\theta), \theta > 0.$$

The `params` argument is of the form `c(`θ`)`.

Isotropic Wave (Cardinal Sine) Kernel. The isotropic wave (cardinal sine) kernel, which is positive-definite for $R^d, d \leq 3$, is given by

$$a(x;\theta) = \begin{cases} \frac{\theta}{x}\sin\left(\frac{x}{\theta}\right), & x \neq 0 \\ 1, & x = 0 \end{cases}$$

where $\theta > 0$. The `params` argument is of the form `c(`θ`)`.

Isotropic Rational Quadratic Kernel. The isotropic rational quadratic kernel, which is positive-definite for $R^d, d \in N$, is defined as

$$a(x;\theta) = \frac{\theta}{x^2+\theta}, \theta > 0.$$

The `params` argument is of the form `c(`θ`)`.

Isotropic Spherical Kernel. The isotropic spherical kernel, which is positive-definite for $R^3, d \leq 3$, is given by

$$a(x;\theta) = \begin{cases} 1 - \frac{3}{2}\frac{x}{\theta} + \frac{1}{2}\left(\frac{x}{\theta}\right)^3, & x < \theta \\ 0, & \text{otherwise} \end{cases}$$

where $\theta > 0$. The `params` argument is of the form `c(`θ`)`.

Isotropic Circular Kernel. The isotropic circular kernel, which is positive-definite for $R^d, d \leq 2$, is given by

$$a(x;\theta) = \begin{cases} \frac{2}{\pi}\arccos\left(\frac{x}{\theta}\right) - \frac{2}{\pi}\frac{x}{\theta}\sqrt{1-\left(\frac{x}{\theta}\right)^2}, & x < \theta \\ 0, & \text{otherwise} \end{cases}$$

where $\theta > 0$. The `params` argument is of the form `c(`θ`)`.

Isotropic Matérn Kernel. The isotropic Matérn kernel, which is positive-definite for $R^d, d \in N$, and when $\nu > 0$, is defined as

$$a(x;\theta,\nu) = \left(\sqrt{2\nu}\frac{x}{\theta}\right)^{\nu}\left(2^{\nu-1}\Gamma(\nu)\right)^{-1} K_{\nu}\left(\sqrt{2\nu}\frac{x}{\theta}\right),$$

where $\theta > 0$ and $K_{\nu}(\cdot)$ is the modified Bessel function of the second kind. The `params` argument is of the form `c(`θ, ν`)`.

Isotropic Bessel Kernel. The isotropic Bessel kernel, which is positive-definite for $R^d, d \in N$, and when $\nu \geq \frac{d}{2} - 1$, is given by

$$a(x;\theta,\nu) = 2^{\nu}\Gamma(\nu+1)J_{\nu}(x/\theta)(x/\theta)^{-\nu},$$

where $\theta > 0$ and $J_{\nu}(\cdot)$ is the Bessel function of the first kind. The `params` argument is of the form `c(`θ, ν, d `)`.

Isotropic Cauchy Kernel. The isotropic Cauchy kernel, which is positive-definite for $R^d, d \in N$, and when $0 < \alpha \leq 2$ and $\beta \geq 0$, is defined by

$$a(x;\theta,\alpha,\beta) = (1 + (x/\theta)^{\alpha})^{-(\beta/\alpha)}, \theta > 0.$$

The `params` argument is of the form `c(`θ, α, β `)`.

Value

A vector or matrix of kernel values.

References

Genton, M. (2001). Classes of Kernels for Machine Learning: A Statistics Perspective. Journal of Machine Learning Research. 2, 299-312. https://doi.org/10.1162/15324430260185646

Table 4.2 in Hristopulos, D. T. (2020). Random Fields for Spatial Data Modeling: A Primer for Scientists and Engineers. Springer. https://doi.org/10.1007/978-94-024-1918-4

Examples

```
x <- c(0.2, 0.4, 0.6)
theta <- 0.9
kernel_ec(x, "gaussian", c(theta))
kernel_ec(x, "exponential", c(theta))
kernel_ec(x, "wave", c(theta))
kernel_ec(x, "rational_quadratic", c(theta))
kernel_ec(x, "spherical", c(theta))
kernel_ec(x, "circular", c(theta))
nu <- 1
kernel_ec(x, "matern", c(theta, nu))
dim <- 1
kernel_ec(x, "bessel_j", c(theta, nu, dim))
alpha <- 1
beta <- 2
kernel_ec(x, "cauchy", c(theta, alpha, beta))
curve(kernel_ec(x, "gaussian", c(theta)), from = 0, to = 5)
curve(kernel_ec(x, "exponential", c(theta)), from = 0, to = 5)
curve(kernel_ec(x, "wave", c(theta)), from = 0, to = 5)
curve(kernel_ec(x, "rational_quadratic", c(theta)), from = 0, to = 5)
curve(kernel_ec(x, "spherical", c(theta)), from = 0, to = 5)
curve(kernel_ec(x, "circular", c(theta)), from = 0, to = 5)
curve(kernel_ec(x, "matern", c(theta, nu)), from = 0, to = 5)
curve(kernel_ec(x, "bessel_j", c(theta, nu, dim)), from = 0, to = 5)
curve(kernel_ec(x, "cauchy", c(theta, alpha, beta)), from = 0, to = 5)
```

---

kernel_est *Kernel Correction for an Estimated Autocovariance Function.*

Description

This function applies kernel correction to an estimated autocovariance function,

$$\widehat{C}_T^{(a)}(h) = \widehat{C}(h) a_T(h),$$

where $a_T(h) := a(h/N_T)$. It uses a kernel $a(\cdot)$ which decays or vanishes to zero (depending on the type of kernel) where $a(0) = 1$. The rate or value at which the kernel vanishes is N_T, which is recommended to be of order $0.1N$, where N is the length of the observation window, however, one may need to play with this value.

Usage

```
kernel_est(
  estCov,
 kernel_name = c("gaussian", "exponential", "wave", "rational_quadratic", "spherical",
    "circular", "bessel_j", "matern", "cauchy"),
  kernel_params = c(),
  N_T = 0.1 * length(estCov),
  maxLag = length(estCov) - 1,
```

```
  x = 0:length(estCov),
  type = c("autocovariance", "autocorrelation"),
  custom_kernel = FALSE
)

## S3 method for class 'CovEsts'
kernel_est(
  estCov,
 kernel_name = c("gaussian", "exponential", "wave", "rational_quadratic", "spherical",
    "circular", "bessel_j", "matern", "cauchy"),
  kernel_params = c(),
  N_T = 0.1 * length(estCov$acf),
  maxLag = length(estCov$acf) - 1,
  x = estCov$lags,
  type = c("autocovariance", "autocorrelation"),
  custom_kernel = FALSE
)

## Default S3 method:
kernel_est(
  estCov,
 kernel_name = c("gaussian", "exponential", "wave", "rational_quadratic", "spherical",
    "circular", "bessel_j", "matern", "cauchy"),
  kernel_params = c(),
  N_T = 0.1 * length(estCov),
  maxLag = length(estCov) - 1,
  x = 0:length(estCov),
  type = c("autocovariance", "autocorrelation"),
  custom_kernel = FALSE
)
```

Arguments

estCov	A vector whose values are an estimate autocovariance function.
kernel_name	The name of the kernel_ec function to be used. Possible values are: gaussian, exponential, wave, rational_quadratic, spherical, circular, bessel_j, matern, cauchy.
kernel_params	A vector of parameters of the kernel function. See kernel_ec for parameters. In the case of gaussian, wave, rational_quadratic, spherical and circular, N_T takes the place of θ. For kernels that require parameters other than θ, such as the Matern kernel, those parameters are passed.
N_T	The range at which the kernel function vanishes at. Recommended to be $0.1N$ when considering all lags. This parameter may be large for a lag small estimation lag.
maxLag	An optional parameter that determines the maximum lag to compute the estimated autocovariance function at. Defaults to `length(estCov) - 1`.
x	A vector of lag indices. Defaults to the sequence `0:length(X)`. Must be at least as large as `maxLag + 1`.

type	Compute either the 'autocovariance' or 'autocorrelation'. Defaults to 'autocovariance'.
custom_kernel	If a custom kernel is to be used or not. Defaults to FALSE. See the examples of corrected_est for usage.

Value

A vector whose values are the kernel corrected autocovariance estimates or CovEsts S3 object (list) with the following values

acf A numeric vector containing the autocovariance/autocorrelation estimates.

lags A numeric vector containing the lag indices used to compute the estimates on, inherited from the argument estCov.

est_type The type of estimate, namely 'autocorrelation' or 'autocovariance', this depends on the argument type.

est_used The estimator function used, in this case, 'kernel_est'.

If a numeric vector is given for the argument estCov, then a numeric vector output is given, and if a CovEsts S3 object is given, a CovEsts object is given as output.

Methods (by class)

- kernel_est(CovEsts): Method for CovEsts objects.
- kernel_est(default): Method for numeric vectors.

Examples

```
X <- rnorm(1000)
Y <- c(X[1], X[2])
for(i in 3:length(X)) { Y[i] <- X[i] - 0.3*X[i - 1] - 0.6*X[i - 2] }
cov_est <- standard_est(Y)
plot(cov_est)
plot(kernel_est(cov_est,
     "bessel_j", kernel_params=c(0, 1), N_T=0.2*length(Y)))
```

---

kernel_symm_ec *1D Isotropic Symmetric Kernels.*

Description

These functions computes values of kernels that have the properties of symmetric probability distributions. Note that kernel_symm() is **deprecated**, please use kernel_symm_ec() instead. For a kernel $a(x)$, the standardised version is $a(x)/\int_{-\infty}^{\infty} a(x)dx$, so that the integral is 1. The symmetric kernels are different to kernel and are used in the functions adjusted_est and truncated_est.

Usage

```
kernel_symm_ec(
  x,
  name = c("gaussian", "wave", "rational_quadratic", "bessel_j"),
  params = c(1)
)
```

Arguments

x	A vector or matrix of arguments of at least length 1 for which the kernel is computed at. Each value can be negative as well as positive.
name	The name of the kernel. Options are: gaussian, wave, rational_quadratic, bessel_j.
params	A vector of parameters for the kernel. See the documentation below for the position of the parameters. All kernels will have a scale parameter as the first value in the vector.

Details

Symmetric Gaussian Kernel. The symmetric Gaussian kernel is defined as

$$a(x;\theta) = \sqrt{\pi\theta}\exp(-x^2/\theta), \theta > 0.$$

The params argument is of the form c(θ).

Symmetric Wave Kernel. The wave (cardinal sine) kernel is given by

$$a(x;\theta) = \begin{cases} (\sqrt{\theta^2}\pi)^{-1}\frac{\theta}{x}\sin\left(\frac{x}{\theta}\right), & x \neq 0 \\ 1, & x = 0 \end{cases}$$

where $\theta > 0$. The params argument is of the form c(θ)

Symmetric Rational Quadratic Kernel. The symmetric rational quadratic kernel is given by

$$a(x;\theta) = (\pi\sqrt{\theta})^{-1}\left(\frac{\theta}{x^2+\theta}\right), \theta > 0.$$

The params argument is of the form c(θ)

Symmetric Besesel Kernel. The symmetric Bessel kernel, which is valid when $\nu \geq \frac{d}{2} - 1$, is given by

$$a(x;\theta,\nu) = \left(\Gamma\left(\frac{1}{2}+\nu\right)/(2\sqrt{\pi}\theta\Gamma(1+\nu))\right)(2^\nu\Gamma(\nu+1)J_\nu(x/\theta)(x/\theta)^{-\nu}), \theta > 0, \nu \geq \frac{d}{2} - 1,$$

where $J_\nu(\cdot)$ is the Bessel function of the first kind and d is the dimension. The params argument is of the form c(θ, ν, d).

Value

A vector or matrix of values.

Examples

```
x <- c(-2, -1, 0, 1, 2)
theta <- 1
kernel_symm_ec(x, "gaussian", c(theta))
kernel_symm_ec(x, "wave", c(theta))
kernel_symm_ec(x, "rational_quadratic", c(theta))
dim <- 1
nu <- 1
kernel_symm_ec(x, "bessel_j", c(theta, nu, dim))
curve(kernel_symm_ec(x, "gaussian", c(theta)), from = -5, to = 5)
curve(kernel_symm_ec(x, "wave", c(theta)), from = -5, to = 5)
curve(kernel_symm_ec(x, "rational_quadratic", c(theta)), from = -5, to = 5)
curve(kernel_symm_ec(x, "bessel_j", c(theta, nu, dim)), from = -5, to = 5)
```

---

lines.BootEsts *Lines Method for BootEsts Objects*

Description

This function plots a BootEsts object as a line onto another plot.

Usage

```
## S3 method for class 'BootEsts'
lines(x, type = "l", ...)
```

Arguments

x	A BootEsts S3 object.
type	Defaults to 'l', see base::plot for the possible values.
...	Additional plotting arguments, refer to graphics::par.

Value

A line of a BootEsts S3 object.

Examples

```
plot(block_bootstrap(c(1, 2, 3), 2))
lines(block_bootstrap(c(1, 2, 3), 2, pd = FALSE), lwd = 2, lty = 3, col = 'blue')
```

lines.CovEsts *Lines Method for CovEsts Objects*

Description

This function plots a CovEsts object as a line onto another plot.

Usage

```
## S3 method for class 'CovEsts'
lines(x, type = "l", ...)
```

Arguments

x	A CovEst S3 object.
type	Defaults to 'l', see base::plot for the possible values.
...	Additional plotting arguments, refer to graphics::par.

Value

A line of a CovEsts S3 object.

Examples

```
plot(standard_est(c(1, 2, 3)))
lines(standard_est(c(1, 2, 3), pd = FALSE))
```

lines.VarioEsts *Lines Method for VarioEsts Objects*

Description

This function plots a VarioEsts object as a line onto another plot.

Usage

```
## S3 method for class 'VarioEsts'
lines(x, type = "l", ...)
```

Arguments

x	A VarioEsts S3 object.
type	Defaults to 'l', see base::plot for the possible values.
...	Additional plotting arguments, refer to graphics::par.

Value

A line of a VarioEsts S3 object.

Examples

```
plot(to_vario(standard_est(c(1, 2, 3))))
lines(to_vario(standard_est(c(1, 2, 3), pd = FALSE)))
```

---

make_pd *Make a Function Positive-Definite*

Description

This function can make a function positive-definite using the methods proposed by P. Hall and his coauthors.

Usage

```
make_pd(x, method.1 = TRUE)

## S3 method for class 'CovEsts'
make_pd(x, method.1 = TRUE)

## Default S3 method:
make_pd(x, method.1 = TRUE)
```

Arguments

x	A vector of numeric values of an estimated autocovariance function.
method.1	Should method 1 be used (TRUE) or method 2 (FALSE).

Details

This function perform positive-definite adjustments proposed by P. Hall and his coauthors.

Method 1 is as follows:

1. Take the discrete Fourier cosine transform, $\widehat{\mathcal{F}}(\theta)$.
2. Compute a modified spectrum $\widetilde{\mathcal{F}}(\theta) = \max(\widehat{\mathcal{F}}(\theta), 0)$ for all sample frequencies.
3. Perform the Fourier inversion to obtain a new estimator.

Method 2 is as follows:

1. Take the discrete Fourier cosine transform $\widehat{\mathcal{F}}(\theta)$.
2. Find the smallest frequency where its associated value in the spectral domain is negative
$$\hat{\theta} = \inf\{\theta > 0 : \widehat{\mathcal{F}}(\theta) < 0\}.$$
3. Compute a modified spectrum $\widetilde{\mathcal{F}} = \widehat{\mathcal{F}}(\theta)\mathbf{1}(\theta < \hat{\theta})$, where $\mathbf{1}(A)$ is the indicator function over the set A.
4. Perform the Fourier inversion.

Value

A vector that is the adjusted function or a `CovEsts` S3 object (list) with the following values

`acf` A numeric vector containing the autocovariance/autocorrelation estimates.

`lags` A numeric vector containing the lag indices used to compute the estimates on, inherited from the argument `x`.

`est_type` The type of estimate, namely 'autocorrelation' or 'autocovariance', inherited from the argument `x`.

`est_used` The estimator function used, inherited from the argument `x`.

`correction_method` Either 'method.1' or 'method.2' depending on the argument `method.1`

If a numeric vector is given for the argument `x`, then a numeric vector output is given, and if a `CovEsts` S3 object is given, a `CovEsts` object is given as output.

Methods (by class)

- `make_pd(CovEsts)`: Method for `CovEsts` objects.
- `make_pd(default)`: Method for numeric vectors.

References

Hall, P. & Patil, P. (1994). Properties of nonparametric estimators of autocovariance for stationary random fields. Probability Theory and Related Fields 99(3), 399-424. https://doi.org/10.1007/bf01199899

Hall, P., Fisher, N. I., & Hoffmann, B. (1994). On the nonparametric estimation of covariance functions. The Annals of Statistics 22(4), 2115-2134. https://doi.org/10.1214/aos/1176325774

Bilchouris, A. & Olenko, A (2025). On Nonparametric Estimation of Covariogram. Austrian Statistical Society 54(1), 112-137. https://doi.org/10.17713/ajs.v54i1.1975

Examples

```
X <- c(1, 2, 3, 4)
make_pd(X)
check_pd(make_pd(X))
check_pd(make_pd(X, method.1 = FALSE))
```

---

max_distance *Maximum Vertical Distance Between Estimated Functions.*

Description

This function computes the maximum vertical distance between functions.

Usage

```
max_distance(est1, est2, lags = c(), plot = FALSE)

## S3 method for class 'CovEsts'
max_distance(est1, est2, lags = c(), plot = FALSE)

## Default S3 method:
max_distance(est1, est2, lags = c(), plot = FALSE)
```

Arguments

`est1`	A numeric vector representing the first estimated autocovariance function.
`est2`	A numeric vector of the same length as `est1` representing the second estimated autocovariance function
`lags`	An optional vector of lags starting from 0 up until some other lag. If empty, a vector of lags is created starting from 0 until `len(est1) - 1`, by 1.
`plot`	A boolean as to whether a plot should be created. By default, no plot is created.

Details

This function computes the maximum vertical distance between functions:

$$D(\hat{C}_1(h), \hat{C}_2(h)) = \max_h \left| \hat{C}_1(h) - \hat{C}_2(h) \right|,$$

where $\hat{C}_1(\cdot)$ and $\hat{C}_2(\cdot)$ are estimated autocovariance functions. It assumes that the estimated values are given for the same set of lags. The vectors of the function values must be of the same length.

Value

A numeric value representing the maximum vertical distance between the two estimated functions.

Methods (by class)

- `max_distance(CovEsts)`: Method for `CovEsts` objects.
- `max_distance(default)`: Method for numeric vectors.

Examples

```
x <- seq(0, 5, by=0.1)
estCov1 <- exp(-x^2)
estCov2 <- exp(-x^2.1)
max_distance(estCov1, estCov2, lags=x)
max_distance(estCov1, estCov2, lags=x, plot = TRUE)
```

mse *MSE Between Estimated Autocovariance Functions.*

Description

This function computes the mean-square difference/error between two autocovariance functions (estimated or theoretical).

Usage

```
mse(est1, est2)

## S3 method for class 'CovEsts'
mse(est1, est2)

## Default S3 method:
mse(est1, est2)
```

Arguments

`est1`	A numeric vector representing the first estimated autocovariance function.
`est2`	A numeric vector of the same length as `est1` representing the second estimated (or theoretical) autocovariance function

Details

This function computes the mean-square difference/error (MSE) between two estimated autocovariance functions (estimated or theoretical). The MSE is defined as

$$\frac{1}{n}\sum_{i=0}^{n}\left(\widehat{C}_1(h_i) - \widehat{C}_2(h_i)\right)^2$$

over a set of lags $\{h_0, h_1, h_2, \ldots, h_n\}$.

Value

A numeric value representing the MSE between two autocovariance functions (estimated or theoretical).

Methods (by class)

- `mse(CovEsts)`: Method for `CovEsts` object.
- `mse(default)`: Method for numeric vectors.

Examples

```
x <- seq(0, 5, by=0.1)
estCov1 <- exp(-x^2)
estCov2 <- exp(-x^2.1)
mse(estCov1, estCov2)
```

`nearest_pd` *Compute the Nearest Positive-Definite Matrix.*

Description

This function computes the nearest positive-definite matrix to some matrix A.

Usage

```
nearest_pd(X)

## S3 method for class 'CovEsts'
nearest_pd(X)

## Default S3 method:
nearest_pd(X)
```

Arguments

`X`	Either a numeric square matrix. If a vector is provided, a matrix will be created of the form found in cyclic_matrix.

Details

This function computes the nearest positive-definite matrix to some matrix A.

The procedure to do so is as follows For a matrix X, compute the symmetric matrix $B = (A + A^T)/2$. Let $B = UH$ be the polar decomposition of B. The nearest positive-definite matrix to X is $X_F = (B + H)/2$.

Unlike shrinking, only an autocorrelation matrix can be returned, not an autocovariance function.

The implementation is a translation of https://au.mathworks.com/matlabcentral/fileexchange/42885-nearestspd#functions_tab .

Value

The closest positive-definite autocorrelation matrix.

Methods (by class)

- `nearest_pd(CovEsts)`: Method for `CovEsts` objects.
- `nearest_pd(default)`: Method for numeric vectors.

References

Higham, N. J. (1988). Computing a nearest symmetric positive semidefinite matrix. Linear Algebra and its Applications, 103, 103-118. https://doi.org/10.1016/0024-3795(88)90223-6

D'Errico, J. (2025). nearestSPD (https://www.mathworks.com/matlabcentral/fileexchange/42885-nearestspd), MATLAB Central File Exchange. Retrieved August 2, 2025.

Examples

```
X <- c(1, 0, -1.1)
nearest_pd(X)
check_pd(nearest_pd(X))
```

`normalise_acf` *Normalise a CovEsts Object*

Description

This function normalises a CovEsts S3 object, that is, turning an autocovariance function into an autocorrelation function. This procedure is a one-way transformation. This function does not support any other type of argument.

Usage

```
normalise_acf(estCov)

## Default S3 method:
normalise_acf(estCov)

## S3 method for class 'CovEsts'
normalise_acf(estCov)
```

Arguments

`estCov`	A CovEst S3 object.

Value

A CovEsts S3 object with a normalised `acf` attribute.

Methods (by class)

- `normalise_acf(default)`: Method for other objects.
- `normalise_acf(CovEsts)`: Method for 'CovEsts' objects.

Examples

```
normalise_acf(standard_est(c(1, 2, 3)))
```

plot.BootEsts *Plot Method for BootEsts Objects*

Description

This function plots a BootEsts object. It plots the confidence region, average bootstrap estimate, and the original nonbootstrapped estimate.

Usage

```
## S3 method for class 'BootEsts'
plot(x, type = "l", xlab = "Lag (h)", ylab = "ACF", ...)
```

Arguments

x	A BootEsts S3 object.
type	Defaults to 'l', see base::plot for the possible values.
xlab	Defaults to 'Lag (h)', see graphics::title for more information.
ylab	Defaults to 'ACF', see graphics::title for more information.
...	Additional plotting arguments, refer to graphics::par.

Value

A plot of a BootEsts S3 object.

Examples

```
plot(block_bootstrap(c(1, 2, 3), 2))
```

plot.CovEsts *Plot Method for CovEsts Objects*

Description

This function plots a CovEsts object, over the lags used to estimate the autocovariance values.

Usage

```
## S3 method for class 'CovEsts'
plot(x, type = "l", xlab = "Lag (h)", ylab = "ACF", ...)
```

Arguments

x	A CovEst S3 object.
type	Defaults to 'l', see base::plot for the possible values.
xlab	Defaults to 'Lag (h)', see graphics::title for more information.
ylab	Defaults to 'ACF', see graphics::title for more information.
...	Additional plotting arguments, refer to graphics::par.

Value

A plot of a CovEsts S3 object.

Examples

```
plot(standard_est(c(1, 2, 3)))
```

---

plot.VarioEsts *Plot Method for VarioEsts Objects*

Description

This function plots a VarioEsts object, over the lags used to estimate the variogram values.

Usage

```
## S3 method for class 'VarioEsts'
plot(x, type = "l", xlab = "Lag (h)", ylab = expression(gamma(h)), ...)
```

Arguments

x	A VarioEsts S3 object.
type	Defaults to 'l', see base::plot for the possible values.
xlab	Defaults to 'Lag (h)', see graphics::title for more information.
ylab	Defaults to expression(gamma(h)), see graphics::title for more information.
...	Additional plotting arguments, refer to graphics::par.

Value

A plot of a VarioEsts S3 object.

Examples

```
plot(to_vario(standard_est(c(1, 2, 3))))
```

print.BootEsts *Print Method for BootEsts Objects*

Description

This function plots a BootEsts object, printing estimtor type, bootstrap type, estimator used, the bootstrap average estimate, the original estimate, and the pointwise confidence interval.

Usage

```
## S3 method for class 'BootEsts'
print(x, n_head = 5, n_tail = 5, digits = NULL, ...)
```

Arguments

x	A BootEsts S3 object.
n_head	The number of 'head' values to print, defaults to 5.
n_tail	The number of 'tail' values to print, defaults to 5.
digits	The number of digits to print. Defaults to NULL. Note, this value is passed into base::format.
...	Addition printing parameters, see base::print.

Value

Prints a BootEsts object.

Examples

```
print(block_bootstrap(c(1, 2, 3), 2))
```

print.CovEsts *Print Method for CovEsts Objects*

Description

This function plots a CovEsts object, printing the estimator type, estimator used, lags and values.

Usage

```
## S3 method for class 'CovEsts'
print(x, n_head = 5, n_tail = 5, digits = NULL, ...)
```

Arguments

x	A CovEst S3 object.
n_head	The number of 'head' values to print, defaults to 5.
n_tail	The number of 'tail' values to print, defaults to 5.
digits	The number of digits to print. Defaults to NULL. Note, this value is passed into base::format.
...	Addition printing parameters, see base::print.

Value

Prints a CovEsts object.

Examples

```
print(standard_est(c(1, 2, 3)))
```

---

print.VarioEsts *Print Method for VarioEsts Objects*

Description

This function plots a VarioEsts object, printing the lags and values.

Usage

```
## S3 method for class 'VarioEsts'
print(x, n_head = 5, n_tail = 5, digits = NULL, ...)
```

Arguments

x	A VarioEsts S3 object.
n_head	The number of 'head' values to print, defaults to 5.
n_tail	The number of 'tail' values to print, defaults to 5.
digits	The number of digits to print. Defaults to NULL. Note, this value is passed into base::format.
...	Addition printing parameters, see base::print.

Value

Prints a VarioEsts object.

Examples

```
print(to_vario(standard_est(c(1, 2, 3))))
```

rho_T1 *Compute $\rho(T\_1)$ used in the Truncated Kernel Regression Estimator.*

Description

This helper function computes $\rho(T_1)$ used in the truncated kernel regression estimator, truncated_est.

Usage

```
rho_T1(
  x,
  meanX,
  T1,
  b,
  xij_mat,
  kernel_name = c("gaussian", "wave", "rational_quadratic", "bessel_j"),
  kernel_params = c(),
  custom_kernel = FALSE
)
```

Arguments

x	A vector of lags.
meanX	The average value of X.
T1	The first trunctation point.
b	Bandwidth parameter, greater than 0.
xij_mat	The matrix of pairwise covariance values.
kernel_name	The name of the symmetric kernel (see kernel_symm_ec) function to be used. Possible values are: gaussian, wave, rational_quadratic, and bessel_j. Alternatively, a custom kernel function can be provided, see the examples.
kernel_params	A vector of parameters of the kernel function. See kernel_symm_ec for parameters.
custom_kernel	If a custom kernel is to be used or not. Defaults to FALSE.

Details

This function computes the following value,

$$\hat{\rho}(T_1) = \left(\sum_{i=1}^{N} \sum_{j=1}^{N} \check{X}_{ij} K((T_1 - (t_i - t_j))/b) \right) \left(\sum_{i=1}^{N} \sum_{j=1}^{N} K((T_1 - (t_i - t_j)))/b) \right)^{-1},$$

where $\check{X}_{ij} = (X(t_i) - \bar{X})(X(t_j) - \bar{X})$, which is then used in truncated_est,

$$\hat{\rho}_1(t) = \begin{cases} \hat{\rho}(t) & 0 \le t \le T_1 \\ \hat{\rho}(T_1)(T_2 - t)(T_2 - T_1)^{-1} & T_1 < t \le T_2 \\ 0 & t > T_2 \end{cases}.$$

Value

The estimated autocovariance function at T_1.

References

Hall, P. & Patil, P. (1994). Properties of nonparametric estimators of autocovariance for stationary random fields. Probability Theory and Related Fields 99(3), 399-424. https://doi.org/10.1007/bf01199899

Hall, P., Fisher, N. I., & Hoffmann, B. (1994). On the nonparametric estimation of covariance functions. The Annals of Statistics 22(4), 2115-2134. https://doi.org/10.1214/aos/1176325774

Examples

```
## Not run:
X <- c(1, 2, 3, 4)
rho_T1(1:4, mean(X), 1, 0.1, Xij_mat(X, mean(X)), "gaussian", c(), FALSE)
my_kernel <- function(x, params) {
  stopifnot(params[1] > 0, length(x) >= 1)
 return(exp(-((abs(x) / params[1])^params[2])) * (2 * params[1] * gamma(1 + 1/params[2])))
}
rho_T1(1:4, mean(X), 1, 0.1, Xij_mat(X, mean(X)), my_kernel, c(0.25), TRUE)

## End(Not run)
```

---

shrinking *Linear Shrinking*

Description

This function corrects an autocovariance/autocorrelation function estimate via linear shrinking of the autocorrelation matrix.

Usage

```
shrinking(estCov, return_matrix = FALSE, target = NULL)

## S3 method for class 'CovEsts'
shrinking(estCov, return_matrix = FALSE, target = NULL)

## Default S3 method:
shrinking(estCov, return_matrix = FALSE, target = NULL)
```

Arguments

estCov	A vector whose values are an estimate autocovariance/autocorrelation function.
return_matrix	A boolean determining whether the shrunken matrix or the corresponding vector is returned. If FALSE, it returns a vector whose values are the shrunken autocorrelation function. Defaults to FALSE.
target	A shrinkage target matrix used in the shrinking process. This should only be used if you wish to use a specific matrix as the target.

Details

This function corrects an autocovariance/autocorrelation function estimate via linear shrinking of the autocorrelation matrix. The shrunken autocorrelation matrix is computed as follows

$$\widetilde{R} = \lambda R + (1 - \lambda) I_p,$$

where $\widetilde{R}$ is the shrunken autocorrelation matrix, R is the original autocorrelation matrix, $\lambda \in [0, 1]$, and I_p is the $p \times p$ identity matrix. λ is chosen in such a away that largest value which still results in a positive-definite matrix. The shrunken matrix will be positive-definite.

Value

A vector with values of the shrunken autocorrelation function, the corresponding matrix (depending on `return_matrix`), or `CovEsts` S3 object (list) with the following values

`acf` A numeric vector containing the shrunken autocovariance/autocorrelation estimates.

`lags` A numeric vector containing the lag indices used to compute the estimates on, inherited from the argument `estCov`.

`est_type` The type of estimate, namely 'autocorrelation' or 'autocovariance', this depends on the argument `type`.

`est_used` The estimator function used, in this case, inherited from the argument `estCov`.

`correction_method` This value is 'shrinking'.

`lambda` The λ value obtained during the shrinking process.

If a numeric vector is given for the argument `estCov`, then a numeric vector output is given, and if a `CovEsts` S3 object is given, a `CovEsts` object is given as output.

Methods (by class)

- `shrinking(CovEsts)`: Method for `CovEsts` objects.
- `shrinking(default)`: Method for numeric vectors

References

Devlin, S. J., Gnanadesikan R. & Kettenring, J. R. (1975). Robust Estimation and Outlier Detection with Correlation Coefficients. Biometrika, 62(3), 531-545. 10.1093/biomet/62.3.531

Rousseeuw, P. J. & Molenberghs, G. (1993). Transformation of Non Positive Semidefinite Correlation Matrices. Communications in Statistics - Theory and Methods, 22(4), 965-984. 10.1080/03610928308831068

Examples

```
estCorr <- c(1, 0.8, 0.5, -1.2)
shrinking(estCorr)
target <- diag(length(estCorr))
shrinking(estCorr, TRUE, target)
```

`solve_shrinking` *Solve Linear Shrinking*

Description

This is an objective function used to select $\lambda \in [0, 1]$ in linear shrinking, see shrinking.

Usage

```
solve_shrinking(par, corr_mat, target)
```

Arguments

`par`	The initial parameter used in the maximisation process.
`corr_mat`	The autocorrelation matrix of the considered time series.
`target`	A shrinkage target matrix used in the shrinking process. This should only be used if you wish to use a specific matrix as the target.

Value

A numeric value that is either equal to $-$`par` or 1.

References

Devlin, S. J., Gnanadesikan R. & Kettenring, J. R. (1975). Robust Estimation and Outlier Detection with Correlation Coefficients. Biometrika, 62(3), 531-545. 10.1093/biomet/62.3.531

Rousseeuw, P. J. & Molenberghs, G. (1993). Transformation of Non Positive Semidefinite Correlation Matrices. Communications in Statistics - Theory and Methods, 22(4), 965-984. 10.1080/03610928308831068

Examples

```
## Not run:
estCorr <- c(1, 0.5, 0)
corr_mat <- cyclic_matrix(estCorr)
solve_shrinking(0.5, corr_mat, diag(length(estCorr)))

## End(Not run)
```

`solve_spline` *Objective Function for WLS.*

Description

This is the objective function to find the weights for each basis function in the minimising spline, see Choi, Li & Wang (2013, p. 617). The parameters must be nonnegative, so a penalty of 10^{12} is given if any parameters are negative. The weights are chosen as per Choi, Li & Wang (2013, p. 617).

Usage

```
solve_spline(par, splines_df, weights)
```

Arguments

`par`	A vector of initial parameters to used in the minimisation process.
`splines_df`	A data frame whose structure is defined in splines_df.
`weights`	A vector of weights, see the description.

Details

Let $\beta = (\beta_0, \ldots, \beta_{m+p})'$ be a vector of model coefficients, $\{f_1^{(p-1)}, \ldots, f_{m+p}^{(p-1)}\}$ be a set of completely monotone basis functions, and $\widehat{C}(\cdot)$ be an estimated covariance function. As per Choi, Li & Wang (2013, p. 617), β can be estimated via weighted-least squares,

$$\hat{\beta}_{WLS} = \arg\min_{\beta_j \geq 0} \sum_{i=1}^{L} w_i \left(\widehat{C}(h_i) - \sum_{j=1}^{m+p} \beta_j f_j^{(p-1)}(h_i^2) \right)^2,$$

where $\{h_1, \ldots, h_L\}$ is a set of lags and $\{w_1, \ldots, w_L\}$ is a set of weights. The set of weights is calculated in splines_est, and they are of the form $w_i = (N - h_i)/((1 - \widehat{C}(h_i))^2)$.

Value

The value of the objective function at those parameters.

References

Choi, I., Li, B. & Wang, X. (2013). Nonparametric Estimation of Spatial and Space-Time Covariance Function. JABES 18, 611-630. https://doi.org/10.1007/s13253-013-0152-z

Examples

```
## Not run:
taus <- get_taus(3, 2)
x <- seq(0, 2, by=0.25)
maxLag <- 4
splines_df <- splines_df(x[1:maxLag], 3, 2, taus)
splines_df$estCov <- exp(-splines_df$lags^2) + 0.001
# pars are the initial parameters used in the minimisation process.
pars <- c(0.5, 0.5, 0.5, 0.5, 0.5)
weights <- c()
X <- rnorm(50)
for(i in 0:(maxLag - 1)) {
  weights <- c(weights, (length(X) - i) / ( (1 - splines_df$estCov[i + 1])^2 ))
}
solve_spline(pars, splines_df, weights)

## End(Not run)
```

`spectral_norm` *Compute the Spectral Norm Between Estimated Functions.*

Description

This function computes the spectral norm of the difference of two estimated autocovariance functions. This function is intended for estimates over lags with a constant difference.

Usage

```
spectral_norm(est1, est2)

## S3 method for class 'CovEsts'
spectral_norm(est1, est2)

## Default S3 method:
spectral_norm(est1, est2)
```

Arguments

`est1`	A numeric vector representing the first estimated autocovariance function.
`est2`	A numeric vector of the same length as `est1` representing the second estimated autocovariance function

Details

This function computes the spectral norm of the difference of two estimated autocovariance functions. Let $D(h) = \hat{C}_1(h) - \hat{C}_2(h)$, where $\hat{C}_1(\cdot)$ and $\hat{C}_2(\cdot)$ are estimated autocovariance functions.

A matrix D is created from $D(\cdot)$,

$$\begin{bmatrix} D(h_0) & D(h_1) & \cdots & D(h_{n-1}) & D(h_n) \\ D(h_1) & D(h_0) & \cdots & D(h_{n-2}) & D(h_{n-1}) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ D(h_{n-1}) & D(h_{n-2}) & \cdots & D(h_0) & D(h_1) \\ D(h_n) & D(h_{n-1}) & \cdots & D(h_1) & D(h_0) \end{bmatrix},$$

over a set of lags $\{h_0, h_1, \ldots, h_N\}$. This matrix is created by cyclic_matrix.

The spectral norm is defined as the largest eigenvalue of D.

Value

The spectral norm of the differences between the two functions.

Methods (by class)

- `spectral_norm(CovEsts)`: Method for `CovEsts` objects.
- `spectral_norm(default)`: Method for numeric vectors.

Examples

```
x <- seq(0, 5, by=0.1)
estCov1 <- exp(-x^2)
estCov2 <- exp(-x^2.1)
spectral_norm(estCov1, estCov2)
```

---

splines_df *Construct Data Frame of Basis Functions.*

Description

This helper function constructs a data frame with the following structure:

- One column for the x-values
- m + p columns values of squared basis functions evaluated at the corresponding x.

Usage

```
splines_df(x, p, m, taus)
```

Arguments

`x`	A vector of lags.
`p`	The order of the splines.
`m`	The number of nonboundary knots.
`taus`	Vector of τs, see get_taus.

Value

A data frame of the above structure.

Examples

```
## Not run:
taus <- get_taus(3, 2)
splines_df(seq(0, 2, by=0.25), 3, 2, taus)

## End(Not run)
```

`splines_est` *Compute the Splines Estimator.*

Description

Compute the estimated covariance function by using the method from Choi, Li & Wang (2013, pp. 614-617).

$$C(\tau) = \sum_{j=1}^{m+p} \beta_j f_j^{(p-1)}(\tau^2),$$

where m is the number of nonboundary knots, p is the order of the splines, τ is the isotropic distance, β_j are nonnegative weights and $f_j^{(p)}$ are basis functions of order p. For optimisation, the Nelder-Mead and L-BFGS-B methods are used, the one which selects parameters which minimises the objective function is chosen.

Usage

```
splines_est(
  X,
  x,
  estCov,
  p,
  m,
  maxLag = length(X) - 1,
  type = c("autocovariance", "autocorrelation"),
  initial_pars = c(),
  control = list(maxit = 1000)
)

## S3 method for class 'CovEsts'
splines_est(
  X,
  x,
  estCov,
  p,
  m,
```

```
  maxLag = length(X) - 1,
  type = c("autocovariance", "autocorrelation"),
  initial_pars = c(),
  control = list(maxit = 1000)
)

## Default S3 method:
splines_est(
  X,
  x,
  estCov,
  p,
  m,
  maxLag = length(X) - 1,
  type = c("autocovariance", "autocorrelation"),
  initial_pars = c(),
  control = list(maxit = 1000)
)
```

Arguments

`X`	A vector representing observed values of the time series.
`x`	A vector of lag indices.
`estCov`	An estimated autocovariance function to fit to (a vector).
`p`	The order of the splines.
`m`	The number of nonboundary knots.
`maxLag`	An optional parameter that determines the maximum lag to compute the estimated autocovariance function at. Defaults to `length(X) - 1`.
`type`	Compute either the 'autocovariance' or 'autocorrelation'. Defaults to 'autocovariance'.
`initial_pars`	An optional vector of parameters - can be used to fine tune the fit. By default, it is a vector of 0.5 whose length is $m + p$.
`control`	An optional list of optimisation parameters used in the optimisation process, see `control` in stats::optim.

Details

Due to the weighting scheme, the autocovariance at lag zero cannot be 1,

$$w_i = \frac{N - i}{1 - C(i)}$$

Value

A vector whose values are the spline autocovariance estimates or a `CovEsts` S3 object (list) with the following values

`acf` A numeric vector containing the autocovariance/autocorrelation estimates.

`lags` A numeric vector containing the lag indices used to compute the estimates on.

`est_type` The type of estimate, namely 'autocorrelation' or 'autocovariance', this depends on the `type` parameter.

`est_used` The estimator function used, in this case, 'splines_est'.

If a numeric vector is given for the argument `estCov`, then a numeric vector output is given, and if a `CovEsts` S3 object is given, a `CovEsts` object is given as output.

Methods (by class)

- `splines_est(CovEsts)`: Method for 'CovEsts' objects.
- `splines_est(default)`: Method for numeric vectors.

References

Choi, I., Li, B. & Wang, X. (2013). Nonparametric Estimation of Spatial and Space-Time Covariance Function. JABES 18, 611-630. https://doi.org/10.1007/s13253-013-0152-z

Examples

```
X <- rnorm(100)
x <- seq(0, 5, by = 0.25)
maxLag <- 5
estCov <- standard_est(X, maxLag = maxLag)
estimated <- splines_est(X, x, estCov, 3, 2, maxLag = maxLag)
estimated
```

`standard_est` *Computes the Standard Estimator of the Autocovariance Function.*

Description

This function computes the following two estimates of the autocovariance function depending on the parameter pd, see the Details section.

Usage

```
standard_est(
  X,
  pd = TRUE,
  maxLag = length(X) - 1,
  x = 0:length(X),
  type = c("autocovariance", "autocorrelation"),
  meanX = mean(X)
)
```

Arguments

X	A vector representing observed values of the time series.
pd	Whether a positive-definite estimate should be used. Defaults to TRUE.
maxLag	An optional parameter that determines the maximum lag to compute the estimated autocovariance function at. Defaults to length(X) - 1.
x	A vector of lag indices. Defaults to the sequence 0:length(X). Must be at least as large as maxLag + 1.
type	Compute either the 'autocovariance' or 'autocorrelation'. Defaults to 'autocovariance'.
meanX	The average value of X. Defaults to mean(X).

Details

For pd = TRUE:

$$\widehat{C}(h) = \frac{1}{N}\sum_{j=1}^{N-h}(X(j) - \bar{X})(X(j+h) - \bar{X}).$$

For pd = FALSE:

$$\widehat{C}(h) = \frac{1}{N-h}\sum_{j=1}^{N-h}(X(j) - \bar{X})(X(j+h) - \bar{X}).$$

This function will generate autocovariance values for lags h from the set $\{0, \ldots, \text{maxLag}\}$.

The positive-definite estimator must be used cautiously when estimating over all lags as the sum of all values of the autocorrelation function equals to $-1/2$. For the nonpositive-definite estimator a similar constant summation property holds.

Value

A CovEsts S3 object (list) with the following values

acf A numeric vector containing the autocovariance/autocorrelation estimates.

lags A numeric vector containing the lag indices used to compute the estimates on.

est_type The type of estimate, namely 'autocorrelation' or 'autocovariance', this depends on the type parameter.

est_used The estimator function used, in this case, 'standard_est'.

References

Bilchouris, A. & Olenko, A (2025). On Nonparametric Estimation of Covariogram. Austrian Statistical Society 54(1), 112-137. https://doi.org/10.17713/ajs.v54i1.1975

Examples

```
X <- c(1, 2, 3)
standard_est(X, pd = FALSE, maxLag = 2, meanX = mean(X))
```

`starting_locs` *Random Block Locations*

Description

This function performs random sampling to obtain random starting locations for block bootstrap.

Usage

```
starting_locs(N, l, k, boot_type = c("moving", "circular"))
```

Arguments

`N`	The length of the observation window.
`l`	The block length considered for bootstrap.
`k`	The number of blocks considered for bootstrap.
`boot_type`	What type of block bootstrap should be used, either 'moving' for moving block bootstrap or 'circular' for circular block bootstrap.

Details

This function performs random sampling to obtain random starting locations for block bootstrap. If `type = 'moving'`, the set $\{1, \ldots, N-\ell+1\}$ is randomly sampled, with replacement, k times to obtain random block locations for moving block bootstrap. If `type = 'circular'`, the set $\{1, \ldots, N\}$ is randomly sampled, with replacement, k times to obtain random block locations for moving block bootstrap.

Value

A vector of length `k` whose values are random block locations.

References

Chapters 2.5 and 2.7 in Lahiri, S. N. (2003). Resampling Methods for Dependent Data. Springer. https://doi.org/10.1007/978-1-4757-3803-2

Künsch, H. R. (1989). The Jackknife and the Bootstrap for General Stationary Observations. The Annals of Statistics 17(3), 1217-1241. https://doi.org/10.1214/aos/1176347265

Politis, D. N. & Romano, J. P. (1991). A Circular Block-Resampling Procedure for Stationary Data. In R. LePage & L. Billard, eds, Exploring the Limits of Bootstrap, Wiley, 263-270.

Examples

```
starting_locs(4, 2, 2)
```

taper *Compute the Function $a(x;\rho)$.*

Description

This helper function computes the taper function for a given window function as

$$a(x;\rho) = \begin{cases} w(2x/\rho) & 0 \le x < \frac{1}{2}\rho, \\ 1 & \frac{1}{2}\rho \le x \le \frac{1}{2} \\ a(1-x;\rho) & \frac{1}{2} < x \le 1 \end{cases},$$

where $w(\cdot)$ is a continuous increasing function with $w(0) = 0, w(1) = 1$, $\rho \in (0, 1]$, and $x \in [0, 1]$. The possible window function choices are found in window_ec.

Usage

```
taper(
  x,
  rho,
  window_name = c("tukey", "triangular", "sine", "power_sine", "blackman",
    "hann_poisson", "welch"),
  window_params = c(1),
  custom_window = FALSE
)
```

Arguments

x	A vector of numbers between 0 and 1 (inclusive).
rho	A scale parameter in $(0, 1]$.
window_name	The name of the window_ec function to be used. Possible values are: tukey, triangular, power_sine, blackman_window, hann_poisson, welch. Alternatively, a custom window function can be provided, see the example.
window_params	A vector of parameters of the window function.
custom_window	If a custom window is to be used or not. Defaults to FALSE.

Value

A vector of taper values.

Examples

```
X <- c(0.1, 0.2, 0.3)
taper(X, 0.5, "tukey")
curve(taper(x, 1, "tukey"), from = 0, to = 1)
curve(taper(x, 1, "power_sine", c(4)), from = 0, to = 1)
my_taper <- function(x, ...) {
  return(x)
}
taper(X, 0.5, my_taper, custom_window = TRUE)
```

`tapered_est` *Compute the Estimated Tapered Autocovariance Function over a Set of Lags.*

Description

This function computes the tapered autocovariance over a set of lags.

Usage

```
tapered_est(
  X,
  rho,
  window_name = c("tukey", "triangular", "sine", "power_sine", "blackman",
    "hann_poisson", "welch"),
  window_params = c(1),
  maxLag = length(X) - 1,
  x = 0:length(X),
  type = c("autocovariance", "autocorrelation"),
  meanX = mean(X),
  custom_window = FALSE
)
```

Arguments

`X`	A vector representing observed values of the time series.
`rho`	A scale parameter in $(0, 1]$.
`window_name`	The name of the window_ec function to be used. Possible values are: tukey, triangular, sine, power_sine, blackman_window, hann_poisson, welch. Alternatively, a custom window_ec function can be provided, see the example in taper.
`window_params`	A vector of parameters of the window function.
`maxLag`	An optional parameter that determines the maximum lag to compute the estimated autocovariance function at. Defaults to `length(X) - 1`.
`x`	A vector of lag indices. Defaults to the sequence `0:length(X)`. Must be at least as large as `maxLag + 1`.
`type`	Compute either the 'autocovariance' or 'autocorrelation'. Defaults to 'autocovariance'.
`meanX`	The average value of `X`. Defaults to `mean(X)`.
`custom_window`	If a custom window is to be used or not. Defaults to `FALSE`.

Details

This function computes the estimated tapered autocovariance over a set of lags,

$$\widehat{C}_N^a(h) = (H_{2,n}(0))^{-1} \sum_{j=1}^{N-h} (X(j) - \bar{X})(X(j+h) - \bar{X}) a((j-1/2)/N; \rho) a((j+h-1/2)/N; \rho),$$

where $a(\cdot)$ is a window function, $\rho \in (0, 1]$ is a scale parameter. This estimator takes into account the edge effect during estimation, assigning a lower weight to values closer to the boundaries and higher weights for observations closer to the middle. This estimator is positive-definite and asymptotically unbiased.

Value

A `CovEsts` S3 object (list) with the following values

`acf` A numeric vector containing the autocovariance/autocorrelation estimates.

`lags` A numeric vector containing the lag indices used to compute the estimates on.

`est_type` The type of estimate, namely 'autocorrelation' or 'autocovariance', this depends on the `type` parameter.

`est_used` The estimator function used, in this case, 'tapered_est'.

References

Dahlhaus R. & Künsch, H. (1987). Edge Effects and Efficient Parameter Estimation for Stationary Random Fields. Biometrika 74(4), 877-882. 10.1093/biomet/74.4.877

Examples

```
X <- c(1, 2, 3)
tapered_est(X, 0.5, "tukey", maxLag = 2)
```

---

to_pacf *Computes the Standard Estimator of the Autocovariance Function.*

Description

This function computes the partial autocorrelation function from an estimated autocovariance or autocorrelation function.

Usage

```
to_pacf(estCov)

## S3 method for class 'CovEsts'
to_pacf(estCov)

## Default S3 method:
to_pacf(estCov)
```

Arguments

`estCov`	A numeric vector representing an estimated autocovariance or autocorrelation function.

Details

This function is a translation of the 'uni_pacf' function in src/library/stats/src/pacf.c of the R source code which is an implementation of the Durbin-Levinson algorithm.

Value

A vector whose values are an estimate partial autocorrelation function or a `CovEsts` S3 object (list) with the following values

`acf` A numeric vector containing the partial autocorrelation estimates.

`lags` A numeric vector containing the lag indices used to compute the estimates on.

`est_type` The type of estimate, namely 'partial'.

`est_used` The estimator function used, in this case, 'to_pacf'.

If a numeric vector is given for the argument `estCov`, then a numeric vector output is given, and if a `CovEsts` S3 object is given, a `CovEsts` object is given as output.

Methods (by class)

- `to_pacf(CovEsts)`: Method for `CovEsts` objects.
- `to_pacf(default)`: Method for numeric vectors.

Examples

```
X <- c(1, 2, 3)
to_pacf(standard_est(X, pd = FALSE, maxLag = 2, meanX = mean(X)))
```

`to_vario` *Autocovariance to Semivariogram*

Description

This function computes an estimated semivariogram using an estimated autocovariance function.

Usage

```
to_vario(estCov)

## Default S3 method:
to_vario(estCov)

## S3 method for class 'CovEsts'
to_vario(estCov)
```

Arguments

`estCov`	A vector whose values are an estimate autocovariance function or a `CovEsts` S3 object.

Details

The semivariogram, $\gamma(h)$ and autocovariance function, $C(h)$, under the assumption of weak stationarity are related as follows:

$$\gamma(h) = C(0) - C(h).$$

When an empirical autocovariance function is considered instead, this relation does not necessarily hold, however, it can be used to obtain a function that is close to a semivariogram, see Bilchouris and Olenko (2025).

Value

A numeric vector whose values are an estimate of the semivariogram or an `VarioEsts` S3 object (list) with the following values

`vario` A numeric vector whose values are an estimate of the semivariogram.

`lags` A numeric vector containing the lag indices used to compute the estimates on, inherited from the argument `estCov`.

`est_used` The estimator function used, in this case, 'to_vario'.

If a numeric vector is given for the argument `estCov`, then a numeric vector output is given, and if a `CovEsts` S3 object is given, a `VarioEsts` object is given as output.

Methods (by class)

- `to_vario(default)`: Method for numeric vectors.
- `to_vario(CovEsts)`: Method for `CovEsts` objects.

References

Bilchouris, A. & Olenko, A (2025). On Nonparametric Estimation of Covariogram. Austrian Statistical Society 54(1), 112-137. 10.17713/ajs.v54i1.1975

Examples

```
X <- c(1, 2, 3)
estCov <- standard_est(X, meanX=mean(X), maxLag = 2, pd=FALSE)
to_vario(estCov)
```

`truncated_est` *Compute the Truncated Kernel Regression Estimator.*

Description

This function computes the truncated kernel regression estimator, based on the kernel regression estimator $\hat{\rho}(\cdot)$, see adjusted_est.

Usage

```
truncated_est(
  X,
  x,
  t,
  T1,
  T2,
  b,
  kernel_name = c("gaussian", "wave", "rational_quadratic", "bessel_j"),
  kernel_params = c(),
  pd = TRUE,
  type = c("autocovariance", "autocorrelation"),
  meanX = mean(X),
  custom_kernel = FALSE,
  parallel = FALSE,
  ncores = parallel::detectCores() - 1,
  cl_export = NULL,
  cl = NULL
)
```

Arguments

X	A vector representing observed values of the time series.
x	A vector of lags.
t	The arguments at which the autocovariance function is calculated at.
T1	The first truncation point, $T_1 > 0$.
T2	The second truncation point, $T_2 > T_1 > 0$.
b	Bandwidth parameter, greater than 0.
kernel_name	The name of the symmetric kernel (see kernel_symm_ec) function to be used. Possible values are: gaussian, wave, rational_quadratic, and bessel_j. Alternatively, a custom kernel function can be provided, see the examples.
kernel_params	A vector of parameters of the kernel function. See kernel_symm_ec for parameters.
pd	Whether a positive-definite estimate should be used. Defaults to TRUE.
type	Compute either the 'autocovariance' or 'autocorrelation'. Defaults to 'autocovariance'.
meanX	The average value of X. Defaults to mean(X).
custom_kernel	If a custom kernel is to be used or not. Defaults to FALSE.
parallel	Whether or not the computations should be done in parallel or not. Defaults to FALSE.
ncores	The number of cores to be used in the parallel computations. Defaults to the number cores - 1 (threads if hyperthreading is available), calculated from parallel::detectCores() - 1.

`cl_export` A vector of any additional functions or variables to export for parallel computations. This may be required if `estimator` is not within the package. Defaults to `NULL`.

`cl` An optional cluster object created by `parallel::makeCluster`. Defaults to `NULL`, which creates a temporary PSOCK cluster.

Details

This function computes the truncated kernel regression estimator,

$$\hat{\rho}_1(t) = \begin{cases} \hat{\rho}(t) & 0 \leq t \leq T_1 \\ \hat{\rho}(T_1)(T_2 - t)(T_2 - T_1)^{-1} & T_1 < t \leq T_2 \\ 0 & t > T_2 \end{cases}$$

where $\hat{\rho}(\cdot)$ is the kernel regression estimator, see adjusted_est.

Compared to adjusted_est, this function brings down the estimate to zero linearly between T_1 and T_2. In the case of short-range dependence, this may be beneficial as it can remove estimation artefacts at large lags.

To make this estimator positive-definite, the following procedure is used:

1. Take the discrete Fourier cosine transform $\widehat{\mathcal{F}}(\theta)$.
2. Find the smallest frequency where its associated value in the spectral domain is negative
$$\hat{\theta} = \inf\{\theta > 0 : \widehat{\mathcal{F}}(\theta)) < 0\}.$$
3. Set all values starting at the frequency to zero.
4. Perform the Fourier inversion.

If $\hat{\theta}$ is a small frequency, most of the spectrum equals zero, resulting in an inaccurate estimate of the autocovariance function, see Bilchouris and Olenko (2025).

Value

A `CovEsts` S3 object (list) with the following values

`acf` A numeric vector containing the autocovariance/autocorrelation estimates.

`lags` A numeric vector containing the lag indices used to compute the estimates on.

`est_type` The type of estimate, namely 'autocorrelation' or 'autocovariance', this depends on the `type` parameter.

`est_used` The estimator function used, in this case, 'truncated_est'.

References

Hall, P. & Patil, P. (1994). Properties of nonparametric estimators of autocovariance for stationary random fields. Probability Theory and Related Fields 99(3), 399-424. https://doi.org/10.1007/bf01199899

Hall, P., Fisher, N. I., & Hoffmann, B. (1994). On the nonparametric estimation of covariance functions. The Annals of Statistics 22(4), 2115-2134. https://doi.org/10.1214/aos/1176325774

Bilchouris, A. & Olenko, A (2025). On Nonparametric Estimation of Covariogram. Austrian Statistical Society 54(1), 112-137. https://doi.org/10.17713/ajs.v54i1.1975

Examples

```
X <- c(1, 2, 3, 4)
truncated_est(X, 1:4, 1:3, 1, 2, 0.1,
                  "gaussian")

my_kernel <- function(x, params) {
  stopifnot(params[1] > 0, length(x) >= 1)
 return(exp(-((abs(x) / params[1])^params[2])) * (2 * params[1] * gamma(1 + 1/params[2])))
}

truncated_est(X, 1:4, 1:3, 1, 2, 0.1, my_kernel, c(0.25), custom_kernel = TRUE)
## Not run:
library(parallel)
X <- c(1, 2, 3, 4)
my_cl <- makePSOCKcluster(2)
truncated_est(X, 1:4, 1:3, 1, 2, 0.1, "gaussian", parallel = TRUE, cl = my_cl)
stopCluster(my_cl)

## End(Not run)
```

`window_ec` *1D Window Functions.*

Description

A window function in this context is a continuous nondecreasing function such that at 0 it is 0, and at 1, it is 1. Note that `window()` is **deprecated**, please use `window_ec()` instead. This computes one of the window functions listed below.

Usage

```
window_ec(
  x,
 name = c("tukey", "triangular", "sine", "power_sine", "blackman", "hann_poisson",
    "welch"),
  params = c(1)
)
```

Arguments

`x`	A vector or matrix of arguments of at least length 1. Each value must be between 0 and 1, inclusive.
`name`	The name of the window. Options are: tukey, triangular, sine, power_sine, blackman, hann_poisson, welch.
`params`	A vector of parameters for the windows. See the documentation below for the position of the parameters.

Details

Tukey Window. The Tukey window is defined as

$$w(x) = \frac{1}{2} - \frac{1}{2}\cos(\pi x), x \in [0, 1].$$

The `params` argument is empty.

Triangular Window. The triangular window is given by

$$w(x) = x, x \in [0, 1].$$

The `params` argument is empty.

Sine Window. The sine window is given by

$$w(x) = \sin(\pi x/2), x \in [0, 1].$$

The `params` argument is empty.

Power Sine Window. The power sine window is given by

$$w(x; a) = \sin^a(\pi x/2), x \in [0, 1], a > 0.$$

The `params` argument is of the form `c(`a`)`.

Blackman Window. The Blackman window is defined as

$$w(x; a) = ((1 - a)/2) - \frac{1}{2}\cos(\pi x) + \frac{a}{2}\cos(2\pi x), x \in [0, 1], a \in [-0.25, 0.25].$$

The `params` argument is of the form `c(`a`)`. It is recommended that $a \in [-0.25, 0.25]$ to ensure that the window is nondecreasi#'ng on $[0, 1]$.

Hann-Poisson Window. The Hann-Poisson window is defined as

$$w(x; a) = \frac{1}{2}(1 - \cos(\pi x))\exp(-(a\,|1 - x|)), x \in [0, 1], a > 0.$$

The `params` argument is of the form `c(`a`)`.

Welch Window. The Welch window is given by

$$w(x) = 1 - (x - 1)^2, x \in [0, 1].$$

The `params` argument is empty.

See the function call examples below.

Value

A vector or matrix of values.

Examples

```
x <- c(0.2, 0.4, 0.6)
window_ec(x, "tukey")
window_ec(x, "triangular")
window_ec(x, "sine")
window_ec(x, "power_sine", c(0.7))
window_ec(x, "blackman", c(0.16))
window_ec(x, "hann_poisson", c(0.7))
window_ec(x, "welch")
curve(window_ec(x, "tukey"), from = 0, to = 1)
curve(window_ec(x, "triangular"), from = 0, to = 1)
curve(window_ec(x, "sine"), from = 0, to = 1)
curve(window_ec(x, "power_sine", c(0.7)), from = 0, to = 1)
curve(window_ec(x, "blackman", c(0.16)), from = 0, to = 1)
curve(window_ec(x, "hann_poisson", c(0.7)), from = 0, to = 1)
curve(window_ec(x, "welch"), from = 0, to = 1)
```

---

window_symm_ec *1D Symmetric Window Functions.*

Description

A symmetric window function in this context are traditional window functions, unlike the window functions. Note that `window_symm()` is **deprecated**, please use `window_symm_ec()` instead. This computes one of the symmetric window functions listed below, all of which are defined for $x \in [-1, 1]$, and are 0 otherwise.

Usage

```
window_symm_ec(
  x,
 name = c("tukey", "triangular", "sine", "power_sine", "blackman", "hann_poisson",
    "welch"),
  params = c(1)
)
```

Arguments

`x`	A vector or matrix of arguments of at least length 1. Each value must be between 0 and 1, inclusive.
`name`	The name of the window. Options are: tukey, triangular, sine, power_sine, blackman, hann_poisson, welch.
`params`	A vector of parameters for the windows. See the documentation below for the position of the parameters.

Details

Tukey Window. The Tukey window is defined as

$$w(x) = \frac{1}{2} + \frac{1}{2}\cos(\pi|x|), x \in [-1, 1].$$

The `params` argument is empty, see the example.

Triangular Window. The triangular window is given by

$$w(x) = 1 - |x|, x \in [-1, 1].$$

The `params` argument is empty, see the example.

Sine Window. The sine window is given by

$$w(x) = 1 - \sin\left(\pi|x|/2\right), x \in [-1, 1].$$

The `params` argument is empty, see the example.

Power Sine Window. The power sine window is given by

$$w(x; a) = 1 - \sin^a(\pi|x|/2), x \in [-1, 1], a > 0.$$

The `params` argument is of the form `c(`a`)`

Blackman Window. The Blackman window is defined as

$$w(x; a) = 1 + ((a-1)/2) + \frac{1}{2}\cos(\pi|x|) - \frac{a}{2}\cos(2\pi|x|), x \in [-1, 1], a \in [-0.25, 0.25].$$

The `params` argument is of the form `c(`a`)`. The standard value of a for this window is 0.16.

Hann-Poisson Window. The Hann-Poisson window is defined as

$$w(x; a) = 1 - \frac{1}{2}(1 - \cos(\pi|x|))\exp(-(a\,|1 - |x||)), x \in [-1, 1], a > 0.$$

The `params` argument is of the form `c(`a`)`

Welch Window. The Welch window is given by

$$w(x) = (|x| - 1)^2, x \in [-1, 1].$$

The `params` argument is empty, see the example.

Value

A vector or matrix of values.

Examples

```
x <- c(-0.6, -0.4, -0.2, 0, 0.2, 0.4, 0.6)
window_symm_ec(x, "tukey")
window_symm_ec(x, "triangular")
window_symm_ec(x, "sine")
window_symm_ec(x, "power_sine", c(0.7))
window_symm_ec(x, "blackman", c(0.16))
```

```
window_symm_ec(x, "hann_poisson", c(0.7))
window_symm_ec(x, "welch")
curve(window_symm_ec(x, "tukey"), from = -1, to = 1)
curve(window_symm_ec(x, "triangular"), from = -1, to = 1)
curve(window_symm_ec(x, "sine"), from = -1, to = 1)
curve(window_symm_ec(x, "power_sine", c(0.7)), from = -1, to = 1)
curve(window_symm_ec(x, "blackman", c(0.16)), from = -1, to = 1)
curve(window_symm_ec(x, "hann_poisson", c(0.7)), from = -1, to = 1)
curve(window_symm_ec(x, "welch"), from = -1, to = 1)
```

`Xij_mat` *Compute X_ij Matrix*

Description

This helper function computes the matrix of pairwise values, X_{ij}, for the kernel regression estimator,

$$X_{ij} = (X_i - \bar{X})(X_j - \bar{X}).$$

Usage

```
Xij_mat(X, meanX = mean(X))
```

Arguments

`X`	A vector of values.
`meanX`	The average value of `X`. Defaults to `mean(X)`.

Value

A matrix of size $N \times N$, where N is the length of the vector X.

References

Hall, P. & Patil, P. (1994). Properties of nonparametric estimators of autocovariance for stationary random fields. Probability Theory and Related Fields 99(3), 399-424. https://doi.org/10.1007/bf01199899

Hall, P., Fisher, N. I., & Hoffmann, B. (1994). On the nonparametric estimation of covariance functions. The Annals of Statistics 22(4), 2115-2134. https://doi.org/10.1214/aos/1176325774

Examples

```
## Not run:
X <- c(1, 2, 3, 4)
Xij_mat(X, mean(X))

## End(Not run)
```

Index