Package ‘SuperGauss’

February 24, 2022

Type Package

Title Superfast Likelihood Inference for Stationary Gaussian Time Series

Version 2.0.3

Date 2022-02-24

Description Likelihood evaluations for stationary Gaussian time series are typically obtained via the Durbin-Levinson algorithm, which scales as $O(n^2)$ in the number of time series observations. This package provides a "superfast" $O(n \log^2 n)$ algorithm written in C++, crossing over with Durbin-Levinson around $n = 300$. Efficient implementations of the score and Hessian functions are also provided, leading to superfast versions of inference algorithms such as Newton-Raphson and Hamiltonian Monte Carlo. The C++ code provides a Toeplitz matrix class packaged as a header-only library, to simplify low-level usage in other packages and outside of R.

License GPL-3

Depends R (>= 3.0.0)

Imports stats, methods, R6, Rcpp (>= 0.12.7), fftw

LinkingTo Rcpp, RcppEigen

Suggests knitr, rmarkdown, testthat, mvtnorm, numDeriv

VignetteBuilder knitr

RoxygenNote 7.1.2

Encoding UTF-8

SystemRequirements fftw3 (>= 3.1.2)

NeedsCompilation yes

Author Yun Ling [aut],
Martin Lysy [aut, cre]

Maintainer Martin Lysy <mlysy@uwaterloo.ca>

Repository CRAN

Date/Publication 2022-02-24 20:40:02 UTC
SuperGauss-package

R topics documented:

SuperGauss-package .................................................. 2
acf2incr ................................................................. 3
acf2msd ................................................................. 4
Cholesky ................................................................. 4
Circulant ................................................................. 5
dnormtz ................................................................. 7
fbm_msd ................................................................. 8
matern_acf .............................................................. 9
msd2acf ................................................................. 9
NormalCirculant ....................................................... 10
NormalToeplitz ....................................................... 12
pex_acf ................................................................. 14
rmormtz ............................................................... 15
SuperGauss-defunct .................................................. 16
toepl.mulf ............................................................ 16
Toeplitz ............................................................... 17

Index 21

SuperGauss-package  Superfast inference for stationary Gaussian time series.

Description

Likelihood evaluations for stationary Gaussian time series are typically obtained via the Durbin-Levinson algorithm, which scales as $O(n^2)$ in the number of time series observations. This package provides a "superfast" $O(n \log^2 n)$ algorithm written in C++, crossing over with Durbin-Levinson around $n = 300$. Efficient implementations of the score and Hessian functions are also provided, leading to superfast versions of inference algorithms such as Newton-Raphson and Hamiltonian Monte Carlo. The C++ code provides a Toeplitz matrix class packaged as a header-only library, to simplify low-level usage in other packages and outside of R.

Details

While likelihood calculations with stationary Gaussian time series generally scale as $O(N^2)$ in the number of observations, this package implements an algorithm which scales as $O(N \log^2 N)$. "Superfast" algorithms for loglikelihood gradients and Hessians are also provided. The underlying C++ code is distributed through a header-only library found in the installed package’s include directory.

Author(s)

Maintainer: Martin Lysy <mlysy@uwaterloo.ca>

Authors:
  • Yun Ling
acf2incr

Examples

# Superfast inference for the timescale parameter
# of the exponential autocorrelation function
exp_acf <- function(lambda) exp(-(1:N-1)/lambda)

# simulate data
lambda0 <- 1
N <- 1000
X <- rnormtz(n = 1, acf = exp_acf(lambda0))

# loglikelihood function
# allocate memory for a NormalToeplitz distribution object
NTz <- NormalToeplitz$new(N)
loglik <- function(lambda) {
  NTz$logdens(z = X, acf = exp_acf(lambda))
  ## dSnorm(X = X, acf = Toep, log = TRUE)
}

# maximum likelihood estimation
optimize(f = loglik, interval = c(.2, 5), maximum = TRUE)

---

acf2incr

Convert position autocorrelations to increment autocorrelations.

Description

Convert the autocorrelation of a stationary sequence \( x = (x_1, \ldots, x_N) \) to that of its increments, \( dx = (x_2 - x_1, \ldots, x_N - x_{N-1}) \).

Usage

acf2incr(acf)

Arguments

acf

Length-\( N \) vector of position autocorrelations.

Value

Length \( N-1 \) vector of increment autocorrelations.

Examples

acf2incr(acf = exp(-0:10)))
acf2msd

Convert autocorrelation of stationary increments to mean squared displacement of positions.

Description

Converts the autocorrelation of a stationary increment sequence \( dx = (x_1 - x_0, ..., x_N - x_{(N-1)}) \) to the mean squared displacement (MSD) of the corresponding positions, i.e., \( MSD_i = E[(x_i - x_0)^2] \).

Usage

acf2msd(acf)

Arguments

acf

Length-N autocorrelation vector of a stationary increment sequence.

Value

Length-N MSD vector of the corresponding positions.

Examples

acf2msd(acf = exp(-(0:10)))

Cholesky

Cholesky multiplication with Toeplitz variance matrices.

Description

Multiplies the Cholesky decomposition of the Toeplitz matrix with another matrix, or solves a system of equations with the Cholesky factor.

Usage

cholZX(Z, acf)

cholXZ(X, acf)

Arguments

Z

Length-N or N x p matrix of residuals.

acf

Length-N autocorrelation vector of the Toeplitz variance matrix.

X

Length-N or N x p matrix of observations.
Details

If \( C = \text{t(chol(toeplitz(acf)))} \), then \( \text{cholZX()} \) computes \( C \%\% Z \) and \( \text{cholZX()} \) computes \( \text{solve}(C, X) \). Both functions use the Durbin-Levinson algorithm.

Value

Size \( N \times p \) residual or observation matrix.

Examples

\[
N <- 10 \\
p <- 2 \\
acf <- \exp(-(1:N - 1)) \\
Z <- \text{matrix(rnorm}(N \times p), N, p) \\
\text{cholZX}(Z = Z, acf = acf) - (\text{t(chol(toeplitz(acf)))} \%\% Z) \\
X <- \text{matrix(rnorm}(N \times p), N, p) \\
\text{cholXZ}(X = X, acf = acf) - \text{solve}(\text{t(chol(toeplitz(acf)))}, X)
\]

Description

Constructor and methods for Circulant matrix objects.

Methods

Public methods:

- \text{Circulant}$\text{new()}$
- \text{Circulant}$\text{size()}$
- \text{Circulant}$\text{set_acf()}$
- \text{Circulant}$\text{get_acf()}$
- \text{Circulant}$\text{set_psd()}$
- \text{Circulant}$\text{get_psd()}$
- \text{Circulant}$\text{has_acf()}$
- \text{Circulant}$\text{prod()}$
- \text{Circulant}$\text{solve()}$
- \text{Circulant}$\text{log_det()}$
- \text{Circulant}$\text{clone()}$

Method \text{new()}: \text{Class constructor.}

Usage:

\text{Circulant}$\text{new}(N, uacf, upsd)$
Arguments:
N  Size of Circulant matrix.
uacf Optional vector of \( \text{Nu} = \text{floor}(N/2)+1 \) unique elements of the autocorrelation.
upsd Optional vector of \( \text{Nu} = \text{floor}(N/2)+1 \) unique elements of the PSD.

Returns: A Circulant object.

Method size(): Get the size of the Circulant matrix.
Usage:
Circulant$size()
Returns: Size of the Circulant matrix.

Method set_acf(): Set the autocorrelation of the Circulant matrix.
Usage:
Circulant$set_acf(uacf)
Arguments:
\( uacf \)  Vector of \( \text{Nu} = \text{floor}(N/2)+1 \) unique elements of the autocorrelation.

Method get_acf(): Get the autocorrelation of the Circulant matrix.
Usage:
Circulant$get_acf()
Returns: The complete autocorrelation vector of length \( N \).

Method set_psd(): Set the PSD of the Circulant matrix.
The power spectral density (PSD) of a Circulant matrix \( C_t = \text{Circulant}(\text{acf}) \) is defined as \( \text{psd} = \text{iFFT}(\text{acf}) \).
Usage:
Circulant$set_psd(upsd)
Arguments:
\( upsd \)  Vector of \( \text{Nu} = \text{floor}(N/2)+1 \) unique elements of the psd.

Method get_psd(): Get the PSD of the Circulant matrix.
Usage:
Circulant$get_psd()
Returns: The complete PSD vector of length \( N \).

Method has_acf(): Check whether the autocorrelation of the Circulant matrix has been set.
Usage:
Circulant$has_acf()
Returns: Logical; TRUE if Circulant$set_acf() has been called.

Method prod(): Circulant matrix-matrix product.
Usage:
Circulant$prod(x)
Arguments:
x Vector or matrix with N rows.
Returns: The matrix product Ct %*% x.

Method solve(): Solve a Circulant system of equations.
Usage:
Circulant$solve(x)
Arguments:
x Optional vector or matrix with N rows.
Returns: The solution in z to the system of equations Ct %*% z = x. If x is missing, returns the inverse of Ct.

Method log_det(): Calculate the log-determinant of the Circulant matrix.
Usage:
Circulant$log_det()
Returns: The log-determinant log(det(Ct)).

Method clone(): The objects of this class are cloneable with this method.
Usage:
Circulant$clone(deep = FALSE)
Arguments:
deep Whether to make a deep clone.

Description
Density of a multivariate normal with Toeplitz variance matrix.

Usage
dnormtz(X, mu, acf, log = FALSE, method = c("gschur", "ltz"))

Arguments

- X Vector of length N or N x n matrix, of which each column is a multivariate observation.
- mu Vector or matrix of mean values of compatible dimensions with X. Defaults to all zeros.
- acf Vector of length N containing the first column of the Toeplitz variance matrix.
- log Logical; whether to return the multivariate normal density on the log scale.
- method Which calculation method to use. Choices are: gschur for a modified version of the Generalized Schur algorithm of Ammar & Gragg (1988), or ltz for the Levinson-Trench-Zohar method. The former scales as O(N log^2 N) whereas the latter scales as O(N^2) and should only be used for N < 300.
Value

Vector of n (log-)densities, one for each column of X.

Examples

# simulate data
N <- 10  # length of each time series
n <- 3  # number of time series
theta <- 0.1
lambda <- 2
mu <- theta^2 * rep(1, N)
acf <- exp(-lambda * (1:N - 1))
X <- rnorm(tseq, acf = acf) + mu

# evaluate log-density
dnorm(tseq, mu, acf, log = TRUE)

fbm_msd

Mean square displacement of fractional Brownian motion.

Description

Mean square displacement of fractional Brownian motion.

Usage

fbm_msd(tseq, H)

Arguments

tseq  Length-N vector of timepoints.
H      Hurst parameter (between 0 and 1).

Details

The mean squared displacement (MSD) of a stochastic process $X_t$ is defined as

$MSD(t) = E[(X_t - X_0)^2]$.

Fractional Brownian motion (fBM) is a continuous Gaussian process with stationary increments, such that its covariance function is entirely defined the MSD, which in this case is $MSD(t) = t^{2H}$.

Value

Length-N vector of mean square displacements.

Examples

fbm_msd(tseq = 1:10, H = 0.4)
**matern_acf**

*Matern autocorrelation function.*

**Description**

Matern autocorrelation function.

**Usage**

```r
matern_acf(tseq, lambda, nu)
```

**Arguments**

- `tseq`: Vector of N time points at which the autocorrelation is to be calculated.
- `lambda`: Timescale parameter.
- `nu`: Smoothness parameter.

**Details**

The Matern autocorrelation is given by

\[
ACF(t) = \frac{1}{\Gamma(\nu)} \left( \frac{\sqrt{2\nu}t}{\lambda} \right)^\nu K_\nu \left( \frac{\sqrt{2\nu}t}{\lambda} \right),
\]

where \( K_\nu(x) \) is the modified Bessel function of second kind.

**Value**

An autocorrelation vector of length N.

**Examples**

```r
matern_acf(tseq = 1:10, lambda = 1, nu = 3/2)
```

---

**msd2acf**

*Convert mean square displacement of positions to autocorrelation of increments.*

**Description**

Converts the mean squared displacement (MSD) of a stationary increments sequence \( x = (x_0, x_1, ..., x_N) \) positions to the autocorrelation of the corresponding increments \( dx = (x_1 - x_0, ..., x_N - x_{(N-1)}) \).

**Usage**

```r
msd2acf(msd)
```
Arguments

msd Length-N MSD vector, i.e., excluding x_0 which is assumed to be zero.

Value

Length-N autocorrelation vector.

Examples

# autocorrelation of fBM increments
msd2acf(msd = fbm_msd(tseq = 0:10, H = .3))

Description

Provides methods for the Normal-Circulant (NCt) distribution, which for a random vector z of length N is defined as

\[ z \sim \text{NCt}(\text{uacf}) \iff z \sim \text{Normal}(0, \text{toeplitz}(\text{acf})), \]

where uacf are the \( \text{Nu} = \text{floor}(N/2)+1 \) unique elements of the autocorrelation vector acf, i.e.,

\[ \text{acf} = \left( \text{uacf}, \text{rev(uacf}[2:(\text{Nu}-1)]), \right), \quad \text{N even}, \]
\[ = \left( \text{uacf}, \text{rev(uacf}[2:\text{Nu}]]), \quad \text{N odd}. \]

Methods

Public methods:

- \text{NormalCirculant}$new()  
- \text{NormalCirculant}$size()  
- \text{NormalCirculant}$logdens()  
- \text{NormalCirculant}$grad_full()  
- \text{NormalCirculant}$clone()  

Method \text{new()}: Class constructor.

Usage:

\text{NormalCirculant}$new(N)

Arguments:

N Size of the NCt random vector.

Returns: A NormalCirculant object.

Method \text{size()}: Get the size of the NCt random vector.

Usage:
NormalCirculant

NormalCirculant$size()

*Returns:* Size of the NCt random vector.

**Method** `logdens()`: Log-density function.

*Usage:*
`NormalCirculant$logdens(z, uacf)`

*Arguments:*
- `z`: Density argument. A vector of length N or an N x n_obs matrix where each column is an N-dimensional observation.
- `uacf`: A vector of length Nu = floor(N/2) containing the first half of the autocorrelation (i.e., first row/column) of the Circulant variance matrix.

*Returns:* A scalar or vector of length n_obs containing the log-density of the NCt evaluated at its arguments.

**Method** `grad_full()`: Full gradient of log-density function.

*Usage:*
`NormalCirculant$grad_full(z, uacf, calc_dldz = TRUE, calc_dldu = TRUE)`

*Arguments:*
- `z`: Density argument. A vector of length N.
- `uacf`: A vector of length Nu = floor(N/2) containing the first half of the autocorrelation (i.e., first row/column) of the Circulant variance matrix.
- `calc_dldz`: Whether or not to calculate the gradient with respect to `z`.
- `calc_dldu`: Whether or not to calculate the gradient with respect to `uacf`.

*Returns:* A list with elements:
- `ldens`: The log-density evaluated at `z` and `uacf`.
- `dldz`: The length-N gradient vector with respect to `z`, if `calc_dldz` = TRUE.
- `dldu`: The length-Nu = floor(N/2)+1 gradient vector with respect to `uacf`, if `calc_dldu` = TRUE.

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*
`NormalCirculant$clone(deep = FALSE)`

*Arguments:*
- `deep`: Whether to make a deep clone.
NormalToeplitz  Multivariate normal with Toeplitz variance matrix.

Description

Provides methods for the Normal-Toeplitz (NTz) distribution defined as

\[ z \sim NTz(\text{acf}) \iff z \sim \text{Normal}(0, \text{toeplitz}(\text{acf})), \]

i.e., for a multivariate normal with mean zero and variance \( Tz = \text{toeplitz}(\text{acf}) \).

Methods

Public methods:

- NormalToeplitz$new()
- NormalToeplitz$size()
- NormalToeplitz$logdens()
- NormalToeplitz$grad()
- NormalToeplitz$hess()
- NormalToeplitz$grad_full()
- NormalToeplitz$clone()

Method new(): Class constructor.

Usage:

NormalToeplitz$new(N)

Arguments:

N  Size of the NTz random vector.

Returns: A NormalToeplitz object.

Method size(): Get the size of the NTz random vector.

Usage:

NormalToeplitz$size()

Returns: Size of the NTz random vector.

Method logdens(): Log-density function.

Usage:

NormalToeplitz$logdens(z, acf)

Arguments:

z  Density argument. A vector of length N or an N x n_obs matrix where each column is an N-dimensional observation.

acf  A vector of length N containing the autocorrelation (i.e., first row/column) of the Toeplitz variance matrix.
NormalToeplitz

**Returns:** A scalar or vector of length n_obs containing the log-density of the NTz evaluated at its arguments.

**Method grad():** Gradient of the log-density with respect to parameters.

**Usage:**
```
NormalToeplitz$grad(z, dz, acf, dacf, full_out = FALSE)
```

**Arguments:**
- `z` Density argument. A vector of length N.
- `dz` An N x n_theta matrix containing the gradient dz/dtheta.
- `acf` A vector of length N containing the autocorrelation of the Toeplitz variance matrix.
- `dacf` An N x n_theta matrix containing the gradient dacf/dtheta.
- `full_out` If TRUE, returns the log-density as well (see 'Value').

**Returns:** A vector of length n_theta containing the gradient of the NTz log-density with respect to theta, or a list with elements ldens and grad consisting of the log-density and the gradient vector.

**Method hess():** Hessian of log-density with respect to parameters.

**Usage:**
```
NormalToeplitz$hess(z, dz, d2z, acf, dacf, d2acf, full_out = FALSE)
```

**Arguments:**
- `z` Density argument. A vector of length N.
- `dz` An N x n_theta matrix containing the gradient dz/dtheta.
- `d2z` An N x n_theta x n_theta array containing the Hessian d^2z/dtheta^2.
- `acf` A vector of length N containing the autocorrelation of the Toeplitz variance matrix.
- `dacf` An N x n_theta matrix containing the gradient dacf/dtheta.
- `d2acf` An N x n_theta x n_theta array containing the Hessian dacf^2/dtheta^2.
- `full_out` If TRUE, returns the log-density and its gradient as well (see 'Value').

**Returns:** An n_theta x n_theta matrix containing the Hessian of the NTz log-density with respect to theta, or a list with elements ldens, grad, and hess consisting of the log-density, its gradient (a vector of size n_theta), and the Hessian matrix, respectively.

**Method grad_full():** Full gradient of log-density function.

**Usage:**
```
NormalToeplitz$grad_full(z, acf, calc_dldz = TRUE, calc_dlda = TRUE)
```

**Arguments:**
- `z` Density argument. A vector of length N.
- `acf` A vector of length N containing the autocorrelation of the Toeplitz variance matrix.
- `calc_dldz` Whether or not to calculate the gradient with respect to z.
- `calc_dlda` Whether or not to calculate the gradient with respect to acf.

**Returns:** A list with elements:
- `ldens` The log-density evaluated at z and acf.
- `dldz` The length-N gradient vector with respect to z, if calc_dldz = TRUE.
The length-$N$ gradient vector with respect to \( \text{acf} \), if \( \text{calc}_{\text{dlda}} = \text{TRUE} \).

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```r
NormalToeplitz$\text{clone}(\text{deep} = \text{FALSE})
```

*Arguments:*

- `deep` Whether to make a deep clone.

---

**pex_acf**  

*Power-exponential autocorrelation function.*

**Description**

Power-exponential autocorrelation function.

**Usage**

```r
\text{pex_acf}(tseq, \lambda, \rho)
```

**Arguments**

- `tseq`  
  Vector of $N$ time points at which the autocorrelation is to be calculated.
- `lambda`  
  Timescale parameter.
- `rho`  
  Power parameter.

**Details**

The power-exponential autocorrelation function is given by:

\[
\text{ACF}(t) = \exp \left\{ \frac{-t}{\lambda \rho} \right\}.
\]

**Value**

An autocorrelation vector of length $N$.

**Examples**

```r
\text{pex_acf}(tseq = 1:10, \lambda = 1, \rho = 2)
```
rnormtz

Simulate a stationary Gaussian time series.

Description

Simulate a stationary Gaussian time series.

Usage

rnormtz(n = 1, acf, Z, fft = TRUE, nkeep, tol = 1e-06)

Arguments

- **n**: Number of time series to generate.
- **acf**: Length-N vector giving the autocorrelation of the series.
- **Z**: Optional size (2N-2) x n or N x n matrix of iid standard normals, to use in the FFT and Durbin-Levinson methods, respectively.
- **fft**: Logical; whether or not to use the O(N log N) FFT-based algorithm of Wood and Chan (1994) or the more stable O(N^2) Durbin-Levinson algorithm. See Details.
- **nkeep**: Length of time series. Defaults to \( N = \text{length}(\text{acf}) \). See Details.
- **tol**: Relative tolerance on negative eigenvalues. See Details.

Details

The FFT method fails when the embedding circulant matrix is not positive definite. This is typically due to one of two things:

1. Roundoff error can make tiny eigenvalues appear negative. For this purpose, argument **tol** can be used to replace all negative eigenvalues by \( \text{tol} \times \text{ev}\_\text{max} \), where \( \text{ev}\_\text{max} \) is the largest eigenvalue.

2. The autocorrelation is decaying too slowly on the given timescale. To mitigate this, argument **nkeep** can be used to supply a longer **acf** than is required, and keep only the first **nkeep** time series observations. For consistency, **nkeep** also applies to Durbin-Levinson method.

Value

Length-\( \text{nkeep} \) vector or size \( \text{nkeep} \times N \) matrix with time series as columns.

Examples

```
N <- 10
acf <- exp(-(1:N - 1)/N)
rnormtz(n = 3, acf = acf)
```
Defunct functions in SuperGauss.

The following functions have been removed from the SuperGauss package:

- `rSnorm()` Please use `rnormtz()` instead.
- `dSnorm()` Please use `dnormtz()` instead.
- `Snorm.grad()` Please use the `grad()` method in the `NormalToeplitz` class.
- `Snorm.hess()` Please use the `hess()` method in the `NormalToeplitz` class.

Toep.mult

Toeplitz matrix multiplication.

Description

Efficient matrix multiplication with Toeplitz matrix and arbitrary matrix or vector.

Usage

toop.mult(acf, X)

Arguments

- `acf` Length-N vector giving the first column (or row) of the Toeplitz matrix.
- `X` Vector or matrix of compatible dimensions with `acf`.

Value

An N-row matrix corresponding to `toeplitz(acf) %*% X`.

Examples

```r
N <- 20
d <- 3
acf <- exp(-(1:N))
X <- matrix(rnorm(N*d), N, d)
toop.mult(acf, X)
```
Description

The Toeplitz class contains efficient methods for linear algebra with symmetric positive definite (i.e., variance) Toeplitz matrices.

Usage

```r
isToeplitz(x)

asToeplitz(x)
```

### S3 method for class 'Toeplitz'

```r
dim(x)
```

Arguments

- `x` An R object.

Details

An N x N Toeplitz matrix $T_z$ is defined by its length-N "autocorrelation" vector $\text{acf}$, i.e., first row/column $T_z$. Thus, for the function `stats::toeplitz()`, we have $T_z = \text{toeplitz}(\text{acf})$.

It is assumed that $\text{acf}$ defines a valid (i.e., positive definite) variance matrix. The matrix multiplication methods still work when this is not the case but the other methods do not (return values typically contain NaNs).

`asToeplitz(x)` attempts to convert its argument to a Toeplitz object by calling `Toeplitz$new(acf = x)`.

`isToeplitz(x)` checks whether its argument is a Toeplitz object.

Methods

**Public methods:**

- `Toeplitz$new()`
- `Toeplitz$print()`
- `Toeplitz$size()`
- `Toeplitz$set_acf()`
- `Toeplitz$get_acf()`
- `Toeplitz$has_acf()`
- `Toeplitz$prod()`
- `Toeplitz$solve()`
- `Toeplitz$log_det()`
- `Toeplitz$trace_grad()`
- `Toeplitz$trace_hess()`
• Toeplitz$clone()

**Method new():** Class constructor.

*Usage:*

Toeplitz$new(N, acf)

*Arguments:*

N  Size of Toeplitz matrix.
acf  Autocorrelation vector of length N.

*Returns: A Toeplitz object.*

**Method print():** Print method.

*Usage:*

Toeplitz$print()

**Method size():** Get the size of the Toeplitz matrix.

*Usage:*

Toeplitz<size>()

*Returns: Size of the Toeplitz matrix. ncol(), nrow(), and dim() methods for Toeplitz objects also work as expected.*

**Method set_acf():** Set the autocorrelation of the Toeplitz matrix.

*Usage:*

Toeplitz$set_acf(acf)

*Arguments:*

acf  Autocorrelation vector of length N.

**Method get_acf():** Get the autocorrelation of the Toeplitz matrix.

*Usage:*

Toeplitz$get_acf()

*Returns: The autocorrelation vector of length N.*

**Method has_acf():** Check whether the autocorrelation of the Toeplitz matrix has been set.

*Usage:*

Toeplitz$has_acf()

*Returns: Logical; TRUE if Toeplitz$set_acf() has been called.*

**Method prod():** Toeplitz matrix-matrix product.

*Usage:*

Toeplitz$prod(x)

*Arguments:*

x  Vector or matrix with N rows.

*Returns: The matrix product Tz %*% x. Tz %*% x and x %*% Tz also work as expected.*
Method solve(): Solve a Toeplitz system of equations.

Usage:
Toeplitz$solve(x, method = c("gschur", "pcg"), tol = 1e-10)

Arguments:
x  Optional vector or matrix with \( N \) rows.
tol  Tolerance level for the pcg method.

Returns: The solution in \( z \) to the system of equations \( Tz \%*% z = x \). If \( x \) is missing, returns the inverse of \( Tz \). solve(Tz,x) and solve(Tz,x,method,tol) also work as expected.

Method log_det(): Calculate the log-determinant of the Toeplitz matrix.

Usage:
Toeplitz$log_det()

Returns: The log-determinant \( \log(\det(Tz)) \). determinant(Tz) also works as expected.

Method trace_grad(): Computes the trace-gradient with respect to Toeplitz matrices.

Usage:
Toeplitz$trace_grad(acf2)

Arguments:
acf2  Length-\( N \) autocorrelation vector of the second Toeplitz matrix. This matrix must be symmetric but not necessarily positive definite.

Returns: Computes the trace of 
\( \text{solve}(Tz, \text{toeplitz}(acf2)) \).
This is used in the computation of the gradient of \( \log(\det(Tz(\theta))) \) with respect to \( \theta \).

Method trace_hess(): Computes the trace-Hessian with respect to Toeplitz matrices.

Usage:
Toeplitz$trace_hess(acf2, acf3)

Arguments:
acf2  Length-\( N \) autocorrelation vector of the second Toeplitz matrix. This matrix must be symmetric but not necessarily positive definite.
acf3  Length-\( N \) autocorrelation vector of the third Toeplitz matrix. This matrix must be symmetric but not necessarily positive definite.

Returns: Computes the trace of 
\( \text{solve}(Tz, \text{toeplitz}(acf2)) \%*% \text{solve}(Tz, \text{toeplitz}(acf3)) \).
This is used in the computation of the Hessian of \( \log(\det(Tz(\theta))) \) with respect to \( \theta \).

Method clone(): The objects of this class are cloneable with this method.

Usage:
Toeplitz$clone(deep = FALSE)

Arguments:
deep  Whether to make a deep clone.
Examples

# construct a Toeplitz matrix
acf <- exp(-(1:5))
Tz <- Toeplitz$new(acf = acf)
# alternatively, can allocate space first
Tz <- Toeplitz$new(N = length(acf))
Tz$set_acf(acf = acf)

# basic methods
Tz$get_acf() # extract the acf
dim(Tz) # == c(nrow(Tz), ncol(Tz))
Tz # print method

# linear algebra methods
X <- matrix(rnorm(10), 5, 2)
Tz %*% X
t(X) %*% Tz
solve(Tz, X)
determinant(Tz) # log-determinant
Index

%*% (Toeplitz), 17

acf2incr, 3
acf2msd, 4
as.Toeplitz (Toeplitz), 17

Cholesky, 4
cholXZ (Cholesky), 4
cholZX (Cholesky), 4
Circulant, 5

determinant (Toeplitz), 17
determinant, Toeplitz-method (Toeplitz), 17

dim(), 18
dim.Toeplitz (Toeplitz), 17
dnormtz, 7
dnormtz(), 16
dSnorm (SuperGauss-defunct), 16

fbm_msd, 8

is.Toeplitz (Toeplitz), 17

matern_acf, 9
msd2acf, 9

ncol(), 18
ncol, Toeplitz-method (Toeplitz), 17
NormalCirculant, 10
NormalToeplitz, 12, 16
nrow(), 18
nrow, Toeplitz-method (Toeplitz), 17

pex_acf, 14

rnormtz, 15
rnormtz(), 16
rSnorm (SuperGauss-defunct), 16

Snorm.grad (SuperGauss-defunct), 16

solve (Toeplitz), 17
solve, Toeplitz, ANY-method (Toeplitz), 17
solve, Toeplitz-method (Toeplitz), 17
stats::toeplitz(), 17
SuperGauss (SuperGauss-package), 2
SuperGauss-defunct, 16
SuperGauss-package, 2
toep.mult, 16
Toeplitz, 17