

# Package: SuperGauss (via r-universe)

August 27, 2024

**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 ( $\geq 3.0.0$ )

**Imports** stats, methods, R6, Rcpp ( $\geq 0.12.7$ ), fftw

**LinkingTo** Rcpp, RcppEigen

**Suggests** knitr, rmarkdown, testthat, mvtnorm, numDeriv

**VignetteBuilder** knitr

**RoxygenNote** 7.1.2

**Encoding** UTF-8

**SystemRequirements** fftw3 ( $\geq 3.1.2$ )

**NeedsCompilation** yes

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

**Maintainer** Martin Lysy <mlysy@uwaterloo.ca>

**Date/Publication** 2022-02-24 20:40:02 UTC

**Repository** <https://mlysy.r-universe.dev>

**RemoteUrl** <https://github.com/cran/SuperGauss>

**RemoteRef** HEAD

**RemoteSha** 098cf8989f2b4c73c04022dca375a7d6a1b8a750

## Contents

SuperGauss-package . . . . .	2
acf2incr . . . . .	3
acf2msd . . . . .	4
Cholesky . . . . .	4
Circulant . . . . .	5
dnormtz . . . . .	7
fbm_msd . . . . .	8
matern_acf . . . . .	9
msd2acf . . . . .	10
NormalCirculant . . . . .	10
NormalToeplitz . . . . .	12
pex_acf . . . . .	14
rnormtz . . . . .	15
SuperGauss-defunct . . . . .	16
toep.mult . . . . .	16
Toeplitz . . . . .	17
<b>Index</b>	<b>21</b>

---

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

**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  $x = (x_1, \dots, x_N)$  to that of its increments,  $dx = (x_2 - x_1, \dots, x_N - x_{(N-1)})$ .

**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	<i>Convert autocorrelation of stationary increments to mean squared displacement of positions.</i>
---------	--

---

**Description**

Converts the autocorrelation of a stationary increment sequence  $dx = (x_1 - x_0, \dots, x_N - x_{(N-1)})$  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	<i>Cholesky multiplication with Toeplitz variance matrices.</i>
----------	---

---

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

**Value**

Size N x p residual or observation matrix.

**Examples**

```
N <- 10
p <- 2
acf <- exp(-(1:N - 1))

Z <- matrix(rnorm(N * p), N, p)
cholZX(Z = Z, acf = acf) - (t(chol(toeplitz(acf))) \%*\% Z)

X <- matrix(rnorm(N * p), N, p)
cholXZ(X = X, acf = acf) - solve(t(chol(toeplitz(acf))), X)
```

---

Circulant

---

*Constructor and methods for Circulant matrix objects.*


---

**Description**

Constructor and methods for Circulant matrix objects.

**Methods****Public methods:**

- `Circulant$new()`
- `Circulant$size()`
- `Circulant$set_acf()`
- `Circulant$get_acf()`
- `Circulant$set_psd()`
- `Circulant$get_psd()`
- `Circulant$has_acf()`
- `Circulant$prod()`
- `Circulant$solve()`

- `Circulant$log_det()`
- `Circulant$clone()`

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

*Usage:*

`Circulant$new(N, uacf, upsd)`

*Arguments:*

`N` Size of Circulant matrix.

`uacf` Optional vector of  $N_u = \text{floor}(N/2)+1$  unique elements of the autocorrelation.

`upsd` Optional vector of  $N_u = \text{floor}(N/2)+1$  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  $N_u = \text{floor}(N/2)+1$  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  $C_t = \text{Circulant}(acf)$  is defined as  $psd = \text{iFFT}(acf)$ .

*Usage:*

`Circulant$set_psd(upsd)`

*Arguments:*

`upsd` Vector of  $N_u = \text{floor}(N/2)+1$  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.

---

dnormtz

*Density of a multivariate normal with Toeplitz variance matrix.*

---

## Description

Density of a multivariate normal with Toeplitz variance matrix.

## Usage

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

**Arguments**

<code>X</code>	Vector of length $N$ or $N \times n$ matrix, of which each column is a multivariate observation.
<code>mu</code>	Vector or matrix of mean values of compatible dimensions with $X$ . Defaults to all zeros.
<code>acf</code>	Vector of length $N$ containing the first column of the Toeplitz variance matrix.
<code>log</code>	Logical; whether to return the multivariate normal density on the log scale.
<code>method</code>	Which calculation method to use. Choices are: <code>gschur</code> for a modified version of the Generalized Schur algorithm of Ammar & Gragg (1988), or <code>ltz</code> 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 <- rnormtz(n, acf = acf) + mu

# evaluate log-density
dnormtz(X, 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**

<code>tseq</code>	Length- $N$ vector of timepoints.
<code>H</code>	Hurst parameter (between 0 and 1).



**Details**

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

$$\text{MSD}(t) = E[(X_t - X_0)^2].$$

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  $\text{MSD}(t) = |t|^{2H}$ .

**Value**

Length-N vector of mean square displacements.

**Examples**

```
fbm_msd(tseq = 1:10, H = 0.4)
```

---

matern_acf	<i>Matern autocorrelation function.</i>
------------	---

---

**Description**

Matern autocorrelation function.

**Usage**

```
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

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

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

**Value**

An autocorrelation vector of length N.

**Examples**

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

---

msd2acf	<i>Convert mean square displacement of positions to autocorrelation of increments.</i>
---------	--

---

**Description**

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

**Usage**

```
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))
```

---

NormalCirculant	<i>Multivariate normal with Circulant variance matrix.</i>
-----------------	--

---

**Description**

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

$$z \sim \text{NCt}(\text{uacf}) \iff z \sim \text{Normal}(\theta, \text{toeplitz}(\text{acf})),$$

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

$$\begin{aligned} \text{acf} &= (\text{uacf}, \text{rev}(\text{uacf}[2:(N_u-1)])), & N \text{ even,} \\ &= (\text{uacf}, \text{rev}(\text{uacf}[2:N_u])), & N \text{ odd.} \end{aligned}$$

**Methods****Public methods:**

- `NormalCirculant$new()`
- `NormalCirculant$size()`
- `NormalCirculant$logdens()`
- `NormalCirculant$grad_full()`
- `NormalCirculant$clone()`

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

*Usage:*

`NormalCirculant$new(N)`

*Arguments:*

N Size of the NCt random vector.

*Returns:* A NormalCirculant object.

**Method** `size()`: Get the size of the NCt random vector.

*Usage:*

`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

$z \sim \text{NTz}(\text{acf}) \iff z \sim \text{Normal}(\mathbf{0}, \text{toeplitz}(\text{acf})),$

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

## 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 × 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.

*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 × n_theta` matrix containing the gradient  $dz/d\theta$ .

`acf` A vector of length `N` containing the autocorrelation of the Toeplitz variance matrix.

`dacf` An `N × n_theta` matrix containing the gradient  $dacf/d\theta$ .

`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 × n_theta` matrix containing the gradient  $dz/d\theta$ .

`d2z` An `N × n_theta × n_theta` array containing the Hessian  $d^2z/d\theta^2$ .

`acf` A vector of length `N` containing the autocorrelation of the Toeplitz variance matrix.

`dacf` An `N × n_theta` matrix containing the gradient  $dacf/d\theta$ .

`d2acf` An `N × n_theta × n_theta` array containing the Hessian  $d^2acf/d\theta^2$ .

`full_out` If `TRUE`, returns the log-density and its gradient as well (see 'Value').

*Returns:* An `n_theta × 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.  
 dlda The length-N gradient vector with respect to acf, if calc\_dlda = TRUE.

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

*Usage:*

```
NormalToeplitz$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

---

pex\_acf

*Power-exponential autocorrelation function.*

---

**Description**

Power-exponential autocorrelation function.

**Usage**

```
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:

$$\text{ACF}(t) = \exp\left\{-\left(t/\lambda\right)^\rho\right\}.$$

**Value**

An autocorrelation vector of length N.

**Examples**

```
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 $(2N-2) \times n$ or $N \times n$ 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 `tol * ev_max`, where `ev_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-`nkeep` vector or size `nkeep`  $\times$  `n` matrix with time series as columns.

### Examples

```
N <- 10
acf <- exp(-(1:N - 1)/N)
rnormtz(n = 3, acf = acf)
```

---

SuperGauss-defunct      *Defunct functions in SuperGauss.*

---

### Description

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

```
toep.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

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



---

Toeplitz

*Constructor and methods for Toeplitz matrix objects.*

---

### Description

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

### Usage

```
is.Toeplitz(x)

as.Toeplitz(x)

## S3 method for class 'Toeplitz'
dim(x)
```

### Arguments

x                    An R object.

### Details

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

It is assumed that  $acf$  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).

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

method Solve method to use. Choices are: gschur for a modified version of the Generalized Schur algorithm of Ammar & Gragg (1988), or pcg for the preconditioned conjugate gradient method of Chen et al (2006). The former is faster and obtains the log-determinant as a direct biproduct. The latter is more numerically stable for long-memory autocorrelations.

tol Tolerance level for the pcg method.

*Returns:* The solution in z to the system of equations  $Tz \text{ %% } z = x$ . 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

```
solve(Tz, 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

```
solve(Tz, toeplitz(acf2)) %% solve(Tz, 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`, 10

`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  
`Snorm.hess` (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