Package ‘SuperGauss’

February 27, 2020

Type Package

Title Superfast Likelihood Inference for Stationary Gaussian Time Series

Version 1.0.2

Date 2020-02-27

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, Rcpp (>= 0.12.7), fftw

LinkingTo Rcpp, RcppEigen

Suggests knitr, rmarkdown, testthat, mvtnorm, numDeriv

VignetteBuilder knitr

RoxygenNote 7.0.2

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 2020-02-27 12:40:02 UTC

R topics documented:

`acf2incr` *Convert position to increment autocorrelations.*

Description

Converts the autocorrelation of a stationary sequence `X` to that of its increments, `dX == diff(X)`.

Usage

```
acf2incr(acf)
```

Arguments

`acf`	Length-`N` vector of position autocorrelations.

Value

Length `N-1` vector if increment autocorrelations.

Examples

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

acf2msd — *Convert autocorrelation of stationary increments to mean squared displacement of posititions.*

Description

Converts the autocorrelation (ACF) of stationary increments to the mean squared displacement (MSD) of the corresponding positions.

Usage

```
acf2msd(acf)
```

Arguments

`acf`	length-`N` ACF vector of a stationary increment sequence.

Details

If $X(t)$ is a stationary increments process, then $\Delta X_0, \Delta X_1, \ldots$ with

$$\Delta X_n = X((n+1)\Delta t) - X(n\Delta t)$$

is a stationary time series. This function converts the ACF of this series into the MSD of the corresponding positions, namely returns the sequence $\eta_1, \ldots, \eta_N$, where $\eta_i = \mathrm{var}(X(i\Delta t))$.

Value

Length-`N` MSD vector of the corresponding positions.

Examples

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

Choleski — *Choleski multiplication with Toeplitz variance matrices.*

Description

Multiplies the Choleski 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(chol(toeplitz(acf)))`, then `cholZX` computes `C %*% Z` and `cholZX` computes `solve(C,X)`. Both functions use the Durbin-Levinson algorithm.

Value

Size `N x p` residual or observation matrix.

Examples

```
N <- 10
p <- 2
W <- matrix(rnorm(N * p), N, p)
acf <- exp(-(1:N - 1))
cholZX(Z = W, acf = acf)
cholXZ(X = W, acf = acf)
```

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

Description

Efficient density evaluation for the multivariate normal distribution with Toeplitz variance matrix.

Usage

```
dSnorm(X, mu, acf, log = FALSE)

dSnormDL(X, mu, acf, log = FALSE)
```

Arguments

`X`	Vector or 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 containing the first column of the Toeplitz variance matrix. For `dSnorm`, can also be a `Toeplitz` object.
`log`	Logical, whether to return the multivariate normal density on the log scale.

Details

`dSnorm` and `dSnormDL` have identical outputs, with the former using the generalized Schur algorithm and the latter, the Durbin-Levinson algorithm, which is more common but slower. `dSnormDL` is provided mainly for speed comparisons.

Value

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

Examples

```
N <- 10
d <- 4
X <- matrix(rnorm(N*d), N, d)
theta <- 0.1
lambda <- 2

mu <- theta^2 * rep(1, N)
acf <- exp(-lambda * (1:N - 1))
acf <- Toeplitz(acf = acf)

dSnorm(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

`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

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

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

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

```
matern.acf(tseq, lambda, nu)
```

Arguments

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

Details

The Matern autocorrelation is given by

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

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` *Convert mean square displacement to autocorrelations.*

Description

Converts the mean squared displacement (MSD) of positions to the autocorrelation (ACF) of the corresponding increments.

Usage

```
msd2acf(msd)
```

Arguments

`msd`	Length-`N` vector of MSDs at regular timepoints `dt,2*dt,...,N*dt`.

Details

For a stationary increments process X_t, converts a sequence $\eta_1, \ldots, \eta_N$ of regularly spaced MSDs,

$$\eta_i = E[(X_{i\Delta t} - X_0)^2],$$

into $\gamma_1, \ldots, \gamma_N$, a sequence of regularly spaced ACFs,

$$\gamma_i = \mathrm{cov}\{X_{(i+1)\Delta t} - X_{i\Delta_i}, X_{\Delta t} - X_0\}.$$

This only produces correct results when `msd` corresponds to equally-spaced observations.

Value

Length `N` vector of ACFs.

Examples

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

`pex.acf` *Power-exponential autocorrelation function.*

Description

Power-exponential autocorrelation function.

Usage

```
pex.acf(tseq, lambda, rho)
```

Arguments

`tseq`	Length-`N` vector of timepoints.
`lambda`	Timescale parameter.
`rho`	Power parameter.

Details

The power-exponential autocorrelation function is given by:

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

Value

Length-`N` autocorrelation vector.

Examples

```
pex.acf(tseq = 1:10, lambda = 1, rho = 2)
```

`rSnorm` *Simulation of a stationary Gaussian time series.*

Description

Simulation of a stationary Gaussian time series.

Usage

```
rSnorm(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 superfast FFT-based algorithm of Chan and Wood or the more stable Durbin-Levinson algorithm. See Details.
`nkeep`	Length of time series. Defaults to `N = length(acf)`. See Details.
`tol`	Relative tolerance on negative eigenvalues. See Details.

Details

The superfast method fails when the circulant matrix is not positive definite. This is typically due to one of two things, for which the FFT algorithm can be tuned with `tol` and `nkeep`:

`tol` Roundoff error can make tiny eigenvalues appear negative. If `evMax` is the maximum eigenvalue, then all negative eigenvalues of magnitude less than `tol * evMax` are mapped to this threshold value. This does not guarantee a positive definite embedding.

`nkeep` The autocorrelation is decaying too slowly on the given timescale. To mitigate this it is possible to increase the time horizon, i.e. input a longer `acf` and keep the first `nkeep` time series observations. For consistency, `nkeep` also applies to Durbin-Levinson method.

Value

Length-`nkeep` vector or size `nkeep x n` matrix with time series as columns.

Examples

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

`Snorm.grad` *Gradient of the loglikelihood of a multivariate normal with Toeplitz variance matrix.*

Description

Superfast evaluation of loglikelihood gradient.

Usage

```
Snorm.grad(X, mu, acf, dmu, dacf)
```

Arguments

X	A length-N vector of multivariate normal observations.
mu	A scalar or length-N vector of means. If missing defaults to the vector of zeros.
acf	A Toeplitz object or length-N vector containing the first column of the Toeplitz variance matrix.
dmu	A length-p vector or N x p matrix of partial derivatives of mu along the columns. If missing defaults to a matrix of zeros.
dacf	An N x p matrix with the partial derivatives of acf along the columns.

Value

A length-p vector containing the gradient of the loglikelihood.

Examples

```
# two parameter inference
acf.fun <- function(theta) theta[2]^2 * exp(-theta[1]*(1:N-1))
mu.fun <- function(theta) theta[1]

# partial derivatives
dacf.fun <- function(theta) {
  ea <- exp(-theta[1]*(1:N-1))
  cbind(-theta[1]*theta[2]^2 * ea, 2*theta[2] * ea)
}
dmu.fun <- function(theta) c(1, 0)

# generate data
N <- 300
theta <- rexp(2)
X <- rSnorm(n = 1, acf = acf.fun(theta)) + mu.fun(theta)

# likelihood gradient
Snorm.grad(X = X, mu = mu.fun(theta), dmu = dmu.fun(theta),
           acf = acf.fun(theta), dacf = dacf.fun(theta))
```

Snorm.hess — *Hessian of the loglikelihood of a multivariate normal with Toeplitz variance matrix.*

Description

Superfast evaluation of loglikelihood Hessian.

Usage

```
Snorm.hess(X, mu, acf, dmu, dacf, d2mu, d2acf)
```

Arguments

`X`	A length-`N` vector of multivariate normal observations.
`mu`	A scalar or length-`N` vector of means. If missing defaults to the vector of zeros.
`acf`	A `Toeplitz` object or length-`N` vector containing the first column of the Toeplitz variance matrix.
`dmu`	A length-`p` vector or `N x p` matrix of partial derivatives of `mu` along the columns. If missing defaults to a matrix of zeros.
`dacf`	An `N x p` matrix with the partial derivatives of `acf` along the columns.
`d2mu`	A `p x p` matrix or `N x p x p` array of second partial derivatives of `mu`. If missing defaults to zeros.
`d2acf`	A `N x p x p` array of second partial derivatives of `acf`.

Value

The `p x p` Hessian matrix of the loglikelihood.

Examples

```
# two parameter inference
acf.fun <- function(theta) theta[2]^2 * exp(-(1:N-1))
mu.fun <- function(theta) theta[1] * (1:N) + log(theta[2] + 1:N)

# partial derivatives
dacf.fun <- function(theta) {
  cbind(0, 2*theta[2] * exp(-(1:N-1)))
}
dmu.fun <- function(theta) cbind(1:N, 1/(theta[2] + 1:N))

# 2nd order partials
d2acf.fun <- function(theta) {
  H <- array(0, dim = c(N, 2, 2))
  H[,2,2] <- 2*exp(-(1:N-1))
  H
}
d2mu.fun <- function(theta) {
  H <- array(0, dim = c(N, 2, 2))
  H[,2,2] <- -1/(theta[2] + 1:N)^2
  H
}

# generate data
N <- 300
theta <- rexp(2)
X <- rSnorm(n = 1, acf = acf.fun(theta)) + mu.fun(theta)

# likelihood Hessian
Snorm.hess(X = X, mu = mu.fun(theta), acf = acf.fun(theta),
           dmu = dmu.fun(theta), dacf = dacf.fun(theta),
           d2mu = d2mu.fun(theta), d2acf = d2acf.fun(theta))
```

SuperGauss *Superfast inference for stationary Gaussian time series.*

Description

Superfast inference for stationary Gaussian time series.

Details

While likelihood calculations with stationary Gaussian time series generally scale as $\mathcal{O}(N^2)$ in the number of observations, this package implements an algorithm which scales as $\mathcal{O}(N \log^2 N)$. "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.

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 <- rSnorm(n = 1, acf = exp.acf(lambda0))

# loglikelihood function
Toep <- Toeplitz(n = N) # allocate memory for a Toeplitz matrix object
loglik <- function(lambda) {
  Toep$setAcf(acf = exp.acf(lambda))
  dSnorm(X = X, acf = Toep, log = TRUE)
}

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

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-class` *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

```
Toeplitz(n, acf)
```

Arguments

`n`	Size of the Toeplitz matrix.
`acf`	Autocorrelation vector of Toeplitz matrix.

Details

It is assumed that the autocorrelation of the `Toeplitz` object defines a valid (i.e., positive definite) variance matrix. The multiplication algorithms still work when this is not the case but the other algorithms do not (return values typically contain `NaN`s).

Value

A `Toeplitz` object.

Methods

If `Toep` is a `Toeplitz` object with first row/column given by `acf`, then:

`Toep$setAcf(acf)` Sets the autocorrelation of the matrix.

`Toep$getAcf()` Gets the autocorrelation of the matrix.

`nrow(Toep)`, `ncol(Toep)`, `dim(Toep)` Selected dimension(s) of the matrix.

`Toep %*% X`, `X %*% Toep` Toeplitz-Matrix and Matrix-Toeplitz multiplication. Also works if `X` is a vector.

`solve(Toep, X)`, `solve(Toep)` Solves Toeplitz systems of equations. When second argument is missing, returns the inverse of the Toeplitz matrix.

`determinant(Toep)` Log-determinant of the Toeplitz matrix, i.e., same thing as `log(det(toeplitz(acf)))`.

`Toep$traceT2(acf2)` If `T1 == toeplitz(acf)` and `T2 == toeplitz(acf2)`, computes the trace of `solve(T1,T2)`. This is used in the computation of the gradient of Gaussian likelihoods with Toeplitz variance matrix.

`Toep$traceT4(acf2, acf3)` If `T1 == toeplitz(acf)`, `T2 == toeplitz(acf2)`, and `T3 == toeplitz(acf3)`, computes the trace of `solve(T1,T2) %*% solve(T1,T3)`. This is used in the computation of the Hessian of Gaussian likelihoods with Toeplitz variance matrix.

Examples

```
# construction
acf <- exp(-(1:5))
Toep <- Toeplitz(acf = acf)
# alternatively, can allocate space first
Toep <- Toeplitz(n = length(acf))
Toep$setAcf(acf = acf)

dim(Toep) # == c(nrow(Toep), ncol(Toep))
Toep # show method
Toep$getAcf() # extract the acf

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

Index