Package: sdetorus (via r-universe)

October 28, 2026

Type Package

Title Statistical Tools for Toroidal Diffusions

Version 0.1.10

Date 2024-03-01

Description Implementation of statistical methods for the estimation of toroidal diffusions. Several diffusive models are provided, most of them belonging to the Langevin family of diffusions on the torus. Specifically, the wrapped normal and von Mises processes are included, which can be seen as toroidal analogues of the Ornstein-Uhlenbeck diffusion. A collection of methods for approximate maximum likelihood estimation, organized in four blocks, is given: (i) based on the exact transition probability density, obtained as the numerical solution to the Fokker-Plank equation; (ii) based on wrapped pseudo-likelihoods; (iii) based on specific analytic approximations by wrapped processes; (iv) based on maximum likelihood of the stationary densities. The package allows the replicability of the results in García-Portugués et al. (2019) <doi:10.1007/s11222-017-9790-2>.

License GPL-3

Depends R (>= 3.6.0), Rcpp, mvtnorm

Suggests rgl, Bessel, manipulate

LinkingTo Rcpp, RcppArmadillo

URL https://github.com/egarpor/sdetorus

BugReports https://github.com/egarpor/sdetorus

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation yes

Author Eduardo García-Portugués [aut, cre] (<https://orcid.org/0000-0002-9224-4111>)

Maintainer Eduardo García-Portugués <edgarcia@est-econ.uc3m.es>

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

Date/Publication 2024-03-02 02:36:01 UTC

RemoteUrl https://github.com/cran/sdetorus

RemoteRef HEAD

RemoteSha e14dc98b3e567673da5cf7f6714cfc6c3d561bea

Contents

alphaToA . . . 3
approxMleWn1D . . . 4
approxMleWn2D . . . 5
approxMleWnPairs . . . 7
crankNicolson1D . . . 8
crankNicolson2D . . . 10
dBvm . . . 12
diffCirc . . . 13
dJp . . . 14
dPsTpd . . . 15
driftJp . . . 19
driftMixIndVm . . . 20
driftMixVm . . . 21
driftMvm . . . 22
driftWn . . . 23
driftWn1D . . . 24
driftWn2D . . . 25
dStatWn2D . . . 26
dTpdMou . . . 27
dTpdOu . . . 28
dTpdPde1D . . . 29
dTpdPde2D . . . 30
dTpdWou . . . 32
dTpdWou1D . . . 33
dTpdWou2D . . . 35
dVm . . . 37
dWn1D . . . 38
euler1D . . . 39
euler2D . . . 40
linesCirc . . . 41
linesTorus . . . 42
linesTorus3d . . . 43
logBesselI0Scaled . . . 44
logLikWouPairs . . . 45
mleMou . . . 46
mleOptimWrapper . . . 47
mleOu . . . 50
mlePde1D . . . 51

mlePde2D 53
periodicTrapRule1D 55
psMle 57
rStatWn2D 60
rTpdWn2D 61
rTrajLangevin 62
rTrajMou 63
rTrajOu 64
rTrajWn1D 65
rTrajWn2D 66
safeSoftMax 67
scoreMatchWnBvm 68
sdetorus 69
sigmaDiff 70
solveTridiag 71
stepAheadWn1D 73
stepAheadWn2D 74
toPiInt 76
torusAxis 77
torusAxis3d 77
unwrapCircSeries 78

Index **80**

`alphaToA` *Valid drift matrices for the Ornstein–Uhlenbeck diffusion in 2D*

Description

Constructs drift matrices A such that `solve(A) %*% Sigma` is symmetric.

Usage

```
alphaToA(alpha, sigma = NULL, rho = 0, Sigma = NULL)

aToAlpha(A, sigma = NULL, rho = 0, Sigma = NULL)
```

Arguments

`alpha`	vector of length `3` containing the `A` matrix. The first two elements are the diagonal.
`sigma`	vector of length `2` containing the **square root** of the diagonal of `Sigma`.
`rho`	correlation of `Sigma`.
`Sigma`	the diffusion matrix of size `c(2, 2)`.
`A`	matrix of size `c(2, 2)`.

Details

The parametrization enforces that `solve(A) %*% Sigma` is symmetric. Positive definiteness is guaranteed if `alpha[3]^2 < rho^2 * (alpha[1] - alpha[2])^2 / 4 + alpha[1] * alpha[2]`.

Value

The drift matrix `A` or the `alpha` vector.

Examples

```
# Parameters
alpha <- 3:1
Sigma <- rbind(c(1, 0.5), c(0.5, 4))

# Covariance matrix
A <- alphaToA(alpha = alpha, Sigma = Sigma)
S <- 0.5 * solve(A) %*% Sigma
det(S)

# Check
aToAlpha(A = alphaToA(alpha = alpha, Sigma = Sigma), Sigma = Sigma)
alphaToA(alpha = aToAlpha(A = A, Sigma = Sigma), Sigma = Sigma)
```

`approxMleWn1D` *Approximate MLE of the WN diffusion in 1D*

Description

Approximate Maximum Likelihood Estimation (MLE) for the Wrapped Normal (WN) in 1D using the wrapped Ornstein–Uhlenbeck diffusion.

Usage

```
approxMleWn1D(data, delta, start, alpha = NA, mu = NA, sigma = NA,
  lower = c(0.01, -pi, 0.01), upper = c(25, pi, 25), vmApprox = FALSE,
  maxK = 2, ...)
```

Arguments

`data`	a matrix of dimension `c(n, p)`.
`delta`	discretization step.
`start`	starting values, a matrix with `p` columns, with each entry representing a different starting value.
`alpha`, `mu`, `sigma`	if their values are provided, the likelihood function is optimized with respect to the rest of unspecified parameters. The number of elements in `start`, `lower` and `upper` has to be modified accordingly (see examples).
`lower`, `upper`	bound for box constraints as in method `"L-BFGS-B"` of `optim`.

vmApprox	flag to indicate von Mises approximation to wrapped normal. See momentMatchWnVm and scoreMatchWnBvm.
maxK	maximum absolute winding number used if circular = TRUE.
...	further parameters passed to mleOptimWrapper.

Details

See Section 3.3 in García-Portugués et al. (2019) for details.

Value

Output from mleOptimWrapper.

References

García-Portugués, E., Sørensen, M., Mardia, K. V. and Hamelryck, T. (2019) Langevin diffusions on the torus: estimation and applications. *Statistics and Computing*, 29(2):1–22. doi:10.1007/s1122201797902

Examples

```
alpha <- 0.5
mu <- 0
sigma <- 2
samp <- rTrajWn1D(x0 = 0, alpha = alpha, mu = mu, sigma = sigma, N = 1000,
                  delta = 0.1)
approxMleWn1D(data = samp, delta = 0.1, start = c(alpha, mu, sigma))
approxMleWn1D(data = samp, delta = 0.1, sigma = sigma, start = c(alpha, mu),
              lower = c(0.01, -pi), upper = c(25, pi))
approxMleWn1D(data = samp, delta = 0.1, mu = mu, start = c(alpha, sigma),
              lower = c(0.01, 0.01), upper = c(25, 25))
```

---

approxMleWn2D *Approximate MLE of the WN diffusion in 2D*

Description

Approximate Maximum Likelihood Estimation (MLE) for the Wrapped Normal (WN) in 2D using the wrapped Ornstein–Uhlenbeck diffusion.

Usage

```
approxMleWn2D(data, delta, start, alpha = rep(NA, 3), mu = rep(NA, 2),
  sigma = rep(NA, 2), rho = NA, lower = c(0.01, 0.01, -25, -pi, -pi,
  0.01, 0.01, -0.99), upper = c(rep(25, 3), pi, pi, 25, 25, 0.99),
  maxK = 2, ...)
```

Arguments

`data`	a matrix of dimension `c(n, p)`.
`delta`	discretization step.
`start`	starting values, a matrix with `p` columns, with each entry representing a different starting value.
`alpha`, `mu`, `sigma`, `rho`	if their values are provided, the likelihood function is optimized with respect to the rest of unspecified parameters. The number of elements in `start`, `lower` and `upper` has to be modified accordingly (see examples).
`lower`, `upper`	bound for box constraints as in method `"L-BFGS-B"` of `optim`.
`maxK`	maximum absolute winding number used if `circular = TRUE`.
`...`	further parameters passed to `mleOptimWrapper`.

Details

See Section 3.3 in García-Portugués et al. (2019) for details.

Value

Output from `mleOptimWrapper`.

References

García-Portugués, E., Sørensen, M., Mardia, K. V. and Hamelryck, T. (2019) Langevin diffusions on the torus: estimation and applications. *Statistics and Computing*, 29(2):1–22. doi:10.1007/s1122201797902

Examples

```
alpha <- c(2, 2, -0.5)
mu <- c(0, 0)
sigma <- c(1, 1)
rho <- 0.2
samp <- rTrajWn2D(x0 = c(0, 0), alpha = alpha, mu = mu, sigma = sigma,
                  rho = rho, N = 1000, delta = 0.1)
approxMleWn2D(data = samp, delta = 0.1, start = c(alpha, mu, sigma, rho))
approxMleWn2D(data = samp, delta = 0.1, alpha = alpha,
              start = c(mu, sigma), lower = c(-pi, -pi, 0.01, 0.01),
              upper = c(pi, pi, 25, 25))
mleMou(data = samp, delta = 0.1, start = c(alpha, mu, sigma),
       optMethod = "Nelder-Mead")
```

`approxMleWnPairs` *Approximate MLE of the WN diffusion in 2D from a sample of initial and final pairs of angles.*

Description

Approximate Maximum Likelihood Estimation (MLE) for the Wrapped Normal (WN) diffusion, using the wrapped Ornstein–Uhlenbeck diffusion and assuming initial stationarity.

Usage

```
approxMleWnPairs(data, delta, start = c(0, 0, 1, 1, 0, 1, 1),
  alpha = rep(NA, 3), mu = rep(NA, 2), sigma = rep(NA, 2), rho = NA,
  lower = c(-pi, -pi, 0.01, 0.01, -25, 0.01, 0.01, -0.99), upper = c(pi,
  pi, 25, 25, 25, 25, 25, 0.99), maxK = 2, expTrc = 30, ...)
```

Arguments

`data`	a matrix of dimension `c(n, p)`.
`delta`	discretization step.
`start`	starting values, a matrix with `p` columns, with each entry representing a different starting value.
`alpha`	vector of length `3` parametrizing the `A` matrix as in `alphaToA`.
`mu`	a vector of length `2` giving the mean.
`sigma`	vector of length `2` containing the **square root** of the diagonal of Σ, the diffusion matrix.
`rho`	correlation coefficient of Σ.
`lower, upper`	bound for box constraints as in method `"L-BFGS-B"` of `optim`.
`maxK`	maximum absolute value of the windings considered in the computation of the WN.
`expTrc`	truncation for exponential: `exp(x)` with `x <= -expTrc` is set to zero. Defaults to `30`.
`...`	further parameters passed to `mleOptimWrapper`.

Value

Output from `mleOptimWrapper`.

Examples

```
mu <- c(0, 0)
alpha <- c(1, 2, 0.5)
sigma <- c(1, 1)
rho <- 0.5
set.seed(4567345)
```

```
begin <- rStatWn2D(n = 200, mu = mu, alpha = alpha, sigma = sigma)
end <- t(apply(begin, 1, function(x) rTrajWn2D(x0 = x, alpha = alpha,
                                               mu = mu, sigma = sigma,
                                               rho = rho, N = 1,
                                               delta = 0.1)[2, ]))
data <- cbind(begin, end)
approxMleWnPairs(data = data, delta = 0.1,
                 start = c(2, pi/2, 2, 0.5, 0, 2, 1, 0.5))
```

`crankNicolson1D` *Crank–Nicolson finite difference scheme for the 1D Fokker–Planck equation with periodic boundaries*

Description

Implementation of the Crank–Nicolson scheme for solving the Fokker–Planck equation

$$p(x,t)_t = -(p(x,t)b(x))_x + \frac{1}{2}(\sigma^2(x)p(x,t))_{xx},$$

where $p(x,t)$ is the transition probability density of the circular diffusion

$$dX_t = b(X_t)dt + \sigma(X_t)dW_t$$

.

Usage

```
crankNicolson1D(u0, b, sigma2, N, deltat, Mx, deltax, imposePositive = 0L)
```

Arguments

`u0`	matrix of size `c(Mx, 1)` giving the initial condition. Typically, the evaluation of a density highly concentrated at a given point. If `nt == 1`, then `u0` can be a matrix `c(Mx, nu0)` containing different starting values in the columns.
`b`	vector of length `Mx` containing the evaluation of the drift.
`sigma2`	vector of length `Mx` containing the evaluation of the squared diffusion coefficient.
`N`	increasing integer vector of length `nt` giving the indexes of the times at which the solution is desired. The times of the solution are `delta * c(0:max(N))[N + 1]`.
`deltat`	time step.
`Mx`	size of the equispaced spatial grid in $[-\pi, \pi)$.
`deltax`	space grid discretization.
`imposePositive`	flag to indicate whether the solution should be transformed in order to be always larger than a given tolerance. This prevents spurious negative values. The tolerance will be taken as `imposePositiveTol` if this is different from `FALSE` or `0`.

Details

The function makes use of `solvePeriodicTridiag` for obtaining implicitly the next step in time of the solution.

If `imposePositive = TRUE`, the code implicitly assumes that the solution integrates to one at any step. This might b unrealistic if the initial condition is not properly represented in the grid (for example, highly concentrated density in a sparse grid).

Value

- If `nt > 1`, a matrix of size `c(Mx, nt)` containing the discretized solution at the required times.
- If `nt == 1`, a matrix of size `c(Mx, nu0)` containing the discretized solution at a fixed time for different starting values.

References

Thomas, J. W. (1995). *Numerical Partial Differential Equations: Finite Difference Methods*. Springer, New York. doi:10.1007/9781489972781

Examples

```
# Parameters
Mx <- 200
N <- 200
x <- seq(-pi, pi, l = Mx + 1)[-c(Mx + 1)]
times <- seq(0, 1, l = N + 1)
u0 <- dWn1D(x, pi/2, 0.05)
b <- driftWn1D(x, alpha = 1, mu = pi, sigma = 1)
sigma2 <- rep(1, Mx)

# Full trajectory of the solution (including initial condition)
u <- crankNicolson1D(u0 = cbind(u0), b = b, sigma2 = sigma2, N = 0:N,
                     deltat = 1 / N, Mx = Mx, deltax = 2 * pi / Mx)

# Mass conservation
colMeans(u) * 2 * pi

# Visualization of tpd
plotSurface2D(times, x, z = t(u), levels = seq(0, 3, l = 50))

# Only final time
v <- crankNicolson1D(u0 = cbind(u0), b = b, sigma2 = sigma2, N = N,
                     deltat = 1 / N, Mx = Mx, deltax = 2 * pi / Mx)
sum(abs(u[, N + 1] - v))
```

`crankNicolson2D` *Crank–Nicolson finite difference scheme for the 2D Fokker–Planck equation with periodic boundaries*

Description

Implementation of the Crank–Nicolson scheme for solving the Fokker–Planck equation

$$p(x,y,t)_t = -(p(x,y,t)b_1(x,y))_x - (p(x,y,t)b_2(x,y))_y +$$

$$+\frac{1}{2}(\sigma_1^2(x,y)p(x,y,t))_{xx} + \frac{1}{2}(\sigma_2^2(x,y)p(x,y,t))_{yy} + (\sigma_{12}(x,y)p(x,y,t))_{xy},$$

where $p(x,y,t)$ is the transition probability density of the toroidal diffusion

$$dX_t = b_1(X_t,Y_t)dt + \sigma_1(X_t,Y_t)dW_t^1 + \sigma_{12}(X_t,Y_t)dW_t^2,$$

$$dY_t = b_2(X_t,Y_t)dt + \sigma_{12}(X_t,Y_t)dW_t^1 + \sigma_2(X_t,Y_t)dW_t^2.$$

Usage

```
crankNicolson2D(u0, bx, by, sigma2x, sigma2y, sigmaxy, N, deltat, Mx, deltax,
  My, deltay, imposePositive = 0L)
```

Arguments

`u0`	matrix of size `c(Mx * My, 1)` giving the initial condition matrix column-wise stored. Typically, the evaluation of a density highly concentrated at a given point. If `nt == 1`, then `u0` can be a matrix `c(Mx * My, nu0)` containing different starting values in the columns.
`bx`, `by`	matrices of size `c(Mx, My)` containing the evaluation of the drift in the first and second space coordinates, respectively.
`sigma2x`, `sigma2y`, `sigmaxy`	matrices of size `c(Mx, My)` containing the evaluation of the entries of the diffusion matrix (it has to be positive definite) `rbind(c(sigma2x, sigmaxy), c(sigmaxy, sigma2y))`.
`N`	increasing integer vector of length `nt` giving the indexes of the times at which the solution is desired. The times of the solution are `deltat * c(0:max(N))[N + 1]`.
`deltat`	time step.
`Mx`, `My`	sizes of the equispaced spatial grids in $[-\pi, \pi)$ for each component.
`deltax`, `deltay`	space grid discretizations for each component.
`imposePositive`	flag to indicate whether the solution should be transformed in order to be always larger than a given tolerance. This prevents spurious negative values. The tolerance will be taken as `imposePositiveTol` if this is different from `FALSE` or `0`.

Details

The function makes use of `solvePeriodicTridiag` for obtaining implicitly the next step in time of the solution.

If `imposePositive = TRUE`, the code implicitly assumes that the solution integrates to one at any step. This might b unrealistic if the initial condition is not properly represented in the grid (for example, highly concentrated density in a sparse grid).

Value

- If `nt > 1`, a matrix of size `c(Mx * My, nt)` containing the discretized solution at the required times with the `c(Mx, My)` matrix stored column-wise.
- If `nt == 1`, a matrix of size `c(Mx * My, nu0)` containing the discretized solution at a fixed time for different starting values.

References

Thomas, J. W. (1995). *Numerical Partial Differential Equations: Finite Difference Methods*. Springer, New York. doi:10.1007/9781489972781

Examples

```
# Parameters
Mx <- 100
My <- 100
N <- 200
x <- seq(-pi, pi, l = Mx + 1)[-c(Mx + 1)]
y <- seq(-pi, pi, l = My + 1)[-c(My + 1)]
m <- c(pi / 2, pi)
p <- c(0, 1)
u0 <- c(outer(dWn1D(x, p[1], 0.5), dWn1D(y, p[2], 0.5)))
bx <- outer(x, y, function(x, y) 5 * sin(m[1] - x))
by <- outer(x, y, function(x, y) 5 * sin(m[2] - y))
sigma2 <- matrix(1, nrow = Mx, ncol = My)
sigmaxy <- matrix(0.5, nrow = Mx, ncol = My)

# Full trajectory of the solution (including initial condition)
u <- crankNicolson2D(u0 = cbind(u0), bx = bx, by = by, sigma2x = sigma2,
                     sigma2y = sigma2, sigmaxy = sigmaxy,
                     N = 0:N, deltat = 1 / N, Mx = Mx, deltax = 2 * pi / Mx,
                     My = My, deltay = 2 * pi / My)

# Mass conservation
colMeans(u) * 4 * pi^2

# Only final time
v <- crankNicolson2D(u0 = cbind(u0), bx = bx, by = by, sigma2x = sigma2,
                     sigma2y = sigma2, sigmaxy = sigmaxy,
                     N = N, deltat = 1 / N, Mx = Mx, deltax = 2 * pi / Mx,
                     My = My, deltay = 2 * pi / My)
sum(abs(u[, N + 1] - v))
```

```
## Not run:
# Visualization of tpd
library(manipulate)
manipulate({
  plotSurface2D(x, y, z = matrix(u[, j + 1], Mx, My),
                main = round(mean(u[, j + 1]) * 4 * pi^2, 4),
                levels = seq(0, 2, l = 21))
  points(p[1], p[2], pch = 16)
  points(m[1], m[2], pch = 16)
}, j = slider(0, N))

## End(Not run)
```

---

dBvm *Bivariate Sine von Mises density*

Description

Evaluation of the bivariate Sine von Mises density and its normalizing constant.

Usage

```
dBvm(x, mu, kappa, logConst = NULL)

constBvm(M = 25, kappa)
```

Arguments

x	a matrix of size c(nx, 2) for evaluating the density.
mu	two-dimensional vector of circular means.
kappa	three-dimensional vector with concentrations $(\kappa_1, \kappa_2, \lambda)$.
logConst	logarithm of the normalizing constant. Computed if NULL.
M	number of terms considered in the series expansion used for evaluating the normalizing constant.

Details

If $\kappa_1 = 0$ or $\kappa_2 = 0$ and $\lambda \neq 0$, then constBvm will perform a Monte Carlo integration of the constant.

Value

A vector of length nx with the evaluated density (dBvm) or a scalar with the normaalizing constant (constBvm).

References

Singh, H., Hnizdo, V. and Demchuk, E. (2002) Probabilistic model for two dependent circular variables, *Biometrika*, 89(3):719–723, doi:10.1093/biomet/89.3.719

Examples

```
x <- seq(-pi, pi, l = 101)[-101]
plotSurface2D(x, x, f = function(x) dBvm(x = x, mu = c(0, pi / 2),
                                         kappa = c(2, 3, 1)),
              fVect = TRUE)
```

`diffCirc` *Lagged differences for circular time series*

Description

Returns suitably lagged and iterated circular differences.

Usage

```
diffCirc(x, circular = TRUE, ...)
```

Arguments

`x`	wrapped or unwrapped angles to be differenced. Must be a vector or a matrix, see details.
`circular`	convenience flag to indicate whether wrapping should be done. If `FALSE`, the function is exactly `diff`.
`...`	parameters to be passed to `diff`.

Details

If `x` is a matrix then the difference operations are carried out row-wise, on each column separately.

Value

The value of `diff(x, ...)`, circularly wrapped. Default parameters give an object of the kind of `x` with one less entry or row.

Examples

```
# Vectors
x <- c(-pi, -pi/2, pi - 0.1, -pi + 0.2)
diffCirc(x) - diff(x)

# Matrices
set.seed(234567)
N <- 100
```

```
x <- t(euler2D(x0 = rbind(c(0, 0)), A = diag(c(1, 1)), sigma = rep(2, 2),
               mu = c(pi, pi), N = N, delta = 1, type = 2)[1, , ])
diffCirc(x) - diff(x)
```

---

dJp *Jones and Pewsey (2005)'s circular distribution*

Description

Computes the circular density of Jones and Pewsey (2005).

Usage

```
dJp(x, mu, kappa, psi, const = NULL)

constJp(mu, kappa, psi, M = 200)
```

Arguments

x	evaluation angles, not necessary in $[\pi, \pi)$.
mu	circular mean.
kappa	non-negative concentration parameter.
psi	shape parameter, see details.
const	normalizing constant, computed with constJp if not provided.
M	grid size for computing the normalizing constant by numerical integration.

Details

Particular interesting choices for the shape parameter are:

- psi = -1: gives the Wrapped Cauchy as stationary density.
- psi = 0: is the sinusoidal drift of the vM diffusion.
- psi = 1: gives the Cardioid as stationary density.

Value

A vector of the same length as x containing the density.

References

Jones, M. C. and Pewsey, A. (2005). A family of symmetric distributions on the circle. *Journal of the American Statistical Association*, 100(472):1422–1428. doi:10.1198/016214505000000286

Examples

```
x <- seq(-pi, pi, l = 200)
plot(x, x, type = "n", ylab = "Density", ylim = c(0, 0.6))
for (i in 0:20) {
  lines(x, dJp(x = x, mu = 0, kappa = 1, psi = -2 + 4 * i / 20),
        col = rainbow(21)[i + 1])
}
```

---

dPsTpd *Wrapped Euler and Shoji–Ozaki pseudo-transition probability densities*

Description

Wrapped pseudo-transition probability densities.

Usage

```
dPsTpd(x, x0, t, method = c("E", "SO", "SO2"), b, jac.b, sigma2, b1, b2,
  circular = TRUE, maxK = 2, vmApprox = FALSE, twokpi = NULL, ...)
```

Arguments

x	a matrix of dimension c(n, p). If a vector is provided, is assumed that p = 1.
x0	a matrix of dimension c(n, p). If all x0 are the same, a matrix of dimension c(1, p) can be passed for better performance. If a vector is provided, is assumed that p = 1.
t	time step between x and x0.
method	a string for choosing "E" (Euler), "SO" (Shoji–Ozaki) or "SO2" (Shoji–Ozaki with Ito's expansion in the drift) method.
b	drift function. Must return a matrix of the same size as x.
jac.b	jacobian of the drift function.
sigma2	diagonal of the diffusion matrix (if univariate, this is the square of the diffusion coefficient). Must return an object of the same size as x.
b1	first derivative of the drift function (univariate). Must return a vector of the same length as x.
b2	second derivative of the drift function (univariate). Must return a vector of the same length as x.
circular	flag to indicate circular data.
maxK	maximum absolute winding number used if circular = TRUE.
vmApprox	flag to indicate von Mises approximation to wrapped normal. See momentMatchWnVm and scoreMatchWnBvm.
twokpi	optional matrix of winding numbers to avoid its recomputation. See details.
...	additional parameters passed to b, b1, b2, jac.b and sigma2.

Details

See Section 3.2 in García-Portugués et al. (2019) for details. `"SO2"` implements Shoji and Ozai (1998)'s expansion with for `p = 1`. `"SO"` is the same expansion, for arbitrary `p`, but considering null second derivatives.

`twokpi` is `repRow(2 * pi * c(-maxK:maxK), n = n)` if `p = 1` and `as.matrix(do.call(what = expand.grid,args = rep(list(2 * pi * c(-maxK:maxK)), p)))` otherwise.

Value

Output from `mleOptimWrapper`.

References

García-Portugués, E., Sørensen, M., Mardia, K. V. and Hamelryck, T. (2019) Langevin diffusions on the torus: estimation and applications. *Statistics and Computing*, 29(2):1–22. doi:10.1007/s1122201797902

Shoji, I. and Ozaki, T. (1998) A statistical method of estimation and simulation for systems of stochastic differential equations. *Biometrika*, 85(1):240–243. doi:10.1093/biomet/85.1.240

Examples

```
# 1D
grid <- seq(-pi, pi, l = 501)[-501]
alpha <- 1
sigma <- 1
t <- 0.5
x0 <- pi/2
# manipulate::manipulate({

  # Drifts
  b <- function(x) driftWn1D(x = x, alpha = alpha, mu = 0, sigma = sigma)
  b1 <- function(x, h = 1e-4) {
    l <- length(x)
    res <- driftWn1D(x = c(x + h, x - h), alpha = alpha, mu = 0,
                     sigma = sigma)
    drop(res[1:l] - res[(l + 1):(2 * l)])/(2 * h)
  }
  b2 <- function(x, h = 1e-4) {
    l <- length(x)
    res <- driftWn1D(x = c(x + h, x, x - h), alpha = alpha, mu = 0,
                     sigma = sigma)
    drop(res[1:l] - 2 * res[(l + 1):(2 * l)] +
          res[(2 * l + 1):(3 * l)]) / (h^2)
  }

  # Squared diffusion
  sigma2 <- function(x) rep(sigma^2, length(x))

  # Plot
```

```
  plot(grid, dTpdPde1D(Mx = length(grid), x0 = x0, t = t, alpha = alpha,
                       mu = 0, sigma = sigma), type = "l",
       ylab = "Density", xlab = "", ylim = c(0, 0.75), lwd = 2)
  lines(grid, dTpdWou1D(x = grid, x0 = rep(x0, length(grid)), t = t,
                        alpha = alpha, mu = 0, sigma = sigma), col = 2)
  lines(grid, dPsTpd(x = grid, x0 = x0, t = t, method = "E", b = b,
                     b1 = b1, b2 = b2, sigma2 = sigma2), col = 3)
  lines(grid, dPsTpd(x = grid, x0 = x0, t = t, method = "SO", b = b,
                     b1 = b1, b2 = b2, sigma2 = sigma2), col = 4)
  lines(grid, dPsTpd(x = grid, x0 = x0, t = t, method = "SO2", b = b,
                     b1 = b1, b2 = b2, sigma2 = sigma2),
        col = 5)
  lines(grid, dPsTpd(x = grid, x0 = x0, t = t, method = "E", b = b,
                     b1 = b1, b2 = b2, sigma2 = sigma2, vmApprox = TRUE),
        col = 6)
  lines(grid, dPsTpd(x = grid, x0 = x0, t = t, method = "SO", b = b,
                     b1 = b1, b2 = b2, sigma2 = sigma2, vmApprox = TRUE),
        col = 7)
  lines(grid, dPsTpd(x = grid, x0 = x0, t = t, method = "SO2", b = b,
                     b1 = b1, b2 = b2, sigma2 = sigma2, vmApprox = TRUE),
        col = 8)
  legend("topright", legend = c("PDE", "WOU", "E", "SO1", "SO2", "EvM",
                                "SO1vM", "SO2vM"), lwd = 2, col = 1:8)

# }, x0 = manipulate::slider(-pi, pi, step = 0.1, initial = -pi),
# alpha = manipulate::slider(0.1, 5, step = 0.1, initial = 1),
# sigma = manipulate::slider(0.1, 5, step = 0.1, initial = 1),
# t = manipulate::slider(0.1, 5, step = 0.1, initial = 1))

# 2D
grid <- seq(-pi, pi, l = 76)[-76]
alpha1 <- 2
alpha2 <- 1
alpha3 <- 0.5
sig1 <- 1
sig2 <- 2
t <- 0.5
x01 <- pi/2
x02 <- -pi/2
# manipulate::manipulate({

  alpha <- c(alpha1, alpha2, alpha3)
  sigma <- c(sig1, sig2)
  x0 <- c(x01, x02)

  # Drifts
  b <- function(x) driftWn2D(x = x, A = alphaToA(alpha = alpha,
                                                 sigma = sigma),
                             mu = rep(0, 2), sigma = sigma)
  jac.b <- function(x, h = 1e-4) {
    l <- nrow(x)
    res <- driftWn2D(x = rbind(cbind(x[, 1] + h, x[, 2]),
                               cbind(x[, 1] - h, x[, 2]),
```

```
                              cbind(x[, 1], x[, 2] + h),
                              cbind(x[, 1], x[, 2] - h)),
                      A = alphaToA(alpha = alpha, sigma = sigma),
                      mu = rep(0, 2), sigma = sigma)
    cbind(res[1:l, ] - res[(l + 1):(2 * l), ],
          res[2 * l + 1:l, ] - res[2 * l + (l + 1):(2 * l), ]) / (2 * h)
  }

  # Squared diffusion
  sigma2 <- function(x) matrix(sigma^2, nrow = length(x) / 2L, ncol = 2)

  # Plot
  old_par <- par(mfrow = c(3, 2))
  plotSurface2D(grid, grid, z = dTpdPde2D(Mx = length(grid),
                                          My = length(grid), x0 = x0,
                                          t = t, alpha = alpha,
                                          mu = rep(0, 2), sigma = sigma),
                levels = seq(0, 1, l = 20), main = "Exact")
  plotSurface2D(grid, grid,
                f = function(x) drop(dTpdWou2D(x = x,
                                               x0 = repRow(x0, nrow(x)),
                                               t = t, alpha = alpha,
                                               mu = rep(0, 2),
                                               sigma = sigma)),
                levels = seq(0, 1, l = 20), fVect = TRUE, main = "WOU")
  plotSurface2D(grid, grid,
                f = function(x) dPsTpd(x = x, x0 = rbind(x0), t = t,
                                       method = "E", b = b, jac.b = jac.b,
                                       sigma2 = sigma2),
                levels = seq(0, 1, l = 20), fVect = TRUE, main = "E")
  plotSurface2D(grid, grid,
                f = function(x) dPsTpd(x = x, x0 = rbind(x0), t = t,
                                       method = "SO", b = b, jac.b = jac.b,
                                       sigma2 = sigma2),
                levels = seq(0, 1, l = 20), fVect = TRUE, main = "SO")
  plotSurface2D(grid, grid,
                f = function(x) dPsTpd(x = x, x0 = rbind(x0), t = t,
                                       method = "E", b = b, jac.b = jac.b,
                                       sigma2 = sigma2, vmApprox = TRUE),
                levels = seq(0, 1, l = 20), fVect = TRUE, main = "EvM")
  plotSurface2D(grid, grid,
                f = function(x) dPsTpd(x = x, x0 = rbind(x0), t = t,
                                       method = "SO", b = b, jac.b = jac.b,
                                       sigma2 = sigma2, vmApprox = TRUE),
                levels = seq(0, 1, l = 20), fVect = TRUE, main = "SOvM")
  par(old_par)

# }, x01 = manipulate::slider(-pi, pi, step = 0.1, initial = -pi),
# x02 = manipulate::slider(-pi, pi, step = 0.1, initial = -pi),
# alpha1 = manipulate::slider(0.1, 5, step = 0.1, initial = 1),
# alpha2 = manipulate::slider(0.1, 5, step = 0.1, initial = 1),
# alpha3 = manipulate::slider(-5, 5, step = 0.1, initial = 0),
# sig1 = manipulate::slider(0.1, 5, step = 0.1, initial = 1),
```

```
# sig2 = manipulate::slider(0.1, 5, step = 0.1, initial = 1),
# t = manipulate::slider(0.01, 5, step = 0.01, initial = 1))
```

`driftJp` *Drift for the JP diffusion*

Description

Drift for the Langevin diffusion associated to the Jones and Pewsey (JP) family of circular distributions.

Usage

```
driftJp(x, alpha, mu, psi)
```

Arguments

`x`	vector with the evaluation points for the drift.
`alpha`	strength of the drift.
`mu`	unconditional mean of the diffusion.
`psi`	shape parameter, see details.

Details

Particular interesting choices for the shape parameter are:

- `psi = -1`: gives the Wrapped Cauchy as stationary density.
- `psi = 0`: is the sinusoidal drift of the vM diffusion.
- `psi = 1`: gives the Cardioid as stationary density.

See Section 2.2.3 in García-Portugués et al. (2019) for details.

Value

A vector of the same length as `x` containing the drift.

References

García-Portugués, E., Sørensen, M., Mardia, K. V. and Hamelryck, T. (2019) Langevin diffusions on the torus: estimation and applications. *Statistics and Computing*, 29(2):1–22. doi:10.1007/s1122201797902

Jones, M. C. and Pewsey, A. (2005). A family of symmetric distributions on the circle. *Journal of the American Statistical Association*, 100(472):1422–1428. doi:10.1198/016214505000000286

Examples

```
x <- seq(-pi, pi, l = 200)
plot(x, x, type = "n", ylab = "drift")
for (i in 0:20) {
  lines(x, driftJp(x = x, alpha = 1, mu = 0, psi = -1 + 2 * i / 20),
        col = rainbow(21)[i + 1])
}
```

`driftMixIndVm` *Drift for the mivM diffusion*

Description

Drift for the Langevin diffusion associated to a mixture of `m` independent (multivariate) von Mises (mivM) of dimension `p`.

Usage

```
driftMixIndVm(x, A, M, sigma, p, expTrc = 30)
```

Arguments

`x`	matrix of size `c(n, p)` with the evaluation points for the drift.
`A`	matrix of size `c(m, p)` giving the strengths of the drifts.
`M`	matrix of size `c(m, p)` giving the means.
`sigma`	diffusion coefficient.
`p`	vector of length `m` giving the proportions. Must add to one.
`expTrc`	truncation for exponential: `exp(x)` with `x <= -expTrc` is set to zero. Defaults to `30`.

Details

`driftMixVm` is more efficient for the circular case. The diffusion matrix is $\sigma \boldsymbol{I}$. See Section 2.2.4 in García-Portugués et al. (2019) for details.

Value

A matrix of the same size as `x` containing the drift.

References

García-Portugués, E., Sørensen, M., Mardia, K. V. and Hamelryck, T. (2019) Langevin diffusions on the torus: estimation and applications. *Statistics and Computing*, 29(2):1–22. doi:10.1007/s1122201797902

Examples

```
# 1D
x <- seq(-pi, pi, l = 200)
plot(x, x, type = "n", ylab = "drift")
for (i in 1:10) {
  lines(x, driftMixIndVm(x = cbind(x), A = cbind(c(2, 2)),
        M = cbind(c(0, -pi + 2 * pi * i / 10)), sigma = 1, p = c(0.5, 0.5)),
        col = rainbow(10)[i])
}

# 2D
x <- seq(-pi, pi, l = 100)
plotSurface2D(x, x, f = function(x) sqrt(rowSums(driftMixIndVm(x = x,
              A = rbind(c(1, 1), c(1, 1)), M = rbind(c(1, 1), c(-1, -1)),
              sigma = 1, p = c(0.25, 0.75))^2)), fVect = TRUE)
```

`driftMixVm` *Drift for the mivM diffusion (circular case)*

Description

Drift for the Langevin diffusion associated to a mixture of `m` independent von Mises (mivM) of dimension one.

Usage

```
driftMixVm(x, alpha, mu, sigma, p, expTrc = 30)
```

Arguments

`x`	vector with the evaluation points for the drift.
`alpha`	vector of length `m` giving the strengths of the drifts.
`mu`	vector of length `m` giving the means.
`sigma`	diffusion coefficient.
`p`	vector of length `m` giving the proportions. Must add to one.
`expTrc`	truncation for exponential: `exp(x)` with `x <= -expTrc` is set to zero. Defaults to `30`.

Details

`driftMixIndVm` is more general, but less efficient for the circular case. See Section 2.2.4 in García-Portugués et al. (2019) for details.

Value

A vector of the same length as `x` containing the drift.

References

García-Portugués, E., Sørensen, M., Mardia, K. V. and Hamelryck, T. (2019) Langevin diffusions on the torus: estimation and applications. *Statistics and Computing*, 29(2):1–22. doi:10.1007/s1122201797902

Examples

```
x <- seq(-pi, pi, l = 200)
plot(x, x, type = "n", ylab = "drift")
for (i in 1:10) {
  lines(x, driftMixVm(x = x, alpha = c(2, 2),
                      mu = c(0, -pi + 2 * pi * i / 10),
                      sigma = 1, p = c(0.5, 0.5)), col = rainbow(10)[i])
}
```

`driftMvm` *Drift for the MvM diffusion*

Description

Drift for the Langevin diffusion associated to the Multivariate von Mises (MvM) in dimension `p`.

Usage

```
driftMvm(x, alpha, mu, A = 0)
```

Arguments

`x`	matrix of size `c(n, p)` with the evaluation points for the drift.
`alpha`	vector of length `p` with the strength of the drift in the diagonal (sin terms).
`mu`	vector of length `p` with the unconditional mean of the diffusion.
`A`	matrix of size `c(p, p)` with the strength of the drift in cross terms (cos-sin terms). The diagonal has to be zero.

Details

See Section 2.2.1 in García-Portugués et al. (2019) for details.

Value

A matrix of the same size as `x` containing the drift.

References

García-Portugués, E., Sørensen, M., Mardia, K. V. and Hamelryck, T. (2019) Langevin diffusions on the torus: estimation and applications. *Statistics and Computing*, 29(2):1–22. doi:10.1007/s1122201797902

Examples

```
# 1D
x <- seq(-pi, pi, l = 200)
plot(x, x, type = "n", ylab = "drift")
for (i in 0:20) {
  lines(x, driftMvm(x = x, alpha = 3 * i / 20, mu = 0, A = 0),
        col = rainbow(21)[i + 1])
}

# 2D
x <- seq(-pi, pi, l = 100)
plotSurface2D(x, x, f = function(x) sqrt(rowSums(driftMvm(x = x,
              alpha = c(2, 2), mu = c(-1, -1),
              A = rbind(c(0, 0), c(0, 0)))^2)),
              fVect = TRUE)
```

`driftWn` *Drift for the WN diffusion*

Description

Drift for the Langevin diffusion associated to the (multivariate) Wrapped Normal (WN) in dimension p.

Usage

```
driftWn(x, A, mu, Sigma, invSigmaA = NULL, maxK = 2, expTrc = 30)
```

Arguments

`x`	matrix of size `c(n, p)` with the evaluation points for the drift.
`A`	matrix of size `c(p, p)` giving the drift strength.
`mu`	vector of length p with the unconditional mean of the diffusion.
`Sigma`	diffusion matrix, of size `c(p, p)`.
`invSigmaA`	the matrix `solve(Sigma) %*% A` (optional).
`maxK`	maximum absolute value of the windings considered in the computation of the WN.
`expTrc`	truncation for exponential: `exp(x)` with `x <= -expTrc` is set to zero. Defaults to `30`.

Details

See Section 2.2.2 in García-Portugués et al. (2019) for details.

`driftWn1D` and `driftWn2D` are more efficient for the 1D and 2D cases.

Value

A matrix of the same size as x containing the drift.

References

García-Portugués, E., Sørensen, M., Mardia, K. V. and Hamelryck, T. (2019) Langevin diffusions on the torus: estimation and applications. *Statistics and Computing*, 29(2):1–22. doi:10.1007/s1122201797902

Examples

```
# 1D
x <- seq(-pi, pi, l = 200)
plot(x, x, type = "n", ylab = "drift")
for (i in 1:20) {
  lines(x, driftWn(x = cbind(x), A = 1 * i / 20, mu = 0, Sigma = 1),
        col = rainbow(20)[i])
}

# 2D
x <- seq(-pi, pi, l = 100)
plotSurface2D(x, x, f = function(x) sqrt(rowSums(
              driftWn(x = x, A = alphaToA(alpha = c(1, 1, 0.5),
                                          sigma = c(1.5, 1.5)), mu = c(0, 0),
              Sigma = diag(c(1.5^2, 1.5^2)))^2)), fVect = TRUE)
```

---

driftWn1D *Drift of the WN diffusion in 1D*

Description

Computes the drift of the WN diffusion in 1D in a vectorized way.

Usage

```
driftWn1D(x, alpha, mu, sigma, maxK = 2L, expTrc = 30)
```

Arguments

x	a vector of length n containing angles. They all must be in $[\pi, \pi)$ so that the truncated wrapping by maxK windings is able to capture periodicity.
alpha	drift parameter.
mu	mean parameter. Must be in $[\pi, \pi)$.
sigma	diffusion coefficient.
maxK	maximum absolute value of the windings considered in the computation of the WN.
expTrc	truncation for exponential: exp(x) with x <= -expTrc is set to zero. Defaults to 30.

Value

A vector of length n containing the drift evaluated at x.

Examples

```
driftWn1D(x = seq(0, pi, l = 10), alpha = 1, mu = 0, sigma = 1, maxK = 2,
          expTrc = 30)
```

`driftWn2D` *Drift of the WN diffusion in 2D*

Description

Computes the drift of the WN diffusion in 2D in a vectorized way.

Usage

```
driftWn2D(x, A, mu, sigma, rho = 0, maxK = 2L, expTrc = 30)
```

Arguments

x	a matrix of dimension c(n, 2) containing angles. They all must be in $[\pi, \pi)$ so that the truncated wrapping by maxK windings is able to capture periodicity.
A	drift matrix of size c(2, 2).
mu	a vector of length 2 giving the mean.
sigma	vector of length 2 containing the **square root** of the diagonal of Σ, the diffusion matrix.
rho	correlation coefficient of Σ.
maxK	maximum absolute value of the windings considered in the computation of the WN.
expTrc	truncation for exponential: exp(x) with x <= -expTrc is set to zero. Defaults to 30.

Value

A matrix of size c(n, 2) containing the drift evaluated at x.

Examples

```
alpha <- 3:1
mu <- c(0, 0)
sigma <- 1:2
rho <- 0.5
Sigma <- diag(sigma^2)
Sigma[1, 2] <- Sigma[2, 1] <- rho * prod(sigma)
A <- alphaToA(alpha = alpha, sigma = sigma, rho = rho)
```

```
x <- rbind(c(0, 1), c(1, 0.1), c(pi, pi), c(-pi, -pi), c(pi / 2, 0))
driftWn2D(x = x, A = A, mu = mu, sigma = sigma, rho = rho)
driftWn(x = x, A = A, mu = c(0, 0), Sigma = Sigma)
```

---

dStatWn2D *Stationary density of a WN diffusion (with diagonal diffusion matrix) in 2D*

Description

Stationary density of the WN diffusion.

Usage

```
dStatWn2D(x, alpha, mu, sigma, rho = 0, maxK = 2L, expTrc = 30)
```

Arguments

x	a matrix of dimension c(n, 2) containing angles. They all must be in $[\pi, \pi)$ so that the truncated wrapping by maxK windings is able to capture periodicity.
alpha	vector of length 3 parametrizing the A matrix as in alphaToA.
mu	a vector of length 2 giving the mean.
sigma	vector of length 2 containing the **square root** of the diagonal of Σ, the diffusion matrix.
rho	correlation coefficient of Σ.
maxK	maximum absolute value of the windings considered in the computation of the WN.
expTrc	truncation for exponential: exp(x) with x <= -expTrc is set to zero. Defaults to 30.

Value

A vector of size n containing the stationary density evaluated at x.

Examples

```
set.seed(345567)
alpha <- c(2, 1, -1)
sigma <- c(1.5, 2)
Sigma <- diag(sigma^2)
A <- alphaToA(alpha = alpha, sigma = sigma)
mu <- c(pi, pi)
dStatWn2D(x = toPiInt(matrix(1:20, nrow = 10, ncol = 2)), mu = mu,
          alpha = alpha, sigma = sigma)
dTpdWou(t = 10, x = toPiInt(matrix(1:20, nrow = 10, ncol = 2)), A = A,
         mu = mu, Sigma = Sigma, x0 = mu)
xth <- seq(-pi, pi, l = 100)
```

```
contour(xth, xth, matrix(dStatWn2D(x = as.matrix(expand.grid(xth, xth)),
                                   alpha = alpha, sigma = sigma, mu = mu),
                         nrow = length(xth), ncol = length(xth)), nlevels = 50)
points(rStatWn2D(n = 1000, mu = mu, alpha = alpha, sigma = sigma), col = 2)
```

---

dTpdMou *Transition probability density of the multivariate OU diffusion*

Description

Transition probability density of the *multivariate* Ornstein–Uhlenbeck (OU) diffusion

$$dX_t = A(\mu - X_t)dt + \Sigma^{\frac{1}{2}} dW_t, X_0 = x_0.$$

Usage

```
dTpdMou(x, x0, t, A, mu, Sigma, eigA = NULL, log = FALSE)

meantMou(t, x0, A, mu, eigA = NULL)

covtMou(t, A, Sigma, eigA = NULL)
```

Arguments

x	matrix of with p columns containing the evaluation points.
x0	initial point.
t	time between observations.
A	the drift matrix, of size c(p, p).
mu	unconditional mean of the diffusion.
Sigma	square of the diffusion matrix, a matrix of size c(p, p).
eigA	optional argument containing eigen(A) for reuse.
log	flag to indicate whether to compute the logarithm of the density.

Details

The transition probability density is a multivariate normal with mean meantMou and covariance covtMou. See dTpdOu for the univariate case (more efficient).

solve(A) %*% Sigma has to be a covariance matrix (symmetric and positive definite) in order to have a proper transition density. For the bivariate case, this can be ensured with the alphaToA function. In the multivariate case, it is ensured if Sigma is isotropic and A is a covariance matrix.

Value

A matrix of the same size as x containing the evaluation of the density.

Examples

```
x <- seq(-4, 4, by = 0.1)
xx <- as.matrix(expand.grid(x, x))
isRStudio <- identical(.Platform$GUI, "RStudio")
if (isRStudio) {
  manipulate::manipulate(
    image(x, x, matrix(dTpdMou(x = xx, x0 = c(1, 2), t = t,
                               A = alphaToA(alpha = c(1, 2, 0.5),
                                            sigma = 1:2),
                               mu = c(0, 0), Sigma = diag((1:2)^2)),
                       nrow = length(x), ncol = length(x)),
          zlim = c(0, 0.25)), t = manipulate::slider(0.1, 5, step = 0.1))
}
```

---

dTpdOu *Transition probability density of the univariate OU diffusion*

Description

Transition probability density of the *univariate* Ornstein–Uhlenbeck (OU) diffusion

$$dX_t = \alpha(\mu - X_t)dt + \sigma dW_t, X_0 = x_0.$$

Usage

```
dTpdOu(x, x0, t, alpha, mu, sigma, log = FALSE)

meantOu(x0, t, alpha, mu)

vartOu(t, alpha, sigma)

covstOu(s, t, alpha, sigma)
```

Arguments

x	vector with the evaluation points.
x0	initial point.
t, s	time between observations.
alpha	strength of the drift.
mu	unconditional mean of the diffusion.
sigma	diffusion coefficient.
log	flag to indicate whether to compute the logarithm of the density.

Details

The transition probability density is a normal density with mean meantOu and variance vartOu. See dTpdMou for the multivariate case (less efficient for dimension one).

Value

A vector of the same length as x containing the evaluation of the density.

Examples

```
x <- seq(-4, 4, by = 0.01)
plot(x, dTpdOu(x = x, x0 = 3, t = 0.1, alpha = 1, mu = -1, sigma = 1),
     type = "l", ylim = c(0, 1.5), xlab = "x", ylab = "Density",
     col = rainbow(20)[1])
for (i in 2:20) {
  lines(x, dTpdOu(x = x, x0 = 3, t = i / 10, alpha = 1, mu = -1, sigma = 1),
        col = rainbow(20)[i])
}
```

---

dTpdPde1D *Transition probability density in 1D by PDE solving*

Description

Computation of the transition probability density (tpd) of the Wrapped Normal (WN) or von Mises (vM) diffusion, by solving its associated Fokker–Planck Partial Differential Equation (PDE) in 1D.

Usage

```
dTpdPde1D(Mx = 500, x0, t, alpha, mu, sigma, type = "WN",
  Mt = ceiling(100 * t), sdInitial = 0.1, ...)
```

Arguments

Mx	size of the equispaced spatial grid in $[-\pi, \pi)$.
x0	point giving the mean of the initial circular density, a WN with standard deviation equal to sdInitial.
t	time separating x0 and the evaluation of the tpd.
alpha	drift parameter.
mu	mean parameter. Must be in $[\pi, \pi)$.
sigma	diffusion coefficient.
type	either "WN" or "vM".
Mt	size of the time grid in $[0, t]$.
sdInitial	the standard deviation of the concentrated WN giving the initial condition.
...	Further parameters passed to crankNicolson1D.

Details

A combination of small sdInitial and coarse space-time discretization (small Mx and Mt) is prone to create numerical instabilities. See Sections 3.4.1, 2.2.1 and 2.2.2 in García-Portugués et al. (2019) for details.

Value

A vector of length `Mx` with the tpd evaluated at `seq(-pi, pi, l = Mx + 1)[-(Mx + 1)]`.

References

García-Portugués, E., Sørensen, M., Mardia, K. V. and Hamelryck, T. (2019) Langevin diffusions on the torus: estimation and applications. *Statistics and Computing*, 29(2):1–22. doi:10.1007/s1122201797902

Examples

```
Mx <- 100
x <- seq(-pi, pi, l = Mx + 1)[-c(Mx + 1)]
x0 <- pi
t <- 0.5
alpha <- 1
mu <- 0
sigma <- 1
isRStudio <- identical(.Platform$GUI, "RStudio")
if (isRStudio) {
  manipulate::manipulate({
  plot(x, dTpdPde1D(Mx = Mx, x0 = x0, t = t, alpha = alpha, mu = 0,
                    sigma = sigma), type = "l", ylab = "Density",
       xlab = "", ylim = c(0, 0.75))
  lines(x, dTpdWou1D(x = x, x0 = rep(x0, Mx), t = t, alpha = alpha, mu = 0,
                     sigma = sigma), col = 2)
  }, x0 = manipulate::slider(-pi, pi, step = 0.01, initial = 0),
  alpha = manipulate::slider(0.01, 5, step = 0.01, initial = 1),
  sigma = manipulate::slider(0.01, 5, step = 0.01, initial = 1),
  t = manipulate::slider(0.01, 5, step = 0.01, initial = 1))
}
```

`dTpdPde2D` *Transition probability density in 2D by PDE solving*

Description

Computation of the transition probability density (tpd) of the Wrapped Normal (WN) or Multivariate von Mises (MvM) diffusion, by solving its associated Fokker–Planck Partial Differential Equation (PDE) in 2D.

Usage

```
dTpdPde2D(Mx = 50, My = 50, x0, t, alpha, mu, sigma, rho = 0,
  type = "WN", Mt = ceiling(100 * t), sdInitial = 0.1, ...)
```

Arguments

Mx, My	sizes of the equispaced spatial grids in $[-\pi, \pi)$ for each component.
x0	point giving the mean of the initial circular density, an isotropic WN with standard deviations equal to sdInitial.
t	time separating x0 and the evaluation of the tpd.
alpha	for "WN", a vector of length 3 parametrizing the A matrix as in alphaToA. For "vM", a vector of length 3 containing c(alpha[1:2], A[1, 2]), from the arguments alpha and A in driftMvm.
mu	vector of length 2 giving the mean.
sigma	for "WN", a vector of length 2 containing the **square root** of the diagonal of the diffusion matrix. For "vM", the standard deviation giving the isotropic diffusion matrix.
rho	for "WN", the correlation of the diffusion matrix.
type	either "WN" or "vM".
Mt	size of the time grid in $[0, t]$.
sdInitial	standard deviations of the concentrated WN giving the initial condition.
...	Further parameters passed to crankNicolson2D.

Details

A combination of small sdInitial and coarse space-time discretization (small Mx and Mt) is prone to create numerical instabilities. See Sections 3.4.2, 2.2.1 and 2.2.2 in García-Portugués et al. (2019) for details.

Value

A matrix of size c(Mx, My) with the tpd evaluated at the combinations of seq(-pi, pi, l = Mx + 1)[-(Mx + 1)] and seq(-pi, pi, l = My + 1)[-(My + 1)].

References

García-Portugués, E., Sørensen, M., Mardia, K. V. and Hamelryck, T. (2019) Langevin diffusions on the torus: estimation and applications. *Statistics and Computing*, 29(2):1–22. doi:10.1007/s1122201797902

Examples

```
M <- 100
x <- seq(-pi, pi, l = M + 1)[-c(M + 1)]
image(x, x, dTpdPde2D(Mx = M, My = M, x0 = c(0, pi), t = 1,
                      alpha = c(1, 1, 0.5), mu = c(pi / 2, 0), sigma = 1:2),
      zlim = c(0, 0.25), col = matlab.like.colorRamps(20),
      xlab = "x", ylab = "y")
```

dTpdWou *Conditional probability density of the WOU process*

Description

Conditional probability density of the Wrapped Ornstein–Uhlenbeck (WOU) process.

Usage

```
dTpdWou(x, t, A, mu, Sigma, x0, maxK = 2, eigA = NULL, invASigma = NULL)
```

Arguments

x	matrix of size c(n, p) with the evaluation points in $[-\pi, \pi)^p$.
t	a scalar containing the times separating x and x0.
A	matrix of size c(p, p) giving the drift strength.
mu	mean parameter. Must be in $[\pi, \pi)$.
Sigma	diffusion matrix, of size c(p, p).
x0	vector of length p with the initial point in $[-\pi, \pi)^p$.
maxK	maximum absolute value of the windings considered in the computation of the WN.
eigA	optional argument containing eigen(A) for reuse.
invASigma	the matrix solve(Sigma) %*% A (optional).

Details

See Section 3.3 in García-Portugués et al. (2019) for details. dTpdWou1D and dTpdWou2D are more efficient implementations for the 1D and 2D cases, respectively.

Value

A vector of length n with the density evaluated at x.

References

García-Portugués, E., Sørensen, M., Mardia, K. V. and Hamelryck, T. (2019) Langevin diffusions on the torus: estimation and applications. *Statistics and Computing*, 29(2):1–22. doi:10.1007/s1122201797902

Examples

```
# 1D
t <- 0.5
alpha <- 1
mu <- 0
sigma <- 1
x0 <- pi
x <- seq(-pi, pi, l = 10)
dTpdWou(x = cbind(x), x0 = x0, t = t, A = alpha, mu = 0, Sigma = sigma^2) -
dTpdWou1D(x = cbind(x), x0 = rep(x0, 10), t = t, alpha = alpha, mu = 0,
          sigma = sigma)

# 2D
t <- 0.5
alpha <- c(2, 1, -1)
sigma <- c(1.5, 2)
rho <- 0.9
Sigma <- diag(sigma^2)
Sigma[1, 2] <- Sigma[2, 1] <- rho * prod(sigma)
A <- alphaToA(alpha = alpha, sigma = sigma, rho = rho)
mu <- c(pi, 0)
x0 <- c(0, 0)
x <- seq(-pi, pi, l = 5)
x <- as.matrix(expand.grid(x, x))
dTpdWou(x = x, x0 = x0, t = t, A = A, mu = mu, Sigma = Sigma) -
dTpdWou2D(x = x, x0 = rbind(x0), t = t, alpha = alpha, mu = mu,
          sigma = sigma, rho = rho)
```

---

dTpdWou1D *Approximation of the transition probability density of the WN diffusion in 1D*

Description

Computation of the transition probability density (tpd) for a WN diffusion.

Usage

```
dTpdWou1D(x, x0, t, alpha, mu, sigma, maxK = 2L, expTrc = 30,
  vmApprox = 0L, kt = 0, logConstKt = 0)
```

Arguments

x	a vector of length n containing angles. They all must be in $[\pi, \pi)$ so that the truncated wrapping by maxK windings is able to capture periodicity.
x0	a vector of length n containing the starting angles. They all must be in $[\pi, \pi)$.
t	a scalar containing the times separating x and x0.
alpha	drift parameter.

mu	mean parameter. Must be in $[\pi, \pi)$.
sigma	diffusion coefficient.
maxK	maximum absolute value of the windings considered in the computation of the WN.
expTrc	truncation for exponential: exp(x) with x <= -expTrc is set to zero. Defaults to 30.
vmApprox	whether to use the von Mises approximation to a wrapped normal (1) or not (0, default).
kt	concentration for the von Mises, a suitable output from momentMatchWnVm (see examples).
logConstKt	the logarithm of the von Mises normalizing constant associated to the concentration kt (see examples)

Details

See Section 3.3 in García-Portugués et al. (2019) for details. See dTpdWou for the general case (less efficient for 2D).

Value

A vector of size n containing the tpd evaluated at x.

References

García-Portugués, E., Sørensen, M., Mardia, K. V. and Hamelryck, T. (2019) Langevin diffusions on the torus: estimation and applications. *Statistics and Computing*, 29(2):1–22. doi:10.1007/s1122201797902

Examples

```
t <- 0.5
alpha <- 1
mu <- 0
sigma <- 1
x0 <- 0.1
dTpdWou1D(x = seq(-pi, pi, l = 10), x0 = rep(x0, 10), t = t, alpha = alpha,
          mu = mu, sigma = sigma, vmApprox = 0)

# von Mises approximation
kt <- scoreMatchWnVm(sigma2 = sigma^2 * (1 - exp(-2 * alpha * t)) / (2 * alpha))
dTpdWou1D(x = seq(-pi, pi, l = 10), x0 = rep(x0, 10), t = t, alpha = alpha,
          mu = mu, sigma = sigma, vmApprox = 1, kt = kt,
          logConstKt = -log(2 * pi * besselI(x = kt, nu = 0,
                                             expon.scaled = TRUE)))
```

dTpdWou2D *Approximation of the transition probability density of the WN diffusion in 2D*

Description

Computation of the transition probability density (tpd) for a WN diffusion (with diagonal diffusion matrix)

Usage

```
dTpdWou2D(x, x0, t, alpha, mu, sigma, rho = 0, maxK = 2L, expTrc = 30)
```

Arguments

x	a matrix of dimension c(n, 2) containing angles. They all must be in $[\pi, \pi)$ so that the truncated wrapping by maxK windings is able to capture periodicity.
x0	a matrix of dimension c(n, 2) containing the starting angles. They all must be in $[\pi, \pi)$. If all x0 are the same, a matrix of dimension c(1, 2) can be passed for better performance.
t	a scalar containing the times separating x and x0.
alpha	vector of length 3 parametrizing the A matrix as in alphaToA.
mu	a vector of length 2 giving the mean.
sigma	vector of length 2 containing the **square root** of the diagonal of Σ, the diffusion matrix.
rho	correlation coefficient of Σ.
maxK	maximum absolute value of the windings considered in the computation of the WN.
expTrc	truncation for exponential: exp(x) with x <= -expTrc is set to zero. Defaults to 30.

Details

The function checks for positive definiteness. If violated, it resets A such that solve(A) %*% Sigma is positive definite.

See Section 3.3 in García-Portugués et al. (2019) for details. See dTpdWou for the general case (less efficient for 1D).

Value

A vector of size n containing the tpd evaluated at x.

References

García-Portugués, E., Sørensen, M., Mardia, K. V. and Hamelryck, T. (2019) Langevin diffusions on the torus: estimation and applications. *Statistics and Computing*, 29(2):1–22. doi:10.1007/s1122201797902

Examples

```
set.seed(3455267)
alpha <- c(2, 1, -1)
sigma <- c(1.5, 2)
rho <- 0.9
Sigma <- diag(sigma^2)
Sigma[1, 2] <- Sigma[2, 1] <- rho * prod(sigma)
A <- alphaToA(alpha = alpha, sigma = sigma, rho = rho)
solve(A) %*% Sigma
mu <- c(pi, 0)
x <- t(euler2D(x0 = matrix(c(0, 0), nrow = 1), A = A, mu = mu,
               sigma = sigma, N = 500, delta = 0.1)[1, , ])

sum(sapply(1:49, function(i) log(dTpdWou(x = matrix(x[i + 1, ], ncol = 2),
                                         x0 = x[i, ], t = 1.5, A = A,
                                         Sigma = Sigma, mu = mu))))

sum(log(dTpdWou2D(x = matrix(x[2:50, ], ncol = 2),
                  x0 = matrix(x[1:49, ], ncol = 2), t = 1.5, alpha = alpha,
                  mu = mu, sigma = sigma, rho = rho)))

lgrid <- 56
grid <- seq(-pi, pi, l = lgrid + 1)[-(lgrid + 1)]
image(grid, grid, matrix(dTpdWou(x = as.matrix(expand.grid(grid, grid)),
                                 x0 = c(0, 0), t = 0.5, A = A,
                                 Sigma = Sigma, mu = mu),
                         nrow = 56, ncol = 56), zlim = c(0, 0.25),
      main = "dTpdWou")
image(grid, grid, matrix(dTpdWou2D(x = as.matrix(expand.grid(grid, grid)),
                                   x0 = matrix(0, nrow = 56^2, ncol = 2),
                                   t = 0.5, alpha = alpha, sigma = sigma,
                                   mu = mu),
                         nrow = 56, ncol = 56), zlim = c(0, 0.25),
      main = "dTpdWou2D")

x <- seq(-pi, pi, l = 100)
t <- 1
image(x, x, matrix(dTpdWou2D(x = as.matrix(expand.grid(x, x)),
                             x0 = matrix(rep(0, 100 * 2), nrow = 100 * 100,
                                         ncol = 2),
                             t = t, alpha = alpha, mu = mu, sigma = sigma,
                             maxK = 2, expTrc = 30),
                             nrow = 100, ncol = 100),
      zlim = c(0, 0.25))
points(stepAheadWn2D(x0 = rbind(c(0, 0)), delta = t / 500,
                     A = alphaToA(alpha = alpha, sigma = sigma), mu = mu,
```

```
sigma = sigma, N = 500, M = 1000, maxK = 2,
expTrc = 30))
```

---

dVm *Density of the von Mises*

Description

Computes the density of a von Mises in a numerically stable way.

Usage

```
dVm(x, mu, kappa)
```

Arguments

x	evaluation angles, not necessary in $[\pi, \pi)$.
mu	circular mean.
kappa	non-negative concentration parameter.

Value

A vector of the same length as x containing the density.

References

Jammalamadaka, S. R. and SenGupta, A. (2001) *Topics in Circular Statistics*. World Scientific, Singapore. doi:10.1142/4031

Examples

```
x <- seq(-pi, pi, l = 200)
plot(x, x, type = "n", ylab = "Density", ylim = c(0, 1))
for (i in 0:20) {
  lines(x, dVm(x = x, mu = 0, kappa = 5 * i / 20),
        col = rainbow(21)[i + 1])
}
```

dWn1D *WN density in 1D*

Description

Computation of the WN density in 1D.

Usage

```
dWn1D(x, mu, sigma, maxK = 2L, expTrc = 30, vmApprox = 0L, kt = 0,
  logConstKt = 0)
```

Arguments

x	a vector of length n containing angles. They all must be in $[\pi, \pi)$ so that the truncated wrapping by maxK windings is able to capture periodicity.
mu	mean parameter. Must be in $[\pi, \pi)$.
sigma	diffusion coefficient.
maxK	maximum absolute value of the windings considered in the computation of the WN.
expTrc	truncation for exponential: exp(x) with x <= -expTrc is set to zero. Defaults to 30.
vmApprox	whether to use the von Mises approximation to a wrapped normal (1) or not (0, default).
kt	concentration for the von Mises, a suitable output from momentMatchWnVm (see examples).
logConstKt	the logarithm of the von Mises normalizing constant associated to the concentration kt (see examples)

Value

A vector of size n containing the density evaluated at x.

Examples

```
mu <- 0
sigma <- 1
dWn1D(x = seq(-pi, pi, l = 10), mu = mu, sigma = sigma, vmApprox = 0)

# von Mises approximation
kt <- scoreMatchWnVm(sigma2 = sigma^2)
dWn1D(x = seq(-pi, pi, l = 10), mu = mu, sigma = sigma, vmApprox = 1, kt = kt,
      logConstKt = -log(2 * pi * besselI(x = kt, nu = 0, expon.scaled = TRUE)))
```

euler1D *Simulation of trajectories of the WN or vM diffusion in 1D*

Description

Simulation of the Wrapped Normal (WN) diffusion or von Mises (vM) diffusion by the Euler method in 1D, for several starting values.

Usage

```
euler1D(x0, alpha, mu, sigma, N = 100L, delta = 0.01, type = 1L,
  maxK = 2L, expTrc = 30)
```

Arguments

x0	vector of length nx0 giving the initial points.
alpha	drift parameter.
mu	mean parameter. Must be in $[\pi, \pi)$.
sigma	diffusion coefficient.
N	number of discretization steps.
delta	discretization step.
type	integer giving the type of diffusion. Currently, only 1 for WN and 2 for vM are supported.
maxK	maximum absolute value of the windings considered in the computation of the WN.
expTrc	truncation for exponential: exp(x) with x <= -expTrc is set to zero. Defaults to 30.

Value

A matrix of size c(nx0, N + 1) containing the nx0 discretized trajectories. The first column corresponds to the vector x0.

Examples

```
N <- 100
nx0 <- 20
x0 <- seq(-pi, pi, l = nx0 + 1)[-(nx0 + 1)]
set.seed(12345678)
samp <- euler1D(x0 = x0, mu = 0, alpha = 3, sigma = 1, N = N,
                delta = 0.01, type = 2)
tt <- seq(0, 1, l = N + 1)
plot(rep(0, nx0), x0, pch = 16, col = rainbow(nx0), xlim = c(0, 1))
for (i in 1:nx0) linesCirc(tt, samp[i, ], col = rainbow(nx0)[i])
```

`euler2D` *Simulation of trajectories of the WN or MvM diffusion in 2D*

Description

Simulation of the Wrapped Normal (WN) diffusion or Multivariate von Mises (MvM) diffusion by the Euler method in 2D, for several starting values.

Usage

```
euler2D(x0, A, mu, sigma, rho = 0, N = 100L, delta = 0.01, type = 1L,
  maxK = 2L, expTrc = 30)
```

Arguments

`x0`	matrix of size `c(nx0, 2)` giving the initial points.
`A`	drift matrix of size `c(2, 2)`.
`mu`	a vector of length `2` giving the mean.
`sigma`	vector of length `2` containing the **square root** of the diagonal of Σ, the diffusion matrix.
`rho`	correlation coefficient of Σ.
`N`	number of discretization steps.
`delta`	discretization step.
`type`	integer giving the type of diffusion. Currently, only `1` for WN and `2` for vM are supported.
`maxK`	maximum absolute value of the windings considered in the computation of the WN.
`expTrc`	truncation for exponential: `exp(x)` with `x <= -expTrc` is set to zero. Defaults to `30`.

Value

An array of size `c(nx0, 2, N + 1)` containing the `nx0` discretized trajectories. The first slice corresponds to the matrix `x0`.

Examples

```
N <- 100
nx0 <- 5
x0 <- seq(-pi, pi, l = nx0 + 1)[-(nx0 + 1)]
x0 <- as.matrix(expand.grid(x0, x0))
nx0 <- nx0^2
set.seed(12345678)
samp <- euler2D(x0 = x0, mu = c(0, 0), A = rbind(c(3, 1), 1:2),
                sigma = c(1, 1), N = N, delta = 0.01, type = 2)
```

```
plot(x0[, 1], x0[, 2], xlim = c(-pi, pi), ylim = c(-pi, pi), pch = 16,
     col = rainbow(nx0))
for (i in 1:nx0) linesTorus(samp[i, 1, ], samp[i, 2, ],
                            col = rainbow(nx0, alpha = 0.5)[i])
```

`linesCirc` *Lines and arrows with vertical wrapping*

Description

Joins the corresponding points with line segments or arrows that exhibit wrapping in $[-\pi, \pi)$ in the vertical axis.

Usage

```
linesCirc(x = seq_along(y), y, col = 1, lty = 1, ltyCross = lty,
  arrows = FALSE, ...)
```

Arguments

`x`	vector with horizontal coordinates.
`y`	vector with vertical coordinates, wrapped in $[-\pi, \pi)$.
`col`	color vector of length 1 or the same length of `x` and `y`.
`lty`	line type as in `par`.
`ltyCross`	specific line type for crossing segments.
`arrows`	flag for drawing arrows instead of line segments.
`...`	further graphical parameters passed to `segments` or `arrows`.

Details

`y` is wrapped to $[-\pi, \pi)$ before plotting.

Value

Nothing. The functions are called for drawing wrapped lines.

Examples

```
x <- 1:100
y <- toPiInt(pi * cos(2 * pi * x / 100) + 0.5 * runif(50, -pi, pi))
plot(x, y, ylim = c(-pi, pi))
linesCirc(x = x, y = y, col = rainbow(length(x)), ltyCross = 2)
plot(x, y, ylim = c(-pi, pi))
linesCirc(x = x, y = y, col = rainbow(length(x)), arrows = TRUE)
```

`linesTorus` *Lines and arrows with wrapping in the torus*

Description

Joins the corresponding points with line segments or arrows that exhibit wrapping in $[-\pi, \pi)$ in the horizontal and vertical axes.

Usage

```
linesTorus(x, y, col = 1, lty = 1, ltyCross = lty, arrows = FALSE, ...)
```

Arguments

`x`	vector with horizontal coordinates, wrapped in $[-\pi, \pi)$.
`y`	vector with vertical coordinates, wrapped in $[-\pi, \pi)$.
`col`	color vector of length 1 or the same length of `x` and `y`.
`lty`	line type as in `par`.
`ltyCross`	specific line type for crossing segments.
`arrows`	flag for drawing arrows instead of line segments.
`...`	further graphical parameters passed to `segments` or `arrows`.

Details

`x` and `y` are wrapped to $[-\pi, \pi)$ before plotting.

Value

Nothing. The functions are called for drawing wrapped lines.

Examples

```
x <- toPiInt(rnorm(50, mean = seq(-pi, pi, l = 50), sd = 0.5))
y <- toPiInt(x + rnorm(50, mean = seq(-pi, pi, l = 50), sd = 0.5))
plot(x, y, xlim = c(-pi, pi), ylim = c(-pi, pi), col = rainbow(length(x)),
     pch = 19)
linesTorus(x = x, y = y, col = rainbow(length(x)), ltyCross = 2)
plot(x, y, xlim = c(-pi, pi), ylim = c(-pi, pi), col = rainbow(length(x)),
     pch = 19)
linesTorus(x = x, y = y, col = rainbow(length(x)), arrows = TRUE)
```

`linesTorus3d` *Lines and arrows with wrapping in the torus*

Description

Joins the corresponding points with line segments or arrows that exhibit wrapping in $[-\pi, \pi)$ in the horizontal and vertical axes.

Usage

```
linesTorus3d(x, y, z, col = 1, arrows = FALSE, ...)
```

Arguments

`x`, `y`	vectors with horizontal coordinates, wrapped in $[-\pi, \pi)$.
`z`	vector with vertical coordinates, wrapped in $[-\pi, \pi)$.
`col`	color vector of length 1 or the same length of `x`, `y`, and `z`.
`arrows`	flag for drawing arrows instead of line segments.
`...`	further graphical parameters passed to `segments` or `arrows`.

Details

`x`, `y`, and `z` are wrapped to $[-\pi, \pi)$ before plotting. `arrows = TRUE` makes sequential calls to `arrow3d`, and is substantially slower than `arrows = FALSE`.

Value

Nothing. The functions are called for drawing wrapped lines.

Examples

```
if (requireNamespace("rgl")) {
  n <- 20
  x <- toPiInt(rnorm(n, mean = seq(-pi, pi, l = n), sd = 0.5))
  y <- toPiInt(rnorm(n, mean = seq(-pi, pi, l = n), sd = 0.5))
  z <- toPiInt(x + y + rnorm(n, mean = seq(-pi, pi, l = n), sd = 0.5))
  rgl::plot3d(x, y, z, xlim = c(-pi, pi), ylim = c(-pi, pi),
              zlim = c(-pi, pi), col = rainbow(n), size = 2,
              box = FALSE, axes = FALSE)
  linesTorus3d(x = x, y = y, z = z, col = rainbow(n), lwd = 2)
  torusAxis3d()
  rgl::plot3d(x, y, z, xlim = c(-pi, pi), ylim = c(-pi, pi),
              zlim = c(-pi, pi), col = rainbow(n), size = 2,
              box = FALSE, axes = FALSE)
  linesTorus3d(x = x, y = y, z = z, col = rainbow(n), ltyCross = 2,
               arrows = TRUE, theta = 0.1 * pi / 180, barblen = 0.1)
  torusAxis3d()
}
```

`logBesselI0Scaled` *Efficient computation of Bessel related functions*

Description

Computation of $\log(I_0(x)) - x$ and the inverse of $A_1(k) = \frac{I_0(k)}{I_1(k)}$.

Usage

```
logBesselI0Scaled(x, splineApprox = TRUE)

a1Inv(x, splineApprox = TRUE)
```

Arguments

`x`	evaluation vector. For `logBesselI0Scaled`, `x` must contain non-negative values. For `a1Inv`, `x` must be in $[0, 1]$.
`splineApprox`	whether to use a pre-computed spline approximation (faster) or not.

Details

Both functions may rely on pre-computed spline interpolations (`logBesselI0ScaledSpline` and `a1InvSpline`). Otherwise, a call to `besselI` is done for $\log(I_0(x)) - x$ and $A_1(k) = x$ is solved numerically. The data in which the interpolation is based is given in the examples.

For `x` larger than `5e4`, the asymptotic expansion of `besselIasym` is employed.

Value

A vector of the same length as `x`.

Examples

```
# Data employed for log besselI0 scaled
x1 <- c(seq(0, 1, by = 1e-4), seq(1 + 1e-2, 10, by = 1e-3),
        seq(10 + 1e-1, 100, by = 1e-2), seq(100 + 1e0, 1e3, by = 1e0),
        seq(1000 + 1e1, 5e4, by = 2e1))
logBesselI0ScaledEvalGrid <- log(besselI(x = x1, nu = 0,
                                         expon.scaled = TRUE))
# save(list = "logBesselI0ScaledEvalGrid",
#      file = "logBesselI0ScaledEvalGrid.rda", compress = TRUE)

# Data employed for A1 inverse
x2 <- rev(c(seq(1e-04, 0.9 - 1e-4, by = 1e-4),
            seq(0.9, 1 - 1e-05, by = 1e-5)))
a1InvEvalGrid <- sapply(x2, function(k) {
  uniroot(f = function(x) k - besselI(x, nu = 1, expon.scaled = TRUE) /
          besselI(x, nu = 0, expon.scaled = TRUE),
          lower = 1e-06, upper = 1e+05, tol = 1e-15)$root
```

```
})
# save(list = "a1InvEvalGrid", file = "a1InvEvalGrid.rda", compress = TRUE)

# Accuracy logBesselI0Scaled
x <- seq(0, 1e3, l = 1e3)
summary(logBesselI0Scaled(x = x, splineApprox = TRUE) -
        logBesselI0Scaled(x = x, splineApprox = FALSE))

# Accuracy a1Inv
y <- seq(0, 1 - 1e-4, l = 1e3)
summary(a1Inv(x = y, splineApprox = TRUE) -
        a1Inv(x = y, splineApprox = FALSE))
```

`logLikWouPairs` *Loglikelihood of WN in 2D when only the initial and final points are observed*

Description

Computation of the loglikelihood for a WN diffusion (with diagonal diffusion matrix) from a sample of initial and final pairs of angles.

Usage

```
logLikWouPairs(x, t, alpha, mu, sigma, rho = 0, maxK = 2L, expTrc = 30)
```

Arguments

`x`	a matrix of dimension `c(n, 4)` of initial and final pairs of angles. Each row is an observation containing $(\phi_0, \psi_0, \phi_t, \psi_t)$. They all must be in $[\pi, \pi)$ so that the truncated wrapping by `maxK` windings is able to capture periodicity.
`t`	either a scalar or a vector of length `n` containing the times the initial and final dihedrals. If `t` is a scalar, a common time is assumed.
`alpha`	vector of length `3` parametrizing the `A` matrix as in `alphaToA`.
`mu`	a vector of length `2` giving the mean.
`sigma`	vector of length `2` containing the **square root** of the diagonal of Σ, the diffusion matrix.
`rho`	correlation coefficient of Σ.
`maxK`	maximum absolute value of the windings considered in the computation of the WN.
`expTrc`	truncation for exponential: `exp(x)` with `x <= -expTrc` is set to zero. Defaults to `30`.

Details

A negative penalty is added if positive definiteness is violated. If the output value is Inf, -100 * N is returned instead.

Value

A scalar giving the final loglikelihood, defined as the sum of the loglikelihood of the initial angles according to the stationary density and the loglikelihood of the transitions from initial to final angles.

Examples

```
set.seed(345567)
x <- toPiInt(matrix(rnorm(200, mean = pi), ncol = 4, nrow = 50))
alpha <- c(2, 1, -0.5)
mu <- c(0, pi)
sigma <- sqrt(c(2, 1))

# The same
logLikWouPairs(x = x, t = 0.5, alpha = alpha, mu = mu, sigma = sigma)
sum(
  log(dStatWn2D(x = x[, 1:2], alpha = alpha, mu = mu, sigma = sigma)) +
  log(dTpdWou2D(x = x[, 3:4], x0 = x[, 1:2], t = 0.5, alpha = alpha, mu = mu,
                sigma = sigma))
)

# Different times
logLikWouPairs(x = x, t = (1:50) / 50, alpha = alpha, mu = mu, sigma = sigma)
```

---

mleMou *Maximum likelihood estimation of the multivariate OU diffusion*

Description

Computation of the maximum likelihood estimator of the parameters of the *multivariate* Ornstein–Uhlenbeck (OU) diffusion from a discretized trajectory $\{X_{\Delta i}\}_{i=1}^{N}$. The objective function to minimize is

$$\sum_{i=2}^{n} \log p_{\Delta}(X_{\Delta i} | X_{\Delta(i-1)}).$$

Usage

```
mleMou(data, delta, alpha = rep(NA, 3), mu = rep(NA, 2), sigma = rep(NA,
  2), start, lower = c(0.01, 0.01, -25, -pi, -pi, 0.01, 0.01),
  upper = c(25, 25, 25, pi, pi, 25, 25), ...)
```

Arguments

`data`	a matrix of size `c(N, p)` with the discretized trajectory of the diffusion.
`delta`	time discretization step.
`alpha, mu, sigma`	arguments to fix a parameter to a given value and perform the estimation on the rest. Defaults to `NA`, meaning that the parameter is estimated. Note that `start`, `lower` and `upper` must be changed accordingly if parameters are fixed, see examples.

start	starting values, a matrix with p columns, with each entry representing a different starting value.
lower, upper	bound for box constraints as in method "L-BFGS-B" of optim.
...	further arguments to be passed to mleOptimWrapper.

Details

The first row in data is not taken into account for estimation. See mleOu for the univariate case (more efficient).

mleMou only handles p = 2 currently. It imposes that Sigma is diagonal and handles the parametrization of A by alphaToA.

Value

Output from mleOptimWrapper.

Examples

```
set.seed(345678)
data <- rTrajMou(x0 = c(0, 0), A = alphaToA(alpha = c(1, 1, 0.5),
                                            sigma = 1:2), mu = c(1, 1),
                 Sigma = diag((1:2)^2), N = 200, delta = 0.5)
mleMou(data = data, delta = 0.5, start = c(1, 1, 0, 1, 1, 1, 2),
       lower = c(0.1, 0.1, -25, -10, -10, 0.1, 0.1),
       upper = c(25, 25, 25, 10, 10, 25, 25), maxit = 500)

# Fixed sigma and mu
mleMou(data = data, delta = 0.5, mu = c(1, 1), sigma = 1:2,
       start = c(1, 1, 0), lower = c(0.1, 0.1, -25), upper = c(25, 25, 25))
```

---

mleOptimWrapper *Optimization wrapper for likelihood-based procedures*

Description

A convenient wrapper to perform local optimization of the likelihood function via nlm and optim including several practical utilities.

Usage

```
mleOptimWrapper(minusLogLik, region = function(pars) list(pars = pars,
  penalty = 0), penalty = 1e+10, optMethod = "Nelder-Mead", start,
  lower = rep(-Inf, ncol(start)), upper = rep(Inf, ncol(start)),
  selectSolution = "lowestLocMin", checkCircular = TRUE, maxit = 500,
  tol = 1e-05, verbose = 0, eigTol = 1e-04, condTol = 10000, ...)
```

Arguments

minusLogLik	function computing the minus log-likelihood function. Must have a single argument containing a vector of length p.
region	function to impose a feasibility region via a penalty. See details.
penalty	imposed penalty if value is not finite.
optMethod	one of the following strings: "nlm", "Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN", or "Brent".
start	starting values, a matrix with p columns, with each entry representing a different starting value.
lower, upper	bound for box constraints as in method "L-BFGS-B" of optim.
selectSolution	which criterion is used for selecting a solution among possible ones, either "lowest", "lowestConv" or "lowestLocMin". "lowest" returns the solution with lowest value in the minusLogLik function. "lowestConv" restricts the search of the best solution among the ones for which the optimizer has converged. "lowestLocMin" in addition imposes that the solution is guaranteed to be a local minimum by examining the Hessian matrix.
checkCircular	logical indicating whether to automatically treat the variables with lower and upper entries equal to -pi and pi as circular. See details.
maxit	maximum number of iterations.
tol	tolerance for convergence (passed to reltol, pgtol or gradtol).
verbose	an integer from 0 to 2 if optMethod = "Nelder-Mead" or from 0 to 4 otherwise giving the amount of information displayed.
eigTol, condTol	eigenvalue and condition number tolerance for the Hessian in order to guarantee a local minimum. Used only if selectSolution = "lowestLocMin".
...	further arguments passed to the optMethod selected. See options in nlm or optim.

Details

If checkCircular = TRUE, then the corresponding lower and upper entries of the circular parameters are set to -Inf and Inf, respectively, and minusLogLik is called with the *principal value* of the circular argument.

If no solution is found satisfying the criterion in selectSolution, NAs are returned in the elements of the main solution.

The Hessian is only computed if selectSolution = "lowestLocMin".

Region feasibility can be imposed by a function with the same arguments as minusLogLik that resets pars in to the boundary of the feasibility region and adds a penalty proportional to the violation of the feasibility region. Note that this is *not the best procedure at all* to solve the constrained optimization problem, but just a relatively flexible and quick approach (for a more advanced treatment of restrictions, see optimization-focused packages). The value must be a list with objects pars and penalty. By default no region is imposed, i.e., region = function(pars) list("pars" = pars, "penalty" = 0). Note that the Hessian is computed from the unconstrained problem, hence localMinimumGuaranteed might be FALSE even if a local minimum to the constrained problem was found.

Value

A list containing the following elements:

- par: estimated minimizing parameters
- value: value of minusLogLik at the minimum.
- convergence: if the optimizer has converged or not.
- message: a character string giving any additional information returned by the optimizer.
- eigHessian: eigenvalues of the Hessian at the minimum. Recall that for the same solution slightly different outputs may be obtained according to the different computations of the Hessian of nlm and optim.
- localMinimumGuaranteed: tests if the Hessian is positive definite (all eigenvalues larger than the tolerance eigTol and condition number smaller than condTol).
- solutionsOutput: a list containing the complete output of the selected method for the different starting values. It includes the extra objects convergence and localMinimumGuaranteed.

Examples

```
# No local minimum
head(mleOptimWrapper(minusLogLik = function(x) -sum((x - 1:4)^2),
                     start = rbind(10:13, 1:2), selectSolution = "lowest"))
head(mleOptimWrapper(minusLogLik = function(x) -sum((x - 1:4)^2),
                     start = rbind(10:13, 1:2),
                     selectSolution = "lowestConv"))
head(mleOptimWrapper(minusLogLik = function(x) -sum((x - 1:4)^2),
                     start = rbind(10:13, 1:2),
                     selectSolution = "lowestLocMin"))

# Local minimum
head(mleOptimWrapper(minusLogLik = function(x) sum((x - 1:4)^2),
                     start = rbind(10:13), optMethod = "BFGS"))
head(mleOptimWrapper(minusLogLik = function(x) sum((x - 1:4)^2),
                     start = rbind(10:13), optMethod = "Nelder-Mead"))

# Function with several local minimum and a 'spurious' one
f <- function(x)  0.75 * (x[1] - 1)^2 -
                  10 / (0.1 + 0.1 * ((x[1] - 15)^2 + (x[2] - 2)^2)) -
                  9.5 / (0.1 + 0.1 * ((x[1] - 15)^2 + (x[2] + 2)^2))
plotSurface2D(x = seq(0, 20, l = 100), y = seq(-3, 3, l = 100), f = f)
head(mleOptimWrapper(minusLogLik = f,
                     start = rbind(c(15, 2), c(15, -2), c(5, 0)),
                     selectSolution = "lowest"))
head(mleOptimWrapper(minusLogLik = f,
                     start = rbind(c(15, 2), c(15, -2), c(5, 0)),
                     selectSolution = "lowestConv"))
head(mleOptimWrapper(minusLogLik = f,
                     start = rbind(c(15, 2), c(15, -2), c(5, 0)),
                     selectSolution = "lowestLocMin", eigTol = 0.01))

# With constraint region
```

```
head(mleOptimWrapper(minusLogLik = function(x) sum((x - 1:2)^2),
                     start = rbind(10:11),
                     region = function(pars) {
                       x <- pars[1]
                       y <- pars[2]
                       if (y <= x^2) {
                         return(list("pars" = pars, "penalty" = 0))
                       } else {
                        return(list("pars" = c(sqrt(y), y),
                                    "penalty" = y - x^2))
                       }
                     }, lower = c(0.5, 1), upper = c(Inf, Inf),
                   optMethod = "Nelder-Mead", selectSolution = "lowest"))
head(mleOptimWrapper(minusLogLik = function(x) sum((x - 1:2)^2),
                     start = rbind(10:11), lower = c(0.5, 1),
                     upper = c(Inf, Inf),optMethod = "Nelder-Mead"))
```

`mleOu` *Maximum likelihood estimation of the OU diffusion*

Description

Computation of the maximum likelihood estimator of the parameters of the *univariate* Ornstein–Uhlenbeck (OU) diffusion from a discretized trajectory $\{X_{\Delta i}\}_{i=1}^{N}$. The objective function to minimize is

$$\sum_{i=2}^{n} \log p_{\Delta}(X_{\Delta i}|X_{\Delta(i-1)}).$$

Usage

```
mleOu(data, delta, alpha = NA, mu = NA, sigma = NA, start,
  lower = c(0.01, -5, 0.01), upper = c(25, 5, 25), ...)
```

Arguments

`data`	a vector of size `N` with the discretized trajectory of the diffusion.
`delta`	time discretization step.
`alpha`, `mu`, `sigma`	arguments to fix a parameter to a given value and perform the estimation on the rest. Defaults to `NA`, meaning that the parameter is estimated. Note that `start`, `lower` and `upper` must be changed accordingly if parameters are fixed, see examples.
`start`	starting values, a matrix with `p` columns, with each entry representing a different starting value.
`lower`, `upper`	bound for box constraints as in method `"L-BFGS-B"` of `optim`.
`...`	further arguments to be passed to `mleOptimWrapper`.

Details

The first element in `data` is not taken into account for estimation. See `mleMou` for the multivariate case (less efficient for dimension one).

Value

Output from `mleOptimWrapper`.

Examples

```
set.seed(345678)
data <- rTrajOu(x0 = 0, alpha = 1, mu = 0, sigma = 1, N = 100, delta = 0.1)
mleOu(data = data, delta = 0.1, start = c(2, 1, 2), lower = c(0.1, -10, 0.1),
      upper = c(25, 10, 25))

# Fixed sigma and mu
mleOu(data = data, delta = 0.1, mu = 0, sigma = 1, start = 2, lower = 0.1,
      upper = 25, optMethod = "nlm")
```

---

mlePde1D *MLE for toroidal process via PDE solving in 1D*

Description

Maximum Likelihood Estimation (MLE) for arbitrary diffusions, based on the transition probability density (tpd) obtained as the numerical solution of the Fokker–Planck Partial Differential Equation (PDE) in 1D.

Usage

```
mlePde1D(data, delta, b, sigma2, Mx = 500, Mt = ceiling(100 * delta),
  sdInitial = 0.1, linearBinning = FALSE, start, lower, upper, ...)
```

Arguments

`data`	a vector of size `N` with the discretized trajectory of the diffusion.
`delta`	time discretization step.
`b`	drift function. Must return a vector of the same size as its argument.
`sigma2`	function giving the squared diffusion coefficient. Must return a vector of the same size as its argument.
`Mx`	size of the equispaced spatial grid in $[-\pi, \pi)$.
`Mt`	size of the time grid in $[0, t]$.
`sdInitial`	the standard deviation of the concentrated WN giving the initial condition.
`linearBinning`	flag to indicate whether linear binning should be applied for the initial values of the tpd, instead of usual simple binning (cheaper). Linear binning is always done in the evaluation of the tpd.

start	starting values, a matrix with p columns, with each entry representing a different starting value.
lower, upper	bound for box constraints as in method "L-BFGS-B" of optim.
...	Further parameters passed to crankNicolson1D.

Details

See Sections 3.4.1 and 3.4.4 in García-Portugués et al. (2019) for details.

Value

Output from mleOptimWrapper.

References

García-Portugués, E., Sørensen, M., Mardia, K. V. and Hamelryck, T. (2019) Langevin diffusions on the torus: estimation and applications. *Statistics and Computing*, 29(2):1–22. doi:10.1007/s1122201797902

Examples

```
# Test in OU
alpha <- 2
mu <- 0
sigma <- 1
set.seed(234567)
traj <- rTrajOu(x0 = 0, alpha = alpha, mu = mu, sigma = sigma, N = 500,
                delta = 0.5)
b <- function(x, pars) pars[1] * (pars[2] - x)
sigma2 <- function(x, pars) rep(pars[3]^2, length(x))

exactOu <- mleOu(traj, delta = 0.5, start = c(1, 1, 2),
                 lower = c(0.1, -pi, 0.1), upper = c(10, pi, 10))
pdeOu <- mlePde1D(data = traj, delta = 0.5, Mx = 100, Mt = 100, b = b,
                  sigma2 = sigma2, start = c(1, 1, 2),
                  lower = c(0.1, -pi, -10), upper = c(10, pi, 10),
                  verbose = 2)
pdeOuLin <- mlePde1D(data = traj, delta = 0.5, Mx = 100, Mt = 100, b = b,
                     sigma2 = sigma2, start = c(1, 1, 2),
                     lower = c(0.1, -pi, -10), upper = c(10, pi, 10),
                     linearBinning = TRUE, verbose = 2)
head(exactOu)
head(pdeOu)
head(pdeOuLin)

# Test in WN diffusion
alpha <- 2
mu <- 0
sigma <- 1
set.seed(234567)
traj <- rTrajWn1D(x0 = 0, alpha = alpha, mu = mu, sigma = sigma, N = 500,
```

```
                 delta = 0.5)

exactOu <- mleOu(traj, delta = 0.5, start = c(1, 1, 2),
                 lower = c(0.1, -pi, 0.1), upper = c(10, pi, 10))
pdeWn <- mlePde1D(data = traj, delta = 0.5, Mx = 100, Mt = 100,
                  b = function(x, pars)
                    driftWn1D(x = x, alpha = pars[1], mu = pars[2],
                              sigma = pars[3]),
                  sigma2 = function(x, pars) rep(pars[3]^2, length(x)),
                  start = c(1, 1, 2), lower = c(0.1, -pi, -10),
                  upper = c(10, pi, 10), verbose = 2)
pdeWnLin <- mlePde1D(data = traj, delta = 0.5, Mx = 100, Mt = 100,
                     b = function(x, pars)
                       driftWn1D(x = x, alpha = pars[1], mu = pars[2],
                                 sigma = pars[3]),
                     sigma2 = function(x, pars) rep(pars[3]^2, length(x)),
                     start = c(1, 1, 2), lower = c(0.1, -pi, -10),
                     upper = c(10, pi, 10), linearBinning = TRUE,
                     verbose = 2)
head(exactOu)
head(pdeWn)
head(pdeWnLin)
```

`mlePde2D` *MLE for toroidal process via PDE solving in 2D*

Description

Maximum Likelihood Estimation (MLE) for arbitrary diffusions, based on the transition probability density (tpd) obtained as the numerical solution of the Fokker–Planck Partial Differential Equation (PDE) in 2D.

Usage

```
mlePde2D(data, delta, b, sigma2, Mx = 50, My = 50, Mt = ceiling(100 *
  delta), sdInitial = 0.1, linearBinning = FALSE, start, lower, upper, ...)
```

Arguments

`data`	a matrix of dimension `c(n, p)`.
`delta`	discretization step.
`b`	drift function. Must return a vector of the same size as its argument.
`sigma2`	function giving the diagonal of the diffusion matrix. Must return a vector of the same size as its argument.
`Mx`, `My`	sizes of the equispaced spatial grids in $[-\pi, \pi)$ for each component.
`Mt`	size of the time grid in $[0, t]$.

sdInitial	standard deviations of the concentrated WN giving the initial condition.
linearBinning	flag to indicate whether linear binning should be applied for the initial values of the tpd, instead of usual simple binning (cheaper). Linear binning is always done in the evaluation of the tpd.
start	starting values, a matrix with p columns, with each entry representing a different starting value.
lower, upper	bound for box constraints as in method "L-BFGS-B" of optim.
...	further parameters passed to mleOptimWrapper.

Details

See Sections 3.4.2 and 3.4.4 in García-Portugués et al. (2019) for details. The function currently includes the region function for imposing a feasibility region on the parameters of the bivariate WN diffusion.

Value

Output from mleOptimWrapper.

References

García-Portugués, E., Sørensen, M., Mardia, K. V. and Hamelryck, T. (2019) Langevin diffusions on the torus: estimation and applications. *Statistics and Computing*, 29(2):1–22. doi:10.1007/s1122201797902

Examples

```
# Test in OU process
alpha <- c(1, 2, -0.5)
mu <- c(0, 0)
sigma <- c(0.5, 1)
set.seed(2334567)
data <- rTrajMou(x0 = c(0, 0), A = alphaToA(alpha = alpha, sigma = sigma),
                 mu = mu, Sigma = diag(sigma^2), N = 500, delta = 0.5)
b <- function(x, pars) sweep(-x, 2, pars[4:5], "+") %*%
                       t(alphaToA(alpha = pars[1:3], sigma = sigma))
sigma2 <- function(x, pars) repRow(sigma^2, nrow(x))

exactOu <- mleMou(data = data, delta = 0.5, sigma = sigma,
                  start = c(1, 1, 0, 2, 2),
                  lower = c(0.1, 0.1, -25, -10, -10),
                  upper = c(25, 25, 25, 10, 10))
head(exactOu, 2)
pdeOu <- mlePde2D(data = data, delta = 0.5, b = b, sigma2 = sigma2,
                  Mx = 10, My = 10, Mt = 10,
                  start = rbind(c(1, 1, 0, 2, 2)),
                  lower = c(0.1, 0.1, -25, -10, -10),
                  upper = c(25, 25, 25, 10, 10), verbose = 2)
head(pdeOu, 2)
pdeOuLin <- mlePde2D(data = data, delta = 0.5, b = b, sigma2 = sigma2,
```

```
                      Mx = 10, My = 10, Mt = 10,
                      start = rbind(c(1, 1, 0, 2, 2)),
                      lower = c(0.1, 0.1, -25, -10, -10),
                      upper = c(25, 25, 25, 10, 10), verbose = 2,
                      linearBinning = TRUE)
head(pdeOuLin, 2)

# Test in WN diffusion
alpha <- c(1, 0.5, 0.25)
mu <- c(0, 0)
sigma <- c(2, 1)
set.seed(234567)
data <- rTrajWn2D(x0 = c(0, 0), alpha = alpha, mu = mu, sigma = sigma,
                     N = 200, delta = 0.5)
b <- function(x, pars) driftWn2D(x = x, A = alphaToA(alpha = pars[1:3],
                                                     sigma = sigma),
                                 mu = pars[4:5], sigma = sigma)
sigma2 <- function(x, pars) repRow(sigma^2, nrow(x))

exactOu <- mleMou(data = data, delta = 0.5, sigma = sigma,
                  start = c(1, 1, 0, 1, 1),
                  lower = c(0.1, 0.1, -25, -25, -25),
                  upper = c(25, 25, 25, 25, 25), optMethod = "nlm")
pdeWn <- mlePde2D(data = data, delta = 0.5, b = b, sigma2 = sigma2,
                  Mx = 20, My = 20, Mt = 10, start = rbind(c(1, 1, 0, 1, 1)),
                  lower = c(0.1, 0.1, -25, -25, -25),
                  upper = c(25, 25, 25, 25, 25), verbose = 2,
                  optMethod = "nlm")
pdeWnLin <- mlePde2D(data = data, delta = 0.5, b = b, sigma2 = sigma2,
                     Mx = 20, My = 20, Mt = 10,
                     start = rbind(c(1, 1, 0, 1, 1)),
                     lower = c(0.1, 0.1, -25, -25, -25),
                     upper = c(25, 25, 25, 25, 25), verbose = 2,
                     linearBinning = TRUE)

head(exactOu)
head(pdeOu)
head(pdeOuLin)
```

`periodicTrapRule1D` *Quadrature rules in 1D, 2D and 3D*

Description

Quadrature rules for definite integrals over intervals in 1D, $\int_{x_1}^{x_2} f(x)dx$, rectangles in 2D, $\int_{x_1}^{x_2}\int_{y_1}^{y_2} f(x,y)dydx$ and cubes in 3D, $\int_{x_1}^{x_2}\int_{y_1}^{y_2}\int_{z_1}^{z_2} f(x,y,z)dzdydx$. The trapezoidal rules assume that the function is periodic, whereas the Simpson rules work for arbitrary functions.

Usage

```
periodicTrapRule1D(fx, endsMatch = FALSE, na.rm = TRUE,
  lengthInterval = 2 * pi)

periodicTrapRule2D(fxy, endsMatch = FALSE, na.rm = TRUE,
  lengthInterval = rep(2 * pi, 2))

periodicTrapRule3D(fxyz, endsMatch = FALSE, na.rm = TRUE,
  lengthInterval = rep(2 * pi, 3))

integrateSimp1D(fx, lengthInterval = 2 * pi, na.rm = TRUE)

integrateSimp2D(fxy, lengthInterval = rep(2 * pi, 2), na.rm = TRUE)

integrateSimp3D(fxyz, lengthInterval = rep(2 * pi, 3), na.rm = TRUE)
```

Arguments

`fx`	vector containing the evaluation of the function to integrate over a uniform grid in $[x_1, x_2]$.
`endsMatch`	flag to indicate whether the values of the last entries of `fx`, `fxy` or `fxyz` are the ones in the first entries (elements, rows, columns, slices). See examples for usage.
`na.rm`	logical. Should missing values (including `NaN`) be removed?
`lengthInterval`	vector containing the lengths of the intervals of integration.
`fxy`	matrix containing the evaluation of the function to integrate over a uniform grid in $[x_1, x_2] \times [y_1, y_2]$.
`fxyz`	three dimensional array containing the evaluation of the function to integrate over a uniform grid in $[x_1, x_2] \times [y_1, y_2] \times [z_1, z_2]$.

Details

The simple trapezoidal rule has a very good performance for periodic functions in 1D and 2D(order of error). The higher dimensional extensions are obtained by iterative usage of the 1D rules.

Value

The value of the integral.

References

Press, W. H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P. (1996). *Numerical Recipes in Fortran 77: The Art of Scientific Computing (Vol. 1 of Fortran Numerical Recipes)*. Cambridge University Press, Cambridge.

Examples

```
# In 1D. True value: 3.55099937
N <- 21
grid <- seq(-pi, pi, l = N)
fx <- sin(grid)^2 * exp(cos(grid))
periodicTrapRule1D(fx = fx, endsMatch = TRUE)
periodicTrapRule1D(fx = fx[-N], endsMatch = FALSE)
integrateSimp1D(fx = fx, lengthInterval = 2 * pi)
integrateSimp1D(fx = fx[-N]) # Worse, of course

# In 2D. True value: 22.31159
fxy <- outer(grid, grid, function(x, y) (sin(x)^2 * exp(cos(x)) +
                                           sin(y)^2 * exp(cos(y))) / 2)
periodicTrapRule2D(fxy = fxy, endsMatch = TRUE)
periodicTrapRule2D(fxy = fxy[-N, -N], endsMatch = FALSE)
periodicTrapRule1D(apply(fxy[-N, -N], 1, periodicTrapRule1D))
integrateSimp2D(fxy = fxy)
integrateSimp1D(apply(fxy, 1, integrateSimp1D))

# In 3D. True value: 140.1878
fxyz <- array(fxy, dim = c(N, N, N))
for (i in 1:N) fxyz[i, , ] <- fxy
periodicTrapRule3D(fxyz = fxyz, endsMatch = TRUE)
integrateSimp3D(fxyz = fxyz)
```

---

psMle *Maximum pseudo-likelihood estimation by wrapped pseudo-likelihoods*

Description

Maximum pseudo-likelihood using the Euler and Shoji–Ozaki pseudo-likelihoods.

Usage

```
psMle(data, delta, method = c("E", "SO", "SO2"), b, jac.b, sigma2, b1, b2,
  start, lower, upper, circular = TRUE, maxK = 2, vmApprox = FALSE, ...)
```

Arguments

data	a matrix of dimension c(n, p).
delta	discretization step.
method	a string for choosing "E" (Euler), "SO" (Shoji–Ozaki) or "SO2" (Shoji–Ozaki with Ito's expansion in the drift) method.
b	drift function. Must return a matrix of the same size as x.
jac.b	jacobian of the drift function.
sigma2	diagonal of the diffusion matrix (if univariate, this is the square of the diffusion coefficient). Must return an object of the same size as x.

b1	first derivative of the drift function (univariate). Must return a vector of the same length as x.
b2	second derivative of the drift function (univariate). Must return a vector of the same length as x.
start	starting values, a matrix with p columns, with each entry representing a different starting value.
lower, upper	bound for box constraints as in method "L-BFGS-B" of optim.
circular	flag to indicate circular data.
maxK	maximum absolute winding number used if circular = TRUE.
vmApprox	flag to indicate von Mises approximation to wrapped normal. See momentMatchWnVm and scoreMatchWnBvm.
...	further parameters passed to mleOptimWrapper.

Details

See Section 3.2 in García-Portugués et al. (2019) for details. "SO2" implements Shoji and Ozai (1998)'s expansion with for p = 1. "SO" is the same expansion, for arbitrary p, but considering null second derivatives.

Value

Output from mleOptimWrapper.

References

García-Portugués, E., Sørensen, M., Mardia, K. V. and Hamelryck, T. (2019) Langevin diffusions on the torus: estimation and applications. *Statistics and Computing*, 29(2):1–22. doi:10.1007/s1122201797902

Shoji, I. and Ozaki, T. (1998) A statistical method of estimation and simulation for systems of stochastic differential equations. *Biometrika*, 85(1):240–243. doi:10.1093/biomet/85.1.240

Examples

```
# Example in 1D

delta <- 0.5
pars <- c(0.25, 0, 2)
set.seed(12345678)
samp <- rTrajWn1D(x0 = 0, alpha = pars[1], mu = pars[2], sigma = pars[3],
                  N = 100, delta = delta)
b <- function(x, pars) driftWn1D(x = x, alpha = pars[1], mu = pars[2],
                                 sigma = pars[3], maxK = 2, expTrc = 30)
b1 <- function(x, pars, h = 1e-4) {
  l <- length(x)
  res <- b(x = c(x + h, x - h), pars = pars)
  drop(res[1:l] - res[(l + 1):(2 * l)])/(2 * h)
}
b2 <- function(x, pars, h = 1e-4) {
```

```
  l <- length(x)
  res <- b(x = c(x + h, x, x - h), pars = pars)
  drop(res[1:l] - 2 * res[(l + 1):(2 * l)] + res[(2 * l + 1):(3 * l)])/(h^2)
}
sigma2 <- function(x, pars) rep(pars[3]^2, length(x))
lower <- c(0.1, -pi, 0.1)
upper <- c(10, pi, 10)
psMle(data = samp, delta = delta, method = "E", b = b, sigma2 = sigma2,
      start = pars, lower = lower, upper = upper)
psMle(data = samp, delta = delta, method = "E", b = b, sigma2 = sigma2,
      start = pars, lower = lower, upper = upper, vmApprox = TRUE)
psMle(data = samp, delta = delta, method = "SO2", b = b, b1 = b1,
      b2 = b2, sigma2 = sigma2, start = pars, lower = lower, upper = upper)
psMle(data = samp, delta = delta, method = "SO2", b = b, b1 = b1,
      b2 = b2, sigma2 = sigma2, start = pars, lower = lower,
      upper = upper, vmApprox = TRUE)
psMle(data = samp, delta = delta, method = "SO", b = b, b1 = b1,
      lower = lower, upper = upper, sigma2 = sigma2, start = pars)
approxMleWn1D(data = samp, delta = delta, start = pars)
mlePde1D(data = samp, delta = delta, b = b, sigma2 = sigma2,
         start = pars, lower = lower, upper = upper)

# Example in 2D

delta <- 0.5
pars <- c(1, 0.5, 0, 0, 0, 1, 2)
set.seed(12345678)
samp <- rTrajWn2D(x0 = c(0, 0), alpha = pars[1:3], mu = pars[4:5],
                  sigma = pars[6:7], N = 100, delta = delta)
b <- function(x, pars) driftWn2D(x = x, A = alphaToA(alpha = pars[1:3],
                                                    sigma = pars[6:7]),
                                 mu = pars[4:5], sigma = pars[6:7], maxK = 2,
                                 expTrc = 30)
jac.b <- function(x, pars, h = 1e-4) {
  l <- nrow(x)
  res <- b(x = rbind(cbind(x[, 1] + h, x[, 2]),
                     cbind(x[, 1] - h, x[, 2]),
                     cbind(x[, 1], x[, 2] + h),
                     cbind(x[, 1], x[, 2] - h)), pars = pars)
  cbind(res[1:l, ] - res[(l + 1):(2 * l), ],
        res[2 * l + 1:l, ] - res[2 * l + (l + 1):(2 * l), ]) / (2 * h)
}
sigma2 <- function(x, pars) matrix(pars[6:7]^2, nrow = length(x) / 2L,
                                   ncol = 2)
lower <- c(0.01, 0.01, -25, -pi, -pi, 0.01, 0.01)
upper <- c(25, 25, 25, pi, pi, 25, 25)
psMle(data = samp, delta = delta, method = "E", b = b, sigma2 = sigma2,
      start = pars, lower = lower, upper = upper)
psMle(data = samp, delta = delta, method = "E", b = b, sigma2 = sigma2,
      start = pars, lower = lower, upper = upper, vmApprox = TRUE)
psMle(data = samp, delta = delta, method = "SO", b = b, jac.b = jac.b,
      sigma2 = sigma2, start = pars, lower = lower, upper = upper)
approxMleWn2D(data = samp, delta = delta, start = c(pars, 0))
```

```
# Set maxit = 5 to test and avoid a very long evaluation
mlePde2D(data = samp, delta = delta, b = b, sigma2 = sigma2, start = pars,
         lower = lower, upper = upper, maxit = 5)
```

`rStatWn2D` *Simulation from the stationary density of a WN diffusion in 2D*

Description

Simulates from the stationary density of the WN diffusion in 2D.

Usage

```
rStatWn2D(n, mu, alpha, sigma, rho = 0)
```

Arguments

`n`	sample size.
`mu`	a vector of length 2 giving the mean.
`alpha`	vector of length 3 parametrizing the `A` matrix as in `alphaToA`.
`sigma`	vector of length 2 containing the **square root** of the diagonal of Σ, the diffusion matrix.
`rho`	correlation coefficient of Σ.

Value

A matrix of dimension `c(n, 2)` containing the samples from the stationary distribution.

Examples

```
set.seed(345567)
alpha <- c(2, 1, -1)
sigma <- c(1.5, 2)
Sigma <- diag(sigma^2)
A <- alphaToA(alpha = alpha, sigma = sigma)
mu <- c(pi, pi)
plot(rStatWn2D(n = 1000, mu = mu, alpha = alpha, sigma = sigma))
points(toPiInt(mvtnorm::rmvnorm(n = 1000, mean = mu,
                                sigma = solve(A) %*% Sigma / 2,
                                method = "chol")), col = 2)
```

rTpdWn2D *Simulation from the approximated transition distribution of a WN diffusion in 2D*

Description

Simulates from the approximate transition density of the WN diffusion in 2D.

Usage

```
rTpdWn2D(n, x0, t, mu, alpha, sigma, rho = 0, maxK = 2L, expTrc = 30)
```

Arguments

n	sample size.
x0	a matrix of dimension c(nx0, 2) giving the starting values.
t	vector of length nx0 containing the times between observations.
mu	a vector of length 2 giving the mean.
alpha	vector of length 3 parametrizing the A matrix as in alphaToA.
sigma	vector of length 2 containing the **square root** of the diagonal of Σ, the diffusion matrix.
rho	correlation coefficient of Σ.
maxK	maximum absolute value of the windings considered in the computation of the WN.
expTrc	truncation for exponential: exp(x) with x <= -expTrc is set to zero. Defaults to 30.

Value

An array of dimension c(n, 2, nx0) containing the n samples of the transition distribution, conditioned on that the process was at x0 at t instants ago. The samples are all in $[\pi, \pi)$.

Examples

```
alpha <- c(3, 2, -1)
sigma <- c(0.5, 1)
mu <- c(pi, pi)
x <- seq(-pi, pi, l = 100)
t <- 0.5
image(x, x, matrix(dTpdWou2D(x = as.matrix(expand.grid(x, x)),
                             x0 = matrix(rep(0, 100 * 2),
                                         nrow = 100 * 100, ncol = 2),
                             t = t, mu = mu, alpha = alpha, sigma = sigma,
                             maxK = 2, expTrc = 30), nrow = 100, ncol = 100),
      zlim = c(0, 0.5))
points(rTpdWn2D(n = 500, x0 = rbind(c(0, 0)), t = t, mu = mu, alpha = alpha,
```

```
                sigma = sigma)[, , 1], col = 3)
points(stepAheadWn2D(x0 = rbind(c(0, 0)), delta = t / 500,
                     A = alphaToA(alpha = alpha, sigma = sigma),
                     mu = mu, sigma = sigma, N = 500, M = 500, maxK = 2,
                     expTrc = 30), col = 4)
```

`rTrajLangevin` *Simulation of trajectories of a Langevin diffusion*

Description

Simulation of an arbitrary Langevin diffusion in dimension `p` by subsampling a fine trajectory obtained by the Euler discretization.

Usage

```
rTrajLangevin(x0, drift, SigDif, N = 100, delta = 0.01, NFine = ceiling(N
  * delta/deltaFine), deltaFine = min(delta/100, 0.001), circular = TRUE,
  ...)
```

Arguments

`x0`	vector of length `p` giving the initial point.
`drift`	drift for the diffusion.
`SigDif`	matrix of size `c(p, p)` giving the infinitesimal (constant) covariance matrix of the diffusion.
`N`	number of discretization steps in the resulting trajectory.
`delta`	discretization step.
`NFine`	number of discretization steps for the fine trajectory. Must be larger than `N`.
`deltaFine`	discretization step for the fine trajectory. Must be smaller than `delta`.
`circular`	whether to wrap the resulting trajectory to $[-\pi, \pi)^p$.
`...`	parameters to be passed to `drift`.

Details

The fine trajectory is subsampled using the indexes `seq(1, NFine + 1, by = NFine / N)`.

Value

A vector of length `N + 1` containing `x0` in the first entry and the discretized trajectory.

Examples

```
isRStudio <- identical(.Platform$GUI, "RStudio")
if (isRStudio) {
  # 1D
  manipulate::manipulate({
    x <- seq(0, N * delta, by = delta)
    plot(x, x, ylim = c(-pi, pi), type = "n",
         ylab = expression(X[t]), xlab = "t")
    linesCirc(x, rTrajLangevin(x0 = 0, drift = driftJp, SigDif = sigma,
                               alpha = alpha, mu = 0, psi = psi, N = N,
                               delta = 0.01))
    }, delta = manipulate::slider(0.01, 5.01, step = 0.1),
    N = manipulate::slider(10, 500, step = 10, initial = 200),
    alpha = manipulate::slider(0.01, 5, step = 0.1, initial = 1),
    psi = manipulate::slider(-2, 2, step = 0.1, initial = 1),
    sigma = manipulate::slider(0.01, 5, step = 0.1, initial = 1))

  # 2D
  samp <- rTrajLangevin(x0 = c(0, 0), drift = driftMvm, alpha = c(1, 1),
                        mu = c(2, -1), A = diag(rep(0, 2)),
                        SigDif = diag(rep(1, 2)), N = 1000, delta = 0.1)
  plot(samp, xlim = c(-pi, pi), ylim = c(-pi, pi), pch = 19, cex = 0.25,
       xlab = expression(X[t]), ylab = expression(Y[t]), col = rainbow(1000))
  linesTorus(samp[, 1], samp[, 2], col = rainbow(1000))
}
```

`rTrajMou` *Simulation of trajectories for the multivariate OU diffusion*

Description

Simulation of trajectories of the *multivariate* Ornstein–Uhlenbeck (OU) diffusion

$$dX_t = A(\mu - X_t)dt + \Sigma^{\frac{1}{2}} dW_t, X_0 = x_0$$

using the exact transition probability density.

Usage

```
rTrajMou(x0, A, mu, Sigma, N = 100, delta = 0.001)
```

Arguments

`x0`	a vector of length `p` containing initial point.
`A`	the drift matrix, of size `c(p, p)`.
`mu`	unconditional mean of the diffusion, a vector of length `p`.
`Sigma`	square of the diffusion matrix, a matrix of size `c(p, p)`.
`N`	number of discretization steps in the resulting trajectory.
`delta`	time discretization step.

Details

The law of the discretized trajectory at *each* time step is a multivariate normal with mean `meantMou` and covariance matrix `covtMou`. See `rTrajOu` for the univariate case (more efficient).

`solve(A) %*% Sigma` has to be a covariance matrix (symmetric and positive definite) in order to have a proper transition density. For the bivariate case, this can be ensured with the `alphaToA` function. In the multivariate case, it is ensured if `Sigma` is isotropic and `A` is a covariance matrix.

Value

A matrix of size `c(N + 1, p)` containing `x0` in the first row and the exact discretized trajectory on the remaining rows.

Examples

```
set.seed(987658)
data <- rTrajMou(x0 = c(0, 0), A = alphaToA(alpha = c(1, 2, 0.5),
                 sigma = 1:2), mu = c(1, 1), Sigma = diag((1:2)^2),
                 N = 200, delta = 0.1)
plot(data, pch = 19, col = rainbow(201), cex = 0.25)
arrows(x0 = data[-201, 1], y0 = data[-201, 2], x1 = data[-1, 1],
       y1 = data[-1, 2], col = rainbow(201), angle = 10, length = 0.1)
```

---

rTrajOu *Simulation of trajectories for the univariate OU diffusion*

Description

Simulation of trajectories of the *univariate* Ornstein–Uhlenbeck (OU) diffusion

$$dX_t = \alpha(\mu - X_t)dt + \sigma dW_t, X_0 = x_0$$

using the exact transition probability density.

Usage

```
rTrajOu(x0, alpha, mu, sigma, N = 100, delta = 0.001)
```

Arguments

`x0`	initial point.
`alpha`	strength of the drift.
`mu`	unconditional mean of the diffusion.
`sigma`	diffusion coefficient.
`N`	number of discretization steps in the resulting trajectory.
`delta`	time discretization step.

Details

The law of the discretized trajectory is a multivariate normal with mean `meantOu` and covariance matrix `covstOu`. See `rTrajMou` for the multivariate case (less efficient for dimension one).

Value

A vector of length `N + 1` containing `x0` in the first entry and the exact discretized trajectory on the remaining elements.

Examples

```
isRStudio <- identical(.Platform$GUI, "RStudio")
if (isRStudio) {
  manipulate::manipulate({
   set.seed(345678);
   plot(seq(0, N * delta, by = delta), rTrajOu(x0 = 0, alpha = alpha, mu = 0,
        sigma = sigma, N = N, delta = delta), ylim = c(-4, 4), type = "l",
        ylab = expression(X[t]), xlab = "t")
   }, delta = manipulate::slider(0.01, 5.01, step = 0.1),
   N = manipulate::slider(10, 500, step = 10, initial = 200),
   alpha = manipulate::slider(0.01, 5, step = 0.1, initial = 1),
   sigma = manipulate::slider(0.01, 5, step = 0.1, initial = 1))
}
```

---

rTrajWn1D *Simulation of trajectories for the WN diffusion in 1D*

Description

Simulation of the Wrapped Normal (WN) diffusion in 1D by subsampling a fine trajectory obtained by the Euler discretization.

Usage

```
rTrajWn1D(x0, alpha, mu, sigma, N = 100, delta = 0.01, NFine = ceiling(N
  * delta/deltaFine), deltaFine = min(delta/100, 0.001))
```

Arguments

`x0`	initial point.
`alpha`	drift parameter.
`mu`	mean parameter. Must be in $[\pi, \pi)$.
`sigma`	diffusion coefficient.
`N`	number of discretization steps in the resulting trajectory.
`delta`	discretization step.
`NFine`	number of discretization steps for the fine trajectory. Must be larger than `N`.
`deltaFine`	discretization step for the fine trajectory. Must be smaller than `delta`.

Details

The fine trajectory is subsampled using the indexes `seq(1, NFine + 1, by = NFine / N)`.

Value

A vector of length `N + 1` containing `x0` in the first entry and the discretized trajectory.

Examples

```
isRStudio <- identical(.Platform$GUI, "RStudio")
if (isRStudio) {
  manipulate::manipulate({
    x <- seq(0, N * delta, by = delta)
    plot(x, x, ylim = c(-pi, pi), type = "n",
         ylab = expression(X[t]), xlab = "t")
    linesCirc(x, rTrajWn1D(x0 = 0, alpha = alpha, mu = 0, sigma = sigma,
                           N = N, delta = 0.01))
    }, delta = slider(0.01, 5.01, step = 0.1),
    N = manipulate::slider(10, 500, step = 10, initial = 200),
    alpha = manipulate::slider(0.01, 5, step = 0.1, initial = 1),
    sigma = manipulate::slider(0.01, 5, step = 0.1, initial = 1))
}
```

---

rTrajWn2D *Simulation of trajectories for the WN diffusion in 2D*

Description

Simulation of the Wrapped Normal (WN) diffusion in 2D by subsampling a fine trajectory obtained by the Euler discretization.

Usage

```
rTrajWn2D(x0, alpha, mu, sigma, rho = 0, N = 100, delta = 0.01,
  NFine = ceiling(N * delta/deltaFine), deltaFine = min(delta/100, 0.001))
```

Arguments

`x0`	vector of length 2 giving the initial point.
`alpha`	vector of length 3 parametrizing the `A` matrix as in `alphaToA`.
`mu`	a vector of length 2 giving the mean.
`sigma`	vector of length 2 containing the **square root** of the diagonal of Σ, the diffusion matrix.
`rho`	correlation coefficient of Σ.
`N`	number of discretization steps in the resulting trajectory.
`delta`	discretization step.
`NFine`	number of discretization steps for the fine trajectory. Must be larger than `N`.
`deltaFine`	discretization step for the fine trajectory. Must be smaller than `delta`.

Details

The fine trajectory is subsampled using the indexes `seq(1, NFine + 1, by = NFine / N)`.

Value

A matrix of size `c(N + 1, 2)` containing `x0` in the first entry and the discretized trajectory.

Examples

```
samp <- rTrajWn2D(x0 = c(0, 0), alpha = c(1, 1, -0.5), mu = c(pi, pi),
                  sigma = c(1, 1), N = 1000, delta = 0.01)
plot(samp, xlim = c(-pi, pi), ylim = c(-pi, pi), pch = 19, cex = 0.25,
     xlab = expression(X[t]), ylab = expression(Y[t]), col = rainbow(1000))
linesTorus(samp[, 1], samp[, 2], col = rainbow(1000))
```

---

safeSoftMax *Safe softmax function for computing weights*

Description

Computes the weights $w_i = \frac{e^{p_i}}{\sum_{j=1}^{k} e^{p_j}}$ from p_i, $i = 1, \ldots, k$ in a safe way to avoid overflows and to truncate automatically to zero low values of w_i.

Usage

```
safeSoftMax(logs, expTrc = 30)
```

Arguments

`logs`	matrix of logarithms where each row contains a set of $p_1, \ldots, p_k$ to compute the weights from.
`expTrc`	truncation for exponential: `exp(x)` with `x <= -expTrc` is set to zero. Defaults to `30`.

Details

The `logs` argument must be always a matrix.

Value

A matrix of the size as `logs` containing the weights for each row.

Examples

```
# A matrix
safeSoftMax(rbind(1:10, 20:11))
rbind(exp(1:10) / sum(exp(1:10)), exp(20:11) / sum(exp(20:11)))

# A row-matrix
safeSoftMax(rbind(-100:100), expTrc = 30)
exp(-100:100) / sum(exp(-100:100))
```

`scoreMatchWnBvm` *Score and moment matching of a univariate or bivariate wrapped normal by a von Mises*

Description

Given a wrapped normal density, find the parameters of a von Mises that matches it according to two characteristics: moments and scores. Score matching estimators are available for univariate and bivariate cases and moment matching only for the former.

Usage

```
scoreMatchWnBvm(Sigma = NULL, invSigma)

scoreMatchWnVm(sigma, sigma2 = NULL)

momentMatchWnVm(sigma, sigma2 = NULL)
```

Arguments

`Sigma`, `invSigma`	covariance or precision matrix of the bivariate wrapped normal.
`sigma`, `sigma2`	standard deviation or variance of the wrapped normal.

Details

If the precision matrix is singular or if there are no solutions for the score matching estimator, `c(0, 0, 0)` is returned.

Value

Vector of parameters $(\kappa_1, \kappa_2, \lambda)$, where $(\kappa_1, \kappa_2, 2\lambda)$ is a suitable input for `kappa` in `dBvm`.

References

Mardia, K. V., Kent, J. T., and Laha, A. K. (2016). Score matching estimators for directional distributions. *arXiv:1604.0847*. https://arxiv.org/abs/1604.08470

Examples

```
# Univariate WN approximation
sigma <- 0.5
curve(dWn1D(x = x, mu = 0, sigma = sigma), from = -pi, to = pi,
      ylab = "Density", ylim = c(0, 1))
curve(dVm(x = x, mu = 0, kappa = momentMatchWnVm(sigma = sigma)), from = -pi,
      to = pi, col = "red", add = TRUE)
curve(dVm(x = x, mu = 0, kappa = scoreMatchWnVm(sigma = sigma)), from = -pi,
      to = pi, col = "green", add = TRUE)

# Bivariate WN approximation

# WN
alpha <- c(2, 1, 1)
sigma <- c(1, 1)
mu <- c(pi / 2, pi / 2)
x <- seq(-pi, pi, l = 101)[-101]
plotSurface2D(x, x, f = function(x) dStatWn2D(x = x, alpha = alpha, mu = mu,
                                              sigma = sigma), fVect = TRUE)
A <- alphaToA(alpha = alpha, sigma = sigma)
S <- 0.5 * solve(A) %*% diag(sigma)

# Score matching
kappa <- scoreMatchWnBvm(Sigma = S)

# dBvm uses lambda / 2 in the exponent
plotSurface2D(x, x, f = function(x) dBvm(x = x, mu = mu,
                                         kappa = c(kappa[1:2], 2 * kappa[3])),
              fVect = TRUE)

# With singular Sigma
invSigma <- matrix(c(1, sqrt(0.999), sqrt(0.999), 1), nrow = 2, ncol = 2)
scoreMatchWnBvm(invSigma = invSigma)
invSigma <- matrix(1, nrow = 2, ncol = 2)
scoreMatchWnBvm(invSigma = invSigma)
```

---

sdetorus *sdetorus - Statistical Tools for Toroidal Diffusions*

Description

Implementation of statistical methods for the estimation of toroidal diffusions. Several diffusive models are provided, most of them belonging to the Langevin family of diffusions on the torus. Specifically, the wrapped normal and von Mises processes are included, which can be seen as toroidal analogues of the Ornstein–Uhlenbeck diffusion. A collection of methods for approximate maximum likelihood estimation, organized in four blocks, is given: (i) based on the exact transition probability density, obtained as the numerical solution to the Fokker-Plank equation; (ii) based on wrapped pseudo-likelihoods; (iii) based on specific analytic approximations by wrapped processes; (iv) based on maximum likelihood of the stationary densities. The package allows the replicability of the results in García-Portugués et al. (2019) <doi:10.1007/s11222-017-9790-2>.

Author(s)

Eduardo García-Portugués.

References

García-Portugués, E., Sørensen, M., Mardia, K. V. and Hamelryck, T. (2019) Langevin diffusions on the torus: estimation and applications. *Statistics and Computing*, 29(2):1–22. doi:10.1007/s1122201797902

`sigmaDiff` *High-frequency estimate of the diffusion matrix*

Description

Estimation of the Σ in the multivariate diffusion

$$dX_t = b(X_t)dt + \Sigma dW_t$$

by the high-frequency estimate

$$\hat{\Sigma} = \frac{1}{N\Delta}\sum_{i=1}^{N}(X_i - X_{i-1})(X_i - X_{i-1})^T$$

Usage

```
sigmaDiff(data, delta, circular = TRUE, diagonal = FALSE,
  isotropic = FALSE)
```

Arguments

`data`	vector or matrix of size `c(N, p)` containing the discretized process.
`delta`	discretization step.
`circular`	whether the process is circular or not.
`diagonal`, `isotropic`	enforce different constraints for the diffusion matrix.

Details

See Section 3.1 in García-Portugués et al. (2019) for details.

Value

The estimated diffusion matrix of size `c(p, p)`.

References

García-Portugués, E., Sørensen, M., Mardia, K. V. and Hamelryck, T. (2019) Langevin diffusions on the torus: estimation and applications. *Statistics and Computing*, 29(2):1–22. doi:10.1007/s1122201797902

Examples

```
# 1D
x <- drop(euler1D(x0 = 0, alpha = 1, mu = 0, sigma = 1, N = 1000,
                  delta = 0.01))
sigmaDiff(x, delta = 0.01)

# 2D
x <- t(euler2D(x0 = rbind(c(pi, pi)), A = rbind(c(2, 1), c(1, 2)),
               mu = c(pi, pi), sigma = c(1, 1), N = 1000,
               delta = 0.01)[1, , ])
sigmaDiff(x, delta = 0.01)
sigmaDiff(x, delta = 0.01, circular = FALSE)
sigmaDiff(x, delta = 0.01, diagonal = TRUE)
sigmaDiff(x, delta = 0.01, isotropic = TRUE)
```

`solveTridiag` *Thomas algorithm for solving tridiagonal matrix systems, with optional presaving of LU decomposition*

Description

Implementation of the Thomas algorithm to solve efficiently the tridiagonal matrix system

$$\begin{aligned} b_1x_1 + c_1x_2 + a_1x_n &= d_1 \\ a_2x_1 + b_2x_2 + c_2x_3 &= d_2 \\ &\vdots \\ a_{n-1}x_{n-2} + b_{n-1}x_{n-1} + c_{n-1}x_n &= d_{n-1} \\ c_nx_1 + a_nx_{n-1} + b_nx_n &= d_n \end{aligned}$$

with $a_1 = c_n = 0$ (usual tridiagonal matrix). If $a_1 \neq 0$ or $c_n \neq 0$ (circulant tridiagonal matrix), then the Sherman–Morrison formula is employed.

Usage

```
solveTridiag(a, b, c, d, LU = 0L)

solveTridiagMatConsts(a, b, c, d, LU = 0L)

solvePeriodicTridiag(a, b, c, d, LU = 0L)

forwardSweepTridiag(a, b, c)

forwardSweepPeriodicTridiag(a, b, c)
```

Arguments

a, b, c	subdiagonal (below main diagonal), diagonal and superdiagonal (above main diagonal), respectively. They all are vectors of length n.
d	vector of constant terms, of length n. For solveTridiagMatConsts, it can be a matrix with n rows.
LU	flag denoting if the forward sweep encoding the LU decomposition is supplied in vectors b and c. See details and examples.

Details

The Thomas algorithm is stable if the matrix is diagonally dominant.

For the periodic case, two non-periodic tridiagonal systems with different constant terms (but same coefficients) are solved using solveTridiagMatConsts. These two solutions are combined by the Sherman–Morrison formula to obtain the solution to the periodic system.

Note that the output of solveTridiag and solveTridiagMatConsts are independent from the values of a[1] and c[n], but solvePeriodicTridiag is not.

If LU is TRUE, then b and c must be precomputed with forwardSweepTridiag or forwardSweepPeriodicTridiag for its use in the call of the appropriate solver, which will be slightly faster.

Value

- solve* functions: the solution, a vector of length n and a matrix with n rows for solveTridiagMatConsts.
- forward* functions: the matrix cbind(b, c) creating the suitable b and c arguments for calling solve* when LU is TRUE.

References

Thomas, J. W. (1995). *Numerical Partial Differential Equations: Finite Difference Methods*. Springer, New York. doi:10.1007/9781489972781

Conte, S. D. and de Boor, C. (1980). *Elementary Numerical Analysis: An Algorithmic Approach*. Third edition. McGraw-Hill, New York. doi:10.1137/1.9781611975208

Examples

```
# Tridiagonal matrix
n <- 10
a <- rnorm(n, 3, 1)
b <- rnorm(n, 10, 1)
c <- rnorm(n, 0, 1)
d <- rnorm(n, 0, 1)
A <- matrix(0, nrow = n, ncol = n)
diag(A) <- b
for (i in 1:(n - 1)) {
  A[i + 1, i] <- a[i + 1]
  A[i, i + 1] <- c[i]
}
```

```
A

# Same solutions
drop(solveTridiag(a = a, b = b, c = c, d = d))
solve(a = A, b = d)

# Presaving the forward sweep (encodes the LU factorization)
LU <- forwardSweepTridiag(a = a, b = b, c = c)
drop(solveTridiag(a = a, b = LU[, 1], c = LU[, 2], d = d, LU = 1))

# With equal coefficient matrix
solveTridiagMatConsts(a = a, b = b, c = c, d = cbind(d, d + 1))
cbind(solve(a = A, b = d), solve(a = A, b = d + 1))
LU <- forwardSweepTridiag(a = a, b = b, c = c)
solveTridiagMatConsts(a = a, b = LU[, 1], c = LU[, 2], d = cbind(d, d + 1), LU = 1)

# Periodic matrix
A[1, n] <- a[1]
A[n, 1] <- c[n]
A

# Same solutions
drop(solvePeriodicTridiag(a = a, b = b, c = c, d = d))
solve(a = A, b = d)

# Presaving the forward sweep (encodes the LU factorization)
LU <- forwardSweepPeriodicTridiag(a = a, b = b, c = c)
drop(solvePeriodicTridiag(a = a, b = LU[, 1], c = LU[, 2], d = d, LU = 1))
```

---

stepAheadWn1D *Multiple simulation of trajectory ends of the WN or vM diffusion in 1D*

Description

Simulates `M` trajectories starting from different initial values `x0` of the WN or vM diffusion in 1D, by the Euler method, and returns their ends.

Usage

```
stepAheadWn1D(x0, alpha, mu, sigma, M, N = 100L, delta = 0.01, type = 1L,
  maxK = 2L, expTrc = 30)
```

Arguments

`x0`	vector of length `nx0` giving the initial points.
`alpha`	drift parameter.
`mu`	mean parameter. Must be in $[\pi, \pi)$.
`sigma`	diffusion coefficient.

M	number of Monte Carlo replicates.
N	number of discretization steps.
delta	discretization step.
type	integer giving the type of diffusion. Currently, only 1 for WN and 2 for vM are supported.
maxK	maximum absolute value of the windings considered in the computation of the WN.
expTrc	truncation for exponential: exp(x) with x <= -expTrc is set to zero. Defaults to 30.

Value

A matrix of size c(nx0, M) containing the M trajectory ends for each starting value x0.

Examples

```
N <- 100
nx0 <- 20
x0 <- seq(-pi, pi, l = nx0 + 1)[-(nx0 + 1)]
set.seed(12345678)
samp1 <- euler1D(x0 = x0, mu = 0, alpha = 3, sigma = 1, N = N,
                 delta = 0.01, type = 2)
tt <- seq(0, 1, l = N + 1)
plot(rep(0, nx0), x0, pch = 16, col = rainbow(nx0), xlim = c(0, 1))
for (i in 1:nx0) linesCirc(tt, samp1[i, ], col = rainbow(nx0)[i])
set.seed(12345678)
samp2 <- stepAheadWn1D(x0 = x0, mu = 0, alpha = 3, sigma = 1, M = 1,
                       N = N, delta = 0.01, type = 2)
points(rep(1, nx0), samp2[, 1], pch = 16, col = rainbow(nx0))
samp1[, N + 1]
samp2[, 1]
```

---

stepAheadWn2D *Multiple simulation of trajectory ends of the WN or MvM diffusion in 2D*

Description

Simulates M trajectories starting from different initial values x0 of the WN or MvM diffusion in 2D, by the Euler method, and returns their ends.

Usage

```
stepAheadWn2D(x0, mu, A, sigma, rho = 0, M = 100L, N = 100L,
  delta = 0.01, type = 1L, maxK = 2L, expTrc = 30)
```

Arguments

x0	matrix of size c(nx0, 2) giving the initial points.
mu	a vector of length 2 giving the mean.
A	drift matrix of size c(2, 2).
sigma	vector of length 2 containing the **square root** of the diagonal of Σ, the diffusion matrix.
rho	correlation coefficient of Σ.
M	number of Monte Carlo replicates.
N	number of discretization steps.
delta	discretization step.
type	integer giving the type of diffusion. Currently, only 1 for WN and 2 for vM are supported.
maxK	maximum absolute value of the windings considered in the computation of the WN.
expTrc	truncation for exponential: exp(x) with x <= -expTrc is set to zero. Defaults to 30.

Value

An array of size c(nx0, 2, M) containing the M trajectory ends for each starting value x0.

Examples

```
N <- 100
nx0 <- 3
x0 <- seq(-pi, pi, l = nx0 + 1)[-(nx0 + 1)]
x0 <- as.matrix(expand.grid(x0, x0))
nx0 <- nx0^2
set.seed(12345678)
samp1 <- euler2D(x0 = x0, mu = c(0, 0), A = rbind(c(3, 1), 1:2),
                 sigma = c(1, 1), N = N, delta = 0.01, type = 2)
plot(x0[, 1], x0[, 2], xlim = c(-pi, pi), ylim = c(-pi, pi), pch = 16,
     col = rainbow(nx0))
for (i in 1:nx0) linesTorus(samp1[i, 1, ], samp1[i, 2, ],
                            col = rainbow(nx0, alpha = 0.75)[i])
set.seed(12345678)
samp2 <- stepAheadWn2D(x0 = x0, mu = c(0, 0), A = rbind(c(3, 1), 1:2),
                       sigma = c(1, 1), M = 2, N = N, delta = 0.01,
                       type = 2)
points(samp2[, 1, 1], samp2[, 2, 1], pch = 16, col = rainbow(nx0))
samp1[, , N + 1]
samp2[, , 1]
```

`toPiInt` *Wrapping of radians to its principal values*

Description

Utilities for transforming a reals into $[-\pi, \pi)$, $[0, 2\pi)$ or $[a, b)$.

Usage

```
toPiInt(x)

to2PiInt(x)

toInt(x, a, b)
```

Arguments

x	a vector, matrix or object for whom `Arithmetic` is defined.
a, b	the lower and upper limits of $[a, b)$.

Details

Note that b is **excluded** from the result, see examples.

Value

The wrapped vector in the chosen interval.

Examples

```
# Wrapping of angles
x <- seq(-3 * pi, 5 * pi, l = 100)
toPiInt(x)
to2PiInt(x)

# Transformation to [1, 5)
x <- 1:10
toInt(x, 1, 5)
toInt(x + 1, 1, 5)

# Transformation to [1, 5]
toInt(x, 1, 6)
toInt(x + 1, 1, 6)
```

torusAxis *Draws pretty axis labels for circular variables*

Description

Wrapper for drawing pretty axis labels for circular variables. To be invoked after `plot` with `axes = FALSE` has been called.

Usage

```
torusAxis(sides = 1:2, twoPi = FALSE, ...)
```

Arguments

`sides`	an integer vector specifying which side of the plot the axes are to be drawn on. The axes are placed as follows: 1 = below, 2 = left, 3 = above, and 4 = right.
`twoPi`	flag indicating that $[0, 2\pi)$ is the support, instead of $[-\pi, \pi)$.
`...`	further parameters passed to `axis`.

Details

The function calls `box`.

Value

This function is usually invoked for its side effect, which is to add axes to an already existing plot.

Examples

```
grid <- seq(-pi, pi, l = 100)
plotSurface2D(grid, grid, f = function(x) sin(x[1]) * cos(x[2]),
              nLev = 20, axes = FALSE)
torusAxis()
```

torusAxis3d *Draws pretty axis labels for circular variables*

Description

Wrapper for drawing pretty axis labels for circular variables. To be invoked after `plot3d` with `axes = FALSE` and `box = FALSE` has been called.

Usage

```
torusAxis3d(sides = 1:3, twoPi = FALSE, ...)
```

Arguments

sides	an integer vector specifying which side of the plot the axes are to be drawn on. The axes are placed as follows: 1 = x, 2 = y, 3 = z.
twoPi	flag indicating that $[0, 2\pi)$ is the support, instead of $[-\pi, \pi)$.
...	further parameters passed to axis3d.

Details

The function calls box3d.

Value

This function is usually invoked for its side effect, which is to add axes to an already existing plot.

Examples

```
if (requireNamespace("rgl")) {
  n <- 50
  x <- toPiInt(rnorm(n, mean = seq(-pi, pi, l = n), sd = 0.5))
  y <- toPiInt(rnorm(n, mean = seq(-pi, pi, l = n), sd = 0.5))
  z <- toPiInt(x + y + rnorm(n, mean = seq(-pi, pi, l = n), sd = 0.5))
  rgl::plot3d(x, y, z, xlim = c(-pi, pi), ylim = c(-pi, pi),
              zlim = c(-pi, pi), col = rainbow(n), size = 2,
              box = FALSE, axes = FALSE)
  torusAxis3d()
}
```

---

unwrapCircSeries *Unwrapping of circular time series*

Description

Completes a circular time series to a linear one by computing the closest wind numbers. Useful for plotting circular trajectories with crossing of boundaries.

Usage

```
unwrapCircSeries(x)
```

Arguments

x	wrapped angles. Must be a vector or a matrix, see details.

Details

If x is a matrix then the unwrapping is carried out row-wise, on each column separately.

Value

A vector or matrix containing a choice of unwrapped angles of x that maximizes linear continuity.

Examples

```
# Vectors
x <- c(-pi, -pi/2, pi - 0.1, -pi + 0.2)
u <- unwrapCircSeries(x)
max(abs(toPiInt(u) - x))
plot(toPiInt(x), ylim = c(-pi, pi))
for(k in -1:1) lines(u + 2 * k * pi)

# Matrices
N <- 100
set.seed(234567)
x <- t(euler2D(x0 = rbind(c(0, 0)), A = diag(c(1, 1)), sigma = rep(1, 2),
                mu = c(pi, pi), N = N, delta = 1, type = 2)[1, , ])
u <- unwrapCircSeries(x)
max(abs(toPiInt(u) - x))
plot(x, xlim = c(-pi, pi), ylim = c(-pi, pi))
for(k1 in -3:3) for(k2 in -3:3) lines(u[, 1] + 2 * k1 * pi,
                                      u[, 2] + 2 * k2 * pi, col = gray(0.5))
```

Index

a1Inv *(*logBesselI0Scaled*)*, 44
alphaToA, 3, *7*, *26, 27*, *31*, *35*, *45*, *47*, *60, 61*, *64*, *66*
approxMleWn1D, 4
approxMleWn2D, 5
approxMleWnPairs, 7
Arithmetic, *76*
arrow3d, *43*
arrows, *41–43*
aToAlpha *(*alphaToA*)*, 3
axis, *77*
axis3d, *78*

besselIasym, *44*
box, *77*
box3d, *78*

constBvm *(*dBvm*)*, 12
constJp *(*dJp*)*, 14
covstOu, *65*
covstOu *(*dTpdOu*)*, 28
covtMou, *27*, *64*
covtMou *(*dTpdMou*)*, 27
crankNicolson1D, 8, *29*, *52*
crankNicolson2D, 10, *31*

dBvm, 12
diff, *13*
diffCirc, 13
dJp, 14
dPsTpd, 15
driftJp, 19
driftMixIndVm, 20, *21*
driftMixVm, *20*, 21
driftMvm, 22, *31*
driftWn, 23
driftWn1D, *23*, 24
driftWn2D, *23*, 25
dStatWn2D, 26
dTpdMou, 27, *28*
dTpdOu, *27*, 28
dTpdPde1D, 29
dTpdPde2D, 30
dTpdWou, 32, *34, 35*
dTpdWou1D, *32*, 33
dTpdWou2D, *32*, 35
dVm, 37
dWn1D, 38

euler1D, 39
euler2D, 40

forwardSweepPeriodicTridiag *(*solveTridiag*)*, 71
forwardSweepTridiag *(*solveTridiag*)*, 71

integrateSimp1D *(*periodicTrapRule1D*)*, 55
integrateSimp2D *(*periodicTrapRule1D*)*, 55
integrateSimp3D *(*periodicTrapRule1D*)*, 55

linesCirc, 41
linesTorus, 42
linesTorus3d, 43
logBesselI0Scaled, 44
logLikWouPairs, 45

meantMou, *27*, *64*
meantMou *(*dTpdMou*)*, 27
meantOu, *28*, *65*
meantOu *(*dTpdOu*)*, 28
mleMou, 46, *51*
mleOptimWrapper, *5–7*, *16*, 47, *47*, *50–52*, *54*, *58*
mleOu, *47*, 50
mlePde1D, 51
mlePde2D, 53
momentMatchWnVm, *5*, *15*, *34*, *38*, *58*
momentMatchWnVm *(*scoreMatchWnBvm*)*, 68

nlm, *48*

optim, *4*, *6, 7*, *47, 48*, *50*, *52*, *54*, *58*

par, *41, 42*
periodicTrapRule1D, 55
periodicTrapRule2D
        *(*periodicTrapRule1D*)*, 55
periodicTrapRule3D
        *(*periodicTrapRule1D*)*, 55
psMle, 57

rStatWn2D, 60
rTpdWn2D, 61
rTrajLangevin, 62
rTrajMou, 63, *65*
rTrajOu, 64, *64*
rTrajWn1D, 65
rTrajWn2D, 66

safeSoftMax, 67
scoreMatchWnBvm, *5*, *15*, *58*, 68
scoreMatchWnVm *(*scoreMatchWnBvm*)*, 68
sdetorus, 69
sdetorus-package *(*sdetorus*)*, 69
segments, *41–43*
sigmaDiff, 70
solvePeriodicTridiag, *9*, *11*
solvePeriodicTridiag *(*solveTridiag*)*, 71
solveTridiag, 71
solveTridiagMatConsts *(*solveTridiag*)*, 71
stepAheadWn1D, 73
stepAheadWn2D, 74

to2PiInt *(*toPiInt*)*, 76
toInt *(*toPiInt*)*, 76
toPiInt, 76
torusAxis, 77
torusAxis3d, 77

unwrapCircSeries, 78

vartOu, *28*
vartOu *(*dTpdOu*)*, 28