Package ‘Sim.DiffProc’

October 18, 2018

Type    Package
Version  4.2
Date     2018-10-18
Title    Simulation of Diffusion Processes
Author   Arsalane Chouaib Guidoum [cre, aut],
          Kamal Boukhetala [aut]
Maintainer Arsalane Chouaib Guidoum <acguidoum@usthb.dz>
Depends  R (>= 2.15.1)
Imports  Deriv (>= 3.8.0), MASS (>= 7.3-30), parallel
Suggests deSolve (>= 1.11), knitr (>= 1.10), rgl (>= 0.93.991),
         rmarkdown (>= 0.8), scatterplot3d (>= 0.3-36), sm (>= 2.2-5.3)
VignetteBuilder knitr
Encoding UTF-8
Description It provides users with a wide range of tools to simulate, estimate, analyze, and visualize the dynamics of stochastic differential systems in both forms Ito and Stratonovich. Statistical analysis with parallel Monte Carlo and moment equations methods of SDEs. Enabled many searchers in different domains to use these equations to modeling practical problems in financial and actuarial modeling and other areas of application, e.g., modeling and simulate of first passage time problem in shallow water using the attractive center (Boukhetala K, 1996) ISBN:1-56252-342-2.
License  GPL (>= 2)
Classification/MSC  37H10, 37M10, 60H05, 60H10, 60H35, 60J60, 65C05,
                    68N15, 68Q10
NeedsCompilation yes
Repository CRAN
Date/Publication 2018-10-17 22:10:03 UTC
### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sim.DiffProc-package</td>
<td>2</td>
</tr>
<tr>
<td>BM</td>
<td>14</td>
</tr>
<tr>
<td>bridgesde1d</td>
<td>16</td>
</tr>
<tr>
<td>bridgesde2d</td>
<td>19</td>
</tr>
<tr>
<td>bridgesde3d</td>
<td>22</td>
</tr>
<tr>
<td>fitsde</td>
<td>26</td>
</tr>
<tr>
<td>fptsde1d</td>
<td>31</td>
</tr>
<tr>
<td>fptsde2d</td>
<td>34</td>
</tr>
<tr>
<td>fptsde3d</td>
<td>38</td>
</tr>
<tr>
<td>HWV</td>
<td>41</td>
</tr>
<tr>
<td>Irates</td>
<td>43</td>
</tr>
<tr>
<td>MCM.sde</td>
<td>44</td>
</tr>
<tr>
<td>MEM.sde</td>
<td>48</td>
</tr>
<tr>
<td>moment</td>
<td>52</td>
</tr>
<tr>
<td>plot2d</td>
<td>53</td>
</tr>
<tr>
<td>rsde1d</td>
<td>54</td>
</tr>
<tr>
<td>rsde2d</td>
<td>56</td>
</tr>
<tr>
<td>rsde3d</td>
<td>59</td>
</tr>
<tr>
<td>snssde1d</td>
<td>62</td>
</tr>
<tr>
<td>snssde2d</td>
<td>65</td>
</tr>
<tr>
<td>snssde3d</td>
<td>69</td>
</tr>
<tr>
<td>st.int</td>
<td>74</td>
</tr>
<tr>
<td>TEX.sde</td>
<td>77</td>
</tr>
</tbody>
</table>

| Index                     | 79   |

---

#### Description

A package for symbolic and numerical computations on scalar and multivariate systems of stochastic differential equations. It provides users with a wide range of tools to simulate, estimate, analyze, and visualize the dynamics of these systems in both forms Itô and Stratonovich. Statistical analysis with parallel Monte-Carlo and moment equations methods of SDE’s. Enabled many searchers in different domains to use these equations to modeling practical problems in financial and actuarial modeling and other areas of application, e.g., modeling and simulate of first passage time problem in shallow water using the attractive center (Boukhetala K, 1996) ISBN:1-56252-342-2.

#### Details

<table>
<thead>
<tr>
<th>Package:</th>
<th>Sim.DiffProc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Package</td>
</tr>
<tr>
<td>Version:</td>
<td>4.2</td>
</tr>
<tr>
<td>Date:</td>
<td>2018-10-18</td>
</tr>
<tr>
<td>License:</td>
<td>GPL (&gt;= 2)</td>
</tr>
</tbody>
</table>
There are main types of functions in this package:

1. Simulation of solution to 1,2 and 3-dim stochastic differential equations of Itô and Stratonovich
types, with different methods.
2. Simulation of solution to 1,2 and 3-dim diffusion bridge of Itô and Stratonovich types, with
different methods.
3. Simulation the first-passage-time (f.p.t) in 1,2 and 3-dim sde of Itô and Stratonovich types.
4. Calculate symbolic ODE’s of moment equations (means and variances-covariance) for 1,2 and
3-dim SDE’s.
5. Monte-Carlo replicates of a statistic applied to 1,2 and 3-dim SDE’s at any time t.
6. Computing the basic statistics (mean, var, median, ...) of the processes at any time t using the
Monte Carlo method.
7. Random number generators (RN’s) to generate 1,2 and 3-dim sde of Itô and Stratonovich
types.
8. Approximate the transition density 1,2 and 3-dim of the processes at any time t.
9. Approximate the density of first-passage-time in 1,2 and 3-dim SDE’s.
10. Computing the stochastic integrals of Itô and Stratonovich types.
11. Estimate drift and diffusion parameters by the method of maximum pseudo-likelihood of the
1-dim stochastic differential equation.
12. Converting Sim.DiffProc objects to LaTeX.
13. Displaying an object inheriting from class "sde" (1,2 and 3 dim).

Main Features

Stochastic integrals:

We consider a simple example to simulation Itô integral, used \texttt{st.int} function:

\[
\int_{t_0}^{t} W^n_s \, dW_s = \frac{1}{n+1} \left[ W^{n+1}_t - W^{n+1}_{t_0} \right] - \frac{n}{2} \int_{t_0}^{t} W^{n-1}_s \, ds
\]

And the Stratonovich integral

\[
\int_{t_0}^{t} W^n_s \circ dW_s = \frac{1}{n+1} \left[ W^{n+1}_t - W^{n+1}_{t_0} \right]
\]

R> f <- expression( w )
R> Itô <- st.int(f,type="Ito",M=500,lower=0,upper=1)
R> Itô
Itô integral:
\[
X(t) = \text{integral} (f(s, w) \ast dw(s)) \\
f(t, w) = w
\]
Summary:
- Number of subinterval: N = 1001
- Number of simulation: M = 500
- Limits of integration: t in [0, 1]

R> summary(Itô)
Monte-Carlo Statistics for integral(f(s, w) \ast dw(s)) at time t = 1
\[
f(t, w) = w
\]
Mean 0.01330
Variance 0.51102
Median -0.28645
Mode -0.42772
First quartile -0.44666
Third quartile 0.22534
Minimum -0.55198
Maximum 4.38802
Skewness 2.27133
Kurtosis 9.27393
Coef-variation 53.75783
3th-order moment 0.82972
4th-order moment 2.42178
5th-order moment 7.60355
6th-order moment 26.72897

R> str <- st.int(f,type="str",M=500,lower=0,upper=1)
R> str
Stratonovich integral:
\[
X(t) = \text{integral} (f(s, w) \circ dw(s)) \\
f(t, w) = w
\]
Summary:
- Number of subinterval: N = 1001
- Number of simulation: M = 500
- Limits of integration: t in [0, 1]

R> summary(str)
Monte-Carlo Statistics for integral (f(s, w) \circ dw(s)) at time t = 1
\[
f(t, w) = w
\]
Mean 0.55655
Variance 0.66663
Median 0.21223
Mode 0.08249
First quartile 0.04269
Third quartile 0.79322
Minimum 0.00000
Maximum 6.70508
SDE's 1,2 and 3-dim:

There are thus two widely used types of stochastic calculus, Stratonovich and Ito, differing in respect of the stochastic integral used. Modelling issues typically dictate which version in appropriate, but once one has been chosen a corresponding equation of the other type with the same solutions can be determined. Thus it is possible to switch between the two stochastic calculi. Specifically, the processes \( \{X_t, t \geq 0\} \) solution to the Ito SDE:

\[
dX_t = f(t, X_t)dt + g(t, X_t)dW_t
\]

where \( \{W_t, t \geq 0\} \) is the standard Wiener process or standard Brownian motion, the drift \( f(t, X_t) \) and diffusion \( g(t, X_t) \) are known functions that are assumed to be sufficiently regular (Lipschitz, bounded growth) for existence and uniqueness of solution; has the same solutions as the Stratonovich SDE:

\[
dX_t = f(t, X_t)dt + g(t, X_t) \circ dW_t
\]

with the modified drift coefficient

\[
f(t, X_t) = f(t, X_t) - \frac{1}{2} g(t, X_t) \frac{\partial g}{\partial x}(t, X_t)
\]

The following examples for different methods of simulation of SDEs (1,2 and 3-dim) use the \texttt{snssde1d}, \texttt{snssde2d} and \texttt{snssde3d} functions.

```r
R> # 1-dim sde
R> f <- expression(2*(3-x))
R> g <- expression(2*x)
R> res1 <- snssde1d(drift=f,diffusion=g,M=1000,x0=1)
R> res1
Itô Sde 1D:
   | dX(t) = 2 * (3 - X(t)) * dt + 2 * X(t) * dW(t)
Method:
   | Euler scheme with order 0.5
Summary:
   | Size of process | N = 1001.
   | Number of simulation | M = 1000.
   | Initial value | x0 = 1.
   | Time of process | t in [0,1].
   | Discretization | Dt = 0.001.
R> res2 <- snssde1d(drift=f,diffusion=g,M=1000,x0=1,type="str")
R> res2
Stratonovitch Sde 1D:
```
Sim.DiffProc-package

\[ \frac{dX(t)}{dt} = 2(3 - X(t)) dt + 2X(t) dW(t) \]

Method:
| Euler scheme with order 0.5
Summary:
| Size of process | N = 1001. 
| Number of simulation | M = 1000. 
| Initial value | x0 = 1. 
| Time of process | t in [0,1]. 
| Discretization | Dt = 0.001.

R> ## 2-dim sde
R> fx <- expression(x-y, y-x)
R> gx <- expression(2*y, 2*x)
R> res2d <- snssde2d(drift=fx,diffusion=gx,x0=c(1,1))
R> res2d

R> ## 3-dim sde
R> fx <- expression(y, 0, 0)
R> gx <- expression(z, 1, 1)
R> res3d <- snssde3d(drift=fx,diffusion=gx,M=1000)
R> res3d

R> plot2d(res2d)

R> plot3d(res3d)
Bridge SDE’s 1,2 and 3-dim:

Simulation of bridge SDEs (1,2 and 3-dim) with `bridgesde1d`, `bridgesde2d` and `bridgesde3d` functions.

```
R> ## 1-dim bridge sde
R> f <- expression(2*(1-x))
R> g <- expression(1)
R> mod1 <- bridgesde1d(drift=f,diffusion=g,x0=2,y=1,M=1000)
R> summary(mod1)  ## Monte-Carlo statistics at T/2=0.5

Monte-Carlo Statistics for X(t) at time t = 0.5
  Crossing realized 843 among 1000

Mean         1.31263
Variance     0.18352
Median       1.30504
Mode         1.46713
First quartile 1.02722
Third quartile 1.60984
Minimum      -0.22080
Maximum      2.83339
Skewness     0.01722
Kurtosis     3.19888
Coef-variation 0.32636
3th-order moment 0.00135
4th-order moment 0.10773
5th-order moment 0.00645
6th-order moment 0.11233
```

```
R> plot(mod1)
```

Density of X(t-t0)|X(t0)=x0 at time t = 1

Data: x (843 obs.); Bandwidth 'bw' = 0.2339

```
x          f(x)
Min.  :0.29822     Min.  :0.01913
1st Qu.:0.64911    1st Qu.:0.13600
```
Estimate the parameters of 1-dim sde:

Consider a process solution of the general stochastic differential equation:

$$dX_t = f(t, X_t, \theta)dt + g(t, X_t, \theta)dW_t$$

The package Sim.DiffProc implements the function \texttt{fitsde} of estimate drift and diffusion parameters $\theta = (\theta_1, \theta_2, \ldots, \theta_p)$ with different methods of maximum pseudo-likelihood of the 1-dim stochastic differential equation.

An example we use a real data, fit with the CKLS model:

$$dX_t = (\theta_1 + \theta_2 X_t)dt + \theta_3 \theta_4 t dW_t$$

we estimate the vector of parameters $\theta = (\theta_1, \theta_2, \theta_3, \theta_4)$, using Euler pseudo-likelihood.

```R
R> fitted <- fitsde(rates, drift = fx, diffusion = gx, pmle = "euler", start = list(theta1 = 1, theta2 = 1, theta3 = 1, theta4 = 1), optim.method = "L-BFGS-B")
R> summary(fitted)
```

Coefficients:

|         |     Estimate |       Std. Error |     z value |   Pr(>|z|) |
|---------|--------------|-----------------|-------------|-----------|
| theta1  | 2.0769516    | 0.2631871       | 7.811917   | < 2.2e-16 |
| theta2  | 0.1302158    | 0.1302158       | 0.997867   | 5.87e-01  |
| theta3  | -0.1302158   | 0.1302158       | -1.000000  | 8.35e-01  |
| theta4  | 1.4513173    | 0.1302158       | 11.111111  | < 2.2e-16 |

R> summary(fitted)

Pseudo maximum likelihood estimation

Method: Euler

```R
R> fitted <- fitsde(rates, drift = fx, diffusion = gx, pmle = "euler", start = list(theta1 = 1, theta2 = 1, theta3 = 1, theta4 = 1),
```
optim.method = "L-BFGS-B")
Coefficients:
    Estimate Std. Error
theta1  2.0769516  0.98838467
theta2  0.2631871  0.19544290
theta3  0.1302158  0.02523105
theta4  1.4513173  0.10323740

-2 log L: 475.7572
R> coef(fitmod)
   theta1  theta2  theta3  theta4
 2.0769516 -0.2631871  0.1302158  1.4513173
R> logLik(fitmod)
      [1] 237.3876
R> AIC(fitmod)
      [1] 483.7572
R> BIC(fitmod)
      [1] 487.1514
R> vcov(fitmod)
           theta1   theta2   theta3   theta4
theta1  0.9769042534 -1.843595796 -2.714334391  0.0011374342
theta2 -1.843595796  3.819793002  6.198499455 -0.0002165286
theta3 -2.714334391  6.198499455 11.698499455 -0.0025457493
theta4  0.0011374342 -2.545749934 -2.945749934  0.0106579616
R> confint(fitmod,level=0.95)
           2.5 %     97.5 %
theta1  0.1379732  4.0141499
theta2 -0.64624812 0.1198740
theta3  0.00876388 0.1796678
theta4  1.24897569 1.6536589

Transition density and Random number generators (RN’s) for 1, 2 and 3-dim sde:

Simulation M-sample for the random variable \( X_{at} \) at time \( t = at \) by a simulated 1, 2 and 3-dim sde, using the functions \( rsde1d, rsde2d \) and \( rsde3d \). And \( dsde1d, dsde2d \) and \( dsde3d \) returns a kernel approximate of transitional densities.

R> f <- expression(-2*(x<=0)+2*(x>0))
R> g <- expression(0.5)
R> res1 <- snsde1d(drift=f,diffusion=g,M=50,type="str",T=10)
R> x <- rsde1d(res1,at=10)
R> x
First-passage-time (f.p.t) in 1,2 and 3-dim sde

The functions \texttt{fptsde1d} (\texttt{fptsde2d} and \texttt{fptsde3d} for 2 and 3-dim) returns a random variable \( \tau(X(t),S(t)) \) "first passage time", is defined as:

\[
\tau(X(t),S(t)) = \begin{cases} 
  t \geq 0; & X(t) \geq S(t), \text{ if } X(t_0) < S(t_0) \\
  t \geq 0; & X(t) \leq S(t), \text{ if } X(t_0) > S(t_0)
\end{cases}
\]

And \texttt{dfptsde1d}, \texttt{dfptsde2d} and \texttt{dfptsde3d} returns a kernel density approximation for first passage time. with \( S(t) \) is through a continuous boundary (barrier).

\begin{verbatim}
R> f <- expression( 0.5*x^2 )
R> g <- expression( sqrt(1+x^2) )
R> St <- expression(-0.5*sqrt(t)+exp(t^2))
R> mod <- snessde1d(drift=f, diffusion=g, x0=2, M=1000)
R> fptmod <- fptsde1d(mod, boundary=St)
R> fptmod

Ito Sde 1D:
| dX(t) = 0.5 * X(t) * t * dt + sqrt(1 + X(t)^2) * dW(t) |
| t in [0,1].

Boundary:
| S(t) = -0.5 * sqrt(t) + exp(t^2)

F.P.T:
| T(S(t),X(t)) = inf{t >= 0 : X(t) <= -0.5 * sqrt(t) + exp(t^2) } |
| Crossing realized 738 among 1000.
R> summary(fptmod)

Monte-Carlo Statistics of F.P.T:
|T(S(t),X(t)) = inf{t >= 0 : X(t) <= -0.5 * sqrt(t) + exp(t^2) } |

Mean 0.47742
Variance 0.07348
Median 0.44831
Mode 0.18582
First quartile 0.23746
Third quartile 0.71321
Minimum 0.03002
\end{verbatim}
Maximum 0.98877
Skewness 0.22793
Kurtosis 1.79959
Coef-variation 0.56778
3th-order moment 0.00454
4th-order moment 0.00972
5th-order moment 0.00134
6th-order moment 0.00158

R> den <- dfptsdeld(mod,boundary=St)
R> den

Kernel density for the F.P.T of X(t)
T(S,X) = \inf\{t \geq 0 : X(t) \leq -0.5 * sqrt(t) + \exp(t^2)\}

Data: fpt (738 obs.); Bandwidth 'bw' = 0.0828

<table>
<thead>
<tr>
<th>x</th>
<th>f(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min.</td>
<td>-0.2095</td>
</tr>
<tr>
<td>1st Qu.</td>
<td>0.1458</td>
</tr>
<tr>
<td>Median</td>
<td>0.5010</td>
</tr>
<tr>
<td>Mean</td>
<td>0.5010</td>
</tr>
<tr>
<td>3rd Qu.</td>
<td>0.8563</td>
</tr>
<tr>
<td>Max.</td>
<td>1.2116</td>
</tr>
</tbody>
</table>

R> ## fpt in 2 and 3-dim sde
R> example(dfptsdeld)
R> example(dfptsdeld)

For other examples see demo(Sim.DiffProc), and for an overview of this package, see browseVignettes('Sim.DiffProc') for more informations.

Requirements
R version >= 3.0.0

Licence
This package and its documentation are usable under the terms of the "GNU General Public License", a copy of which is distributed with the package.

Author(s)
A.C. Guidoum <acguidoum@usthb.dz> and K. Boukhetala <kboukhetala@usthb.dz> (Dept. Probability and Statistics, USTHB, Algeria).
Please send comments, error reports, etc. to the author via the addresses email.

References


See Also

sde, ymia, QPot, DiffusionRgqd, fptdApprox.

---

**BM**  
*Brownian motion, Brownian bridge, geometric Brownian motion, and arithmetic Brownian motion simulators*

**Description**

The (S3) generic function for simulation of brownian motion, brownian bridge, geometric brownian motion, and arithmetic brownian motion.

**Usage**

```r
BM(N, ...)  
BB(N, ...)  
GBM(N, ...)  
ABM(N, ...)  
```

```r
# Default S3 method:  
BM(N =1000,M=1,x0=0,t0=0,T=1,Dt=NULL, ...)  
# Default S3 method:  
BB(N =1000,M=1,x=0,y=0,t0=0,T=1,Dt=NULL, ...)  
# Default S3 method:  
GBM(N =1000,M=1,x0=1,t0=0,T=1,Dt=NULL,theta=1,sigma=1, ...)  
# Default S3 method:  
ABM(N =1000,M=1,x0=0,t0=0,T=1,Dt=NULL,theta=1,sigma=1, ...)  
```

**Arguments**

- **N**  
  number of simulation steps.
- **M**  
  number of trajectories.
- **x0**  
  initial value of the process at time $t_0$.
- **y**  
  terminal value of the process at time $T$ of the BB.
- **t0**  
  initial time.
- **T**  
  final time.
- **Dt**  
  time step of the simulation (discretization). If it is **NULL** a default $\Delta t = \frac{T-t_0}{N}$.
- **theta**  
  the interest rate of the ABM and GBM.
- **sigma**  
  the volatility of the ABM and GBM.
- **...**  
  potentially further arguments for (non-default) methods.
Details

The function BM returns a trajectory of the standard Brownian motion (Wiener process) in the time interval \([t_0, T]\). Indeed, for \(W(dt)\) it holds true that \(W(dt) \rightarrow W(dt) - W(0) \rightarrow \mathcal{N}(0, dt)\), where \(\mathcal{N}(0, 1)\) is normal distribution Normal.

The function BB returns a trajectory of the Brownian bridge starting at \(x_0\) at time \(t_0\) and ending at \(y\) at time \(T\); i.e., the diffusion process solution of stochastic differential equation:

\[
dX_t = \frac{y - X_t}{T - t} dt + dW_t
\]

The function GBM returns a trajectory of the geometric Brownian motion starting at \(x_0\) at time \(t_0\); i.e., the diffusion process solution of stochastic differential equation:

\[
dX_t = \theta X_t dt + \sigma X_t dW_t
\]

The function ABM returns a trajectory of the arithmetic Brownian motion starting at \(x_0\) at time \(t_0\); i.e., the diffusion process solution of stochastic differential equation:

\[
dX_t = \theta dt + \sigma dW_t
\]

Value

\(X\) an visible ts object.

Author(s)

A.C. Guidoum, K. Boukhetala.

References


See Also

This functions BM, BBridge and GBM are available in other packages such as "sde".

Examples

```r
op <- par(mfrow = c(2, 2))

## Brownian motion
set.seed(1234)
X <- BM(M = 100)
plot(X, plot.type="single")
lines(as.vector(time(X)), rowMeans(X), col="red")
```
bridgesde1d

Simulation of 1-D Bridge SDE

Description

The (S3) generic function `bridgesde1d` for simulation of 1-dim bridge stochastic differential equations, Itô or Stratonovich type, with different methods.

Usage

`bridgesde1d(N, ...)`

## Default S3 method:

`bridgesde1d(N = 1000, M = 1, x0 = 0, y = 0, t0 = 0, T = 1, Dt = NULL, 
drift = 0.5, diffusion = 0.5, type = c("ito", "str"), method = c("euler", "milstein", "predcorr", "smilstein", "taylor", "heun", "rk1", "rk2", "rk3"), ...)`

## S3 method for class 'bridgesde1d'

`summary(object, at, digits = NULL, ...)`

## S3 method for class 'bridgesde1d'

`time(x, ...)`

## S3 method for class 'bridgesde1d'

`mean(x, at, ...)`

## S3 method for class 'bridgesde1d'

`Median(x, at, ...)`

## S3 method for class 'bridgesde1d'

`Mode(x, at, ...)`

## S3 method for class 'bridgesde1d'

`lines(x, plot.type = "single", col = "red")`
Arguments

N  number of simulation steps.
M  number of trajectories.
x0  initial value of the process at time t0.
y  terminal value of the process at time T.
t0  initial time.
T  final time.
Dt  time step of the simulation (discretization). If it is NULL a default \[ \Delta t = \frac{T-t_0}{N} \].
drift  drift coefficient: an expression of two variables t and x.
diffusion  diffusion coefficient: an expression of two variables t and x.
alpha, mu  weight of the predictor-corrector scheme; the default alpha = 0.5 and mu = 0.5.
type  if type="ito" simulation diffusion bridge of Itô type, else type="str" simulation diffusion bridge of Stratonovich type; the default type="ito".
method  numerical methods of simulation, the default method = "euler"; see sssde1d.
x, object  an object inheriting from class "bridgesde1d".
at  time between t0 and T. Monte-Carlo statistics of the solution \( X_t \) at time at. The default at = T/2.
digits  integer, used for number formatting.
...  potentially further arguments for (non-default) methods.
Details

The function `bridgesde1d` returns a trajectory of the diffusion bridge starting at \( x \) at time \( t_0 \) and ending at \( y \) at time \( T \).

The methods of approximation are classified according to their different properties. Mainly two criteria of optimality are used in the literature: the strong and the weak (orders of) convergence. The method of simulation can be one among: Euler-Maruyama Order 0.5, Milstein Order 1, Milstein Second-Order, Predictor-Corrector method, Itô-Taylor Order 1.5, Heun Order 2 and Runge-Kutta Order 1, 2 and 3.

An overview of this package, see `browseVignettes('Sim.DiffProc')` for more informations.

Value

`bridgesde1d` returns an object inheriting from class "bridgesde1d".

- **X**: an invisible \( \text{ts} \) object.
- **drift**: drift coefficient.
- **diffusion**: diffusion coefficient.
- **C**: indices of crossing realized of \( X(t) \).
- **type**: type of sde.
- **method**: the numerical method used.

Author(s)

A.C. Guidoum, K. Boukhetala.

References


See Also

`bridgesde2d` and `bridgesde3d` for 2 and 3-dim.

`DBridge` in package "sde".

Examples

```r
## Example 1: Ito bridge sde
## Ito Bridge sde
## dX(t) = 2*(1-X(t)) *dt + dW(t)
## x0 = 2 at time t0=0 , and y = 1 at time T=1
set.seed(1234)

f <- expression(2*(1-x))
g <- expression(1)
```
bridgesde2d

Simulation of 2-D Bridge SDE's

Description

The (S3) generic function bridgesde2d for simulation of 2-dim bridge stochastic differential equations, Itô or Stratonovich type, with different methods.

Usage

bridgesde2d(N, ...)

## Default S3 method:
bridgesde2d(N = 1000, M = 1, x0 = c(0, 0),
y = c(0, 0), t0 = 0, T = 1, Dt=NULL, drift, diffusion,
alpha = 0.5, mu = 0.5,type = c("ito", "str"),method =
c("euler", "milstein", "predcorr", "smilstein", "taylor",
"heun", "rk1", "rk2", "rk3"), ...

## S3 method for class 'bridgesde2d'
summary(object, at,  
digits=NULL, ...)  
## S3 method for class 'bridgesde2d'
time(x, ...)  
## S3 method for class 'bridgesde2d'
mean(x, at, ...)  
## S3 method for class 'bridgesde2d'
Median(x, at, ...)  
## S3 method for class 'bridgesde2d'
Mode(x, at, ...)  
## S3 method for class 'bridgesde2d'
quantile(x, at, ...)  
## S3 method for class 'bridgesde2d'
kurtosis(x, at, ...)  
## S3 method for class 'bridgesde2d'
skewness(x, at, ...)  
## S3 method for class 'bridgesde2d'
min(x, at, ...)  
## S3 method for class 'bridgesde2d'
max(x, at, ...)  
## S3 method for class 'bridgesde2d'
moment(x, at, ...)  
## S3 method for class 'bridgesde2d'
cv(x, at, ...)  
## S3 method for class 'bridgesde2d'
bconfint(x, at, ...)  

## S3 method for class 'bridgesde2d'
plot(x, ...)  
## S3 method for class 'bridgesde2d'
lines(x, ...)  
## S3 method for class 'bridgesde2d'
points(x, ...)  
## S3 method for class 'bridgesde2d'
plot2d(x, ...)  
## S3 method for class 'bridgesde2d'
lines2d(x, ...)  
## S3 method for class 'bridgesde2d'
points2d(x, ...)

Arguments

N number of simulation steps.
bridgesde2d

number of trajectories.

x0 initial value (numeric vector of length 2) of the process \( X_t \) and \( Y_t \) at time \( t_0 \).
y terminal value (numeric vector of length 2) of the process \( X_t \) and \( Y_t \) at time \( T \).
t0 initial time.
T final time.
Dt time step of the simulation (discretization). If it is NULL a default \( \Delta t = \frac{T-t_0}{N} \).
drift drift coefficient: an expression of three variables \( t, x \) and \( y \) for process \( X_t \) and \( Y_t \).
diffusion diffusion coefficient: an expression of three variables \( t, x \) and \( y \) for process \( X_t \) and \( Y_t \).
alpha, mu weight of the predictor-corrector scheme; the default \( \alpha = 0.5 \) and \( \mu = 0.5 \).
type if type=\"ito\" simulation diffusion bridge of Itô type, else type=\"str\" simulation diffusion bridge of Stratonovich type; the default type=\"ito\".
method numerical methods of simulation, the default method = \"euler\"; see ssnsde2d.
x, object an object inheriting from class \"bridgesde2d\".
at time between \( t_0 \) and \( T \). Monte-Carlo statistics of the solution \( (X_t, Y_t) \) at time \( t \). The default at = \( T/2 \).
digits integer, used for number formatting.
... potentially further arguments for (non-default) methods.

Details

The function bridgesde2d returns a mts of the diffusion bridge starting at \( x \) at time \( t_0 \) and ending at \( y \) at time \( T \).

The methods of approximation are classified according to their different properties. Mainly two criteria of optimality are used in the literature: the strong and the weak (orders of) convergence. The method of simulation can be one among: Euler-Maruyama Order \( 0.5 \), Milstein Order 1, Milstein Second-Order, Predictor-Corrector method, Itô-Taylor Order 1.5, Heun Order 2 and Runge-Kutta Order 2 and 3.

An overview of this package, see browseVignettes('Sim.DiffProc') for more informations.

Value

bridgesde2d returns an object inheriting from class \"bridgesde2d\".

\( X, Y \) an invisible mts (2-dim) object \( (X(t), Y(t)) \).

driftX, driftY drift coefficient of \( X(t) \) and \( Y(t) \).

diffX, diffY diffusion coefficient of \( X(t) \) and \( Y(t) \).

\( C_X, C_Y \) indices of crossing realized of \( X(t) \) and \( Y(t) \).

type type of sde.

method the numerical method used.
bridgesde3d

Simulation of 3-D Bridge SDE’s

Description

The (S3) generic function `bridgesde3d` for simulation of 3-dim bridge stochastic differential equations, Itô or Stratonovich type, with different methods.
Usage

bridgesde3d(N, ...)
## Default S3 method:
bridgesde3d(N=1000, M=1, x0=c(0,0,0),
    y=c(0,0,0), t0 = 0, T = 1, Dt=NULL, drift, diffusion,
    alpha = 0.5, mu = 0.5, type = c("ito", "str"), method =
    c("euler", "milstein","predcorr","smilstein", "taylor",
    "heun","rk1", "rk2", "rk3"), ...)

## S3 method for class 'bridgesde3d'
summary(object, at,
    digits=NULL, ...)
## S3 method for class 'bridgesde3d'
time(x, ...)
## S3 method for class 'bridgesde3d'
mean(x, at, ...)
## S3 method for class 'bridgesde3d'
Median(x, at, ...)
## S3 method for class 'bridgesde3d'
Mode(x, at, ...)
## S3 method for class 'bridgesde3d'
quantile(x, at, ...)
## S3 method for class 'bridgesde3d'
kurtosis(x, at, ...)
## S3 method for class 'bridgesde3d'
skewness(x, at, ...)
## S3 method for class 'bridgesde3d'
min(x, at, ...)
## S3 method for class 'bridgesde3d'
max(x, at, ...)
## S3 method for class 'bridgesde3d'
moment(x, at, ...)
## S3 method for class 'bridgesde3d'
cv(x, at, ...)
## S3 method for class 'bridgesde3d'
bconfint(x, at, ...)

## S3 method for class 'bridgesde3d'
plot(x, ...)
## S3 method for class 'bridgesde3d'
lines(x, ...)
## S3 method for class 'bridgesde3d'
points(x, ...)
## S3 method for class 'bridgesde3d'
plot3D(x, display = c("persp","rgl"), ...)
Arguments

- N: number of simulation steps.
- M: number of trajectories.
- x0: initial value (numeric vector of length 3) of the process $X_t$, $Y_t$ and $Z_t$ at time $t_0$.
- y: terminal value (numeric vector of length 3) of the process $X_t$, $Y_t$ and $Z_t$ at time $T$.
- t0: initial time.
- T: final time.
- Dt: time step of the simulation (discretization). If it is NULL a default $\Delta t = \frac{T-t_0}{N}$.
- drift: drift coefficient: an expression of four variables $t$, $x$, $y$ and $z$ for process $X_t$, $Y_t$ and $Z_t$.
- diffusion: diffusion coefficient: an expression of four variables $t$, $x$, $y$ and $z$ for process $X_t$, $Y_t$ and $Z_t$.
- alpha: weight $\alpha$ of the predictor-corrector scheme; the default $\alpha = 0.5$.
- mu: weight $\mu$ of the predictor-corrector scheme; the default $\mu = 0.5$.
- type: if type="ito" simulation diffusion bridge of Itô type, else type="str" simulation diffusion bridge of Stratonovich type; the default type="ito".
- method: numerical methods of simulation, the default method = "euler"; see sssde3d.
- x, object: an object inheriting from class "bridgesde3d".
- at: time between $t_0$ and $T$. Monte-Carlo statistics of the solution $(X_t, Y_t, Z_t)$ at time at. The default at = $T/2$.
- digits: integer, used for number formatting.
- display: "persp" perspective and "rgl" plots.
- ...: potentially further arguments for (non-default) methods.

Details

The function bridgesde3d returns a mts of the diffusion bridge starting at $x$ at time $t_0$ and ending at $y$ at time $T$.

The methods of approximation are classified according to their different properties. Mainly two criteria of optimality are used in the literature: the strong and the weak (orders of) convergence. The method of simulation can be one among: Euler-Maruyama Order 0.5, Milstein Order 1, Milstein Second-Order, Predictor-Corrector method, Itô-Taylor Order 1.5, Heun Order 2 and Runge-Kutta Order 1, 2 and 3.

An overview of this package, see browseVignettes('Sim.DiffProc') for more informations.

Value

bridgesde3d returns an object inheriting from class "bridgesde3d".

- X, Y, Z: an invisible mts (3-dim) object $(X(t), Y(t), Z(t))$.
- driftX, drifty, driftz: drift coefficient of $X(t)$, $Y(t)$ and $Z(t)$.
bridgesde3d

diffx, diffy, diffz
diffusion coefficient of X(t), Y(t) and Z(t).
Cx, Cy, Cz
indices of crossing realized of X(t), Y(t) and Z(t).
type
type of sde.
method
the numerical method used.

Author(s)
A.C. Guidoum, K. Boukhetala.

References

See Also
bridgesde1d for simulation of 1-dim SDE. DBridge in package "sde".
bridgesde2d for simulation of 2-dim SDE.

Examples
```r
## dX(t) = 4*(-1-X(t))*Y(t) dt + 0.2 * dW1(t) ; x01 = 0 and y01 = 0
## dY(t) = 4*(1-Y(t)) *X(t) dt + 0.2 * dW2(t) ; x02 = -1 and y02 = -2
## dZ(t) = 4*(1-Z(t)) *Y(t) dt + 0.2 * dW3(t) ; x03 = 0.5 and y03 = 0.5
## W1(t), W2(t) and W3(t) three independent Brownian motion
set.seed(1234)

fx <- expression(4*(-1-x)*y, 4*(1-y)*x, 4*(1-z)*y)
gx <- rep(expression(0,2),3)

res <- bridgesde3d(x0=c(0,-1,0.5),y=c(0,-2,0.5),drift=fx,diffusion=gx,M=200)

summary(res) ## Monte-Carlo statistics at time T/2=0.5
summary(res,at=0.25) ## Monte-Carlo statistics at time 0.25
summary(res,at=0.75) ## Monte-Carlo statistics at time 0.75

plot(res,type="n")
lines(time(res),apply(res$X,1,mean),col=3,lwd=2)
lines(time(res),apply(res$Y,1,mean),col=4,lwd=2)
lines(time(res),apply(res$Z,1,mean),col=5,lwd=2)
legend("topleft",c(expression(E(X[t])),expression(E(Y[t])),
expression(E(Z[t]))),lty=1,inset = .01,col=c(3,4,5))

plot3D(res,display = "persp",main="3-dim bridge sde")
```
fitsde

Maximum Pseudo-Likelihood Estimation of 1-D SDE

Description

The (S3) generic function "fitsde" of estimate drift and diffusion parameters by the method of maximum pseudo-likelihood of the 1-dim stochastic differential equation.

Usage

fitsde(data, ...)  

## Default S3 method:  
fitsde(data, drift, diffusion, start = list(), pmle = c("euler","kessler",  
"ozaki", "shoji"), optim.method = "L-BFGS-B",  
lower = -Inf, upper = Inf, ...)  

## S3 method for class 'fitsde'  
summary(object, ...)  

## S3 method for class 'fitsde'  
coef(object, ...)  

## S3 method for class 'fitsde'  
vcov(object, ...)  

## S3 method for class 'fitsde'  
logLik(object, ...)  

## S3 method for class 'fitsde'  
AIC(object, ...)  

## S3 method for class 'fitsde'  
BIC(object, ...)  

## S3 method for class 'fitsde'  
confint(object, parm, level=0.95, ...)  

Arguments

data a univariate time series (ts class).
drift drift coefficient: an expression of two variables t, x and theta a vector of parameters of sde. See Examples.
diffusion diffusion coefficient: an expression of two variables t, x and theta a vector of parameters of sde. See Examples.
start named list of starting values for optimizer. See Examples.

pmle a character string specifying the method; can be either: "euler" (Euler pseudo-likelihood), "ozaki" (Ozaki pseudo-likelihood), "shoji" (Shoji pseudo-likelihood), and "kessler" (Kessler pseudo-likelihood).

optim.method the method for optim.

lower, upper bounds on the variables for the "Brent" or "L-BFGS-B" method.
object an object inheriting from class "fitsde."
parm: a specification of which parameters are to be given confidence intervals, either a vector of names (example `parm='theta1'`). If missing, all parameters are considered.

level: the confidence level required.

... potentially further arguments to pass to `optim`.

**Details**

The function `fitsde` returns a pseudo-likelihood estimators of the drift and diffusion parameters in 1-dim stochastic differential equation. The `optim` optimizer is used to find the maximum of the negative log pseudo-likelihood. An approximate covariance matrix for the parameters is obtained by inverting the Hessian matrix at the optimum.

The `pmle` of pseudo-likelihood can be one among: "euler": Euler pseudo-likelihood, "ozaki": Ozaki pseudo-likelihood, "shoji": Shoji pseudo-likelihood, and "kessler": Kessler pseudo-likelihood.

An overview of this package, see `browseVignettes('Sim.DiffProc')` for more informations.

**Value**

`fitsde` returns an object inheriting from `class "fitsde"`.

**Author(s)**

A.C. Guidoum, K. Boukhetala.

**References**


**See Also**

dcEuler, dcElerian, dcOzaki, dcShoji, dcKessler and dcSim for approximated conditional law of a diffusion process. gmm estimator of the generalized method of moments by Hansen, and HPloglik these functions are useful to calculate approximated maximum likelihood estimators when the transition density of the process is not known, in package "sde".

qMLE in package "yuima" calculate quasi-likelihood and ML estimator of least squares estimator.

**Examples**

```
# Example 1:

# Model GBM (BS)
# dX(t) = theta1 * X(t) * dt + theta2 * x * dW(t)
# Simulation of data
set.seed(1234)

X <- GBM(N = 1000, theta=4, sigma=1)
# Estimation: true theta=c(4,1)
fx <- expression(theta[1]*x)
gx <- expression(theta[2]*x)

fres <- fitsde(data=X, drift=fx, diffusion=gx, start = list(theta1=1, theta2=1),
               lower=c(0,0))

fres
summary(fres)
coef(fres)
loglik(fres)
AIC(fres)
BIC(fres)
vcov(fres)
confint(fres, level=0.95)
```

```
# Example 2:

# Nonlinear mean reversion (Ait-Sahalia) model
# dX(t) = (theta1 + theta2*x + theta3*x^2) * dt + theta4 * x*theta5 * dW(t)
# Simulation of the process X(t)
set.seed(1234)
```
fitsde

f <- expression(1 - 11*x + 2*x^2)
g <- expression(x^0.5)
res <- ssnssd1c(drift=f,diffusion=g,M=1,N=1000,Dt=0.001,x0=5)
mydata1 <- res

## Estimation
## true param theta= c(1,-11,2,1,0.5)
true <- c(1,-11,2,1,0.5)
mlle <- eval(formals(fitsde.default)$mlle)

fx <- expression(theta[1] + theta[2]*x + theta[3]*x^2)
gx <- expression(theta[4]*x^theta[5])

fres <- lapply(1:4, function(i) fitsde(mydata1, drift=fx, diffusion=gx,
  mlle=mlle[i], start = list(theta1=1,theta2=1,theta3=1,theta4=1,theta5=1),
  optim.method = "L-BFGS-B"))

Coef <- data.frame(true,do.call("cbind",lapply(1:4,function(i) coef(fres[[i]]))))
names(Coef) <- c("True",mlle)

Summary <- data.frame(do.call("rbind",lapply(1:4,function(i) logLik(fres[[i]]))),
  do.call("rbind",lapply(1:4,function(i) AIC(fres[[i]]))),
  do.call("rbind",lapply(1:4,function(i) BIC(fres[[i]]))),
  row.names=mlle)
names(Summary) <- c("logLik","AIC","BIC")

Coef
Summary

```
```

Example 3:

## dX(t) = (theta1*x*x+theta2*x+theta3*tan(x)) * dt + theta3*t * dW(t)
## Simulation of data
set.seed(1234)

f <- expression(2*x^2+tan(x))
g <- expression(1.25*t)
sim <- ssnssd1c(drift=f,diffusion=g,M=1,N=1000,Dt=0.001,x0=10)
mydata2 <- sim

## Estimation
## true param theta= c(2,-1,1.25)
true <- c(2,-1,1.25)

fx <- expression(theta[1]*x*x+theta[2]*x+theta[3]*tan(x))
gx <- expression(theta[3]*t)

fres <- lapply(1:4, function(i) fitsde(mydata2, drift=fx, diffusion=gx,
  mlle=mlle[i], start = list(theta1=1,theta2=1,theta3=1),
  optim.method = "L-BFGS-B"))

Coef <- data.frame(true,do.call("cbind",lapply(1:4,function(i) coef(fres[[i]]))))
names(Coef) <- c("True",mlle)

Summary <- data.frame(do.call("rbind",lapply(1:4,function(i) logLik(fres[[i]]))),
  do.call("rbind",lapply(1:4,function(i) AIC(fres[[i]]))),
  do.call("rbind",lapply(1:4,function(i) BIC(fres[[i]]))),
  row.names=mlle)
names(Summary) <- c("logLik","AIC","BIC")

Coef
Summary

```

Example 3:
```r
row.names=pmle)
names(Summary) <- c("logLik","AIC","BIC")
Coef
Summary

##### Example 4:

## Application to real data
## CKLS modele vs CIR modele
## CKLS (mod): dX(t) = (theta1+theta2* X(t))* dt + theta3 * X(t)^theta4 * dW(t)
## CIR (mod): dX(t) = (theta1+theta2* X(t))* dt + theta3 * sqrt(X(t)) * dW(t)
set.seed(1234)
data(Irates)
rates <- Irates,"r1"
rates <- window(rates, start=1964.471, end=1989.333)

fx1 <- expression(theta[1]+theta[2]*x)
gx1 <- expression(theta[3]*x^theta[4])
gx2 <- expression(theta[3]*sqrt(x))

fitmod1 <- fitsde(rates,drift=fx1,diffusion=gx1,pmle="euler",start = list(theta1=1,theta2=1, theta3=1,theta4=1),optim.method = "L-BFGS-B")
fitmod2 <- fitsde(rates,drift=fx1,diffusion=gx2,pmle="euler",start = list(theta1=1,theta2=1, theta3=1),optim.method = "L-BFGS-B")
summary(fitmod1)
summary(fitmod2)
coef(fitmod1)
coef(fitmod2)
confint(fitmod1,parm=c(’theta2’,’theta3’))
confint(fitmod2,parm=c(’theta2’,’theta3’))
AIC(fitmod1)
AIC(fitmod2)

## Display
## CKLS Modele
op <- par(mfrow = c(1, 2))
theta <- coef(fitmod1)
N <- length(rates)
res <- sdsnse(drift=fx1,diffusion=gx1,M=200,t0=time(rates)[1],T=time(rates)[N],
Dt=deltat(rates),x0=rates[1],N)
plot(res,plot.type="single",ylim=c(0,40))
lines(rates,col=2,lwd=2)
legend("topleft",c("real data","CKLS modele"),inset = .01,col=c(2,1),lwd=2,cex=0.8)

## CIR Modele
theta <- coef(fitmod2)
res <- sdsnse(drift=Fx1,diffusion=gx2,M=200,t0=time(rates)[1],T=time(rates)[N],
Dt=deltat(rates),x0=rates[1],N)
plot(res,plot.type="single",ylim=c(0,40))
lines(rates,col=2,lwd=2)
legend("topleft",c("real data","CIR modele"),inset = .01,col=c(2,1),lwd=2,cex=0.8)
```
**fptsde1d**

*Approximate densities and random generation for first passage time in 1-D SDE*

**Description**


**Usage**

```r
fptsde1d(object, ...)
dfptsde1d(object, ...)
```

## Default S3 method:
fptsde1d(object, boundary, ...)
## S3 method for class 'fptsde1d'
summary(object, digits=NULL, ...)
## S3 method for class 'fptsde1d'
mean(x, ...)
## S3 method for class 'fptsde1d'
Median(x, ...)
## S3 method for class 'fptsde1d'
Mode(x, ...)
## S3 method for class 'fptsde1d'
quantile(x, ...)
## S3 method for class 'fptsde1d'
kurtosis(x, ...)
## S3 method for class 'fptsde1d'
skewness(x, ...)
## S3 method for class 'fptsde1d'
min(x, ...)
## S3 method for class 'fptsde1d'
max(x, ...)
## S3 method for class 'fptsde1d'
moment(x, ...)
## S3 method for class 'fptsde1d'
cv(x, ...)

## Default S3 method:
dfptsde1d(object, ...)
## S3 method for class 'dfptsde1d'
plot(x, hist=FALSE, ...)
Arguments

object: an object inheriting from class `ssndeQd` for `fptsdeQd`, and `fptsdeQd` for `dfptsdeQd`.

boundary: an `expression` of a constant or time-dependent boundary.

x: an object inheriting from class `dfptsdeQd`.

hist: if `hist=TRUE` plot histogram. Based on `truehist` function.

digits: integer, used for number formatting.

... potentially further arguments for (non-default) methods, such as `density` for `dfptsdeQd`.

Details

The function `fptsdeQd` returns a random variable \( \tau(X(t),S(t)) \) "first passage time", is defined as:

\[
\tau(X(t),S(t)) = \{ t \geq 0; X_t \geq S(t) \}, \quad \text{if} \quad X(t_0) < S(t_0)
\]

\[
\tau(X(t),S(t)) = \{ t \geq 0; X_t \leq S(t) \}, \quad \text{if} \quad X(t_0) > S(t_0)
\]

And `dfptsdeQd` returns a kernel density approximation for \( \tau(X(t),S(t)) \) "first passage time" with \( S(t) \) is through a continuous boundary (barrier).

An overview of this package, see `browseVignettes('Sim.DiffProc')` for more informations.

Value

`dfptsdeQd` gives the density estimate of fpt. `fptsdeQd` generates random of fpt.

Author(s)

A.C. Guidoum, K. Boukhetala.

References


See Also

- `fptsde2d` and `fptsde3d` simulation fpt for 2 and 3-dim SDE.
- `FPTL` for computes values of the first passage time location (FPTL) function, and `Approx.fpt.density` for approximate first-passage-time (f.p.t.) density in package "fptdApprox".
- `GQD.Tipassage` for compute the First Passage Time Density of a GQD With Time Inhomogeneous Coefficients in package "DiffusionRgqd".

Examples

```r
## Example 1: Ito SDE
## dX(t) = -4*X(t) * dt + 0.5*dW(t)
## S(t) = 0 (constant boundary)
set.seed(1234)

# SDE 1d
f <- expression(-4*x)
g <- expression(0.5)
mod <- snssde1d(drift=f,diffusion=g,x0=2,M=1000)

# boundary
St <- expression(0)

# random
out <- fptsde1d(mod, boundary=St)
out
summary(out)
# density approximate
den <- dfptsde1d(out)
den
plot(den)

## Example 2: Stratonovich SDE
## dX(t) = 0.5*X(t)*t * dt + sqrt(1+X(t)^2) * dW(t)
## S(t) = -0.5*sqrt(t) + exp(t^2) (time-dependent boundary)
set.seed(1234)

# SDE 1d
f <- expression(0.5*x*t)
g <- expression(sqrt(1+x^2))
mod2 <- snssde1d(drift=f,diffusion=g,x0=2,M=1000,type="srt")
```

# boundary
St <- expression(-0.5*sqrt(t)+exp(t^2))

# random
out2 <- fptsde1d(mod2,boundary=St)
out2
summary(out2)
# density approximate
plot(dfptsde1d(out2,bw='ucv'))

## Example 3: fptsde1d vs fptdApproximate
## Not run:
f <- expression( -0.5*x+0.5*5 )
g <- expression( 1 )
St <- expression(5+0.25*sin(2*pi*t))
mod <- snssde1d(drift=f,diffusion=g,boundary=St,x0=3,T=10,N=10*4,M =10000)
mod

# random
out3 <- fptsde1d(mod,boundary=St)
out3
summary(out3)
# density approximate:
library("fptdApprox")
# Under 'fptdApprox':
# Define the diffusion process and give its transitional density:
OU <- diffproc(c("alpha*x + beta","sigma^2",
"dnorm((x-(y*exp(alpha*(t-s)) - beta*(1 - exp(alpha*(t-s))))/alpha)/
(sigma*sqrt(((exp(2*alpha*(t-s)) - 1)/(2*alpha))),(0,1))/
(sigma*sqrt(((exp(2*alpha*(t-s)) - 1)/(2*alpha)))))",
"pnorm(x, y*exp(alpha*(t-s)) - beta*(1 - exp(alpha*(t-s))))/alpha,
sigma*sqrt(((exp(2*alpha*(t-s)) - 1)/(2*alpha)))))")
# Approximate the first passage time density for OU, starting in X_0 = 3
# passing through 5+0.25*sin(2*pi*t) on the time interval [0,10]:
res <- Approx.fpt.density(OU, 0, 10, 3,"5+0.25*sin(2*pi*t)", list(alpha=-0.5,beta=0.5*5,sigma=1))

##
plot(dfptsde1d(out3,bw='ucv'),main = 'fptsde1d vs fptdApproximate')
lines(res$y~res$x, type = 'l',lwd=2)
legend('topright', lty = c('solid', 'dashed'), col = c(1, 2),
legend = c('fptdApproximate', 'fptsde1d'), lwd = 2, bty = 'n')

## End(Not run)
Description


Usage

fptsde2d(object, ...)  
dfptsde2d(object, ...)

## Default S3 method:
fptsde2d(object, boundary, ...)
## S3 method for class 'fptsde2d'
summary(object, digits=NULL, ...)
## S3 method for class 'fptsde2d'
mean(x, ...)
## S3 method for class 'fptsde2d'
Median(x, ...)
## S3 method for class 'fptsde2d'
Mode(x, ...)
## S3 method for class 'fptsde2d'
quantile(x, ...)
## S3 method for class 'fptsde2d'
kurtosis(x, ...)
## S3 method for class 'fptsde2d'
skewness(x, ...)
## S3 method for class 'fptsde2d'
min(x, ...)
## S3 method for class 'fptsde2d'
max(x, ...)
## S3 method for class 'fptsde2d'
moment(x, ...)
## S3 method for class 'fptsde2d'
cv(x, ...)

## Default S3 method:
dfptsde2d(object, pdf=c("Joint","Marginal"), ...)
## S3 method for class 'dfptsde2d'
plot(x,display=c("persp","rgl","image","contour"),
     hist=FALSE, ...)

Arguments

object an object inheriting from class snssde2d for fptsde2d, and fptsde2d for dfptsde2d.
boundary an expression of a constant or time-dependent boundary.
pdf probability density function Joint or Marginal.
x an object inheriting from class fptsde2d.
digits integer, used for number formatting.
display plots.

if hist=TRUE plot histogram. Based on truehist function.

... potentially further arguments for (non-default) methods. arguments to be passed to methods, such as density for marginal density and kde2d for joint density.

Details

The function fptsde2d returns a random variable \((\tau(X(t),S(t)), \tau(Y(t),S(t)))\) ”first passage time”, is defined as:

\[
\tau(X(t),S(t)) = \begin{cases} 
0; & X(t) \geq S(t) \\
\{ t \geq 0; X(t) < S(t) \} & \text{if } X(t_0) < S(t_0)
\end{cases}
\]

and:

\[
\tau(Y(t),S(t)) = \begin{cases} 
0; & Y(t) \geq S(t) \\
\{ t \geq 0; Y(t) < S(t) \} & \text{if } Y(t_0) < S(t_0)
\end{cases}
\]

And dfptsde2d returns a kernel density approximation for \((\tau(X(t),S(t)), \tau(Y(t),S(t)))\) ”first passage time”. with \(S(t)\) is through a continuous boundary (barrier).

An overview of this package, see browseVignettes(’Sim.DiffProc’) for more informations.

Value

dfptsde2d gives the kernel density approximation for fpt. fptsde2d generates random of fpt.

Author(s)

A.C. Guidoum, K. Boukhetala.

References


**See Also**

- `fptsde1d` for simulation fpt in sde 1-dim. `fptsde3d` for simulation fpt in sde 3-dim.
- `FPTL` for computes values of the first passage time location (FPTL) function, and `Approx.fpt.density` for approximate first-passage-time (f.p.t.) density in package "fptdApprox".
- `GQD.Tipassage` for compute the First Passage Time Density of a GQD With Time Inhomogeneous Coefficients in package "DiffusionRgqd".

**Examples**

```r
## dX(t) = 5*(-1-Y(t))*X(t) * dt + 0.5 * dW1(t)
## dY(t) = 5*(-1-X(t))*Y(t) * dt + 0.5 * dW2(t)
## x0 = 2, y0 = -2, and barrier -3+5*t.
## W1(t) and W2(t) two independent Brownian motion
set.seed(1234)

# SDE's 2d
fx <- expression(5*(-1-y)*x , 5*(-1-x)*y)
gx <- expression(0.5 , 0.5)
mod2d <- ssnsde2d(drift=fx,diffusion=gx,x0=c(2,-2),M=100)

# boundary
St <- expression(-1+5*t)

# random fpt
out <- fptsde2d(mod2d,boundary=St)
out
summary(out)

# Marginal density
denM <- dfptsde2d(out,pdf="M")
denM
plot(denM)

# Joint density
denJ <- dfptsde2d(out,pdf="J",n=200,lims=c(0.28,0.4,0.04,0.13))
denJ
plot(denJ)
plot(denJ,display="image")
plot(denJ,display="image",drawpoints=TRUE,cex=0.5,pch=19,col.pt='green')
plot(denJ,display="contour")
```
fptsde3d

Approximate densities and random generation for first passage time in 3-D SDE's

Description

Kernel density and random generation for first-passage-time (f.p.t) in 3-dim stochastic differential equations.

Usage

fptsde3d(object, ...)
dfptsde3d(object, ...)

## Default S3 method:
fptsde3d(object, boundary, ...)
## S3 method for class 'fptsde3d'
summary(object, digits=NULL, ...)
## S3 method for class 'fptsde3d'
mean(x, ...)
## S3 method for class 'fptsde3d'
Median(x, ...)
## S3 method for class 'fptsde3d'
Mode(x, ...)
## S3 method for class 'fptsde3d'
quantile(x, ...)
## S3 method for class 'fptsde3d'
kurtosis(x, ...)
## S3 method for class 'fptsde3d'
skewness(x, ...)
## S3 method for class 'fptsde3d'
min(x, ...)
## S3 method for class 'fptsde3d'
max(x, ...)
## S3 method for class 'fptsde3d'
moment(x, ...)
## S3 method for class 'fptsde3d'
cv(x, ...)

## Default S3 method:
dfptsde3d(object, pdf=c("Joint","Marginal"), ...)
## S3 method for class 'dfptsde3d'
plot(x, display="rgl", hist=FALSE, ...)

plot(denJ, display="contour", color.palette=colorRampPalette(c('white','green','blue','red')))
Arguments

- **object**: an object inheriting from class `snssde3d` for `fptsde3d`, and `fptsde3d` for `dfptsde3d`.
- **boundary**: an expression of a constant or time-dependent boundary.
- **pdf**: probability density function Joint or Marginal.
- **x**: an object inheriting from class `dfptsde3d`.
- **digits**: integer, used for number formatting.
- **display**: display plots.
- **hist**: if hist=TRUE plot histogram. Based on `truehist` function.
- **...**: potentially arguments to be passed to methods, such as `density` for marginal density and `sm.density` for joint density.

Details

The function `fptsde3d` returns a random variable \((\tau(X(t),S(t)), \tau(Y(t),S(t)), \tau(Z(t),S(t)))\) "first passage time", is defined as:

\[
\begin{align*}
\tau(X(t),S(t)) &= \{ t \geq 0; X(t) \geq S(t) \}, \quad \text{if } X(t_0) < S(t_0) \\
\tau(Y(t),S(t)) &= \{ t \geq 0; Y(t) \geq S(t) \}, \quad \text{if } Y(t_0) < S(t_0) \\
\tau(Z(t),S(t)) &= \{ t \geq 0; Z(t) \geq S(t) \}, \quad \text{if } Z(t_0) < S(t_0)
\end{align*}
\]

and:

\[
\begin{align*}
\tau(X(t),S(t)) &= \{ t \geq 0; X(t) \leq S(t) \}, \quad \text{if } X(t_0) > S(t_0) \\
\tau(Y(t),S(t)) &= \{ t \geq 0; Y(t) \leq S(t) \}, \quad \text{if } Y(t_0) > S(t_0) \\
\tau(Z(t),S(t)) &= \{ t \geq 0; Z(t) \leq S(t) \}, \quad \text{if } Z(t_0) > S(t_0)
\end{align*}
\]

And `dfptsde3d` returns a marginal kernel density approximation for \((\tau(X(t),S(t)), \tau(Y(t),S(t)), \tau(Z(t),S(t)))\) "first passage time", with \(S(t)\) is through a continuous boundary (barrier).

An overview of this package, see `browseVignettes('Sim.DiffProc')` for more informations.

Value

`dfptsde3d` gives the marginal kernel density approximation for fpt. `fptsde3d` generates random of fpt.

Author(s)

A.C. Guidoum, K. Boukhetala.
References


See Also

- `fptsde1d` for simulation fpt in sde 1-dim. `fptsde2d` for simulation fpt in sde 2-dim.
- `FPTL` for computes values of the first passage time location (FPTL) function, and `Approx.fpt.density` for approximate first-passage-time (f.p.t.) density in package "fptdApprox".
- `GQD.TIPassage` for compute the First Passage Time Density of a GQD With Time Inhomogeneous Coefficients in package "DiffusionRgqd".

Examples

```r
## dX(t) = 4*(-1-X(t))*Y(t) dt + 0.2 * dW_1(t)
## dY(t) = 4*(1-Y(t)) *X(t) dt + 0.2 * dW_2(t)
## dZ(t) = 4*(1-Z(t)) *Y(t) dt + 0.2 * dW_3(t)
## x0 = 0, y0 = -2, z0 = 0, and barrier -3+5*t.
## W_1(t), W_2(t) and W_3(t) three independent Brownian motion
## set.seed(1234)

# SDE's 3d

fx <- expression(4*(-1-x)*y, 4*(1-y)*x, 4*(1-z)*y)
gx <- rep(expression(0,2),3)
mod3 <- snssde3d(drift=fx,diffusion=gx,M=500)
```
The (S3) generic function for simulation of Hull-White/Vasicek or gaussian diffusion models, and Ornstein-Uhlenbeck process.

**Usage**

\[
\text{HWV}(N, \ldots) \\
\text{OU}(N, \ldots)
\]

### Default S3 method:

\[
\text{HWV}(N = 100, M = 1, x_0 = 2, t_0 = 0, T = 1, \text{Dt} = \text{NULL}, \text{mu} = 4, \text{theta} = 1, \text{sigma} = 0.1, \ldots) \\
\text{OU}(N = 100, M = 1, x_0 = 2, t_0 = 0, T = 1, \text{Dt} = \text{NULL}, \text{mu} = 4, \text{sigma} = 0.2, \ldots)
\]

**Arguments**

- **N** number of simulation steps.
- **M** number of trajectories.
- **x\_0** initial value of the process at time \(t_0\).
- **t\_0** initial time.
The function `hwv` returns a trajectory of the **Hull-White/Vasicek process** starting at \( x_0 \) at time \( t_0 \); i.e., the diffusion process solution of stochastic differential equation:

\[
dX_t = \mu (\theta - X_t) \, dt + \sigma dW_t
\]

The function `ou` returns a trajectory of the **Ornstein-Uhlenbeck** starting at \( x_0 \) at time \( t_0 \); i.e., the diffusion process solution of stochastic differential equation:

\[
dX_t = -\mu X_t \, dt + \sigma dW_t
\]

**Constraints:** \( \mu, \sigma > 0 \).

Please note that the process is stationary only if \( \mu > 0 \).

**Value**

- \( X \) an `ts` object.

**Author(s)**

A.C. Guidoum, K. Boukhetala.

**References**


**See Also**

- `rcOU` and `rsOU` for conditional and stationary law of Vasicek process are available in "sde".

**Examples**

```r
## Hull-White/Vasicek Models
## dX(t) = 4 * (2.5 - X(t)) * dt + 1 * dW(t), X0=10
set.seed(1234)

X <- HWV(N=1000,M=10,mu = 4, theta = 2.5,sigma = 1,x0=10)
plot(X,plot.type="single")
lines(as.vector(time(X)),rowMeans(X),col="red")
```
Irates

## Ornstein-Uhlenbeck Process

```r
## dX(t) = -4 * X(t) * dt + 1 * dW(t) , X0=2
set.seed(1234)

X <- OU(N=1000,M=10,mu = 4,sigma = 1,x0=10)
plot(X,plot.type="single")
lines(as.vector(time(X)),rowMeans(X),col="red")
```

<table>
<thead>
<tr>
<th>Irates</th>
<th>Monthly Interest Rates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Description

- monthly observations from 1946–12 to 1991–02
- **number of observations**: 531
- **observation**: country
  - **country**: United–States

### Usage

```r
data(Irates)
```

### Format

A time serie containing:

- **r1**: interest rate for a maturity of 1 months (% per year).
- **r2**: interest rate for a maturity of 2 months (% per year).
- **r3**: interest rate for a maturity of 3 months (% per year).
- **r5**: interest rate for a maturity of 5 months (% per year).
- **r6**: interest rate for a maturity of 6 months (% per year).
- **r11**: interest rate for a maturity of 11 months (% per year).
- **r12**: interest rate for a maturity of 12 months (% per year).
- **r36**: interest rate for a maturity of 36 months (% per year).
- **r60**: interest rate for a maturity of 60 months (% per year).
- **r120**: interest rate for a maturity of 120 months (% per year).

### Source


These datasets *Irates* are in package "Ecdat".
References


Examples

data(rates)
rates <- Irate["r1"]
rates <- window(rates, start=1964.471, end=1989.333)

## CKLS modele vs CIR modele
## CKLS : dx(t) = (theta1+theta2* X(t))* dt + theta3 * X(t)*theta4 * dW(t)

fx <- expression(theta[1]+theta[2]*x)
gx <- expression(theta[3]*x*theta[4])
fitmod <- fitsde(rates,drift=fx,diffusion=gx,pml="euler",start = list(theta1=1,theta2=1,
theta3=1,theta4=1),optim.method = "L-BFGS-B")
theta <- coef(fitmod)

N <- length(rates)
res <- snssde1(drift=fx,diffusion=gx,M=1000,t0=time(rates)[1],T=time(rates)[N],
Dt=deltat(rates),x0=rates[1],N=N)

plot(res,type="n",ylim=c(0,35))
lines(rates,col=2,lwd=2)
lines(time(res),apply(res$X,1,mean),col=3,lwd=2)
lines(time(res),apply(res$X,1,bconfint,level=0.95)[1],col=4,lwd=2)
lines(time(res),apply(res$X,1,bconfint,level=0.95)[2],col=4,lwd=2)
legend("topleft",c("real data","mean path",
paste(”bound of”,95," confidence")),inset = .01, col=2:4,lwd=2,cex=0.8)

MCM.sde

Parallel Monte-Carlo Methods for SDE's

Description

Generate R Monte-Carlo (version parallel) replicates of a statistic applied to SDE’s (1,2 and 3 dim) for the two cases Ito and Stratonovich interpretations.

Usage

MCM.sde(model, ...)

## Default S3 method:
MCM.sde(model, statistic, R = 1000, time, exact = NULL,
   names = NULL, level = 0.95, parallel = c("no", "multicore", "snow"),
   ncpus = getOption("ncpus", 1L), cl = NULL, ...)

## S3 method for class 'MCM.sde'
plot(x,index = 1,type=c("all","hist","qqplot","boxplot","CI"), ...)
Arguments

- **model**: an object from class `snssde1d`, `snssde2d` and `snssde3d`.
- **statistic**: a function which when applied to model returns a vector containing the statistic(s) of interest.
- **R**: the number of Monte-Carlo replicates. Usually this will be a single positive integer.
- **time**: the time when estimating the statistic(s) of interest time between \( t_0 \) and \( T \). The default is \( t = T \).
- **exact**: a named list giving the exact statistic(s) if it exists otherwise \( \text{exact} = \text{NULL} \).
- **names**: named the statistic(s) of interest. The default names = \( c(\mu_1", \mu_2", \ldots) \).
- **level**: the confidence level(s) of the required interval(s).
- **parallel**: the type of parallel operation to be used (if any). The default is \( \text{parallel} = \text{"no"} \).
- **ncpus**: integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs.
- **cl**: an optional parallel or snow cluster for use if \( \text{parallel} = \text{"snow"} \).
- **x**: an object inheriting from class "MCM.sde".
- **index**: the index of the variable of interest within the output of "MCM.sde".
- **type**: the type of plot of the Monte-Carlo estimation of the variable of interest. The default is \( \text{type} = \text{"all"} \).
- **...**: potentially further arguments for (non-default) methods.

Details

We have here developed Monte-Carlo methods whose essence is the use of repeated experiments to evaluate a statistic(s) of interest in SDE’s. For example estimation of moments as: mean, variance, covariance (and other as median, mode, quantile,...). With the standard error and the confidence interval for these estimators.

An overview of this package, see `browseVignettes('Sim.DiffProc')` for more informations.

Value

The returned value is an object of class "MCM.sde", containing the following components:

- **mod**: The SDE’s used (class: `snssde1d`, `snssde2d` and `snssde3d`).
- **dim**: Dimension of the model.
- **call**: The original call to "MCM.sde".
- **Fn**: The function statistic as passed to "MCM.sde".
- **ech**: A matrix with \( \text{sum}(R) \) column each of which is a Monte-Carlo replicate of the result of calling statistic.
- **time**: The time when estimating the statistic(s) of interest.
- **name**: named of statistic(s) of interest.
- **MC**: Table contains simulation results of statistic(s) of interest: Estimate, Bias (if exact available), Std.Error and Confidence interval.
**Note**

When parallel = "multicore" is used are not available on Windows, parallel = "snow" is primarily intended to be used on multi-core Windows machine where parallel = "multicore" is not available. For more details see Q.E.McCallum and S.Weston (2011).

**Author(s)**

A.C. Guidoum, K. Boukhetala.

**References**


**See Also**

MEM.sde moment equations methods for SDE’s.

**Examples**

```r
## Example 1: (1 dim)
## dX(t) = 3*(1-X(t)) dt + 0.5 * dW(t), X(0)=5, t in [0,10]
## set the model 1d
f <- expression(3*(1-x)); g <- expression(0.5)
model1d <- snssde1d(drift=f,diffusion=g,x0=5,T=10,M=50)

## function of the statistic(s) of interest.
sde.fun1d <- function(data, i){
  d <- data[i,]
  return(c(mean(d),Mode(d),var(d)))
}

mc.sde1d = MCM.sde(model=model1d,statistic=sde.fun1d,R=100,exact=list(Me=1,Mo=1,Va=0.5^2/6),
                   names=c("Me(10)","Mo(10)","Va(10)"))
mc.sde1d
plot(mc.sde1d,index=1)
plot(mc.sde1d,index=2)
plot(mc.sde1d,index=3)

## Example 2: with Parallel computing
## Not run:
model1d <- snssde1d(drift=f,diffusion=g,x0=5,T=10,M=1000)
## On Windows or Unix
```
mc.sde1d = MCM.sde(model=mod1d, statistic=sde.fun1d, R=1000, exact=list(Me=1, Mo=1, Va=0.5^2/6), names=c("Me(10)"", "Mo(10)","Va(10)"), parallel="snow", ncpus=parallel::detectCores())

mc.sde1d

# On Unix only
mc.sde1d = MCM.sde(model=mod1d, statistic=sde.fun1d, R=1000, exact=list(Me=1, Mo=1, Va=0.5^2/6), names=c("Me(10)"", "Mo(10)","Va(10)"), parallel="multicore", ncpus=parallel::detectCores())

mc.sde1d

## End(Not run)

## Example 3: (2 dim)
## dX(t) = 1/\mu\times(theta-X(t)) \, dt + \sqrt{\sigma} \, dW(t),
## dY(t) = X(t) \, dt + 0 \times dW(t)
## Not run:
## Set the model 2d
\mu=0.75; \sigma=0.1; \theta=2
x0=0; y0=0; init=c(x=0, y=0)
f <- expression(1/\mu*(theta-x), x)
g <- expression(sqrt(\sigma), 0)
OU <- ssnsde2d(drift=f, diffusion=g, M=1000, Dt=0.01, x0=init)

## function of the statistic(s) of interest.
sde.fun2d <- function(data, i){
  d <- data[i,]
  return(c(mean(d$x), mean(d$y), var(d$x), var(d$y), cov(d$x, d$y)))
}

## Monte-Carlo at time = 5
mc.sde2d.a = MCM.sde(model=OUI, statistic=sde.fun2d, R=100, time=5, parallel="snow", ncpus=parallel::detectCores())
mc.sde2d.a

## Monte-Carlo at time = 10
mc.sde2d.b = MCM.sde(model=OUI, statistic=sde.fun2d, R=100, time=10, parallel="snow", ncpus=parallel::detectCores())
mc.sde2d.b

## Compared with exact values at time 5 and 10
E_x <- function(t) theta+(x0-theta)*exp(-t/\mu)
V_x <- function(t) 0.5\sigma\mu*(1-exp(-2*(t/\mu)))
E_y <- function(t) y0+theta+t+(x0-theta)*\mu*(1-exp(-t/\mu))
V_y <- function(t) \sigma\mu^2*(1-exp(-t/\mu))+0.5*(1-exp(-2*(t/\mu)))
cov_xy <- function(t) 0.5\sigma\mu^2*2*(1-2*exp(-t/\mu)+exp(-2*(t/\mu)))

## at time=5
mc.sde2d.a = MCM.sde(model=OUI, statistic=sde.fun2d, R=100, time=5, exact=list(m1=E_x(5), m2=E_y(5), s1=V_x(5), s2=V_y(5), c12=cov_xy(5)), parallel="snow", ncpus=parallel::detectCores())
mc.sde2d.a

plot(mc.sde2d.a, index=1)
plot(mc.sde2d.a, index=2)

## at time=10
mc.sde2d.b = MCM.sde(model=OUI, statistic=sde.fun2d, R=100, time=10, exact=list(m1=E_x(10), m2=E_y(10), s1=V_x(10), s2=V_y(10), c12=cov_xy(10)), parallel="snow", ncpus=parallel::detectCores())

plot(mc.sde2d.b, index=1)
plot(mc.sde2d.b, index=2)
### Usage

MEM.sde(drift, diffusion, ...)

#### Default S3 method:

MEM.sde(drift, diffusion, type = c("ito", "str"), solve = FALSE, 
  parms = NULL, init = NULL, time = NULL, ...)

#### S3 method for class 'MEM.sde'

summary(object, at , ...)
Arguments

- **drift**: drift coefficient: an expression 1-dim(t,x), 2-dim(t,x,y) or 3-dim(t,x,y,z).
- **diffusion**: diffusion coefficient: an expression 1-dim(t,x), 2-dim(t,x,y) or 3-dim(t,x,y,z).
- **type**: type of process "ito" or "Stratonovich"; the default type="ito".
- **solve**: if solve=TRUE solves a system of ordinary differential equations.
- **parms**: parameters passed to drift and diffusion.
- **init**: the initial (state) values for the ODE system. for 1-dim (m=x0,S=0), 2-dim (m1=x0,m2=y0,S1=0,S2=0,C12=0) and for 3-dim (m1=x0,m2=y0,m3=z0,S1=0,S2=0,S3=0,C12=0,C13=0, see examples.
- **time**: time sequence (vector) for which output is wanted; the first value of time must be the initial time.
- **object, at**: an object inheriting from class "MEM.sde" and summaries at any time at.
- **...**: potentially arguments to be passed to methods, such as `ode` for solver for ODE’s.

Details

The stochastic transition is approximated by the moment equations, and the numerical treatment is required to solve these equations from above with given initial conditions.

An overview of this package, see `browseVignettes('Sim.DiffProc')` for more informations.

Value

Symbolic ODE’s of means and variances-covariance. If solve=TRUE approximate the moment of SDE’s at any time.

Author(s)

A.C. Guidoum, K. Boukhetala.

References


See Also

- `MCM.sde` Monte-Carlo methods for SDE’s.
Examples

library(deSolve)

## Example 1: 1-dim
## dX(t) = mu * X(t) * dt + sigma * X(t) * dW(t)
## Symbolic ODE's of mean and variance
f <- expression(mu*x)
g <- expression(sigma*x)
res1 <- MEM.sde(drift=f,diffusion=g)
res2 <- MEM.sde(drift=f,diffusion=g,type="str")
res1
res2

## numerical approximation of mean and variance
para <- c(mu=2,sigma=0.5)
t <- seq(0,1,by=0.001)
init <- c(m=1,s=0)
res1 <- MEM.sde(drift=f,diffusion=g,solve=TRUE,init=init,parms=para,time=t)
res1
matplot(res1$sol.ode,main="Mean and Variance of X(t), type Ito")
plot(res1$sol.ode,select=c("m","S"))

## approximation at time = 0.75
summary(res1,at=0.75)

##
res2 <- MEM.sde(drift=f,diffusion=g,solve=TRUE,init=init,parms=para,time=t,type="str")
res2
matplot(res2$sol.ode,main="Mean and Variance of X(t), type Stratonovich")
plot(res2$sol.ode,select=c("m","S"))

## approximation at time = 0.75
summary(res2,at=0.75)

## Comparison:
plot(res1$sol.ode, res2$sol.ode,ylab = c("m(t)"),select="m",xlab = "Time",
col = c("red", "blue"))
plot(res1$sol.ode, res2$sol.ode,ylab = c("S(t)"),select="S",xlab = "Time",
col = c("red", "blue"))

## Example 2: 2-dim
## dX(t) = 1/mu*(theta-X(t)) dt + sqrt(sigma) * dW1(t),
## dY(t) = X(t) dt + 0 * dW2(t)
## Not run:
para=c(mu=0.75, sigma=0.1, theta=2)
init=c(m=0,m2=0,s1=0,s2=0,c12=0)
t <- seq(0,10,by=0.001)
f <- expression(1/mu*(theta-x), x)
g <- expression(sqrt(sigma),0)
res2d <- MEM.sde(drift=f,diffusion=g,solve=TRUE,init=init,parms=para,time=t)
res2d

## Exact moment

mu=0.75; sigma=0.1; theta=2; x0=0; y0=0
E_x <- function(t) theta*(x0-theta)*exp(-t/mu)
V_x <- function(t) 0.5*sigma*mu *(1-exp(-2*(t/mu)))
E_y <- function(t) y0-theta+t*(x0-theta)*mu *(1-exp(-t/mu))
V_y <- function(t) sigma*mu^3*((t/mu)-2*(1-exp(-t/mu)))*0.5*(1-exp(-2*(t/mu)))
cov_xy <- function(t) 0.5*sigma*mu^2 *(1-2*exp(-t/mu)+exp(-2*(t/mu)))

# summary(res2d,at=5)
E_x(5);E_y(5);V_x(5);V_y(5);cov_xy(5)

matplot.0D(res2d$sol.ode,select=c("m1"))
curve(E_x,add=TRUE,col="red")

# plot
plot(res2d$sol.ode)
matplot.0D(res2d$sol.ode,select=c("S1","S2","C12"))
plot(res2d$sol.ode[,"m1"], res2d$sol.ode[,"m2"], xlab = "m1(t)",
ylab = "m2(t)", type = "l",lwd = 2)

hist(res2d$sol.ode,select=c("m1","m2"), col = c("darkblue", "red", "orange", "black"))

# Example3: 3-dim
# dx(t) = sigma*(Y(t)-X(t)) dt + 0.1 * dW1(t)
# dy(t) = (rho*X(t)-Y(t)-X(t)*Z(t)) dt + 0.1 * dW2(t)
# dz(t) = (X(t)*Y(t)-bet*Z(t)) dt + 0.1 * dW3(t)
f <- expression(sigma*(y-x),rho*x-y-x*z,x*y-bet*z)
g <- expression(0.1,0.1,0.1)
# Symbolic moments equations
res3d = MEM.sde(drift=f,diffusion=g)

# Numerical approximation
para=c(sigma=10,rho=28,bet=8/3)
ini=c(m1=1,m2=1,m3=1,S1=0,S2=0,S3=0,C12=0,C13=0,C23=0)
res3d = MEM.sde(drift=f,diffusion=g,solve=T,parms=para,init=ini,time=seq(0,1,by=0.01))

summary(res3d,at=0.25)
summary(res3d,at=0.50)
summary(res3d,at=0.75)

plot(res3d$sol.ode)
matplot.0D(res3d$sol.ode,select=c("m1","m2","m3"))
matplot.0D(res3d$sol.ode,select=c("S1","S2","S3"))
matplot.0D(res3d$sol.ode,select=c("C12","C13","C23"))

library(rgl)
plot3d(res3d$sol.ode[,"m1"], res3d$sol.ode[,"m2"],res3d$sol.ode[,"m3"], xlab = "m1(t)",
ylab = "m2(t)",zlab="m3(t)", type = "l",lwd = 2,box=f)

# End(Not run)
Monte-Carlo statistics of SDE's

Description

Generic function for compute the kurtosis, skewness, median, mode and coefficient of variation (relative variability), moment and confidence interval of class "sde".

Usage

```r
## Default S3 method:
bconfint(x, level = 0.95, ...)
## Default S3 method:
kurtosis(x, ...)
## Default S3 method:
moment(x, order = 1, center = TRUE, ...)
## Default S3 method:
cv(x, ...)
## Default S3 method:
skewness(x, ...)
## Default S3 method:
Median(x, ...)
## Default S3 method:
Mode(x, ...)
```

Arguments

- `x`: an object inheriting from class "sde".
- `order`: order of moment.
- `center`: if TRUE is a central moment.
- `level`: the confidence level required.
- `...`: potentially further arguments for (non-default) methods.

Author(s)

A.C. Guidoum, K. Boukhetala.

Examples

```r
## Example 1:
## dX(t) = 2*(3-X(t)) *dt + dW(t)
set.seed(1234)

f <- expression(2*(3-x))
g <- expression(1)
mod <- ssnsdeldrift=f, diffusion=g, M=10000, T=5)
## Monte-Carlo statistics of 5000 trajectory of X(t) at final time T of 'mod'
```
**plot2d**

- summary(mod)
- kurtosis(mod)
- skewness(mod)
- mean(mod)
- Median(mod)
- Mode(mod)
- moment(mod, order = 4)
- cv(mod)
- bconfint(mod, level = 0.95) ## of mean

---

**plot2d**  
*Plotting for Class SDE*

**Description**

Generic function for plotting.

**Usage**

```r
## Default S3 method:
plot2d(x, ...)
## Default S3 method:
lines2d(x, ...)
## Default S3 method:
points2d(x, ...)
## Default S3 method:
plot3d(x, display = c("persp","rgl"), ...)
```

**Arguments**

- `x` an object inheriting from class `snssde2d`, `snssde3d`, `bridgesde2d` and `bridgesde3d`.
- `display` "persp" perspective or "rgl" plots.
- `...` other graphics parameters, see `par` in package "graphics", `scatterplot3d` in package "scatterplot3d" and `plot3d` in package "rgl".

**Details**

The 2 and 3-dim plot of class sde.

**Author(s)**

A.C. Guidoum, K. Boukhetala.
Examples

```r
## Example 1:
set.seed(1234)

fx <- rep(expression(0),2)
gx <- rep(expression(1),2)

res <- snssde2d(drift=fx,diffusion=gx,N=10000)
plot2d(res,type="l")

## Example 2:
set.seed(1234)

fx <- rep(expression(0),3)
gx <- rep(expression(1),3)

res <- snssde3d(drift=fx,diffusion=gx,N=10000)
plot3D(res,display="persp")
plot3D(res,display="rgl")
```

---

rsde1d  

**Approximate transitional densities and random generation for 1-D SDE**

Description

Transition density and random generation for \( X(t) \mid X(s)=x \) of the 1-dim SDE.

Usage

```r
rsde1d(object, ...)
dsde1d(object, ...)

## Default S3 method:
rsde1d(object, at, ...)

## Default S3 method:
dsde1d(object, at, ...)

## S3 method for class 'dsde1d'
plot(x,hist=FALSE, ...)
```

Arguments

- `object`: an object inheriting from class `snssde1d` and `bridgesde1d`.
- `at`: time between \( s=t0 \) and \( t=T \). The default \( at = T \).
- `x`: an object inheriting from class `dsde1d`.
- `hist`: if `hist=TRUE` plot histogram. Based on `truehist` function.
- `...`: potentially arguments to be passed to methods, such as `density` for kernel density.
Details

The function \texttt{rsde1d} returns a \( M \) random variable \( x_{t=at} \) realize at time \( t = at \) defined by:

\[
x_{t=at} = \{ t \geq 0; x = X_{t=at} \}
\]

And \texttt{dsde1d} returns a transition density approximation for \( X(t-s) | X(s)=x0 \). with \( t = at \) is a fixed time between \( t0 \) and \( T \).

An overview of this package, see \texttt{browseVignettes('Sim.DiffProc')} for more informations.

Value

\texttt{dsde1d} gives the transition density estimate of \( X(t-s) | X(s)=x0 \). \texttt{rsde1d} generates random of \( X(t-s) | X(s)=x0 \).

Author(s)

A.C. Guidoum, K. Boukhetala.

See Also

density Kernel density estimation in "stats" package.
kde Kernel density estimate for 1- to 6-dimensional data in "ks" package.
sm.density Nonparametric density estimation in one, two or three dimensions in "sm" package.
rng random number generators in "yuima" package.
dcSim Pedersen's simulated transition density in "sde" package.
rcBS, rcCIR, rcOU and rsOU in package "sde".
dcBS, dcCIR, dcOU and dsOU in package "sde".
QoD.density Generate the transition density of a scalar generalized quadratic diffusion.

Examples

```r
## Example 1:
## dX(t) = (-2*(X(t)<=0)+2*(X(t)>0)) *dt + 0.5 * dW(t)
set.seed(1234)

f <- expression(-2*(x<=0)+2*(x>0))
g <- expression(0.5)
res1 <- snssde1d(drift=f,diffusion=g,M=5000)
x <- rsde1d(res1, at = 1)
summary(x)
dens1 <- dsde1d(res1, at = 1)
dens1
plot(dens1,main="Transition density of X(t=1)|X(s=0)=0") # kernel estimated
plot(dens1,hist=TRUE) # histogramme

## Example 2:
## Transition density of standard Brownian motion W(t) at time = 0.5
```
Approximate transitional densities and random generation for 2-D SDE's.

**Description**

Transition density and random generation for the joint and marginal of \((X(t-s),Y(t-s) \mid X(s)=x_0,Y(s)=y_0)\) of the SDE's 2-d.

**Usage**

```rlen3d(object, ...)dsde2d(object, ...)## Default S3 method:rsde2d(object, at, ...)## Default S3 method:dsde2d(object, pdf=c("Joint","Marginal"), at, ...)## S3 method for class 'dsde2d'plot(x,display=c("persp","rgl","image","contour"),hist=FALSE,...)
```
Arguments

- **object**: an object inheriting from class `snnssde2d` and `bridgesde2d`.
- **at**: time between \( s=0 \) and \( t=T \). The default \( at = T \).
- **pdf**: probability density function Joint or Marginal.
- **x**: an object inheriting from class `dsde2d`.
- **display**: display plots.
- **hist**: if `hist=TRUE` plot histogram. Based on `truehist` function.

- **...**: potentially potentially arguments to be passed to methods, such as `density` for marginal density and `kde2d` for joint density.

Details

The function `rsde2d` returns a \( M \) random variable \( x_{t=at}, y_{t=at} \) realize at time \( t = at \).

And `dsde2d` returns a bivariate density approximation for \( (X(t-s), Y(t-s) \mid X(s)=x0, Y(s)=y0) \) with \( t = at \) is a fixed time between \( t0 \) and \( T \).

An overview of this package, see `browseVignettes('Sim.DiffProc')` for more informations.

Value

- `dsde2d` gives the bivariate density approximation for \( (X(t-s), Y(t-s) \mid X(s)=x0, Y(s)=y0) \).
- `rsde2d` generates random of the couple \( (X(t-s), Y(t-s) \mid X(s)=x0, Y(s)=y0) \).

Author(s)

A.C. Guidoum, K. Boukhetala.

See Also

- `kde2d` Two-dimensional kernel density estimation in "MASS" package.
- `kde` Kernel density estimate for 1- to 6-dimensional data in "ks" package.
- `sm.density` Nonparametric density estimation in one, two or three dimensions in "sm" package.
- `rng` random number generators in "yuima" package.
- `BiGQD.density` Generate the transition density of a bivariate generalized quadratic diffusion model (2D GQD).

Examples

```r
## Example: 1
set.seed(1234)

# SDE's 2d
fx <- expression(3*(2-y), 2*x)
gx <- expression(1, y)
mod2d <- snnssde2d(drift=fx, diffusion=gx, x0=c(1, 2), M=1000)

# random
```

r2d <- rsde2d(mod2d, at=0.5)
summary(r2d)

# Marginal density
denM <- dsde2d(mod2d, pdf="M", at=0.5)
denM
plot(denM)

# Joint density
denJ <- dsde2d(mod2d, pdf="J", n=200, at=0.5, lims=c(-3,4,0,6))
denJ
plot(denJ)
plot(denJ, display="contour")

## Example RZ bivariate transition density of R brownian motion (W1(t),W2(t)) in [0,1]
## Not run:
B2d <- sssde2d(drift=rep(expression(0)), diffusion=rep(expression(1)), M=10000)
for (i in seq(B2d$DT,B2d$T, by=B2d$DT)){
  plot(dsde2d(B2d, at = i, lims=c(-3,3,-3,3), n=100),
       display="contour", main=paste0("Transition Density \n t = ",i))
}

## Example SZ
## Not run:
fx <- expression(4*(x-y) , 4*(1-x-y )
gx <- expression(0.25*y, 0.2*x)
mod2d1 <- sssde2d(drift=fx, diffusion=gx, x0=c(x0=1, y0=-1),
                 M=5000, type="str")

# Marginal transition density
for (i in seq(mod2d1$DT, mod2d1$T, by=mod2d1$DT)){
  plot(dsde2d(mod2d1, pdf="M", at = i), main=
       paste0("Marginal Transition Density \n t = ",i))
}

# Bivariate transition density
for (i in seq(mod2d1$DT, mod2d1$T, by=mod2d1$DT)){
  plot(dsde2d(mod2d1, at = i, lims=c(-1,2,-1,1), n=100),
       display="contour", main=paste0("Transition Density \n t = ",i))
}

## End(Not run)

## Example TZ bivariate transition density of R bridge brownian motion (W1(t),W2(t)) in [0,1]
## Not run:
B2d <- bridgesde2d(drift=rep(expression(0)), 2),
diffusion=rep(expression(1,2),M=5000)
for (i in seq(0.01,0.99,by=0.03))
plot(dsde2d(B2d, at = i,lims=c(-3,3,-3,3),
n=1000),display="contour",main=
paste0('Transition Density \n t = ',i))
}
## End(Not run)

## Example 5: Bivariate Transition Density of bridge
## Ornstein-Uhlenbeck process and its integral in [0,5]
## dx(t) = 4*(-1-X(t)) dt + 0.2 dW1(t)
## dy(t) = X(t) dt + 0 dW2(t)
## x01 = 0, y01 = 0
## x02 = 0, y02 = 0
## Not run:
fx <- expression(4*(-1-x) , x)
gx <- expression(0.2 , 0)
oui <- bridgesde2d(drift=fx,diffusion=gx,Dt=0.005,M=1000)
for (i in seq(0.01,0.99,by=0.01))
plot(dsde2d(oui, at = i,lims=c(-1.2,0.2,-2.5,0.2),n=1000),
display="contour",main=paste0('Transition Density \n t = ',i))
}
## End(Not run)

rsde3d Approximate transitional densities and random generation for 3-D SDE's

Description

Transition density and random generation for the joint and marginal of \(X(t-s),Y(t-s),Z(t-s)\) \(| X(s)=x0,Y(s)=y0,Z(s)=z0\) of the SDE's 3-d.

Usage

rsde3d(object, ...)
dsde3d(object, ...)

## Default S3 method:
rsde3d(object, at, ...)

## Default S3 method:
dsde3d(object, pdf=c("Joint","Marginal"), at, ...)
## S3 method for class 'dsde3d'
plot(x,display="rgl",hist=FALSE,...)
Arguments

- object: an object inheriting from class `snsSde3d` and `bridgesSde3d`.
- at: time between \( s = 0 \) and \( t = T \). The default \( at = T \).
- pdf: probability density function Joint or Marginal.
- x: an object inheriting from class `dsde3d`.
- display: display plots.
- hist: if `hist=TRUE` plot histogram. Based on `truehist` function.
- ...: potentially arguments to be passed to methods, such as `density` for marginal density and `sm.density` for joint density.

Details

The function `rsde3d` returns a random variable \( x_{t=at}, y_{t=at}, z_{t=at} \) realize at time \( t = at \).

And `dsde3d` returns a trivariate kernel density approximation for \( X(t-s), Y(t-s), Z(t-s) \) \( X(s)=x0, Y(s)=y0, Z(s)=z0 \).

with \( t = at \) is a fixed time between \( t0 \) and \( T \).

An overview of this package, see `browseVignettes('Sim.DiffProc')` for more informations.

Value

- `dsde3d` gives the trivariate density approximation \( (X(t-s), Y(t-s), Z(t-s)) \mid X(s)=x0, Y(s)=y0, Z(s)=z0 \).
- `rsde3d` generates random of the \( (X(t-s), Y(t-s), Z(t-s)) \mid X(s)=x0, Y(s)=y0, Z(s)=z0 \).

Author(s)

A.C. Guidoum, K. Boukhetala.

See Also

- `kde` Kernel density estimate for 1- to 6-dimensional data in "ks" package.
- `sm.density` Nonparametric density estimation in one, two or three dimensions in "sm" package.
- `kde3d` Compute a three dimension kernel density estimate in "misc3d" package.
- `rng` random number generators in "yuima" package.
- `rcBS, rcCIR, rcOU` and `rsOU` in package "sde".

Examples

```r
## Example 1: Ito sde
## dX(t) = (2*(Y(t)>0)-2*(Z(t)<=0)) dt + 0.2 * dW1(t)
## dY(t) = -2*X(t) dt + 0.2 * dW2(t)
## dZ(t) = -2*Z(t) dt + 0.2 * dW3(t)
## W1(t), W2(t) and W3(t) three independent Brownian motion
set.seed(1234)
fx <- expression(2*(y>0)-2*(z<=0), -2*x, -2*z)
gx <- rep(expression(0.2),3)
mod3d1 <- snsSde3d(x0=c(0,2,-2),drift=fx,diffusion=gx,M=1000,Dt=0.003)
```
# random at t = 0.75
r3d1 <- rsde3d(mod3d1, at=0.75)
summary(r3d1)

# Marginal transition density at t=0.75, t0=0

denM <- dsde3d(mod3d1, pdf="M", at=0.75)
denM
plot(denM)

# for Joint transition density at t=0.75; t0=0
# Multiple isosurfaces
## Not run:
denJ <- dsde3d(mod3d1, pdf="J", at= 0.75)
denJ
plot(denJ, display="rgl")

## End(Not run)

## Example 2: Stratonovich sde
## dX(t) = Y(t)* dt + X(t) o dW1(t)
## dY(t) = (4*( 1-X(t)^2 )* Y(t) - X(t))* dt + 0.2 o dW2(t)
## dZ(t) = (4*( 1-X(t)^2 )* Z(t) - X(t))* dt + 0.2 o dW3(t)
set.seed(1234)

fx <- expression(y, (4*( 1-x^2 )* y - x), (4*( 1-x^2 )* z - x))
gx <- expression(x, 0.2, 0.2)
mod3d2 <- snssde3d(drift=fx, diffusion=gx, M=1000, type="str")

## random
r3d2 <- rsde3d(mod3d2)
summary(r3d2)

## Marginal transition density at t=1, t0=0

denM <- dsde3d(mod3d2, pdf="M")
denM
plot(denM)

## for Joint transition density at t=1; t0=0
## Multiple isosurfaces
## Not run:
denJ <- dsde3d(mod3d2, pdf="J")
denJ
plot(denJ, display="rgl")

## End(Not run)

## Example 3: Tivariate Transition Density of 3 Brownian motion (W1(t),W2(t),W3(t)) in [0,1]
## Not run:
B3d <- snssde3d(drift=rep(expression(0), 3), diffusion=rep(expression(1), 3), M=500)
for (i in seq(B3d$D, B3d$T, by = B3d$Dt)){
  plot(dsde3d(B3d, at = i, pdf = "J"), box = F, main = paste0("Transition Density t = ", i))
}

## End(Not run)

---

**snssde1d Simulation of 1-D Stochastic Differential Equation**

**Description**

The (S3) generic function `snssde1d` of simulation of solution to 1-dim stochastic differential equation of Itô or Stratonovich type, with different methods.

**Usage**

```r
snssde1d(N, ...)
## Default S3 method:
snssde1d(N = 1000, M = 1, x0 = 0, t0 = 0, T = 1, Dt = NULL, 
  drift, diffusion, alpha = 0.5, mu = 0.5, type = c("ito", "str"), 
  method = c("euler", "milstein", "predcorr", "smilstein", "taylor", 
    "heun", "rk1", "rk2", "rk3"), ...)

## S3 method for class 'snssde1d'
summary(object, at, digits = NULL, ...)
## S3 method for class 'snssde1d'
time(x, ...)
## S3 method for class 'snssde1d'
mean(x, at, ...)
## S3 method for class 'snssde1d'
Median(x, at, ...)
## S3 method for class 'snssde1d'
Mode(x, at, ...)
## S3 method for class 'snssde1d'
quantile(x, at, ...)
## S3 method for class 'snssde1d'
kurtosis(x, at, ...)
## S3 method for class 'snssde1d'
min(x, at, ...)
## S3 method for class 'snssde1d'
max(x, at, ...)
## S3 method for class 'snssde1d'
skewness(x, at, ...)
## S3 method for class 'snssde1d'
moment(x, at, ...)
## S3 method for class 'snssde1d'
```

---
cv(x, at, ...)
## S3 method for class 'snssde1d'
bconfint(x, at, ...)

## S3 method for class 'snssde1d'
plot(x, ...)
## S3 method for class 'snssde1d'
lines(x, ...)
## S3 method for class 'snssde1d'
points(x, ...)

### Arguments

- **N**: number of simulation steps.
- **M**: number of trajectories (Monte-Carlo).
- **x0**: initial value of the process at time \( t_0 \).
- **t0**: initial time.
- **T**: ending time.
- **dt**: time step of the simulation (discretization). If it is **NULL** a default \( \Delta t = \frac{T-t_0}{N} \).
- **drift**: drift coefficient: an **expression** of two variables \( t \) and \( x \).
- **diffusion**: diffusion coefficient: an **expression** of two variables \( t \) and \( x \).
- **alpha**, **mu**: weight of the predictor-corrector scheme; the default \( \alpha = 0.5 \) and \( \mu = 0.5 \).
- **type**: if type="ito" simulation sde of Ito type, else type="str" simulation sde of Stratonovich type; the default type="ito".
- **method**: numerical methods of simulation, the default method = "euler".
- **x**, **object**: an object inheriting from class "snssde1d".
- **at**: time between \( t_0 \) and \( T \). Monte-Carlo statistics of the solution \( X_t \) at time \( at \). The default at = \( T \).
- **digits**: integer, used for number formatting.
- **...**: potentially further arguments for (non-default) methods.

### Details

The function `snssde1d` returns a **ts** \( x \) of length \( N+1 \); i.e. solution of the sde of Ito or Stratonovich types; If \( dt \) is not specified, then the best discretization \( \Delta t = \frac{T-t_0}{N} \).

The Ito stochastic differential equation is:

\[
dX(t) = a(t, X(t))dt + b(t, X(t))dW(t)
\]

Stratonovich sde:

\[
dX(t) = a(t, X(t))dt + b(t, X(t)) \circ dW(t)
\]

The methods of approximation are classified according to their different properties. Mainly two criteria of optimality are used in the literature: the strong and the weak (orders of) convergence. The method of simulation can be one among: Euler-Maruyama Order 0.5, Milstein Order 1, Milstein Second-Order, Predictor-Corrector method, Itô-Taylor Order 1.5, Heun Order 2 and Runge-Kutta Order 1, 2 and 3.

An overview of this package, see `browseVignettes('Sim.DiffProc')` for more informations.
Value

\texttt{snssde1d} returns an object inheriting from \texttt{class "snssde1d"}.  

\begin{itemize}
  \item \texttt{x} \hspace{1cm} an invisible \texttt{ts} object.
  \item \texttt{drift} \hspace{1cm} drift coefficient.
  \item \texttt{diffusion} \hspace{1cm} diffusion coefficient.
  \item \texttt{type} \hspace{1cm} type of sde.
  \item \texttt{method} \hspace{1cm} the numerical method used.
\end{itemize}

Author(s)

A.C. Guidoum, K. Boukhetala.

References

Friedman, A. (1975). \textit{Stochastic differential equations and applications}. Volume 1, ACADEMIC PRESS.


See Also

\texttt{snssde2d} and \texttt{snssde3d} for 2 and 3-dim sde.

\texttt{sde.sim} in package "sde".

simulate in package "yuima".
Examples

```r
## Example 1: Ito sde
## dX(t) = 2*(3-X(t)) dt + 2*X(t) dW(t)
set.seed(1234)

f <- expression(2*(3-x) )
g <- expression(1)
mod1 <- snssde2d(drift=f, diffusion=g, M=4000, x0=10, Dt=0.01)
mod1
summary(mod1)
## Not run:
plot(mod1)
lines(time(mod1), apply(mod1$x, 1, mean), col=2, lwd=2)
lines(time(mod1), apply(mod1$x, 1, bconfint, level=0.95)[1,], col=4, lwd=2)
legend("topright", c("mean path", paste("bound of", 95, " percent confidence")),
       inset = .01, col=c(2,4), lwd=2, cex=0.8)

## End(Not run)

## Example 2: Stratonovich sde
## dX(t) = ((2-X(t))/(2-t)) dt + X(t) o dW(t)
set.seed(1234)

f <- expression(((2-x)/(2-t))
g <- expression(x)
mod2 <- snssde2d(type="str", drift=f, diffusion=g, M=4000, x0=1, method="milstein")
mod2
summary(mod2, at = 0.25)
summary(mod2, at = 1)
## Not run:
plot(mod2)
lines(time(mod2), apply(mod2$x, 1, mean), col=2, lwd=2)
lines(time(mod2), apply(mod2$x, 1, bconfint, level=0.95)[1,], col=4, lwd=2)
legend("topleft", c("mean path", paste("bound of", 95, " percent confidence")),
       inset = .01, col=c(2,4), lwd=2, cex=0.8)

## End(Not run)
```

Description

The S3 generic function `snssde2d` of simulation of solutions to 2-dim stochastic differential equations of Itô or Stratonovich type, with different methods.
Usage

snssde2d(N, ...)
## Default S3 method:
snssde2d(N = 1000, M = 1, x0 = c(0, 0), t0 = 0, T = 1, Dt=NULL,
         drift=NULL, diffusion, alpha = 0.5, mu = 0.5, type = c("ito", "str"),
         method = c("euler", "milstein", "predcorr", "smilstein", "taylor",
                    "heun", "rk1", "rk2", "rk3"), ...)

## S3 method for class 'snssde2d'
summary(object, at, digits=NULL,...)
## S3 method for class 'snssde2d'
time(x, ...)
## S3 method for class 'snssde2d'
mean(x, at, ...)
## S3 method for class 'snssde2d'
Median(x, at, ...)
## S3 method for class 'snssde2d'
Mode(x, at, ...)
## S3 method for class 'snssde2d'
quantile(x, at, ...)
## S3 method for class 'snssde2d'
kurtosis(x, at, ...)
## S3 method for class 'snssde2d'
skewness(x, at, ...)
## S3 method for class 'snssde2d'
min(x, at, ...)
## S3 method for class 'snssde2d'
max(x, at, ...)
## S3 method for class 'snssde2d'
moment(x, at, ...)
## S3 method for class 'snssde2d'
vcv(x, at, ...)
## S3 method for class 'snssde2d'
bconfint(x, at, ...)

## S3 method for class 'snssde2d'
plot(x, ...)
## S3 method for class 'snssde2d'
lines(x, ...)
## S3 method for class 'snssde2d'
points(x, ...)
## S3 method for class 'snssde2d'
plot2d(x, ...)
## S3 method for class 'snssde2d'
lines2d(x, ...)
## S3 method for class 'snssde2d'
points2d(x, ...)
Arguments

- **N**: number of simulation steps.
- **M**: number of trajectories (Monte-Carlo).
- **x0**: initial values x0=(x,y) of the process \(X_t\) and \(Y_t\) at time \(t_0\).
- **t0**: initial time.
- **T**: ending time.
- **Dt**: time step of the simulation (discretization). If it is NULL a default \(\Delta t = \frac{T-t_0}{N}\).
- **drift**: drift coefficient: an expression of three variables \(t, x, y\) for process \(X_t\) and \(Y_t\).
- **diffusion**: diffusion coefficient: an expression of three variables \(t, x, y\) for process \(X_t\) and \(Y_t\).
- **alpha, mu**: weight of the predictor-corrector scheme; the default \(\alpha = 0.5\) and \(\mu = 0.5\).
- **type**: if type="ito" simulation sde of Itô type, else type="str" simulation sde of Stratonovich type; the default type="ito".
- **method**: numerical methods of simulation, the default method = "euler".
- **x, object**: an object inheriting from class "snssde2d".
- **at**: time between \(t_0\) and \(T\). Monte-Carlo statistics of the solutions \((X_t, Y_t)\) at time \(t\). The default at = \(T\).
- **digits**: integer, used for number formatting.
- **...**: potentially further arguments for (non-default) methods.

Details

The function snssde2d returns a mts x of length \(N+1\); i.e. solution of the 2-dim sde \((X_t, Y_t)\) of Itô or Stratonovich types; If \(Dt\) is not specified, then the best discretization \(\Delta t = \frac{T-t_0}{N}\).

The 2-dim Itô stochastic differential equation is:

\[
\begin{align*}
\frac{dX(t)}{dt} &= a(t, X(t), Y(t)) + b(t, X(t), Y(t))dW_1(t) \\
\frac{dY(t)}{dt} &= a(t, X(t), Y(t)) + b(t, X(t), Y(t))dW_2(t)
\end{align*}
\]

2-dim Stratonovich sde:

\[
\begin{align*}
\frac{dX(t)}{dt} &= a(t, X(t), Y(t)) + b(t, X(t), Y(t)) \circ dW_1(t) \\
\frac{dY(t)}{dt} &= a(t, X(t), Y(t)) + b(t, X(t), Y(t)) \circ dW_2(t)
\end{align*}
\]

\(W_1(t), W_2(t)\) two standard Brownian motion independent.

The methods of approximation are classified according to their different properties. Mainly two criteria of optimality are used in the literature: the strong and the weak (orders of) convergence. The method of simulation can be one among: Euler–Maruyama Order 0.5, Milstein Order 1, Milstein Second-Order, Predictor-Corrector method, Itô-Taylor Order 1.5, Heun Order 2 and Runge-Kutta Order 1, 2 and 3.

An overview of this package, see browseVignettes('Sim.DiffProc') for more informations.
Value

\texttt{snssde2d} returns an object inheriting from \texttt{class "snssde2d".}

\begin{itemize}
  \item \texttt{X, Y} \hspace{1cm} \text{an invisible \texttt{mts} (2-dim) object \texttt{(X(t),Y(t)).}}
  \item \texttt{driftx, drifty} \hspace{1cm} \text{drift coefficient of \texttt{X(t)} and \texttt{Y(t)}.}
  \item \texttt{diffx, diffy} \hspace{1cm} \text{diffusion coefficient of \texttt{X(t)} and \texttt{Y(t).}}
  \item \texttt{type} \hspace{1cm} \text{type of sde.}
  \item \texttt{method} \hspace{1cm} \text{the numerical method used.}
\end{itemize}

Author(s)

A.C. Guidoum, K. Boukhetala.

References

Friedman, A. (1975). \textit{Stochastic differential equations and applications}. Volume 1, ACADEMIC PRESS.


See Also

\texttt{snssde3d} for 3-dim sde.

simulate in package "yuima".
## Examples

### Example 1: Ito sde

```r
# dx(t) = 4*(-1-X(t))*Y(t) dt + 0.2 dW1(t)
# dy(t) = 4*(1-Y(t))*X(t) dt + 0.2 dW2(t)
set.seed(1234)

fx <- expression(4*(-1-x)*y , 4*(1-y)*x )
gx <- expression(0.25*y, 0.2*x)

mod2d1 <- snessde2d(drift=fx, diffusion=gx, x0=c(x0=1,y0=-1), M=1000)
mod2d1
summary(mod2d1)

```

```r
dev.new()
plot(mod2d1,type="n")
mx <- apply(mod2d1$X,1,mean)
my <- apply(mod2d1$Y,1,mean)
lines(time(mod2d1),mx,col=1)
lines(time(mod2d1),my,col=2)
legend("topright",c(expression(E(X[t])),expression(E(Y[t]))),lty=1,inset = .01,col=c(1,2),cex=0.95)

```

```r
dev.new()
plot2d(mod2d1) ## in plane (O,X,Y)
lines(my=mx,col=2)
```

### Example 2: Stratonovich sde

```r
# dx(t) = Y(t) dt + 0 o dW1(t)
# dy(t) = (4*(1-X(t)^2)*Y(t) - X(t) ) dt + 0.2 o dW2(t)
set.seed(1234)

fx <- expression( y , (4*(-1-x^2 )* y - x))
gx <- expression( 0 , 0.2)

mod2d2 <- snessde2d(drift=fx, diffusion=gx, type="str", T=1000, N=10000)
mod2d2
plot(mod2d2,pos=2)
dev.new()
plot(mod2d2,union = FALSE)
dev.new()
plot2d(mod2d2, type="n") ## in plane (O,X,Y)
points2d(mod2d2,col=rgb(0,100,0,50,maxColorValue=255), pch=16)
```

---

**snssde3d**  
*Simulation of 3-D Stochastic Differential Equation*
Description

The (S3) generic function `snssde3d` of simulation of solutions to 3-dim stochastic differential equations of Itô or Stratonovich type, with different methods.

Usage

```r
snssde3d(N, ...)  # Default S3 method:
snssde3d(N = 1000, M = 1, x0 = c(0, 0, 0), t0 = 0, T = 1, Dt = NULL,
  drift, diffusion, alpha = 0.5, mu = 0.5, type = c("ito", "str"),
  method = c("euler", "milstein", "predcorr", "smilstein", "taylor",
  "heun", "rk1", "rk2", "rk3"), ...)

# S3 method for class 'snssde3d'
summary(object, at, digits = NULL, ...)
# S3 method for class 'snssde3d'
time(x, ...)
# S3 method for class 'snssde3d'
mean(x, at, ...)
# S3 method for class 'snssde3d'
Median(x, at, ...)
# S3 method for class 'snssde3d'
Mode(x, at, ...)
# S3 method for class 'snssde3d'
quantile(x, at, ...)
# S3 method for class 'snssde3d'
kurtosis(x, at, ...)
# S3 method for class 'snssde3d'
skewness(x, at, ...)
# S3 method for class 'snssde3d'
min(x, at, ...)
# S3 method for class 'snssde3d'
max(x, at, ...)
# S3 method for class 'snssde3d'
moment(x, at, ...)
# S3 method for class 'snssde3d'
cv(x, at, ...)
# S3 method for class 'snssde3d'
bconfint(x, at, ...)
# S3 method for class 'snssde3d'
plot(x, ...)
# S3 method for class 'snssde3d'
lines(x, ...)
# S3 method for class 'snssde3d'
points(x, ...)
# S3 method for class 'snssde3d'
plot3D(x, display = c("persp", "rgl"), ...)
```
Arguments


\begin{align*}
N & \quad \text{number of simulation steps.} \\
M & \quad \text{number of trajectories.} \\
x_0 & \quad \text{initial value of the process } X_t, Y_t \text{ and } Z_t \text{ at time } t_0. \\
t_0 & \quad \text{initial time.} \\
T & \quad \text{ending time.} \\
Dt & \quad \text{time step of the simulation (discretization). If it is } \text{NULL a default } \Delta t = \frac{T-t_0}{N}. \\
\text{drift} & \quad \text{drift coefficient: an expression of four variables } t, x, y \text{ and } z \text{ for process } X_t, Y_t \text{ and } Z_t. \\
\text{diffusion} & \quad \text{diffusion coefficient: an expression of four variables } t, x, y \text{ and } z \text{ for process } X_t, Y_t \text{ and } Z_t. \\
\alpha, \mu & \quad \text{weight of the predictor-corrector scheme; the default } \alpha = 0.5 \text{ and } \mu = 0.5. \\
\text{type} & \quad \text{if type=} \text{"ito" simulation sde of Itô type. else type=} \text{"str" simulation sde of Stratonovich type; the default type=} \text{"ito".} \\
\text{method} & \quad \text{numerical methods of simulation, the default method } = \text{"euler".} \\
x, \text{object} & \quad \text{an object inheriting from class } \text{"snsde3d".} \\
\text{at} & \quad \text{time between } t_0 \text{ and } T. \text{ Monte-Carlo statistics of the solutions } (X_t, Y_t, Z_t) \text{ at time } \text{at. The default } \text{at } = T. \\
\text{digits} & \quad \text{integer, used for number formatting.} \\
\text{display} & \quad \text{"persp" perspective or "rgl" plots.} \\
\ldots & \quad \text{potentially further arguments for (non-default) methods.}
\end{align*}

Details

The function snsde3d returns a mts x of length N+1; i.e. solution of the 3-dim sde \((X_t, Y_t, Z_t)\) of Ito or Stratonovich types; If Dt is not specified, then the best discretization \(\Delta t = \frac{T-t_0}{N}\).

The 3-dim Ito stochastic differential equation is:

\begin{align*}
\frac{dX}{dt} &= a(t, X(t), Y(t), Z(t))dt + b(t, X(t), Y(t), Z(t))dW_1(t) \\
\frac{dY}{dt} &= a(t, X(t), Y(t), Z(t))dt + b(t, X(t), Y(t), Z(t))dW_2(t) \\
\frac{dZ}{dt} &= a(t, X(t), Y(t), Z(t))dt + b(t, X(t), Y(t), Z(t))dW_3(t)
\end{align*}

\begin{align*}
\frac{dX}{dt} &= a(t, X(t), Y(t), Z(t))dt + b(t, X(t), Y(t), Z(t)) \circ dW_1(t) \\
\frac{dY}{dt} &= a(t, X(t), Y(t), Z(t))dt + b(t, X(t), Y(t), Z(t)) \circ dW_2(t) \\
\frac{dZ}{dt} &= a(t, X(t), Y(t), Z(t))dt + b(t, X(t), Y(t), Z(t)) \circ dW_3(t)
\end{align*}

\(W_1(t), W_2(t), W_3(t)\) three standard Brownian motion independent.

The methods of approximation are classified according to their different properties. Mainly two criteria of optimality are used in the literature: the strong and the weak (orders of) convergence. The method of simulation can be one among: Euler-Maruyama Order 0.5, Milstein Order 1, Milstein Second-Order, Predictor-Corrector method, Itô-Taylor Order 1.5, Heun Order 2 and Runge-Kutta Order 1, 2 and 3.

An overview of this package, see \texttt{browseVignettes(‘Sim.DiffProc’)} for more informations.
Value

snssde3d returns an object inheriting from `class "snssde3d"`.

- X, Y, Z: an invisible mts (3-dim) object (X(t),Y(t),Z(t)).
- driftx, drifty, driftz: drift coefficient of X(t), Y(t) and Z(t).
- diffx, diffy, diffz: diffusion coefficient of X(t), Y(t) and Z(t).
- type: type of sde.
- method: the numerical method used.

Author(s)

A.C. Guidoum, K. Boukhetala.

References

Friedman, A. (1975). *Stochastic differential equations and applications*. Volume 1, ACADEMIC PRESS.


See Also

snssde1d and snssde2d for 1- and 2-dim sde.
sde.sim in package "sde". simulate in package "yuima".

Examples

```r
## Example 1: Ito sde
## dX(t) = (2*(Y(t)>0)-2*(Z(t)<=0)) dt + 0.2 * dW1(t)
## dY(t) = -2*Y(t) dt + 0.2 * dW2(t)
## dZ(t) = -2*Z(t) dt + 0.2 * dW3(t)
## W1(t), W2(t) and W3(t) three independent Brownian motion
## set.seed(1234)

fx <- expression(2*(y>0)-2*(z<=0) , -2*y, -2*z)
gx <- rep(expression(0.2),3)

mod3d1 <- snssde3d(x0=c(0,2,-2),drift=fx,diffusion=gx,M=500,Dt=0.003)
mod3d1
summary(mod3d1)
##
## dev.new()
plot(mod3d1,type="n")
mx <- apply(mod3d1$X,1,mean)
my <- apply(mod3d1$Y,1,mean)
mz <- apply(mod3d1$Z,1,mean)
lines(time(mod3d1),mx,col=1)
lines(time(mod3d1),my,col=2)
lines(time(mod3d1),mz,col=3)
legend("topright",c(expression(E(X[t])),expression(E(Y[t])),
expression(E(Z[t]))),lty=1,inset = .01,col=c(1,2,3),cex=0.95)
##
## dev.new()
plot3D(mod3d1,display="persp")  ## in space (0,X,Y,Z)

## Example 2: Stratonovich sde
## dX(t) = Y(t)* dt
## dY(t) = (4*( 1-X(t)^2 )* Y(t) - X(t))* dt + 0.2 o dW2(t)
## dZ(t) = (4*( 1-X(t)^2 )* Z(t) - X(t))* dt + 0.2 o dW3(t)
## set.seed(1234)

fx <- expression( y, (4*( 1-x^2 )* y - x), (4*( 1-x^2 )* z - x))
gx <- expression( 0, 0.2, 0.2)

mod3d2 <- snssde3d(drift=fx,diffusion=gx,N=10000,T=100,type="str")
mod3d2
##
## dev.new()
plot(mod3d2,pos=2)
##
## dev.new()
plot(mod3d2,union = FALSE)
```
## Description

The (S3) generic function `st.int` of simulation of stochastic integrals of Itô or Stratonovich type.

### Usage

```r
st.int(expr, ...)  
## Default S3 method:
st.int(expr, lower = 0, upper = 1, M = 1, subdivisions = 1000L,
      type = c("ito", "str"), ...)

## S3 method for class 'st.int'
summary(object, at, digits=NULL, ...)
## S3 method for class 'st.int'
time(x, ...)
## S3 method for class 'st.int'
mean(x, at, ...)
## S3 method for class 'st.int'
Median(x, at, ...)
## S3 method for class 'st.int'
Mode(x, at, ...)
## S3 method for class 'st.int'
quantile(x, at, ...)
## S3 method for class 'st.int'
kurtosis(x, at, ...)
## S3 method for class 'st.int'
min(x, at, ...)
## S3 method for class 'st.int'
max(x, at, ...)
## S3 method for class 'st.int'
skewness(x, at, ...)
## S3 method for class 'st.int'
moment(x, at, ...)
## S3 method for class 'st.int'
cv(x, at, ...)
## S3 method for class 'st.int'
bconfint(x, at, ...)

## S3 method for class 'st.int'
plot(x, ...)
```
Arguments

expr an expression of two variables \( t \) (time) and \( w \) (w: standard Brownian motion).
lower, upper the lower and upper end points of the interval to be integrate.
M number of trajectories (Monte-Carlo).
subdivisions the maximum number of subintervals.
type Itô or Stratonovich integration.
x, object an object inheriting from class "st.int".
at time between lower and upper. Monte-Carlo statistics of stochastic integral at time at. The default at = upper.
digits integer, used for number formatting.
... potentially further arguments for (non-default) methods.

Details

The function \texttt{stNint} returns a \texttt{ts} \( x \) of length \( N+1 \); i.e. simulation of stochastic integrals of Itô or Stratonovich type.

The Itô interpretation is:

\[
\int_{t_0}^t f(s) \, dW_s = \lim_{N \to \infty} \sum_{i=1}^{N} f(t_{i-1})(W_{t_i} - W_{t_{i-1}})
\]

The Stratonovich interpretation is:

\[
\int_{t_0}^t f(s) \circ dW_s = \lim_{N \to \infty} \sum_{i=1}^{N} f(t_{i-1}) \left( \frac{t_i + t_{i-1}}{2} \right) (W_{t_i} - W_{t_{i-1}})
\]

An overview of this package, see \texttt{browseVignettes('Sim.DiffProc')} for more informations.

Value

\texttt{st.int} returns an object inheriting from \texttt{class }"st.int".

\( x \) the final simulation of the integral, an invisible \texttt{ts} object.
\texttt{fun} function to be integrated.
\texttt{type} type of stochastic integral.
\texttt{subdivisions} the number of subintervals produced in the subdivision process.

Author(s)

A.C. Guidoum, K. Boukhetala.
References


See Also

snnssde1d, snssde2d and snssde3d for 1,2 and 3-dim sde.

Examples

## Example 1: Ito integral
## f(t,w(t)) = int(exp(w(t) - 0.5*t) * dw(s)) with t in [0,1]
set.seed(1234)

f <- expression( exp(w-0.5*t) )
mod1 <- st.int(expr=f,type="ito",M=50,lower=0,upper=1)
mod1
summary(mod1)
## Display
plot(mod1)
lines(time(mod1),apply(mod1$X,1,mean),col=2,lwd=2)
lines(time(mod1),apply(mod1$X,1,bconfint,level=0.95)[1,],col=4,lwd=2)
lines(time(mod1),apply(mod1$X,1,bconfint,level=0.95)[2,],col=4,lwd=2)
legend("topleft",c("mean path","bound of", "95% percent confidence"),
inset = .01,col=c(2,4),lwd=2,cex=0.8)

## Example 2: Stratonovich integral
## f(t,w(t)) = int(w(s) o dw(s)) with t in [0,1]
set.seed(1234)

g <- expression( w )
mod2 <- st.int(expr=g,type="str",M=50,lower=0,upper=1)
mod2
summary(mod2)
## Display
plot(mod2)
lines(time(mod2),apply(mod2$X,1,mean),col=2,lwd=2)
lines(time(mod2),apply(mod2$X,1,bconfint,level=0.95)[1,],col=4,lwd=2)
lines(time(mod2),apply(mod2$X,1,bconfint,level=0.95)[2,],col=4,lwd=2)
legend("topleft",c("mean path","bound of", "95% percent confidence"),
inset = .01,col=c(2,4),lwd=2,cex=0.8)
Description

These methods produce the related LaTeX table and mathematic expression for Sim.DiffProc environment.

Usage

TEX.sde(object, ...)

## Default S3 method:
TEX.sde(object, ...)

Arguments

object an objects from class MCM.sde and MEM.sde. Or an R vector of expression of SDEs, i.e., drift and diffusion coefficients.
... arguments to be passed to kable function if object from class MCM.sde.

Details

New tools for constructing tables and mathematical expressions with Sim.DiffProc package. An overview of this package, see browseVignettes('Sim.DiffProc') for more informations.

Author(s)

A.C. Guidoum

References


See Also

kable create tables in LaTeX, HTML, Markdown and reStructuredText.
tolatex converting R Objects to BibTeX or LaTeX.
**Examples**

```r
## LaTeX mathematic for an R expression of SDEs
## Copy and paste the following output in your LaTeX file

# Example 1

f <- expression(-mu * x)
g <- expression(mu2 * sqrt(x))
TEX.sde(object = c(drift = f, diffusion = g))

# Example 2

f <- expression(mu*cos(mu2+z),mu*sin(mu2+z),0)
g <- expression(sigma,sigma,alpha)
TEX.sde(object = c(drift = f, diffusion = g))

## LaTeX mathematic for object of class 'MEM.sde'
## Copy and paste the following output in your LaTeX file

# Example 3

mem.mod3d <- MEM.sde(drift = f, diffusion = g)
TEX.sde(object = mem.mod3d)

## LaTeX table for object of class 'MCM.sde'
## Copy and paste the following output in your LaTeX file

# Example 4

## Not run:
mu1=0.25; mu2=3; sigma=0.05; alpha=0.03
mod3d <- snessde3d(drift=f,diffusion=g,x0=c(x=0,y=0,z=0),M=100,T=10)
stat.fun3d <- function(data, i){
d <- data[i,]
return(c(mean(d$x),mean(d$y),mean(d$z),
        var(d$x),var(d$y),var(d$z)))
}
mcm.mod3d = MCM.sde(mod3d,statistic=stat.fun3d,R=10,parallel="snow",ncpus=parallel::detectCores(),
names=c("m1","m2","m3","S1","S2","S3"))
TEX.sde(object = mcm.mod3d, booktabs = TRUE, align = "r", caption = "LaTeX table for Monte Carlo results generated by \code{TEX.sde()} method."

## End(Not run)
```
Index

*Topic **BM**
  BM, 14
*Topic **LaTeX**
  \text{TEX}.sde, 77
*Topic **Monte-Carlo**
  MCM. sde, 44
*Topic **datasets**
  Irates, 43
*Topic **fit**
  fitsde, 26
*Topic **fpt**
  fptsde1d, 31
  fptsde2d, 34
  fptsde3d, 38
*Topic **moment equations**
  MEM. sde, 48
*Topic **mts**
  bridgesde1d, 16
  bridgesde2d, 19
  bridgesde3d, 22
  fptsde2d, 34
  fptsde3d, 38
  MCM. sde, 44
  rsde1d, 54
  rsde2d, 56
  rsde3d, 59
  snssde1d, 62
  snssde2d, 65
  snssde3d, 69
*Topic **package**
  Sim.DiffProc-package, 2
*Topic **parallel**
  MCM. sde, 44
*Topic **random generators**
  rsde1d, 54
  rsde2d, 56
  rsde3d, 59
*Topic **sde**
  BM, 14
  bridgesde1d, 16
  bridgesde2d, 19
  bridgesde3d, 22
  fitsde, 26
  fptsde1d, 31
  fptsde2d, 34
  fptsde3d, 38
  HWV, 41
  MCM. sde, 44
  MEM. sde, 48
  rsde1d, 54
  rsde2d, 56
  rsde3d, 59
  snssde1d, 62
  snssde2d, 65
  snssde3d, 69
  st.int, 74
*Topic **transition density**
  rsde1d, 54
  rsde2d, 56
  rsde3d, 59
*Topic **ts**
  BM, 14
  bridgesde1d, 16
  bridgesde2d, 19
  bridgesde3d, 22
  fitsde, 26
  fptsde1d, 31
  fptsde2d, 34
  fptsde3d, 38
  HWV, 41
  MCM. sde, 44
  rsde1d, 54
  rsde2d, 56
  rsde3d, 59
  snssde1d, 62
  snssde2d, 65
  snssde3d, 69
  st.int, 74
ABM (BM), 14
AIC. fitsde (fitsde), 26
Approx. fpt. density, 33, 37, 40

BB (BM), 14
BBridge, 15
bconfint (moment), 52
bconfint. bridgesde1d (bridgesde1d), 16
bconfint. bridgesde2d (bridgesde2d), 19
bconfint. bridgesde3d (bridgesde3d), 22
bconfint. snssde1d (snssde1d), 62
bconfint. snssde2d (snssde2d), 65
bconfint. snssde3d (snssde3d), 69
bconfint. st. int (st. int), 74
BIC. fitsde (fitsde), 26
BiGQD. density, 57
BM, 14, 15
bridgesde1d, 7, 16, 22, 25, 54
bridgesde2d, 7, 18, 19, 25, 53, 57
bridgesde3d, 7, 18, 22, 53, 60
browseVignettes, 11, 18, 21, 24, 27, 32, 36, 39, 43, 49, 55, 57, 60, 63, 67, 71, 75, 77

character, 26
class, 18, 21, 24, 27, 64, 68, 72, 75
coeff. fitsde (fitsde), 26
confint. fitsde (fitsde), 26
cv (moment), 52
cv. bridgesde1d (bridgesde1d), 16
cv. bridgesde2d (bridgesde2d), 19
cv. bridgesde3d (bridgesde3d), 22
cv. fptsde1d (fptsde1d), 31
cv. fptsde2d (fptsde2d), 34
cv. fptsde3d (fptsde3d), 38
cv. snssde1d (snssde1d), 62
cv. snssde2d (snssde2d), 65
cv. snssde3d (snssde3d), 69
cv. st. int (st. int), 74

DBridge, 18, 22, 25
dcBS, 55
dcCIR, 55
dcElerian, 28
dcEuler, 28
dcKessler, 28
dcOU, 55
dcOzaki, 28
dcShoji, 28
dcSim, 28, 55
density, 32, 36, 39, 54, 55, 57, 60
dfptsde1d, 10
dfptsde1d (fptsde1d), 31
dfptsde2d, 10
dfptsde2d (fptsde2d), 34
dfptsde3d, 10
dfptsde3d (fptsde3d), 38
dsde1d, 9
dsde1d (rsde1d), 54
ndsde2d, 9
dsde2d (rsde2d), 56
ndsde3d, 9
ndsde3d (rsde3d), 59
dsOU, 55

expression, 17, 21, 24, 26, 32, 35, 39, 49, 63, 67, 71, 75, 77

fitsde, 8, 26
FPTL, 33, 37, 40
fptsde1d, 10, 31, 32, 37, 40
fptsde2d, 10, 33, 34, 35, 40
fptsde3d, 10, 33, 37, 38, 39

GBM, 15
GBM (BM), 14
gmm, 28
GQD. density, 55
GQD. Tipassage, 33, 37, 40

HPloglik, 28
HWV, 41

Irates, 43, 43

kable, 77
kde, 55, 57, 60
kde2d, 36, 57
kde3d, 60
kurtosis (moment), 52
kurtosis. bridgesde1d (bridgesde1d), 16
kurtosis. bridgesde2d (bridgesde2d), 19
kurtosis. bridgesde3d (bridgesde3d), 22
kurtosis. fptsde1d (fptsde1d), 31
kurtosis. fptsde2d (fptsde2d), 34
kurtosis. fptsde3d (fptsde3d), 38
kurtosis. snssde1d (snssde1d), 62
kurtosis. snssde2d (snssde2d), 65
kurtosis. snssde3d (snssde3d), 69
INDEX

kurtosisNstNint (st.int), 74

lines.bridges1d (bridges1d), 16
lines.bridges2d (bridges2d), 19
lines.bridges3d (bridges3d), 22
lines.snss1d (snss1d), 62
lines.snss2d (snss2d), 65
lines.snss3d (snss3d), 69
lines.st.int (st.int), 74
lines2d (plot2d), 53
lines2d.bridges2d (bridges2d), 19
lines2d.snss2d (snss2d), 65
logLik.fitsde (fitsde), 26

max.bridges1d (bridges1d), 16
max.bridges2d (bridges2d), 19
max.bridges3d (bridges3d), 22
max.fitsde1d (fitsde1d), 31
max.fitsde2d (fitsde2d), 34
max.fitsde3d (fitsde3d), 38
max.snss1d (snss1d), 62
max.snss2d (snss2d), 65
max.snss3d (snss3d), 69
max.st.int (st.int), 74
MCM.sde, 44, 49, 77

mean.bridges1d (bridges1d), 16
mean.bridges2d (bridges2d), 19
mean.bridges3d (bridges3d), 22
mean.fitsde1d (fitsde1d), 31
mean.fitsde2d (fitsde2d), 34
mean.fitsde3d (fitsde3d), 38
mean.snss1d (snss1d), 62
mean.snss2d (snss2d), 65
mean.snss3d (snss3d), 69
mean.st.int (st.int), 74
Median (moment), 52
Median.bridges1d (bridges1d), 16
Median.bridges2d (bridges2d), 19
Median.bridges3d (bridges3d), 22
Median.fitsde1d (fitsde1d), 31
Median.fitsde2d (fitsde2d), 34
Median.fitsde3d (fitsde3d), 38
Median.snss1d (snss1d), 62
Median.snss2d (snss2d), 65
Median.snss3d (snss3d), 69
Median.st.int (st.int), 74
MEM.sde, 46, 48, 77

min.bridges1d (bridges1d), 16
min.bridges2d (bridges2d), 19

min.fitsde1d (fitsde1d), 22
min.fitsde2d (fitsde2d), 34
min.fitsde3d (fitsde3d), 38
min.snss1d (snss1d), 62
min.snss2d (snss2d), 65
min.snss3d (snss3d), 69
min.st.int (st.int), 74
missing, 42
Mode (moment), 52
Mode.bridges1d (bridges1d), 16
Mode.bridges2d (bridges2d), 19
Mode.bridges3d (bridges3d), 22
Mode.fitsde1d (fitsde1d), 31
Mode.fitsde2d (fitsde2d), 34
Mode.fitsde3d (fitsde3d), 38
Mode.snss1d (snss1d), 62
Mode.snss2d (snss2d), 65
Mode.snss3d (snss3d), 69
Mode.st.int (st.int), 74
moment, 52
moment.bridges1d (bridges1d), 16
moment.bridges2d (bridges2d), 19
moment.bridges3d (bridges3d), 22
moment.fitsde1d (fitsde1d), 31
moment.fitsde2d (fitsde2d), 34
moment.fitsde3d (fitsde3d), 38
moment.snss1d (snss1d), 62
moment.snss2d (snss2d), 65
moment.snss3d (snss3d), 69
moment.st.int (st.int), 74

Normal, 15
NULL, 14, 17, 21, 24, 63, 67, 71

ode, 49
optim, 26, 27
OU (HWV), 41

par, 53
plot.bridges1d (bridges1d), 16
plot.bridges2d (bridges2d), 19
plot.bridges3d (bridges3d), 22
plot.dfitsde1d (fitsde1d), 31
plot.dfitsde2d (fitsde2d), 34
plot.dfitsde3d (fitsde3d), 38
plot.dsde1d (rsde1d), 54
plot.dsde2d (rsde2d), 56
plot.dsde3d (rsde3d), 59
plot.MCM.sde (MCM.sde), 44
plot.snsdse1d (snsdse1d), 62
plot.snsdse2d (snsdse2d), 65
plot.snsdse3d (snsdse3d), 69
plot.st.int (st.int), 74
plot2d, 53
plot2d.bridgesde2d (bridgesde2d), 19
plot2d.snsdse2d (snsdse2d), 65
plot3D (plot2d), 53
plot3d, 53
plot3d.bridgesde3d (bridgesde3d), 22
plot3D.snsdse3d (snsdse3d), 69
points.bridgesde1d (bridgesde1d), 16
points.bridgesde2d (bridgesde2d), 19
points.bridgesde3d (bridgesde3d), 22
points.snsdse1d (snsdse1d), 62
points.snsdse2d (snsdse2d), 65
points.snsdse3d (snsdse3d), 69
points.st.int (st.int), 74
points2d (plot2d), 53
points2d.bridgesde2d (bridgesde2d), 19
points2d.snsdse2d (snsdse2d), 65
print.bridgesde1d (bridgesde1d), 16
print.bridgesde2d (bridgesde2d), 19
print.bridgesde3d (bridgesde3d), 22
print.dfptsde1d (dfptsde1d), 31
print.dfptsde2d (dfptsde2d), 34
print.dfptsde3d (dfptsde3d), 38
print.dsde1d (dsde1d), 54
print.dsde2d (dsde2d), 56
print.rsde3d (rsde3d), 59
print.fitsde (fitsde), 26
print.fptsde1d (fptsde1d), 31
print.fptsde2d (fptsde2d), 34
print.fptsde3d (fptsde3d), 38
print.MCM.sde (MCM.sde), 44
print.MEM.sde (MEM.sde), 48
print.snsdse1d (snsdse1d), 62
print.snsdse2d (snsdse2d), 65
print.snsdse3d (snsdse3d), 69
print.st.int (st.int), 74
quantile.bridgesde1d (bridgesde1d), 16
quantile.bridgesde2d (bridgesde2d), 19
quantile.bridgesde3d (bridgesde3d), 22
quantile.fptsde1d (fptsde1d), 31
quantile.fptsde2d (fptsde2d), 34
quantile.fptsde3d (fptsde3d), 38
quantile.snsdse1d (snsdse1d), 62
quantile.snsdse2d (snsdse2d), 65
quantile.snsdse3d (snsdse3d), 69
quantile.st.int (st.int), 74
rcBS, 55, 60
rcCIR, 55, 60
rcOU, 42, 55, 60
rng, 55, 57, 60
rsde1d, 9, 54
rsde2d, 9, 56
rsde3d, 9, 59
rsOU, 42, 55, 60
scatterplot3d, 53
dse.sim, 64, 73
Sim.DiffProc (Sim.DiffProc-package), 2
Sim.DiffProc-package, 2
skewness (moment), 52
skewness.bridgesde1d (bridgesde1d), 16
skewness.bridgesde2d (bridgesde2d), 19
skewness.bridgesde3d (bridgesde3d), 22
skewness.fptsde1d (fptsde1d), 31
skewness.fptsde2d (fptsde2d), 34
skewness.fptsde3d (fptsde3d), 38
skewness.snsdse1d (snsdse1d), 62
skewness.snsdse2d (snsdse2d), 65
skewness.snsdse3d (snsdse3d), 69
skewness.st.int (st.int), 74
sm.density, 39, 55, 57, 60
snsdse1d, 5, 17, 32, 45, 54, 62, 73, 76
snsdse2d, 5, 21, 35, 45, 53, 57, 64, 65, 73, 76
snsdse3d, 5, 24, 39, 45, 53, 60, 64, 68, 69, 76
st.int, 3, 74
summary.bridgesde1d (bridgesde1d), 16
summary.bridgesde2d (bridgesde2d), 19
summary.bridgesde3d (bridgesde3d), 22
summary.fitsde (fitsde), 26
summary.fptsde1d (fptsde1d), 31
summary.fptsde2d (fptsde2d), 34
summary.fptsde3d (fptsde3d), 38
summary.MEM.sde (MEM.sde), 48
summary.snsdse1d (snsdse1d), 62
summary.snsdse2d (snsdse2d), 65
summary.snsdse3d (snsdse3d), 69
summary.st.int (st.int), 74
TEX.sde, 77
time.bridgesde1d (bridgesde1d), 16
time.bridgesde2d (bridgesde2d), 19
INDEX

time.bridgesde3d (bridgesde3d), 22
time.snssde1d (snssde1d), 62
time.snssde2d (snssde2d), 65
time.snssde3d (snssde3d), 69
time.st.int (st.int), 74
toLatex, 77
truehist, 32, 36, 39, 54, 57, 60
ts, 18, 26, 63, 64, 75
vcov.fitsde (fitsde), 26