Package 'DiffusionRgqd'

August 29, 2016

Version 0.1.3

Title Inference and Analysis for Generalized Quadratic Diffusions

Description Tools for performing inference and analysis on a class of quadratic diffusion processes for both scalar and bivariate diffusion systems. For scalar diffusions, a module is provided for solving first passage time problems for both time-homogeneous and time-inhomogeneous GQDs.

URL https://github.com/eta21

BugReports https://github.com/eta21/DiffusionRgqd/issues

MailingList Please send questions and comments to etiennead@gmail.com.

Depends R (>= 3.2.1)

Imports Rcpp, RcppArmadillo, rgl, colorspace

LinkingTo Rcpp, RcppArmadillo

Suggests knitr, coda, fptdApprox, Quandl, R.rsp

License GPL (>= 2)

VignetteBuilder knitr, R.rsp

NeedsCompilation yes

Author Etienne A.D. Pienaar [aut, cre],
Melvin M. Varughese [ctb]

Maintainer Etienne A.D. Pienaar <etiennead@gmail.com>

Repository CRAN

Date/Publication 2016-08-26 09:36:56

R topics documented:

`DiffusionRgqd-package` *A package for Performing Inference and Analysis on Generalized Quadratic Diffusion Processes (GQDs).*

Description

DiffusionRgqd is a toolbox for performing analysis and inference on a class of diffusion processes with quadratic drift and diffusion. The package consists of functions for performing likelihood based inference and transitional density approximations for both 1D and 2D GQDs. For scalar diffusions, a module is provided for solving first passage time problems for both time homogeneous and time inhomogeneous GQDs.

Details

Package:	DiffusionRgqd
Type:	Package
Version:	0.1.2
Date:	2015-12-01
License:	GPL (>= 2)

The package is designed around an interface whereby the user supplies standard R-functions dictating the functional form of the coefficients of the GQD. For example, for scalar GQDs nested within the stochastic differential equation:

$$dX_t = (G_0(t) + G_1(t)X_t + G_2(t)X_t^2)dt + \sqrt{Q_0(t) + Q_1(t)X_t + Q_2(t)X_t^2}dW_t,$$

the user supplies `G0(t)`,`G1(t)` and `Q1(t)` etc. These coefficients may depend on a both vector of parameters and time. The package handles all the necessary mathematics and algorithmic construc-

tion. Furthermore, computational efficiency is optimized by constructing algorithms in C++ using the **Rcpp** and **RcppArmadillo** libraries.

Functions included in the package:

`BiGQD.density`	:	Generate the transitional density of a 2D GQD.
`BiGQD.mcmc`*	:	Conduct inference via MCMC on a 2D GQD.
`BiGQD.mle`*	:	Calculate MLEs for a 2D GQD.
`GQD.density`	:	Generate the transitional density of a 1D GQD.
`GQD.mcmc`*	:	Conduct inference via MCMC on a 1D GQD.
`GQD.mle`*	:	Calculate MLEs for a 1D GQD.
`GQD.passage`*	:	Approximate the first passage time density of a time homogeneous GQD to a barrier.
`GQD.TIpassage`*	:	Approximate the first passage time density of a time-inhomogeneous GQD to a barrier.

* Functions use C++.

Author(s)

Etienne A.D. Pienaar: <etiennead@gmail.com>

Maintainer: Etienne A.D. Pienaar (<etiennead@gmail.com>)

References

Updates available on GitHub at https://github.com/eta21.

See Also

`BiGQD.mcmc`, `BiGQD.mle`, `GQD.mcmc`, `GQD.dic`, `GQD.mle`,`GQD.remove`, `GQD.passage`, `GQD.TIpassage` and `GQD.density`.

Examples

```
## Not run:
example(GQD.density)
example(BiGQD.density)

## End(Not run)
```

`BiGQD.density` *Generate the Transition Density of a Bivariate Generalized Quadratic Diffusion Model (2D GQD).*

Description

`BiGQD.density` generates approximate transitional densities for bivariate generalized quadratic diffusions (GQDs). Given a starting coordinate, (`Xs`, `Ys`), the approximation is evaluated over a lattice `Xt` x `Yt` for an equispaced discretization (intervals of width `delt`) of the transition time horizon [`s`, `t`].

`BiGQD.density()` approximates the transitional density of jump diffusions of the form:

$$d\mathbf{X}_t = \boldsymbol{\mu}(\mathbf{X}_t, t)dt + \boldsymbol{\sigma}(\mathbf{X}_t, t)d\mathbf{W}_t$$

where

$$\mu^{(1)}(\mathbf{X}_t, t) = \sum_{i+j\leq 2} a_{ij}(t) X_t^i Y_t^j,$$

$$\mu^{(2)}(\mathbf{X}_t, t) = \sum_{i+j\leq 2} b_{ij}(t) X_t^i Y_t^j,$$

and

$$\boldsymbol{\sigma}(X_t, Y_t, t)\boldsymbol{\sigma}'(X_t, Y_t, t) = (\gamma^{(i,j)}(\mathbf{X}_t, t))_{i,j=1,2}$$

such that

$$\gamma^{(1,1)}(\mathbf{X}_t, t) = \sum_{i+j\leq 2} c_{ij}(t) X_t^i Y_t^j,$$

$$\gamma^{(1,2)}(\mathbf{X}_t, t) = \sum_{i+j\leq 2} d_{ij}(t) X_t^i Y_t^j,$$

$$\gamma^{(2,1)}(\mathbf{X}_t, t) = \sum_{i+j\leq 2} e_{ij}(t) X_t^i Y_t^j,$$

$$\gamma^{(2,2)}(\mathbf{X}_t, t) = \sum_{i+j\leq 2} f_{ij}(t) X_t^i Y_t^j.$$

Usage

```
BiGQD.density(Xs, Ys, Xt, Yt, s, t, delt=1/100, Dtype='Saddlepoint', print.output=TRUE,
              eval.density=TRUE)
```

Arguments

`Xt`	x-Coordinates of the lattice at which to evaluate the transition density.
`Yt`	y-Coordinates of the lattice at which to evaluate the transition density.
`Xs`	Initial x-coordinate.
`Ys`	Initial y-coordinate.
`s`	Starting time of the diffusion.
`t`	Final time at which to evaluate the transition density.
`delt`	Step size for numerical solution of the cumulant system. Also used for the discretization of the transition time-horizon. See warnings **[1]** and **[2]**.
`Dtype`	The density approximant to use. Valid types are `"Saddlepoint"` (default) or `"Edgeworth"`.
`print.output`	If `TRUE` information about the model and algorithm is printed to the console.
`eval.density`	If `TRUE`, the density is evaluated in addition to calculating the moment eqns.

Details

`BiGQD.density` constructs an approximate transition density for a class of quadratic diffusion models. This is done by first evaluating the trajectory of the cumulants/moments of the diffusion numerically as the solution of a system of ordinary differential equations over a time horizon `[s,t]` split into equi-distant points `delt` units apart. Subsequently, the resulting cumulants/moments are carried into a density approximant (by default, a saddlepoint approximation) in order to evaluate the transtion surface.

Value

`density`	3D Array containing approximate density values. Note that the 3rd dimension represents time.
`Xmarginal`	2D Array containing approximate Xt-marginal density values (calculated using the univariate saddlepoint approximation).
`Ymarginal`	2D Array containing approximate Yt-marginal density values (calculated using the univariate saddlepoint approximation).
`Xt`	Copy of x-coordinates.
`Yt`	Copy of y-coordinates.
`time`	A vector containing the time mesh at which the density was evaluated.
`cumulants`	A matrix giving the cumulants of the diffusion. Cumulants are indicated by row-names.

Warning

Warning [1]: The system of ODEs that dictate the evolution of the cumulants do so approximately. Thus, although it is unlikely such cases will be encountered in inferential contexts, it is worth checking (by simulation) whether cumulants accurately replicate those of the target GQD. Furthermore, it may in some cases occur that the cumulants are indeed accurate whilst the density approximation fails. This can again be verified by simulation.

Warning [2]: The parameter `delt` is also used as the stepsize for solving a system of ordinary differential equations (ODEs) that govern the evolution of the cumulants of the diffusion. As such `delt` is required to be small for highly non-linear models in order to ensure sufficient accuracy.

Author(s)

Etienne A.D. Pienaar: `<etiannead@gmail.com>`

References

Updates available on GitHub at https://github.com/eta21.

Daniels, H.E. 1954 Saddlepoint approximations in statistics. *Ann. Math. Stat.*, **25**:631–650.

Eddelbuettel, D. and Romain, F. 2011 Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, **40**(8):1–18,. URL http://www.jstatsoft.org/v40/i08/.

Eddelbuettel, D. 2013 *Seamless R and C++ Integration with Rcpp*. New York: Springer. ISBN 978-1-4614-6867-7.

Eddelbuettel, D. and Sanderson, C. 2014 Rcpparmadillo: Accelerating R with high-performance C++ linear algebra. *Computational Statistics and Data Analysis*, **71**:1054–1063. URL http://dx.doi.org/10.1016/j.csda.2013.02.005.

Feagin, T. 2007 A tenth-order Runge-Kutta method with error estimate. *In Proceedings of the IAENG Conf. on Scientifc Computing*.

Varughese, M.M. 2013 Parameter estimation for multivariate diffusion systems. *Comput. Stat. Data An.*, **57**:417–428.

See Also

See BiGQD.mcmc and BiGQD.mle for likelihood based inference procedures for bivariate GQDs.

Examples

```
#=============================================================================
# Generate the transition density of a stochastic perturbed Lotka-Volterra
# preditor-prey model, with state-dependent volatility:
# dX = (1.5X-0.4*X*Y)dt          +sqrt(0.05*X)dWt
# dY = (-1.5Y+0.4*X*Y-0.2*Y^2)dt +sqrt(0.10*Y)dBt
#-----------------------------------------------------------------------------

# Remove any existing coefficients
GQD.remove()

# Define the X dimesnion coefficients
a10 = function(t){1.5}
a11 = function(t){-0.4}
c10 = function(t){0.05}

# Define the Y dimension coefficients
b01 = function(t){-1.5}
b11 = function(t){0.4}
b02 = function(t){-0.2}
f01 = function(t){0.1}

# Approximate the transition density
res = BiGQD.density(5,5,seq(3,8,length=25),seq(2,6,length=25),0,10,1/100)


#-----------------------------------------------------------------------------
# Visuallize the density
#-----------------------------------------------------------------------------

par(ask=FALSE)
# Load simulated trajectory of the joint expectation:
data(SDEsim3)
attach(SDEsim3)

# We will simulate some trajectories (crudely) as well:
N=1000; delt= 1/100  # 1000 trajectories
```

```
X1=rep(5,N)          # Initial values for each trajectory
X2=rep(5,N)

for(i in 1:1001)
{
  # Applly Euler-Murayama scheme to the LV-model
  X1=pmax(X1+(a10(d)*X1+a11(d)*X1*X2)*delt+sqrt(c10(d)*X1)*rnorm(N,sd=sqrt(delt)),0)
  X2=pmax(X2+(b01(d)*X2+b11(d)*X1*X2+b02(d)*X2^2)*delt+sqrt(f01(d)*X2)*rnorm(N,sd=sqrt(delt)),0)

  # Now illustrate the density:
  filled.contour(res$Xt,res$Yt,res$density[,,i],
  main=paste0('Transition Density \n (t = ',res$time[i],')'),
  color.palette=colorRampPalette(c('white','green','blue','red'))
  ,xlab='Prey',ylab='Preditor',plot.axes=
   {
     # Add simulated trajectories
     points(X2~X1,pch=20,col='grey47',cex=0.01)
     # Add trajectory of simulated expectation
     lines(my~mx,col='grey57')
     # Show the predicted expectation from BiGQD.density()
     points(res$cumulants[5,i]~res$cumulants[1,i],bg='white',pch=21,cex=1.5)
     axis(1);axis(2);
     # Add a legend
     legend('topright',lty=c('solid',NA,NA),col=c('grey57','grey47','black'),
            pch=c(NA,20,21),legend=c('Simulated Expectation','Simulated Trajectories'
            ,'Predicted Expectation'))
   })
}

#==============================================================================
```

`BiGQD.mcmc` *MCMC Inference on Bivariate Generalized Quadratic Diffusions (2D GQDs).*

Description

`BiGQD.mcmc()` uses parametrised coefficients (provided by the user as R-functions) to construct a C++ program in real time that allows the user to perform Bayesian inference on the resulting diffusion model. Given a set of starting parameters and other input parameters, a MCMC chain is returned for further analysis.

`BiGQD.density()` approximates the transitional density of jump diffusions of the form:

$$d\mathbf{X}_t = \boldsymbol{\mu}(\mathbf{X}_t, t)dt + \boldsymbol{\sigma}(\mathbf{X}_t, t)d\mathbf{W}_t$$

where

$$\mu^{(1)}(\mathbf{X}_t, t) = \sum_{i+j\leq 2} a_{ij}(t) X_t^i Y_t^j,$$

$$\mu^{(2)}(\mathbf{X}_t, t) = \sum_{i+j\leq 2} b_{ij}(t) X_t^i Y_t^j,$$

and

$$\boldsymbol{\sigma}(X_t, Y_t, t)\boldsymbol{\sigma}'(X_t, Y_t, t) = (\gamma^{(i,j)}(\mathbf{X}_t, t))_{i,j=1,2}$$

such that

$$\gamma^{(1,1)}(\mathbf{X}_t, t) = \sum_{i+j\leq 2} c_{ij}(t) X_t^i Y_t^j,$$

$$\gamma^{(1,2)}(\mathbf{X}_t, t) = \sum_{i+j\leq 2} d_{ij}(t) X_t^i Y_t^j,$$

$$\gamma^{(2,1)}(\mathbf{X}_t, t) = \sum_{i+j\leq 2} e_{ij}(t) X_t^i Y_t^j,$$

$$\gamma^{(2,2)}(\mathbf{X}_t, t) = \sum_{i+j\leq 2} f_{ij}(t) X_t^i Y_t^j.$$

Usage

```
BiGQD.mcmc(X, time, mesh=10, theta, sds, updates, burns=min(round(updates/2),25000),
         RK.order=4, exclude=NULL, plot.chain=TRUE, Tag=NA, Dtype='Saddlepoint',
         recycle=FALSE, rtf=runif(2), wrt=FALSE, print.output=TRUE, palette = "mono")
```

Arguments

X	A matrix containing rows of data points to be modelled. Although observations are allowed to be non-equidistant, observations in both dimensions are assumed to occur at the same time epochs (i.e. time gives the time signature for both dimensions).
time	A vector containing the time epochs at which observations were made.
mesh	The number of mesh points in the time discretization.
theta	The parameter vector of the process. theta are taken as the starting values of the MCMC chain and gives the dimension of the parameter vector used to calculate the DIC. Care should be taken to ensure that each element in theta is in fact used within the coefficient-functions, otherwise redundant parameters will be counted in the calculation of the DIC.
sds	Proposal distribution standard deviations. That is, for the i-th parameter the proposal distribution is ~ *Normal(...,*sds*[i]^2).*
updates	The number of MCMC updates/iterations to perform (including burn-in).
burns	The number of MCMC updates/iterations to burn.
exclude	Vector indicating which transitions to exclude from the analysis. Default = NULL.
plot.chain	If = TRUE (default), a trace plot of the MCMC chain will be made along with a trace of the acceptance rate.
RK.order	The order of the Runge-Kutta solver used to approximate the trajectories of cumulants. Must be 4 (default) or 10.
Tag	Tag can be used to name (tag) an MCMC run e.g. Tag='Run_1'

Dtype	The density approximant to use. Valid types are "Saddlepoint" (default), "Edgeworth" or "Normal".
recycle	Whether or not to recycle the roots calculated for the saddlepoint approximation over succesive updates.
rtf	Starting vector for internal use.
wrt	If TRUE a .cpp file will be written to the current directory. For bug report diagnostics.
print.output	If TRUE information about the model and algorithm is printed to the console.
palette	Colour palette for drawing trace plots. Default palette = 'mono', otherwise a qualitative palette will be used.

Value

par.matrix	A matrix containing the MCMC chain on theta.
acceptence.rate	A vector containing the acceptance rate of the MCMC at every iteration.
model.info	A list of variables pertaining to inference calculations.
model.info$elapsed.time	The runtime, in h/m/s format,of the MCMC procedure (excluding compile time).
model.info$time.homogeneous	'No' if the model has time-homogeneous coefficients and 'Yes' otherwise.
model.info$p	The dimension of theta.
model.info$DIC	Calculated Deviance Information Criterion.
model.info$pd	Effective number of parameters (see model.info$DIC).

Syntactical jargon

Synt. [1]: The coefficients of the 2D GQD may be parameterized using the reserved variable theta. For example:

```
a00 <- function(t){theta[1]*(theta[2]+sin(2*pi*(t-theta[3])))}.
```

Synt. [2]: Due to syntactical differences between R and C++ special functions have to be used when terms that depend on t. When the function cannot be separated in to terms that contain a single t, the prod(a,b) function must be used. For example:

```
a00 <- function(t){0.1*(10+0.2*sin(2*pi*t)+0.3*prod(sqrt(t),1+cos(3*pi*t)))}.
```

Here sqrt(t)*cos(3*pi*t) constitutes the product of two terms that cannot be written i.t.o. a single t. To circumvent this isue, one may use the prod(a,b) function.

Synt. [3]: Similarly, the ^ - operator is not overloaded in C++. Instead the pow(x,p) function may be used to calculate x^p. For example sin(2*pi*t)^3 in:

```
a00 <- function(t){0.1*(10+0.2*pow(sin(2*pi*t),3))}.
```

Warning

Warning [1]: The parameter mesh is used to discretize the transition horizons between successive observations. It is thus important to ensure that mesh is not too small when large time differences are present in time. Check output for max(dt) and divide by mesh.

Note

Note [1]: When `plot.chain` is `TRUE`, a trace plot is created of the resulting MCMC along with the acceptance rate at each update. This may save time when scrutinizing initial MCMC runs.

Author(s)

Etienne A.D. Pienaar <etiannead@gmail.com>

References

Updates available on GitHub at https://github.com/eta21.

Daniels, H.E. 1954 Saddlepoint approximations in statistics. *Ann. Math. Stat.*, **25**:631–650.

Eddelbuettel, D. and Romain, F. 2011 Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, **40**(8):1–18,. URL http://www.jstatsoft.org/v40/i08/.

Eddelbuettel, D. 2013 *Seamless R and C++ Integration with Rcpp*. New York: Springer. ISBN 978-1-4614-6867-7.

Eddelbuettel, D. and Sanderson, C. 2014 Rcpparmadillo: Accelerating R with high-performance C++ linear algebra. *Computational Statistics and Data Analysis*, **71**:1054–1063. URL http://dx.doi.org/10.1016/j.csda.2013.02.005.

Feagin, T. 2007 A tenth-order Runge-Kutta method with error estimate. *In Proceedings of the IAENG Conf. on Scientifc Computing*.

Varughese, M.M. 2013 Parameter estimation for multivariate diffusion systems. *Comput. Stat. Data An.*, **57**:417–428.

See Also

`GQD.remove`, `BiGQD.mle`, `GQD.mcmc`, `GQD.mle`, `GQD.passage` and `GQD.TIpassage`.

Examples

```
#============================================================================
# This example simulates a bivariate time homogeneous diffusion and shows how
# to conduct inference using BiGQD.mcmc(). We fit two competing models and then
# use the output to select a winner.
#----------------------------------------------------------------------------

  data(SDEsim2)
  data(SDEsim2)
  attach(SDEsim2)
  # Have a look at the time series:
 plot(Xt~time,type='l',col='blue',ylim=c(2,10),main='Simulated Data',xlab='Time (t)',ylab='State',
      axes=FALSE)
  lines(Yt~time,col='red')
  expr1=expression(dX[t]==2(Y[t]-X[t])*dt+0.3*sqrt(X[t]*Y[t])*dW[t])
  expr2=expression(dY[t]==(5-Y[t])*dt+0.5*sqrt(Y[t])*dB[t])
  text(50,9,expr1)
  text(50,8.5,expr2)
```

```
 axis(1,seq(0,100,5))
 axis(1,seq(0,100,5/10),tcl=-0.2,labels=NA)
 axis(2,seq(0,20,2))
 axis(2,seq(0,20,2/10),tcl=-0.2,labels=NA)

#-------------------------------------------------------------------------------
# Define the coefficients of a proposed model
#-------------------------------------------------------------------------------
 GQD.remove()
 a00 <- function(t){theta[1]*theta[2]}
 a10 <- function(t){-theta[1]}
 c00 <- function(t){theta[3]*theta[3]}

 b00 <- function(t){theta[4]}
 b01 <- function(t){-theta[5]}
 f00 <- function(t){theta[6]*theta[6]}

 theta.start <- c(3,3,3,3,3,3)
 prop.sds    <- c(0.15,0.16,0.04,0.99,0.19,0.04)
 updates     <- 50000
 X           <- cbind(Xt,Yt)

 # Define prior distributions:
 priors=function(theta){dunif(theta[1],0,100)*dunif(theta[4],0,100)}

 # Run the MCMC procedure
 m1=BiGQD.mcmc(X,time,10,theta.start,prop.sds,updates)

#-------------------------------------------------------------------------------
# Remove old coefficients and define the coefficients of a new model
#-------------------------------------------------------------------------------
 GQD.remove()
 a10 <- function(t){-theta[1]}
 a01 <- function(t){theta[1]*theta[2]}
 c11 <- function(t){theta[3]*theta[3]}

 b00 <- function(t){theta[4]*theta[5]}
 b01 <- function(t){-theta[4]}
 f01 <- function(t){theta[6]*theta[6]}

 theta.start <- c(3,3,3,3,3,3)
 prop.sds    <- c(0.16,0.02,0.01,0.18,0.12,0.01)

 # Define prior distributions:
 priors=function(theta){dunif(theta[1],0,100)*dunif(theta[4],0,100)}

 # Run the MCMC procedure
 m2=BiGQD.mcmc(X,time,10,theta.start,prop.sds,updates)

# Compare estimates:
 GQD.estimates(m1)
 GQD.estimates(m2)
```

```
#-------------------------------------------------------------------------
# Compare the two models
#-------------------------------------------------------------------------

 GQD.dic(list(m1,m2))


#=========================================================================
```

BiGQD.mle *Calculate Maximum Likelihood Estimates for a 2D GQD Model.*

Description

`BiGQD.mle()` uses parametrised coefficients (provided by the user as R-functions) to construct a C++ program in real time that allows the user to perform maximum likelihood inference on the resulting diffusion model.

`BiGQD.density()` approximates the transitional density of jump diffusions of the form:

$$d\mathbf{X}_t = \boldsymbol{\mu}(\mathbf{X}_t, t)dt + \boldsymbol{\sigma}(\mathbf{X}_t, t)d\mathbf{W}_t$$

where

$$\mu^{(1)}(\mathbf{X}_t, t) = \sum_{i+j\leq 2} a_{ij}(t) X_t^i Y_t^j,$$

$$\mu^{(2)}(\mathbf{X}_t, t) = \sum_{i+j\leq 2} b_{ij}(t) X_t^i Y_t^j,$$

and

$$\boldsymbol{\sigma}(X_t, Y_t, t)\boldsymbol{\sigma}'(X_t, Y_t, t) = (\gamma^{(i,j)}(\mathbf{X}_t, t))_{i,j=1,2}$$

such that

$$\gamma^{(1,1)}(\mathbf{X}_t, t) = \sum_{i+j\leq 2} c_{ij}(t) X_t^i Y_t^j,$$

$$\gamma^{(1,2)}(\mathbf{X}_t, t) = \sum_{i+j\leq 2} d_{ij}(t) X_t^i Y_t^j,$$

$$\gamma^{(2,1)}(\mathbf{X}_t, t) = \sum_{i+j\leq 2} e_{ij}(t) X_t^i Y_t^j,$$

$$\gamma^{(2,2)}(\mathbf{X}_t, t) = \sum_{i+j\leq 2} f_{ij}(t) X_t^i Y_t^j.$$

Usage

```
BiGQD.mle(X, time, mesh=10, theta, control=NULL, method='Nelder-Mead', RK.order=4,
        exclude=NULL, Tag=NA, Dtype='Saddlepoint', rtf= runif(2,-1,1), wrt=FALSE,
          print.output=TRUE)
```

Arguments

`X`	A matrix containing rows of data points to be modelled. Though observations are allowed to be non-equidistant, observations in both dimensions are assumed to occur at the same time epochs.
`time`	A vector containing the time epochs at which observations were made.
`mesh`	The number of mesh points in the time discretisation.
`theta`	The parameter vector starting values.
`control`	List of control variables to be used by `optim`.
`method`	Method to be used by `optim`.
`exclude`	Vector indicating which transitions to exclude from the analysis. Default = `NULL`.
`RK.order`	The order of the Runge-Kutta solver used to approximate the trajectories of cumulants. Must be 4 (default) or 10.
`Tag`	`Tag` can be used to name (tag) an MCMC run e.g. `Tag='Run_1'`
`Dtype`	The density approximant to use. Valid types are `"Saddlepoint"` (default), `"Edgeworth"` or `"Normal"`.
`rtf`	Starting vector for internal use.
`wrt`	If `TRUE` a .cpp file will be written to the current directory. For bug report diagnostics.
`print.output`	If `TRUE` information about the model and algorithm is printed to the console.

Value

`opt`	The output from optim.
`model.info`	A list of variables pertaining to inference calculations.
`model.info$elapsed.time`	The runtime, in h/m/s format,of the MCMC procedure (excluding compile time).
`model.info$time.homogeneous`	'No' if the model has time-homogeneous coefficients and 'Yes' otherwise.
`model.info$p`	The dimension of `theta`.

Syntactical jargon

Synt. [1]: The coefficients of the 2D GQD may be parameterized using the reserved variable `theta`. For example:

`a00 <- function(t){theta[1]*(theta[2]+sin(2*pi*(t-theta[3])))}`.

Synt. [2]: Due to syntactical differences between R and C++ special functions have to be used when terms that depend on `t`. When the function cannot be separated in to terms that contain a single `t`, the `prod(a,b)` function must be used. For example:

`a00 <- function(t){0.1*(10+0.2*sin(2*pi*t)+0.3*prod(sqrt(t),1+cos(3*pi*t)))}`.

Here sqrt(t)*cos(3*pi*t) constitutes the product of two terms that cannot be written i.t.o. a single `t`. To circumvent this isue, one may use the `prod(a,b)` function.

Synt. [3]: Similarly, the `^` - operator is not overloaded in C++. Instead the `pow(x,p)` function may be used to calculate x^p. For example sin(2*pi*t)^3 in:

`a00 <- function(t){0.1*(10+0.2*pow(sin(2*pi*t),3))}`.

Warning

Warning [1]: The parameter `mesh` is used to discretize the transition horizons between successive observations. It is thus important to ensure that `mesh` is not too small when large time differences are present in `time`. Check output for max(dt) and divide by `mesh`.

Warning [2]: Note that minus the likelihood is minimized, as such the `optim` output (hessian) needs to be adjusted accordingly if used for calculating confidence intervals. Furthermore, `GQD.mle` may be temperamental under certain conditions

Author(s)

Etienne A.D. Pienaar: <etiennead@gmail.com>

References

Updates available on GitHub at https://github.com/eta21.

Daniels, H.E. 1954 Saddlepoint approximations in statistics. *Ann. Math. Stat.*, **25**:631–650.

Eddelbuettel, D. and Romain, F. 2011 Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, **40**(8):1–18,. URL http://www.jstatsoft.org/v40/i08/.

Eddelbuettel, D. 2013 *Seamless R and C++ Integration with Rcpp*. New York: Springer. ISBN 978-1-4614-6867-7.

Eddelbuettel, D. and Sanderson, C. 2014 Rcpparmadillo: Accelerating R with high-performance C++ linear algebra. *Computational Statistics and Data Analysis*, **71**:1054–1063. URL http://dx.doi.org/10.1016/j.csda.2013.02.005.

Feagin, T. 2007 A tenth-order Runge-Kutta method with error estimate. *In Proceedings of the IAENG Conf. on Scientifc Computing*.

Varughese, M.M. 2013 Parameter estimation for multivariate diffusion systems. *Comput. Stat. Data An.*, **57**:417–428.

See Also

`GQD.remove`, `BiGQD.mcmc`, `GQD.mcmc`, `GQD.mle`, `GQD.passage` and `GQD.TIpassage`.

Examples

```
#============================================================================
# This example simulates a bivariate time homogeneous diffusion and shows how
# to conduct inference using BiGQD.mle(). We fit two competing models and then
# use the output to select a winner.
#----------------------------------------------------------------------------

  data(SDEsim2)
  data(SDEsim2)
  attach(SDEsim2)
  # Have a look at the time series:
 plot(Xt~time,type='l',col='blue',ylim=c(2,10),main='Simulated Data',xlab='Time (t)',ylab='State',
      axes=FALSE)
  lines(Yt~time,col='red')
```

```
 expr1=expression(dX[t]==2(Y[t]-X[t])*dt+0.3*sqrt(X[t]*Y[t])*dW[t])
 expr2=expression(dY[t]==(5-Y[t])*dt+0.5*sqrt(Y[t])*dB[t])
 text(50,9,expr1)
 text(50,8.5,expr2)
 axis(1,seq(0,100,5))
 axis(1,seq(0,100,5/10),tcl=-0.2,labels=NA)
 axis(2,seq(0,20,2))
 axis(2,seq(0,20,2/10),tcl=-0.2,labels=NA)

#-------------------------------------------------------------------------------
# Define the coefficients of a proposed model
#-------------------------------------------------------------------------------
 GQD.remove()
 a00 <- function(t){theta[1]*theta[2]}
 a10 <- function(t){-theta[1]}
 c00 <- function(t){theta[3]*theta[3]}

 b00 <- function(t){theta[4]}
 b01 <- function(t){-theta[5]}
 f00 <- function(t){theta[6]*theta[6]}

 theta.start <- c(3,3,3,3,3,3)
 X           <- cbind(Xt,Yt)

 # Calculate MLEs
 m1=BiGQD.mle(X,time,10,theta.start)

#-------------------------------------------------------------------------------
# Remove old coefficients and define the coefficients of a new model
#-------------------------------------------------------------------------------
 GQD.remove()
 a10 <- function(t){-theta[1]}
 a01 <- function(t){theta[1]*theta[2]}
 c11 <- function(t){theta[3]*theta[3]}

 b00 <- function(t){theta[4]*theta[5]}
 b01 <- function(t){-theta[4]}
 f01 <- function(t){theta[6]*theta[6]}

 theta.start <- c(3,3,3,3,3,3)

 # Calculate MLEs
 m2=BiGQD.mle(X,time,10,theta.start)

# Compare estimates:
 GQD.estimates(m1)
 GQD.estimates(m2)

#-------------------------------------------------------------------------------
# Compare the two models
#-------------------------------------------------------------------------------

 GQD.aic(list(m1,m2))
```

```
#============================================================================
```

`fpt.sim.times` *Simulated First Passage Times for a Time-Inhomogeneous Non-Linear Diffusion*

Description

A dataset of 500000 simulated first passage times of a diffusion:

$$dX_t = \theta_1 X_t(10 + 0.2\sin(2\pi t) + 0.3\sqrt{t}(1 + \cos(3\pi t)) - X_t)dt + \sqrt{0.1}X_t dB_t$$

starting in X_1 = 8 traversing a fixed barrier at B = 8.

Usage

```
data("SDEsim6")
```

Format

`fpt.sim.times` 500 000 simulated first passage times.

Examples

```
data("SDEsim6")
hist(fpt.sim.times, freq = FALSE, col = 'gray85', border = 'white',
  main = 'First Passage Time Density', ylab = 'Density', xlab = 'Time',
  breaks = 100)
```

`GQD.aic` *Summarize MLE Selection Output for a List of GQD.mle or BiGQD.mle objects.*

Description

`GQD.aic()` summarizes the MCMC output from a list of `GQD.mle()` objects. This may be used to neatly summarize the MCMC output of various models fitted to a given dataset.

Usage

```
GQD.aic(model.list, type = "col")
```

Arguments

model.list	A list of GQD.mle() objects.
type	Shoould output be of row ('row') or column ('col') format.

Details

GQD.aic() summarizes the output from various models fitted via GQD.mle(). By ranking them according to DIC. [=] indicates which model has the minimal DIC.

	Convergence	p	min.likelihood	AIC	BIC	N
Model 1	0	5	171.5576	[=] 353.1152	[=] 369.6317	201
Model 2	0	5	185.7518	381.5036	398.0201	201

Value

A data frame with summary of model output. See Details.

Author(s)

Etienne A.D. Pienaar: <etiannead@gmail.com>

References

Updates available on GitHub at https://github.com/eta21.

See Also

GQD.mle

Examples

```
#============================================================================
# Simulate a time inhomogeneous diffusion.
#----------------------------------------------------------------------------

  data(SDEsim1)
  attach(SDEsim1)
  par(mfrow=c(1,1))
  expr1=expression(dX[t]==2*(5+3*sin(0.5*pi*t)-X[t])*dt+0.5*sqrt(X[t])*dW[t])
  plot(Xt~time,type='l',col='blue',xlab='Time (t)',ylab=expression(X[t]),main=expr1)

 #----------------------------------------------------------------------------
 # Define coefficients of the process.
 #----------------------------------------------------------------------------

  GQD.remove()
  G0 <- function(t){theta[1]*(theta[2]+theta[3]*sin(0.25*pi*t))}
  G1 <- function(t){-theta[1]}
```

```
Q0 <- function(t){theta[4]*theta[4]}

theta.start  <- c(1,1,1,1)                    # Starting values for the chain
mesh.points  <- 10                            # Number of mesh points

m1 <- GQD.mle(Xt,time,mesh=mesh.points,theta=theta.start)

GQD.remove()

G1 <- function(t){theta[1]*(theta[2]+theta[3]*sin(0.25*pi*t))}
G2 <- function(t){-theta[1]}
Q2 <- function(t){theta[4]*theta[4]}

theta.start  <- c(1,1,1,1)                    # Starting values for the chain
mesh.points  <- 10                            # Number of mesh points

m2 <- GQD.mle(Xt,time,mesh=mesh.points,theta=theta.start)

# Check estimates:
GQD.estimates(m1)
GQD.estimates(m2)

# Compare models:
GQD.aic(list(m1,m2))

#==============================================================================
```

`GQD.density` *Generate the Transition Density of a Scalar Generalized Quadratic Diffusion (GQD).*

Description

`GQD.density` approximates the transition density of a scalar generalized quadratic diffusion model (GQD). Given an initial value for the diffusion, `Xs`, the approximation is evaluated for all `Xt` at equispaced time-nodes given by splitting [`s`, `t`] into subintervals of length `delt`.

`GQD.density()` approximates transitional densities of jump diffusions of the form:

$$dX_t = \mu(X,t)dt + \sigma(X_t,t)dW_t + dP_t$$

where

$$\mu(X,t) = G_0(t,\theta) + G_1(t,\theta)X_t + G_2(t,\theta)X_t^2$$

and

$$\sigma(X,t) = \sqrt{Q_0(t,\theta) + Q_1(t,\theta)X_t + Q_2(t,\theta)X_t^2}.$$

Usage

```
GQD.density(Xs, Xt, s, t, delt=1/100, Dtype='Saddlepoint', Trunc=c(4,4),
            P=100, alpha=0, lower=0, upper=50, print.output=TRUE,
            eval.density=TRUE)
```

Arguments

Xs	Initial value of the process at time s.
Xt	Vector of values at which the transition density is to be evaluated over the trajectory of the transition density from time s to t.
s	The starting time of the process.
t	The time horizon up to and including which the transitional density is evaluated.
delt	Size of the time increments at which successive evaluations are made.
Dtype	Character string indicating the type of density approximation (see details) to use. Types: 'Saddlepoint', 'Normal', 'Gamma', 'InvGamma' and 'Beta' are supported (default = 'Saddlepoint').
Trunc	Vector of length 2 containing the cumulant truncation order and the density truncation order respectively. May take on values 4, 6 and 8 with the constraint that Trunc[1] >= Trunc[2]. Default is c(4,4).
P	Normalization parameter indicating the number of points to use when normalizing members of the Pearson system (see details)
alpha	Normalization parameter controlig the mesh concentration when normalizing members of the Pearson system (see details). Increasing alpha decreases concentration around the mean and vice versa (default alpha = 0).
lower,upper	Lower and upper bounds for the normalization range.
print.output	If TRUE information about the model and algorithm is printed to the console.
eval.density	If TRUE, the density is evaluated in addition to calculating the moment eqns.

Details

GQD.density constructs an approximate transition density for a class of quadratic diffusion models. This is done by first evaluating the trajectory of the cumulants/moments of the diffusion numerically as the solution of a system of ordinary differential equations over a time horizon [s,t] split into equi-distant points delt units apart. Subsequently, the resulting cumulants/moments are carried into a density approximant (by default, a saddlepoint approximation) in order to evaluate the transtion surface.

Value

density	A matrix giving the density over the spatio-temporal mesh whose vertices are defined by paired permutations of the elements of X_t and time
Xt	A vector of points defining the state space at which the density was evaluated(recycled from input).
time	A vector of time points at which the density was evaluated.

cumulants	A matrix giving the cumulants of the diffusion. Row i gives the i-th cumulant.
moments	A matrix giving the moments of the diffusion. Row i gives the i-th cumulant.
mesh	A matrix giving the mesh used for normalization of the density.

Warning

Warning [1]: The system of ODEs that dictate the evolution of the cumulants do so approximately. Thus, although it is unlikely such cases will be encountered in inferential contexts, it is worth checking (by simulation) whether cumulants accurately replicate those of the target GQD. Furthermore, it may in some cases occur that the cumulants are indeed accurate whilst the density approximation fails. This can again be verified by simulation after which alternate density approximants may be specified through the variable `Dtype`.

Warning [2]: The parameter `delt` is also used as the stepsize for solving a system of ordinary differential equations (ODEs) that govern the evolution of the cumulants of the diffusion. As such `delt` is required to be small for highly non-linear models in order to ensure sufficient accuracy.

Author(s)

Etienne A.D. Pienaar: <etiannead@gmail.com>

References

Updates available on GitHub at https://github.com/eta21.

Daniels, H.E. 1954 Saddlepoint approximations in statistics. *Ann. Math. Stat.*, **25**:631–650.

Feagin, T. 2007 A tenth-order Runge-Kutta method with error estimate. *In Proceedings of the IAENG Conf. on Scientifc Computing*.

Varughese, M.M. 2013 Parameter estimation for multivariate diffusion systems. *Comput. Stat. Data An.*, **57**:417–428.

See Also

See `GQD.mcmc` and `GQD.mle` for likelihood based inference procedures for GQDs.

Examples

```
#==============================================================================
# Generate the transition density of a CIR process with time dependant
# drift and volatility.
#------------------------------------------------------------------------------

  # Remove any existing coefficients
  GQD.remove()

  # Define drift Coefficients. Note that the limiting mean is sinusoidal.
  G0 <- function(t){2*(10+sin(2*pi*(t-0.5)))}
  G1 <- function(t){-2}

  # Define sinusoidal diffusion coefficient with `faster' oscillation.
```

```
  Q1 <- function(t){0.25*(1+0.75*(sin(4*pi*t)))}

  states     <-  seq(5,15,1/10) # State values
  initial    <-  8              # Starting value of the process
  Tmax       <-  5              # Time horizon
  Tstart     <-  1              # Time starts at 1
  increment  <-  1/100          # Incremental time steps

  # Generate the transitional density
  M <- GQD.density(Xs=initial,Xt=states,s=Tstart,t=Tmax,delt=increment)

  if(require(rgl, quietly = TRUE))
  {
    open3d(windowRect=c(50,50,640+50,50+640),zoom=0.95)
    persp3d(x=M$Xt,y=M$time,z=M$density,col=3,box=FALSE,xlab='State (X_t)',
            ylab='Time (t)',zlab='Density f(X_t|X_s)')
    play3d(spin3d(axis=c(0,0,1), rpm=3), duration=10)
  }else
  {
    persp(x=M$Xt,y=M$time,z=M$density,col=3,xlab='State (X_t)',ylab='Time (t)',
          zlab='Density f(X_t|X_s)',border=NA,shade=0.5,theta=145)
  }

#==============================================================================
# Generate the transition density for a diffusion process with restricted domain.
# The diffusion has reflective boundaries at 0 and 1.
#------------------------------------------------------------------------------

  GQD.remove()

  G0 <- function(t){0.4*(0.5+sin(2*pi*t))}
  G1 <- function(t){-0.4}
  Q1 <- function(t){0.25}
  Q2 <- function(t){-0.25}

  states     <-  seq(0.005,0.995,1/200)
  initial    <-  0.5
  Tmax       <-  5
  increment  <-  1/50

  # Generate the transitional density
  M <- GQD.density(Xs=initial,Xt=states,s=0,t=Tmax,delt=increment,
                   Dtype='Beta',Trunc=c(8,8))

  if(require(rgl, quietly = TRUE))
  {
    open3d(windowRect=c(50,50,640+50,50+640),zoom=0.95)
    persp3d(x=M$Xt,y=M$time,z=M$density,col='steelblue',box=FALSE,
            xlab='State (X_t)',ylab='Time (t)',zlab='Density f(X_t|X_s)')
    play3d(spin3d(axis=c(0,0,1), rpm=3), duration=10)
  }else
  {
    persp(x=M$Xt,y=M$time,z=M$density,col=3,xlab='State (X_t)',ylab='Time (t)',
```

```
        zlab='Density f(X_t|X_s)',border=NA,shade=0.5,theta=145)
  }
#==============================================================================
```

`GQD.dic` *Summarize MCMC Selection Output for a List of GQD.mcmc or BiGQD.mcmc objects.*

Description

`GQD.dic()` summarizes the MCMC output from a list of `GQD.mcmc()` objects. This may be used to neatly summarize the MCMC output of various models fitted to a given dataset.

Usage

```
GQD.dic(model.list, type = "col")
```

Arguments

`model.list`	A list of `GQD.mcmc()` objects.
`type`	Shoould output be of row (`'row'`) or column (`'col'`) format.

Details

`GQD.dic()` summarizes the output from various models fitted via `GQD.mcmc()`. By ranking them according to DIC. `[=]` indicates which model has the minimal DIC.

	Elapsed_Time	Time_Homogeneous	p	DIC	pD	N
Model 1	00:00:47	No	5.00	389.30	3.92	201
Model 2	00:01:00	No	5.00	[=]386.45	4.13	201
Model 3	00:02:50	No	5.00	391.37	3.94	201

Value

A data frame with summary of model output. See Details.

Author(s)

Etienne A.D. Pienaar: <etiannead@gmail.com>

References

Updates available on GitHub at https://github.com/eta21.

See Also

GQD.mcmc

Examples

```
#============================================================================
# Simulate a time inhomogeneous diffusion.
#----------------------------------------------------------------------------

 data(SDEsim1)
 attach(SDEsim1)
 par(mfrow=c(1,1))
 expr1=expression(dX[t]==2*(5+3*sin(0.5*pi*t)-X[t])*dt+0.5*sqrt(X[t])*dW[t])
 plot(Xt~time,type='l',col='blue',xlab='Time (t)',ylab=expression(X[t]),main=expr1)


 #----------------------------------------------------------------------------
 # Define coefficients of model 1
 #----------------------------------------------------------------------------

  # Remove any existing coeffients. If none are pressent NAs will be returned, but
  # this is a safeguard against overlapping.
    GQD.remove()

  # Define time dependant coefficients. Note that all functions have a single argument.
  # This argument has to be `t' in order for the dependancy to be recognized.
  # theta does not have to be defined as an argument.

  G0 <- function(t){theta[1]*(theta[2]+theta[3]*sin(0.25*pi*t))}
  G1 <- function(t){-theta[1]}
  Q0 <- function(t){theta[4]*theta[4]}

  theta.start   <-  c(1,1,1,1)                    # Starting values for the chain
  proposal.sds  <-  c(0.8,0.1,0.1,0.1)            # Std devs for proposal distributions
  mesh.points   <-  10                            # Number of mesh points
  updates       <-  50000                         # Perform 50000 updates

  priors=function(theta){dnorm(theta[1],5,5)}
  m1 <- GQD.mcmc(Xt,time,mesh=mesh.points,theta=theta.start,sds=proposal.sds,
                 updates=updates)


#----------------------------------------------------------------------------
# Define coefficients of model 2
#----------------------------------------------------------------------------

 GQD.remove()
 G0 <- function(t){theta[1]*(theta[2]+theta[3]*sin(0.25*pi*t))}
 G1 <- function(t){-theta[1]}
 Q1 <- function(t){theta[4]*theta[4]}
```

```
 proposal.sds  <-  c(0.8,0.1,0.1,0.1)
 m2 <- GQD.mcmc(Xt,time,mesh=mesh.points,theta=theta.start,sds=proposal.sds,
               updates=updates)

 # Checkthe parameter estimates:
 GQD.estimates(m2,thin = 200)
#-------------------------------------------------------------------------------
# Summarize the output from the models.
#-------------------------------------------------------------------------------

GQD.dic(list(m1,m2))

#===============================================================================
```

`GQD.estimates` *Extract Parmaeter Estimates from* `.mle()` *or* `.mcmc()` *Objects.*

Description

`GQD.estimates()` calculates parameter estimates from `.mle()` or `.mcmc()` model objects.

Usage

```
GQD.estimates(x, thin = 100, burns, CI = c(0.05, 0.95), corrmat =
         FALSE, acf.plot = TRUE, palette = 'mono')
```

Arguments

`x`	List object of type 'GQD.mcmc' or 'GQD.mle'. That is, when `model =GQD.mcmc()` then `model` constitutes an appropriate object for `x`.
`thin`	Thinnging level for parameter chain.
`burns`	Number of MCMC updates to discard before calculating estimates.
`CI`	Credibility interval quantiles (for MCMC chains).
`corrmat`	If TRUE, an estimated correlation matrix is returned in addition to the estimate vector.
`acf.plot`	If TRUE, an acf plot is drawn for each element of the parameter chain.
`palette`	Colour palette for drawing trace plots. Default `palette = 'mono'`, otherwise a qualitative palette will be used.

Value

Data frame with parameter estimates and appropriate interval statistics.

Author(s)

Etienne A.D. Pienaar: `<etiannead@gmail.com>`

References

Updates available on GitHub at https://github.com/eta21.

See Also

GQD.mcmc, GQD.mle, BiGQD.mcmc and BiGQD.mle.

Examples

```
#==============================================================================
# This example simulates a time inhomogeneous diffusion and shows how to conduct
# inference using GQD.mcmc
#------------------------------------------------------------------------------
library(DiffusionRgqd)
data(SDEsim1)
par(mfrow=c(1,1))
x <- SDEsim1
plot(x$Xt~x$time,type='l',col='blue')
#------------------------------------------------------------------------------
# Define parameterized coefficients of the process, and set up starting
# parameters.
# True model: dX_t = 2(5+3sin(0.25 pi t)-X_t)dt+0.5sqrt(X_t)dW_t
#------------------------------------------------------------------------------

# Remove any existing coeffients. If none are pressent NAs will be returned, but
# this is a safeguard against overlapping.
GQD.remove()

# Define time dependant coefficients. Note that all functions have a single argument.
# This argument has to be `t' in order for the dependancy to be recognized.
# theta does not have to be defined as an argument.

G0 <- function(t){theta[1]*(theta[2]+theta[3]*sin(0.25*pi*t))}
G1 <- function(t){-theta[1]}
Q1 <- function(t){theta[4]*theta[4]}

theta.start  <- c(1,1,1,1)                  # Starting values for the chain
proposal.sds <- c(0.4,0.3,0.2,0.1)*1/2      # Std devs for proposal distributions
mesh.points  <- 10                          # Number of mesh points
updates      <- 50000                       # Perform 50000 updates

#------------------------------------------------------------------------------
# Run the MCMC procedure for the model defined above
#------------------------------------------------------------------------------

m1 <- GQD.mcmc(x$Xt,x$time,mesh=mesh.points,theta=theta.start,sds=proposal.sds,
               updates=updates)

# Calculate estimates:
GQD.estimates(m1,thin=200)
#==============================================================================
```

GQD.mcmc *MCMC Inference on Generalized Quadratic Diffusion Models (GQDs).*

Description

GQD.mcmc() uses parametrised coefficients (provided by the user as R-functions) to construct a C++ program in real time that allows the user to perform Bayesian inference on the resulting jump diffusion model. Given a set of starting parameters, a MCMC chain is returned for further analysis.

GQD.mcmc() performs inference using the Metropolis-Hastings algorithm for diffusions of the form:

$$dX_t = \mu(X,t)dt + \sigma(X_t,t)dW_t$$

where

$$\mu(X,t) = G_0(t,\theta) + G_1(t,\theta)X_t + G_2(t,\theta)X_t^2$$

and

$$\sigma(X,t) = \sqrt{Q_0(t,\theta) + Q_1(t,\theta)X_t + Q_2(t,\theta)X_t^2}.$$

Usage

```
GQD.mcmc(X, time, mesh=10, theta, sds, updates, burns=min(round(updates/2),25000),
         Dtype='Saddle', Trunc=c(4,4), RK.order=4, P=200, alpha=0,
         lower=min(na.omit(X))/2, upper=max(na.omit(X))*2,
         exclude=NULL, plot.chain=TRUE, Tag=NA, wrt=FALSE, print.output=TRUE,
         palette = 'mono')
```

Arguments

X	Time series (vector) of discretely observed points of the process of interest. These may be non-equidistant observations (see time).
time	A vector of time-stamps associated with each observation in X.
mesh	The number mesh points between any two given data points.
theta	The parameter vector of the process. theta are taken as the starting values of the MCMC chain and gives the dimension of the parameter vector used to calculate the DIC. Care should be taken to ensure that each element in theta is in fact used within the coefficient-functions, otherwise redundant parameters will be counted in the calculation of the DIC.
sds	Proposal distribution standard deviations. That is, for the i-th parameter the proposal distribution is ~ *Normal(...,*sds*[i]^2)*
updates	The number of chain updates (including burned updates) to perform.
burns	The number of updates to burn. That is, the first burns values are omitted from the inference, although the entire chain is returned.

exclude	Vector indicating which transitions to exclude from the analysis. Default = NULL.
plot.chain	If TRUE (default), a trace plot is made of the resulting MCMC chain (see details).
RK.order	The order of the Runge-Kutta solver used to approximate the trajectories of cumulants. Must be 4 or (default) 10.
Dtype	Character string indicating the type of density approximation (see details) to use. Types: 'Saddlepoint', 'Normal', 'Gamma', 'InvGamma' and 'Beta' are supported (default = 'Saddlepoint').
Trunc	Vector of length 2 containing the cumulant truncation order and the density truncation order respectively. May take on values 4, 6 and 8 with the constraint that Trunc[1] >= Trunc[2]. Default is c(4,4).
P	Normalization parameter indicating the number of points to use when normalizing members of the Pearson system (see details)
alpha	Normalization parameter controlig the mesh concentration when normalizing members of the Pearson system (see details). Increasing alpha decreases concentration around the mean and vice versa (default alpha = 0).
lower,upper	Lower and upper bounds for the normalization range.
Tag	Tag can be used to name (tag) an MCMC run e.g. Tag='Run_1'
wrt	If TRUE a .cpp file will be written to the current directory. For bug report diagnostics.
print.output	If TRUE information about the model and algorithm is printed to the console.
palette	Colour palette for drawing trace plots. Default palette = 'mono', otherwise a qualitative palette will be used.

Details

GQD.mcmc() operates by searching the workspace for functions with names that match the coefficients of the predefined stochastic differential equation. Only the required coefficients need to be specified e.g. G0(t),G1(t) and Q0(t) for an Ornstein-Uhlenbeck model. Unspecified coefficients are ignored. When a new model is to be defined, the current model may be removed from the workspace by using the GQD.remove function, after which the new coefficients may be supplied.

Value

par.matrix	A matrix containing the MCMC chain on theta.
acceptence.rate	A vector containing the acceptance rate of the MCMC at every iteration.
model.info	A list of variables pertaining to inference calculations.
model.info$elapsed.time	The runtime, in h/m/s format,of the MCMC procedure (excluding compile time).
model.info$time.homogeneous	'No' if the model has time-homogeneous coefficients and 'Yes' otherwise.
model.info$p	The dimension of theta.
model.info$DIC	Calculated Deviance Information Criterion.
model.info$pd	Effective number of parameters (see model.info$DIC).

Syntactical jargon

Synt. [1]: The coefficients of the GQD may be parameterized using the reserved variable `theta`. For example:

`G0 <- function(t){theta[1]*(theta[2]+sin(2*pi*(t-theta[3])))}`.

Synt. [2]: Due to syntactical differences between R and C++ special functions have to be used when terms that depend on `t`. When the function cannot be separated in to terms that contain a single `t`, the `prod(a,b)` function must be used. For example:

`G0 <- function(t){0.1*(10+0.2*sin(2*pi*t)+0.3*prod(sqrt(t),1+cos(3*pi*t)))}`.

Here sqrt(t)*cos(3*pi*t) constitutes the product of two terms that cannot be written i.t.o. a single `t`. To circumvent this isue, one may use the `prod(a,b)` function.

Synt. [3]: Similarly, the ^ - operator is not overloaded in C++. Instead the `pow(x,p)` function may be used to calculate x^p. For example sin(2*pi*t)^3 in:

`G0 <- function(t){0.1*(10+0.2*pow(sin(2*pi*t),3))}`.

Note

Note [1]: When `plot.chain` is `TRUE`, a trace plot is created of the resulting MCMC along with the acceptance rate at each update. This may save time when scrutinizing initial MCMC runs.

Author(s)

Etienne A.D. Pienaar: `<etiennead@gmail.com>`

References

Updates available on GitHub at https://github.com/eta21.

Daniels, H.E. 1954 Saddlepoint approximations in statistics. *Ann. Math. Stat.*, **25**:631–650.

Eddelbuettel, D. and Romain, F. 2011 Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, **40**(8):1–18,. URL http://www.jstatsoft.org/v40/i08/.

Eddelbuettel, D. 2013 *Seamless R and C++ Integration with Rcpp*. New York: Springer. ISBN 978-1-4614-6867-7.

Eddelbuettel, D. and Sanderson, C. 2014 Rcpparmadillo: Accelerating R with high-performance C++ linear algebra. *Computational Statistics and Data Analysis*, **71**:1054–1063. URL http://dx.doi.org/10.1016/j.csda.2013.02.005.

Feagin, T. 2007 A tenth-order Runge-Kutta method with error estimate. *In Proceedings of the IAENG Conf. on Scientifc Computing*.

Varughese, M.M. 2013 Parameter estimation for multivariate diffusion systems. *Comput. Stat. Data An.*, **57**:417–428.

See Also

`GQD.remove`, `GQD.mle`, `BiGQD.mcmc`, `BiGQD.mle`, `GQD.passage` and `GQD.TIpassage`.

Examples

```
#============================================================================
# This example simulates a time inhomogeneous diffusion and shows how to conduct
# inference using GQD.mcmc
#----------------------------------------------------------------------------
 data(SDEsim1)
 attach(SDEsim1)
 par(mfrow=c(1,1))
 expr1=expression(dX[t]==2*(5+3*sin(0.5*pi*t)-X[t])*dt+0.5*sqrt(X[t])*dW[t])
 plot(Xt~time,type='l',col='blue',xlab='Time (t)',ylab=expression(X[t]),main=expr1)
#----------------------------------------------------------------------------
# Define parameterized coefficients of the process, and set up starting
# parameters.
# True model: dX_t = 2X_t(5+3sin(0.25 pi t)-X_t)dt+0.5X_tdW_t
#----------------------------------------------------------------------------

# Remove any existing coeffients. If none are pressent NAs will be returned, but
# this is a safeguard against overlapping.
GQD.remove()

# Define time dependant coefficients. Note that all functions have a single argument.
# This argument has to be `t' in order for the dependancy to be recognized.
# theta does not have to be defined as an argument.

G0 <- function(t){theta[1]*(theta[2]+theta[3]*sin(0.25*pi*t))}
G1 <- function(t){-theta[1]}
Q1 <- function(t){theta[4]*theta[4]}

theta.start  <- c(1,1,1,1)                    # Starting values for the chain
proposal.sds <- c(0.4,0.3,0.2,0.1)*1/2        # Std devs for proposal distributions
mesh.points  <- 10                            # Number of mesh points
updates      <- 50000                         # Perform 50000 updates

#----------------------------------------------------------------------------
# Run the MCMC procedure for the model defined above
#----------------------------------------------------------------------------

m1 <- GQD.mcmc(Xt,time,mesh=mesh.points,theta=theta.start,sds=proposal.sds,
               updates=updates)

# Calculate estimates:
GQD.estimates(m1,thin=200)
#============================================================================
```

GQD.mle *MLEs for Generalized Quadratic Diffusion Models (GQDs).*

Description

`GQD.mle()` constructs a C++ program in real time that allows the user to perform maximum likelihood inference on scalar GQDs. Given a set of starting parameters, the maximum likelihood estimates are calculated via minimization of minus the likelihood function via the built-in R-function `optim`.

`GQD.mcmc()` performs inference using the Metropolis-Hastings algorithm for diffusions of the form:

$$dX_t = \mu(X,t)dt + \sigma(X_t,t)dW_t$$

where

$$\mu(X,t) = G_0(t,\theta) + G_1(t,\theta)X_t + G_2(t,\theta)X_t^2$$

and

$$\sigma(X,t) = \sqrt{Q_0(t,\theta) + Q_1(t,\theta)X_t + Q_2(t,\theta)X_t^2}.$$

Usage

```
GQD.mle(X, time, mesh=10, theta, control=NULL, method='Nelder-Mead', Dtype='Saddle',
       Trunc=c(4,4), RK.order=4, P=200, alpha=0, lower=min(X)/2, upper=max(X)*2,
        exclude=NULL, Tag=NA, wrt=FALSE, print.output=TRUE)
```

Arguments

`X`	Time series (vector) of discretely observed points of the process of interest. These may be non-equidistant observations (see `time`).
`time`	A vector of time-stamps associated with each observation in `X`.
`mesh`	The number mesh points between any two given data points.
`theta`	The set of starting parameters for the optimization routine.
`control`	List of control variables to be used by `optim`.
`method`	Method to be used by `optim`.
`exclude`	Vector indicating which transitions to exclude from the analysis. Default = `NULL`.
`RK.order`	The order of the Runge-Kutta scheme used. Value must be 4 or (default) 10.
`Dtype`	Character string indicating the type of density approximation (see details) to use. Types: `'Saddlepoint'`, `'Normal'`, `'Gamma'`, `'InvGamma'` and `'Beta'` are supported (default = `'Saddlepoint'`).
`Trunc`	Vector of length 2 containing the cumulant truncation order and the density truncation order respectively. May take on values 4, 6 and 8 with the constraint that `Trunc[1] >= Trunc[2]`. Default is `c(4,4)`.
`P`	Normalization parameter indicating the number of points to use when normalizing members of the Pearson system (see details)
`alpha`	Normalization parameter controlig the mesh concentration when normalizing members of the Pearson system (see details). Increasing `alpha` decreases concentration around the mean and vice versa (default `alpha = 0`).
`lower,upper`	Lower and upper bounds for the normalization range.
`Tag`	`Tag` can be used to name (tag) an MCMC run e.g. `Tag='Run_1'`

wrt	If TRUE a .cpp file will be written to the current directory. For bug report diagnostics.
print.output	If TRUE information about the model and algorithm is printed to the console.

Value

opt	The output from optim.
model.info	A list of variables pertaining to inference calculations.
model.info$elapsed.time	The run-time, in h/m/s format,of the MCMC procedure (excluding compile time).
model.info$time.homogeneous	'No' if the model has time-homogeneous coefficients and 'Yes' otherwise.
model.info$p	The dimension of theta.

Syntactical jargon

Synt. [1]: The coefficients of the GQD may be parameterized using the reserved variable theta. For example:

`G0 <- function(t){theta[1]*(theta[2]+sin(2*pi*(t-theta[3])))}`.

Synt. [2]: Due to syntactical differences between R and C++ special functions have to be used when terms that depend on t. When the function cannot be separated in to terms that contain a single t, the prod(a,b) function must be used. For example:

`G0 <- function(t){0.1*(10+0.2*sin(2*pi*t)+0.3*prod(sqrt(t),1+cos(3*pi*t)))}`.

Here sqrt(t)*cos(3*pi*t) constitutes the product of two terms that cannot be written i.t.o. a single t. To circumvent this isue, one may use the prod(a,b) function.

Synt. [3]: Similarly, the ^ - operator is not overloaded in C++. Instead the pow(x,p) function may be used to calculate x^p. For example sin(2*pi*t)^3 in:

`G0 <- function(t){0.1*(10+0.2*pow(sin(2*pi*t),3))}`.

Warning

Warning [1]: The parameter mesh is used to discretize the transition horizons between successive observations. It is thus important to ensure that mesh is not too small when large time differences are present in time. Check output for max(dt) and divide by mesh.

Warning [2]: Note that minus the likelihood is minimized, as such the optim output (hessian) needs to be adjusted accordingly if used for calculating confidence intervals. Furthermore, GQD.mle may be temperamental under certain conditions

Author(s)

Etienne A.D. Pienaar: <etiennead@gmail.com>

References

Updates available on GitHub at https://github.com/eta21.

Daniels, H.E. 1954 Saddlepoint approximations in statistics. *Ann. Math. Stat.*, **25**:631–650.

Eddelbuettel, D. and Romain, F. 2011 Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, **40**(8):1–18,. URL http://www.jstatsoft.org/v40/i08/.

Eddelbuettel, D. 2013 *Seamless R and C++ Integration with Rcpp*. New York: Springer. ISBN 978-1-4614-6867-7.

Eddelbuettel, D. and Sanderson, C. 2014 Rcpparmadillo: Accelerating R with high-performance C++ linear algebra. *Computational Statistics and Data Analysis*, **71**:1054–1063. URL http://dx.doi.org/10.1016/j.csda.2013.02.005.

Feagin, T. 2007 A tenth-order Runge-Kutta method with error estimate. *In Proceedings of the IAENG Conf. on Scientifc Computing*.

Varughese, M.M. 2013 Parameter estimation for multivariate diffusion systems. *Comput. Stat. Data An.*, **57**:417–428.

See Also

GQD.remove, GQD.mcmc, BiGQD.mcmc, BiGQD.mle, GQD.passage and GQD.TIpassage.

Examples

```
#============================================================================
# Simulate a time inhomogeneous diffusion.
#----------------------------------------------------------------------------

  data(SDEsim1)
  attach(SDEsim1)
  par(mfrow=c(1,1))
  expr1=expression(dX[t]==2*(5+3*sin(0.5*pi*t)-X[t])*dt+0.5*sqrt(X[t])*dW[t])
  plot(Xt~time,type='l',col='blue',xlab='Time (t)',ylab=expression(X[t]),main=expr1)

 #----------------------------------------------------------------------------
 # Define coefficients of the process.
 #----------------------------------------------------------------------------

  GQD.remove()
  G0 <- function(t){theta[1]*(theta[2]+theta[3]*sin(0.25*pi*t))}
  G1 <- function(t){-theta[1]}
  Q0 <- function(t){theta[4]*theta[4]}

  theta.start  <- c(1,1,1,1)                    # Starting values for the chain
  mesh.points  <- 10                            # Number of mesh points

  m1 <- GQD.mle(Xt,time,mesh=mesh.points,theta=theta.start)

  GQD.remove()

  G0 <- function(t){theta[1]*(theta[2]+theta[3]*sin(0.25*pi*t))}
```

```
G1 <- function(t){-theta[1]}
Q1 <- function(t){theta[4]*theta[4]}

theta.start  <- c(1,1,1,1)                    # Starting values for the chain
mesh.points  <- 10                            # Number of mesh points

m2 <- GQD.mle(Xt,time,mesh=mesh.points,theta=theta.start)

# Check estimates:
GQD.estimates(m1)
GQD.estimates(m2)

# Compare models:
GQD.aic(list(m1,m2))

#==============================================================================
```

GQD.passage *Calculate the First Passage Time Density of a Time-Homogeneous GQD Process to a Fixed Barrier.*

Description

GQD.passage uses the cumulant truncation procedure of Varughese (2013) in conjunction with a Volterra-type integral equation developed by Buonocore et al. (1987) in order to approximate the first passage time density of a time-homogeneous univariate GQD

$$dX_t = (theta[1] + theta[2]X_t + theta[3]X_t^2)dt + \sqrt{theta[4] + theta[5]X_t + theta[6]X_t^2}dW_t,$$

to a fixed barrier.

Usage

```
GQD.passage(Xs, B, theta, t, delt)
```

Arguments

Xs	Initial value of the process.
B	Fixed barrier (>Xs).
theta	Parameter vector giving the coefficients of the time-homogeneous GQD.
t	The time horizon up to and including which the density is to be evaluated.
delt	Stepsize for the solution of the first passage time density.

Value

density	The approximate first passage time density.
time	The time points at which the approximation is evaluated.
prob.coverage	The approximate cumulative probability coverage.

Warning

Warning [1]: Some instability may occur when `delt` is large or where the saddlepoint approximation fails. As allways it is important to check both the validity of the diffusion process under the given parameters and the quality of the sadlepoint approximation. For a given set of parameters the latter can be checked using `GQD.density`.

Warning [2]: The first passage time problem is considered from one side only i.e. Xs<B. For Xs>B one may equivalently consider Yt=-X_t, apply Ito's lemma and proceed as above.

Note

Note [1]: Since time-homogeneity is assumed for the present implementation, the interface of `GQD.mcmc` etc. is discarded and the model is inferred from the non-zero values of `theta`. For example `theta = c(0.5*10,-0.5,0,0.5^2,0,0)` defines an Ornstein-Uhlenbeck model.

Author(s)

Etienne A.D. Pienaar: <etiannead@gmail.com>

References

Updates available on GitHub at https://github.com/eta21.

A. Buonocore, A. Nobile, and L. Ricciardi. 1987 A new integral equation for the evaluation of first-passage- time probability densities. *Adv. Appl. Probab.* **19**:784–800.

Daniels, H.E. 1954 Saddlepoint approximations in statistics. *Ann. Math. Stat.*, **25**:631–650.

Eddelbuettel, D. and Romain, F. 2011 Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, **40**(8):1–18,. URL http://www.jstatsoft.org/v40/i08/.

Eddelbuettel, D. 2013 *Seamless R and C++ Integration with Rcpp*. New York: Springer. ISBN 978-1-4614-6867-7.

Eddelbuettel, D. and Sanderson, C. 2014 Rcpparmadillo: Accelerating r with high-performance C++ linear algebra. *Computational Statistics and Data Analysis*, **71**:1054–1063. URL http://dx.doi.org/10.1016/j.csda.2013.02.005.

Feagin, T. 2007 A tenth-order Runge-Kutta method with error estimate. *In Proceedings of the IAENG Conf. on Scientific Computing*.

R. G. Jaimez, P. R. Roman and F. T. Ruiz. 1995 A note on the volterra integral equation for the first-passage-time probability density. *Journal of applied probability*, 635–648.

Varughese, M.M. 2013 Parameter estimation for multivariate diffusion systems. *Comput. Stat. Data An.*, **57**:417–428.

See Also

`GQD.density` for functions that generate the transitional density of GQDs. `GQD.mcmc` and `GQD.remove` for MCMC procedures to perform inference on GQDs.

Examples

```
#============================================================================
# Calculate the first passage time from state X_0 = 7 to X_T =10 for
# various diffusions, with T the first passage time.
#============================================================================

  res1 <- GQD.passage(7,10,c(0.1*10,-0.1,0,0.2,0,0),25,1/100)
  res2 <- GQD.passage(7,10,c(0.1*10,-0.1,0,0,0.2,0),25,1/100)
  res3 <- GQD.passage(7,10,c(0,0.1*10,-0.1,0.5^2,0,0),25,1/100)
  res4 <- GQD.passage(7,10,c(0,0.1*10,-0.1,0,0,0.1^2),25,1/100)

  expr1 <- expression(dX[t]==(1-0.1*X[t])*dt+sqrt(0.2)*dW[t])
  expr2 <- expression(dX[t]==(1-0.1*X[t])*dt+sqrt(0.2*X[t])*dW[t])
  expr3 <- expression(dX[t]==(1*X[t]-0.1*X[t]^2)*dt+0.5*dW[t])
  expr4 <- expression(dX[t]==(1*X[t]-0.1*X[t]^2)*dt+0.1*X[t]*dW[t])


#============================================================================
# Plot the resulting first passage time densities.
#============================================================================

  par(mfrow=c(2,2))
 plot(res1$density~res1$time,type='l',main=expr1,xlab='Time (t)',ylab='density',col='blue')
 plot(res2$density~res2$time,type='l',main=expr2,xlab='Time (t)',ylab='density',col='blue')
 plot(res3$density~res3$time,type='l',main=expr3,xlab='Time (t)',ylab='density',col='blue')
 plot(res4$density~res4$time,type='l',main=expr4,xlab='Time (t)',ylab='density',col='blue')
```

`GQD.plot` *Quick Plots for DiffusionRgqd Objects*

Description

`GQD.plot()` recognizes output objects calculated using routines from the **DiffusionRgqd** package and subsequently constructs an appropriate plot, for example a perspective plot of a transition density.

Usage

```
GQD.plot(x, thin = 1, burns, h = FALSE, palette = "mono")
```

Arguments

`x`	Generic GQD-objects, i.e. `res = GQD.density()`.
`thin`	Thinning interval for `.mcmc` objects.
`burns`	Number of parameter draws to discard for `.mcmc` objects.

h	if TRUE a histogram is drawn i.s.o. a trace plot.
palette	Colour palette for drawing trace plots. Default palette = 'mono', otherwise a qualitative palette will be used.

Value

Varies in accordance with input type.

Author(s)

Etienne A.D. Pienaar: <etiannead@gmail.com>

References

Updates available on GitHub at https://github.com/eta21.

See Also

GQD.mcmc, GQD.mle, GQD.density, BiGQD.density etc.

Examples

```
# Remove any existing coefficients
GQD.remove()

# Define drift Coefficients. Note that the limiting mean is sinusoidal.
G0 <- function(t){2*(10+sin(2*pi*(t-0.5)))}
G1 <- function(t){-2}

# Define sinusoidal diffusion coefficient with `faster' oscillation.
Q1 <- function(t){0.25*(1+0.75*(sin(4*pi*t)))}

states     <-  seq(5,15,1/10) # State values
initial    <-  8              # Starting value of the process
Tmax       <-  5              # Time horizon
Tstart     <-  1              # Time starts at 1
increment  <-  1/100          # Incremental time steps

# Generate the transitional density
M <- GQD.density(Xs=initial,Xt=states,s=Tstart,t=Tmax,delt=increment)

GQD.plot(M)
```

GQD.remove *Remove the Coefficients of a GQD Model.*

Description

Removes any existing coefficient functions from the current workspace.

Usage

```
GQD.remove()
```

Details

GQD.remove clears the workspace of functions with names that match the coefficients of the 1D GQD. This may be used when more than one model is specified in a given session.

Value

No value is returned.

Note

GQD.remove simply searches the workspace for functions with definitions that match the form of the DiffusionRgqd interface and removes them from the workspace, freeing up the user to redefine a diffusion with new coefficients.

Author(s)

Etienne A.D. Pienaar: <etiennead@gmail.com>

References

Updates available on GitHub at https://github.com/eta21.

See Also

GQD.density and BiGQD.density.

GQD.TIpassage *Compute the First Passage Time Density of a GQD With Time Inhomogeneous Coefficients.*

Description

GQD.TIpassage() solves first passage time problems for GQDs with time dependent coefficients:

$$dX_t = (G_0(t) + G_1(t)X_t + G_2(t)X_t^2)dt + \sqrt{Q_0(t) + Q_1(t)X_t + Q_2(t)X_t^2}dW_t$$

to a fixed barrier. The function combines the cumulant truncation procedure of Varughese (2013) with a numerical solution to a non-singular Volterra integral equation for the first passage time density, developed by Buonocore et al. (1987), in order to generate an approximate solution.

Usage

```
GQD.TIpassage(Xs,B, s, t, delt, theta=c(0), IEQ.type='Buonocore', wrt=FALSE)
```

Arguments

Xs	Initial value of the diffusion process at time tmin.
B	Fixed barrier (or first constant in static barier transform - see detail **[1]**).
s	Starting time for the diffusion process (see detail **[2]**).
t	The time horizon up to and including which the density is to be evaluated.
delt	Stepsize for the solution of the first passage time density.
theta	Parameter vector for parameters contained in the coefficients (if required).
IEQ.type	Currently only IEQ.type = "Buonocore" is supported.
wrt	If TRUE a .cpp file will be written to the current directory. For bug report diagnostics.

Details

Detail [1]: First passage throug a time dependant barrier may be analised by applying the transform:

$$Y_t = X_t - B_t,$$

if B_t may can be decomposed as

$$B_t = k + f(t).$$

By applying Ito's lemma the revised drift and diffusion functionals, and first passage parameters may be inferred.

Detail [2]: The starting time is of particular importance when the drift and/or diffusion terms are time-inhomogeneous. Take care to select the correct starting time - especially if the drift or diffusion components whch are time dependant have poles or singular points in the time domain.

Value

`density`	The approximate first passage time density.
`time`	The time points at which the approximation is evaluated.
`prob.coverage`	The approximate cumulative probability coverage.

Warning

Warning [1]: Some instability may occur when `delt` is large or where the saddlepoint approximation fails. As allways it is important to check both the validity of the diffusion process under the given parameters and the quality of the saddlepoint approximation. For a given set of parameters the latter can be checked using `GQD.density`.

Warning [2]: The first passage time problem is considered from one side only i.e. Xs<B. For Xs>B one may equivalently consider Yt=-X_t, apply Ito's lemma and proceed as above.

Note

Note [1]: The coefficients od the GQD may be parameterized using the reserved variable `theta`. For example:

```
G0 <- function(t){theta[1]*(theta[2]+sin(2*pi*(t-theta[3])))}
```

may be used so long as values are asigned in the function call, say

```
GQD.TIpassage(Xs=3,B=11,tmin=1,tmax=10,delt=1/100,theta=c(1,10,0.5)).
```

Note [2]: Due to syntactical differences between R and C++ special functions have to be used when terms that depend on `t`. When the function cannot be separated in to terms that contain a single `t`, the `prod(a,b)` function must be used. For example (see examples below):

```
G0 <- function(t){0.1*(10+0.2*sin(2*pi*t)+0.3*prod(sqrt(t),1+cos(3*pi*t)))}.
```

Here sqrt(t)*cos(3*pi*t) constitutes the product of two terms that cannot be written i.t.o. a single `t`. To circumvent this isue, one may use the `prod(a,b)` function.

Note [3]: Similarly, the ^ - operator is not overloaded in C++. Instead the `pow(x,p)` function may be used to calculate x^p. For example sin(2*pi*t)^3 in:

```
G0 <- function(t){0.1*(10+0.2*pow(sin(2*pi*t),3))}.
```

Note [4]: `delt` is used as the stepsize of the Runge-Kutta method used to numerically solve a system of ODEs used to approximate the cumulants of the underlying diffusion procees. The 10th-order scheme of Feagin (2007) is used as the default method.

Author(s)

Etienne A.D. Pienaar: `<etiennead@gmail.com>`

References

Updates available on GitHub at `https://github.com/eta21`.

A. Buonocore, A. Nobile, and L. Ricciardi. 1987 A new integral equation for the evaluation of first-passage- time probability densities. *Adv. Appl. Probab.* **19**:784–800.

Daniels, H.E. 1954 Saddlepoint approximations in statistics. *Ann. Math. Stat.*, **25**:631–650.

Eddelbuettel, D. and Romain, F. 2011 Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, **40**(8):1–18,. URL http://www.jstatsoft.org/v40/i08/.

Eddelbuettel, D. 2013 *Seamless R and C++ Integration with Rcpp*. New York: Springer. ISBN 978-1-4614-6867-7.

Eddelbuettel, D. and Sanderson, C. 2014 Rcpparmadillo: Accelerating r with high-performance C++ linear algebra. *Computational Statistics and Data Analysis*, **71**:1054–1063. URL http://dx.doi.org/10.1016/j.csda.2013.02.005.

Feagin, T. 2007 A tenth-order Runge-Kutta method with error estimate. *In Proceedings of the IAENG Conf. on Scientific Computing*.

R. G. Jaimez, P. R. Roman and F. T. Ruiz. 1995 A note on the volterra integral equation for the first-passage-time probability density. *Journal of applied probability*, 635–648.

Varughese, M.M. 2013 Parameter estimation for multivariate diffusion systems. *Comput. Stat. Data An.*, **57**:417–428.

See Also

GQD.density for functions that generate the transitional density of GQDs and GQD.mcmc for MCMC procedures for GQD models.

Examples

```
#=============================================================================
# Generate the first passage time density of a CIR process with time
# dependant drift to a fixed barrier.
#-----------------------------------------------------------------------------

  # Remove any existing coefficients.
  GQD.remove()

  # Define the coefficients of the process.
  G0 <- function(t){10+0.5*sin(2*pi*t)}
  G1 <- function(t){-1}
  Q1 <- function(t){0.25}


  #Define the parameters of the first passage time problem.
  delt  <- 1/100         # The stepsize for the solution
  X0 <- 8                # The initial value of the process
  BB <- 11               # Fixed barrier
  T0 <- 2                # Starting time of the diffusion
  TT <- 10                # Time horizon of the computation

  # Run the calculation
  res1 <- GQD.TIpassage(X0,BB,T0,TT,delt)

  # Remove any existing coefficients.
  GQD.remove()

  # Redefine the coefficients.
```

```
  G0 <- function(t){ 0.1*(10+0.2*sin(2*pi*t)+0.3*prod(sqrt(t),1+cos(3*pi*t)))}
  G1 <- function(t){-0.1*(1+0.2*sin(2*pi*t))}
  Q1 <- function(t){0.25}

  # Redefine the parameters of the f.p.t. problem.

  delt  <- 1/100
  X0 <- 8
  BB <- 11
  T0 <- 1
  TT <- 10

  # Run the calculation
  res2 <- GQD.TIpassage(X0,BB,T0,TT,delt)

#=============================================================================
# Plot the two solutions.
#=============================================================================
  expr1 <- expression(dX[t]==(10+0.5*sin(2*pi*t)-X[t])*dt+0.25*sqrt(X[t])*dW[t])
  expr2 <- expression(dX[t]==(0.1*(10+0.2*sin(2*pi*t)+0.3*sqrt(t)*(1+cos(3*pi*t))
                      -0.1*(1+0.2*sin(2*pi*t))*X[t])*dt+0.25*sqrt(X[t])*dW[t]))


  par(mfrow=c(1,1))
  plot(res1$density~res1$time,type='l',col='blue',
       ylab='Density',xlab='Time',main=expr1,cex.main=0.95)

  plot(res2$density~res2$time,type='l',col='red',
       ylab='Density',xlab='Time',main =expr2,cex.main=0.95)

#=============================================================================
# Let's see how sensitive the first passage density is w.r.t a speed parameter
# of a non-linear diffusion.
#=============================================================================

GQD.remove()
# Redefine the coefficients with a parameter theta:
G1 <- function(t){theta[1]*(10+0.2*sin(2*pi*t)+0.3*prod(sqrt(t),1+cos(3*pi*t)))}
G2 <- function(t){-theta[1]}
Q2 <- function(t){0.1}
# Now just give a value for the parameter in the standard fashion:
res3=GQD.TIpassage(8,12,1,4,1/100,theta=c(0.5))

plot(res3$density~res3$time,type='l',col=2,ylim=c(0,1.0),
main='First Passage Time Density',ylab='Density',xlab='Time',cex.main=0.95)
# Change the parameter and see the effect on the f.p.t. density.
th.seq=seq(0.1,0.5,1/20)
for(i in 2:length(th.seq))
{
  res3=GQD.TIpassage(8,12,1,4,1/100,,theta=c(th.seq[i]))
  lines(res3$density~res3$time,type='l',col=rainbow(10)[i])
}
lines(res3$density~res3$time,type='l',col=rainbow(10)[i],lwd=2)
```

```
legend('topright',legend=th.seq,col=rainbow(10),lty='solid',cex=0.75,
title=expression(theta[1]))
#============================================================================
```

RcppArmadillo-Functions *A Junk Funktion For Build Purposes*

Description

This function was created as a filler in order for the package to build correctly.

Usage

```
junkfunction()
```

Details

This function was created as a filler in order for the package to build correctly.

Value

`junkfunction()` does nothing useful.

Author(s)

Etienne A.D. Pienaar

References

See the documentation for Armadillo, and RcppArmadillo, for more details.

SDEsim1 *A Simulated Diffusion with Sinusoidal Drift and State-Dependant Diffusion Coefficient.*

Description

The dataset contains discretely sampled observations for a simulated stochastic differential equation (SDE) with dynamics:

$$dX_t = 2(5 + 3\sin(0.25\pi t) - X_t)dt + 0.5\sqrt{X_t}dW_t$$

where `dW_t` is standard Brownian motion, `t` is time and `X_0 = 7`.

Usage

```
data(SDEsim1)
```

Format

A data frame with 401 observations with the following variables:

1. `Xt` : A numeric vector of simulated observations.
2. `time` : A numeric vector of time nodes at which `Xt` was observed (`time[i+1]-time[i] = 1/4`).

Details

The process was simulated by numerically solving the SDE using a Euler-Maruyama scheme with stepsize = 1/2000. Subsequently each 200-th observation was recorded in order to construct the resulting time series.

Author(s)

Etienne A.D. Pienaar: <etiannead@gmail.com>

References

Updates available on GitHub at https://github.com/eta21.

Examples

```
data(SDEsim1)
attach(SDEsim1)
par(mfrow=c(1,1))
expr1=expression(dX[t]==2*(5+3*sin(0.5*pi*t)-X[t])*dt+0.5*sqrt(X[t])*dW[t])
plot(Xt~time,type='l',col='blue',xlab='Time (t)',ylab=expression(X[t]),main=expr1)
```

SDEsim2 *A Simulated Non-Linear Bivariate Diffusion*

Description

The dataset contains discretely sampled observations for a simulated stochastic differential equation (SDE) with dynamics:

$$dX_t = 2.0(Y_t - X_t)dt + 0.3\sqrt{X_t Y_t}dW_t$$

$$dY_t = 1.0(5 - Y_t)dt + 0.5\sqrt{Y_t}dB_t$$

where `dW_t` and `dB_t` are standard Brownian motions, `t` is time and `X_0 = 5`, `Y_0 = 5`.

Usage

```
data("SDEsim2")
```

Format

A data frame with 801 observations on the following 3 variables.

`Xt` Xt trajectory of the diffusion.

`Yt` Yt trajectory of the diffusion.

`time` Time vector (`time[i+1]-time[i] = 1/8`).

Examples

```
data(SDEsim2)
 data(SDEsim2)
 attach(SDEsim2)
 # Have a look at the time series:
 plot(Xt~time,type='l',col='blue',ylim=c(2,10),main='Simulated Data',xlab='Time (t)',ylab='State',
      axes=FALSE)
 lines(Yt~time,col='red')
 expr1=expression(dX[t]==2(Y[t]-X[t])*dt+0.3*sqrt(X[t]*Y[t])*dW[t])
 expr2=expression(dX[t]==(5-Y[t])*dt+0.5*sqrt(Y[t])*dB[t])
 text(50,9,expr1)
 text(50,8.5,expr2)
 axis(1,seq(0,100,5))
 axis(1,seq(0,100,5/10),tcl=-0.2,labels=NA)
 axis(2,seq(0,20,2))
```

SDEsim3 *Simulated Stochastic Lotka-Volterra Eqns*

Description

The dataset contains discretely sampled observations for a simulated stochastic differential equation (SDE) with dynamics:

$$dX_t = (1.5X_t - 0.4X_tY_t)dt + \sqrt{0.05X_t}dW_t$$

$$dY_t = (-1.5Y_t + 0.4X_tY_t - 0.2Y_t^2)dt + \sqrt{0.1X_t}dB_t$$

where `dW_t` and `dB_t` are standard Brownian motions, `t` is time and `X_0 = 5`, `Y_0 = 5`.

Usage

```
data("SDEsim3")
```

Format

A data frame with 1001 observations on the following 3 variables.

`time` A numeric vector of time nodes at which means are calculated (`time[i+1]-time[i] = 1/4`).

`mx` Mean Xt trajectory of the diffusion.

`my` Mean Yt trajectory of the diffusion.

SDEsim4 *A Simulated Non-Linear Bivariate Diffusion With Time-Inhomogeneous Coefficients*

Description

The dataset contains discretely sampled observations for a simulated stochastic differential equation (SDE) with dynamics:

$$dX_t = (1.0(7.5 - X_t) + 1.5Y_t)dt + 0.5\sqrt{X_t Y_t}dW_t$$

$$dY_t = (1.5(5 - Y_t) + 3\sin(0.25\pi t))dt + 0.25\sqrt{Y_t}dB_t$$

where `dW_t` and `dB_t` are standard Brownian motions, `t` is time and `X_0 = 10`, `Y_0 = 5`.

Usage

```
data("SDEsim4")
```

Format

A data frame with 401 observations on the following 3 variables.

`Xt` Xt trajectory of the diffusion.

`Yt` Yt trajectory of the diffusion.

`time` Time vector.

Examples

```
data(SDEsim4)
attach(SDEsim4)
# Have a look at the time series:
plot(Xt~time,type='l',col='blue',ylim=c(0,25),main='Simulated Data',
xlab='Time (t)',ylab='State',axes=FALSE)
lines(Yt~time,col='red')
axis(1,seq(0,100,5))
axis(1,seq(0,100,5/10),tcl=-0.2,labels=NA)
axis(2,seq(0,25,2))
axis(2,seq(0,25,2/10),tcl=-0.2,labels=NA)
```

SDEsim5 *Simulated First Passage Times for a Time-Inhomogeneous Non-Linear Diffusion*

Description

A histogram with counts/density values for a 500000 simulated first passage times of a diffusion:

$$dX_t = \theta_1 X_t(10 + 0.2\sin(2\pi t) + 0.3\sqrt{t}(1 + \cos(3\pi t)) - X_t)dt + \sqrt{0.1}X_t dB_t$$

starting in X_1 = 8 traversing a fixed barrier at B = 8.

Usage

```
data("SDEsim5")
```

Format

A data frame with 92 observations on the following 3 variables.

`counts` Histogram counts for each bin.

`density` Approximate density values.

`mids` Midpoints for each bin.

Examples

```
data(SDEsim5)
plot(SDEsim5$density ~ c(SDEsim5$mids-diff(SDEsim5$mids)[1] / 2), type = 's',
  lty = 'solid', lwd = 1, xlab = 'time', ylab = 'Density')
```

SDEsim6 *Simulated First Passage Times for a Time-Inhomogeneous Non-Linear Diffusion*

Description

A dataset of 500000 simulated first passage times of a diffusion:

$$dX_t = \theta_1 X_t(10 + 0.2\sin(2\pi t) + 0.3\sqrt{t}(1 + \cos(3\pi t)) - X_t)dt + \sqrt{0.1}X_t dB_t$$

starting in X_1 = 8 traversing a fixed barrier at B = 8.

Usage

```
data("SDEsim6")
```

Format

`fpt.sim.times` 500 000 simulated first passage times.

Examples

```
data("SDEsim6")
hist(fpt.sim.times, freq = FALSE, col = 'gray85', border = 'white',
  main = 'First Passage Time Density', ylab = 'Density', xlab = 'Time',
  breaks = 100)
```

Index