Package 'DiffusionRjgqd'

August 29, 2016

Version 0.1.1

Title Inference and Analysis for Jump Generalized Quadratic Diffusions

Description
Tools for performing inference and analysis on a class of quadratic jump diffusion processes.

URL https://github.com/eta21

BugReports https://github.com/eta21/DiffusionRjgqd/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 DiffusionRgqd, knitr, coda, 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 2016-08-16 19:46:05

R topics documented:

DiffusionRjgqd-package 2
BiJGQD.density 3
BiJGQD.mcmc 6
JGQD.density 10
JGQD.dic 13
JGQD.estimates 14
JGQD.mcmc 15
JGQD.plot 18

JGQD.remove 20
JSDEsim1 21
JSDEsim2 21
JSDEsim3 22
RcppArmadillo-Functions 23

Index **24**

`DiffusionRjgqd-package`

A package for Performing Inference and Analysis on Generalized Quadratic Jump Diffusion Processes (JGQDs).

Description

DiffusionRjgqd is a toolbox for performing analysis and inference on a class of jump diffusion processes with quadratic drift and diffusion with state-dependent and time inhomogeneous jump mechanisms. The package consists of functions for performing likelihood based inference and transitional density approximations for both 1D and 2D JGQDs.

Details

Package:	DiffusionRjgqd
Type:	Package
Version:	0.1.0
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 JGQD after which the package handles all the necessary mathematics and algorithmic construction. Furthermore, computational efficiency is optimized by constructing algorithms in C++ using the **Rcpp** and **RcppArmadillo** libraries.

Functions included in the package:

`BiJGQD.density`	:	Generate the transitional density of a 2D JGQD.
`BiJGQD.mcmc`*	:	Conduct inference via MCMC on a 2D JGQD.
`JGQD.density`	:	Generate the transitional density of a 1D JGQD.
`JGQD.mcmc`*	:	Conduct inference via MCMC on a 1D JGQD.

* 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

BiJGQD.mcmc, JGQD.mcmc, JGQD.dic,JGQD.remove and JGQD.density.

Examples

```
## Not run:
example(JGQD.density)
example(BiJGQD.density)

## End(Not run)
```

BiJGQD.density *Generate the Transition Density of a Bivariate Jump Generalized Quadratic Diffusion Model (2D JGQD).*

Description

BiJGQD.density generates approximate transitional densities for bivariate generalized quadratic jump diffusions (JGQDs). 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].

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

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

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

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

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

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

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

and $d\mathbf{P}_t = J(\mathbf{X}_t, t) d\mathbf{N}_t$ describes a Poisson process with jump matrix:

$$\mathbf{J}(\mathbf{X}_t, t, \dot{\mathbf{z}}_t)^{(1,1)} = \dot{z}_{11},$$
$$\mathbf{J}(\mathbf{X}_t, t, \dot{\mathbf{z}}_t)^{(2,1)} = \dot{z}_{21},$$

if jumps are additive or

$$\mathbf{J}(\mathbf{X}_t, t, \dot{\mathbf{z}}_t)^{(1,1)} = \dot{z}_{11} X_t,$$
$$\mathbf{J}(\mathbf{X}_t, t, \dot{\mathbf{z}}_t)^{(2,1)} = \dot{z}_{21} Y_t,$$

if jumps are multiplicative. For purposes of this package the arrival rate is decomposed as:

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

and

$$(\dot{z}_{11}, \dot{z}_{21})' \sim \text{MVN}(\boldsymbol{\mu}_J, \boldsymbol{\Sigma}_J).$$

Usage

```
BiJGQD.density(Xs, Ys, Xt, Yt, s, t, delt, Dtype, Jdist, Jtype, print.output,
               eval.density)
```

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"`.
`Jdist`	Valid entries are 'MVNormal' (currently).
`Jtype`	Valid types are 1 or 2.
`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

`BiJGQD.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.
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 `BiJGQD.mcmc` and `JGQD.density`.

Examples

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

 # For detailed notes and examples on how to use the BiJGQD.density() function, see
 # the following vignette:

 RShowDoc('Part_3_Bivariate_Diffusions',type='html','DiffusionRjgqd')

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

`BiJGQD.mcmc` *MCMC Inference on Bivariate Jump Generalized Quadratic Diffusions (2D JGQDs).*

Description

`BiJGQD.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. The user may specify any model within the JGQD framework by defining parametrised functions giving the form of the coefficients of the model.

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

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

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

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

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

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

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

and $d\mathbf{P}_t = J(\mathbf{X}_t, t)d\mathbf{N}_t$ describes a Poisson process with jump matrix:

$$\mathbf{J}(\mathbf{X}_t, t, \dot{\mathbf{z}}_t)^{(1,1)} = \dot{z}_{11},$$

$$\mathbf{J}(\mathbf{X}_t, t, \dot{\mathbf{z}}_t)^{(2,1)} = \dot{z}_{21},$$

if jumps are additive or

$$\mathbf{J}(\mathbf{X}_t, t, \dot{\mathbf{z}}_t)^{(1,1)} = \dot{z}_{11}X_t,$$

$$\mathbf{J}(\mathbf{X}_t, t, \dot{\mathbf{z}}_t)^{(2,1)} = \dot{z}_{21}Y_t,$$

if jumps are multiplicative. For purposes of this package the arrival rate is decomposed as:

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

and

$$(\dot{z}_{11}, \dot{z}_{21})' \sim \text{MVN}(\boldsymbol{\mu}_J, \boldsymbol{\Sigma}_J).$$

Usage

```
BiJGQD.mcmc(X, time, mesh = 10, theta, sds, updates = 10,
          burns = min(round(updates/2),25000), exclude = NULL, plot.chain = TRUE,
            RK.order = 4, wrt = FALSE, Tag = NA, Dtype = "Saddlepoint",
            Jdist = "MVNormal", Jtype ='Add', adapt = 0, print.output = TRUE,
            decode = 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".
Jdist	Valid entries are 'MVNormal' (currently).
Jtype	Valid types are 1 or 2.
adapt	For development purposes.
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.
decode	Should the algorithm estimate jump detection probabilities? Default value is TRUE.
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).
decode.prob	Estimated jump detection probabilities.

Syntactical jargon

Synt. [1]: The coefficients of the 2D JGQD 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

`JGQD.remove` and `JGQD.mcmc`.

Examples

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

 # For detailed notes and examples on how to use the BiJGQD.mcmc() function, see
 # the following vignette:

 RShowDoc('Part_4_Likelihood_Inference',type='html','DiffusionRjgqd')

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

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

Description

`JGQD.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`.

`JGQD.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

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

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

and $dP_t = j(X_t, \dot{z}_t)dN_t$ describes a Poisson process with intensity:

$$\lambda(X,t) = \lambda_0(t,\theta) + \lambda_1(t,\theta)X_t + \lambda_2(t,\theta)X_t^2,$$

jump coefficient

$$j(X_t, \dot{z}_t) = \dot{z}_t$$

or

$$j(X_t, \dot{z}_t) = \dot{z}_t X_t$$

and

$$\dot{z}_t \sim \text{Normal}(\mu_J(t,\theta), \sigma_J^2(t,\theta)),$$

$$\dot{z}_t \sim \text{Exponential}(\lambda_J(t,\theta)),$$

$$\dot{z}_t \sim \text{Gamma}(\alpha_J(t,\theta), \beta_J(t,\theta)),$$

or

$$\dot{z}_t \sim \text{Laplace}(a_J(t,\theta), b_J(t,\theta)).$$

Usage

```
JGQD.density(Xs = 4, Xt = seq(5, 8, 1/10), s = 0, t = 5, delt =1/100,
             Jdist = "Normal", Jtype = "Add", Dtype = "Saddlepoint",
             Trunc = c(8, 4), factorize = FALSE, factor.type = "Diffusion",
              beta, 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' and 'Edgeworth' 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 and 8 with the constraint that Trunc[1] >= Trunc[2]. Default is c(4,4).
Jdist	Valid entries are 'Normal', 'Exponential', 'Gamma' or 'Laplace'.
Jtype	Valid types are 'Add' or 'Mult'.
factorize	Should factorization be used (default = TRUE).
factor.type	Can be used to envoke a special factorization in order to evaluate Hawkes processes nested within the JGQD framework.
beta	Variable used for a special case jump structure (for research purposes).
print.output	If TRUE, model information is printed to the console.
eval.density	If TRUE, the density is evaluated in addition to calculating the moment eqns.

Details

JGQD.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.

Interface

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 jump 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.

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 `JGQD.mcmc` and `BiJGQD.density`.

Examples

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

 # For detailed notes and examples on how to use the JGQD.density() function, see
 # the following vignette:

 RShowDoc('Part_2_Transition_Densities',type='html','DiffusionRjgqd')

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

JGQD.dic *Summarize MCMC Selection Output for a List of JGQD.mcmc or Bi-JGQD.mcmc Objects.*

Description

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

Usage

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

Arguments

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

Details

JGQD.dic() summarizes the output from various models fitted via JGQD.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

JGQD.mcmc

Examples

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

 # For detailed notes and examples on how to use the BiJGQD.dic() function, see
 # the following vignette:

 RShowDoc('Part_4_Likelihood_Inference',type='html','DiffusionRjgqd')

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

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

Description

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

Usage

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

Arguments

x	List object of type 'JGQD.mcmc' or 'JGQD.mle'. That is, when model =JGQD.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

JGQD.mcmc, BiJGQD.mcmc.

Examples

```
example(JGQD.mcmc)
```

JGQD.mcmc *MCMC Inference on Jump Generalized Quadratic Diffusion Models (JGQDs).*

Description

JGQD.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. The structure of the model is predefined and coefficients may be provided for models nested within the generalized quadratic diffusion framework.

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

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

where

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

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

and $dP_t = j(X_t,\dot{z}_t)dN_t$ describes a Poisson process with intensity:

$$\lambda(X,t) = \lambda_0(t,\theta) + \lambda_1(t,\theta)X_t + \lambda_2(t,\theta)X_t^2,$$

jump coefficient

$$j(X_t,\dot{z}_t) = \dot{z}_t$$

or

$$j(X_t,\dot{z}_t) = \dot{z}_t X_t$$

and

$$\dot{z}_t \sim \text{Normal}(\mu_J(t,\theta), \sigma_J^2(t,\theta)),$$

$$\dot{z}_t \sim \text{Exponential}(\lambda_J(t,\theta)),$$

$$\dot{z}_t \sim \text{Gamma}(\alpha_J(t,\theta), \beta_J(t,\theta)),$$

or

$$\dot{z}_t \sim \text{Laplace}(a_J(t,\theta), b_J(t,\theta)).$$

Usage

```
JGQD.mcmc(X, time, mesh = 10, theta, sds, updates = 1000,
          burns = min(round(updates/2), 25000), Jtype = "Add", Jdist = "Normal",
         Dtype = "Saddlepoint", RK.order = 4, exclude = NULL, plot.chain = TRUE,
          wrt = FALSE, Tag = NA, factorize = TRUE, print.output = TRUE,
          decode = 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' is supported in the current version of the software.
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.
Jdist	Valid entries are 'Normal', 'Expnential', 'Gamma' and 'Laplace'.
Jtype	Valid types are 'Add' or 'Mult'.
factorize	Should factorization be used (default = TRUE).
print.output	If TRUE, model information is printed to the console.
decode	Should the algorithm estimate jump detection probabilities? Default value is TRUE.
palette	Colour palette for drawing trace plots. Default palette = 'mono', otherwise a qualitative palette will be used.

Details

`JGQD.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 `JGQD.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`).
`decode.prob`	Estimated jump detection probabilities.

Syntactical jargon

Synt. [1]: The coefficients of the JGQD 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

JGQD.remove, BiJGQD.mcmc.

Examples

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

 # For detailed notes and examples on how to use the JGQD.mcmc() function, see
 # the following vignette:

 RShowDoc('Part_4_Likelihood_Inference',type='html','DiffusionRjgqd')

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

JGQD.plot *Quick Plots for DiffusionRjgqd Objects*

Description

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

Usage

```
JGQD.plot(x, thin = 1, burns, h = FALSE,  palette = 'mono')
```

Arguments

x	Generic JGQD-objects, i.e. `res = JGQD.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

JGQD.mcmc, JGQD.density, BiJGQD.density etc.

Examples

```
#============================================================================
# Plot the transitional density of a jump diffusion
#----------------------------------------------------------------------------
rm(list=ls(all=TRUE))
library(DiffusionRjgqd)

JGQD.remove()
# Define the jump diffusion using the DiffusionRjgqd syntax:
G1=function(t){0.2*5+0.1*sin(2*pi*t)}
G2=function(t){-0.2}
Q1=function(t){0.2}

# State dependent intensity:
Lam0 = function(t){1}
Lam1    = function(t){0.1}

# Normally distributed jumps: N(1,0.2)
Jmu    = function(t){1.0}
Jsig   = function(t){0.2}
# Normal distribution is the default:
res_1  = JGQD.density(4,seq(2,10,1/10),0,2.5,1/100,factorize=FALSE)

JGQD.plot(res_1)
```

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

Description

Removes any existing coefficient functions from the current workspace.

Usage

```
JGQD.remove()
```

Details

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

Value

No value is returned.

Note

JGQD.remove simply searches the workspace for functions with definitions that match the form of the DiffusionRjgqd 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

JGQD.density and BiJGQD.density.

JSDEsim1 *Simulated Trajectory of a Scalar Jump Diffusion.*

Description

Simulated trajectory of a scalar jump diffusion with Normal distributed jumps ariving with stochastic intensity. The dynamics of the process is given by the jump SDE:

$$dX_t = (5 - X_t)dt + 0.1\sqrt{X_t}dB_t + dP_t$$

where $dP_t = \dot{z}_t dN_t$ describes a Poisson process with intensity:

$$\lambda(X_t) = 0.5X_t$$

and

$$\dot{z} \sim \mathrm{N}(0.5, 0.5^2)$$

Usage

```
data("JSDEsim1")
```

Format

A data frame with 251 observations on the following 2 variables.

`Xt` Observed trajectory.

`time` Time index for the observed trajectory

`contains_jump` Indicator variable to indicate if a given transition contains a jump (ignore the first value).

Examples

```
data(JSDEsim1)
```

JSDEsim2 *Simulated Trajectory of a Bivariate Jump Diffusion.*

Description

Simulated trajectory of a bivariate jump diffusion with Normally distributed jumps . The dynamics of the process is given by the jump SDE:

$$dX_t = 0.5(2 + Y_t - X_t)dt + 0.1\sqrt{X_t Y_t}dB_t + dP_t^1$$

$$dX_t = (5 - X_t)dt + 0.1\sqrt{Y_t}dW_t + dP_t^2$$

where

$$dP_t^1 = \dot{z}_1 dN_t^1$$
$$dP_t^2 = \dot{z}_2 dN_t^1$$

describes a bivariate Poisson process with single excitations ariving with intensity:

$$\lambda(X_t, Y_t) = 1$$

and

$$\{\dot{z}_1, \dot{z}_2\}' \sim \text{Bivariate Normal}(\{0.5, 0.5\}', diag(\{0.5, 0.5\}))$$

.

Usage

```
data("JSDEsim2")
```

Format

A data frame with 501 observations on the following 3 variables.

`Xt` a numeric vector

`Yt` a numeric vector

`time` a numeric vector

Examples

```
data(JSDEsim2)
```

`JSDEsim3` *Jump Observations for a Bivariate Simulated Dataset.*

Description

Jump observations for a simulated trajectory of a bivariate jump diffusion with Normally distributed jumps. The dynamics of the process is given by the jump SDE:

$$dX_t = 0.5(2 + Y_t - X_t)dt + 0.1\sqrt{X_t Y_t}dB_t + dP_t^1$$
$$dX_t = (5 - X_t)dt + 0.1\sqrt{Y_t}dW_t + dP_t^2$$

where

$$dP_t^1 = \dot{z}_1 dN_t^1$$
$$dP_t^2 = \dot{z}_2 dN_t^1$$

describes a bivariate Poisson process with single excitations ariving with intensity:

$$\lambda(X_t, Y_t) = 1$$

and

$$\{\dot{z}_1, \dot{z}_2\}' \sim \text{Bivariate Normal}(\{0.5, 0.5\}', diag(\{0.5, 0.5\}))$$

.

Usage

```
data("JSDEsim3")
```

Format

A data frame with 100001 observations on the following 3 variables.

`Xjumps` Observed jumps in the X-dimension.

`Yjumps` Observed jumps in the X-dimension.

`Jtime` Times at which jumps are observed.

Examples

```
data(JSDEsim3)
## maybe str(JSDEsim3) ; plot(JSDEsim3) ...
```

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

```
junkfunction2()
```

Details

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

Value

`junkfunction2()` does nothing useful.

Author(s)

Etienne A.D. Pienaar

References

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

Index

∗Topic **C++**
BiJGQD.mcmc, 6
DiffusionRjgqd-package, 2
JGQD.mcmc, 15
∗Topic **MCMC**
BiJGQD.mcmc, 6
∗Topic **Moments**
BiJGQD.density, 3
∗Topic **Syntax**
BiJGQD.mcmc, 6
∗Topic **Transition density**
BiJGQD.density, 3
∗Topic **datasets**
JSDEsim1, 21
JSDEsim2, 21
JSDEsim3, 22
∗Topic **deviance information criterion (DIC)**
JGQD.dic, 13
∗Topic **mcmc**
JGQD.mcmc, 15
∗Topic **moments**
JGQD.density, 10
∗Topic **package**
DiffusionRjgqd-package, 2
∗Topic **plot**
JGQD.plot, 18
∗Topic **remove models**
JGQD.remove, 20
∗Topic **syntax**
JGQD.mcmc, 15
∗Topic **transition density**
JGQD.density, 10

BiJGQD.density, *2*, 3, *12*, *19*, *20*
BiJGQD.mcmc, *2*, *3*, *5*, 6, *15*, *18*

DiffusionRjgqd (DiffusionRjgqd-package), 2
DiffusionRjgqd-package, 2

JGQD.density, *2*, *3*, *5*, 10, *19*, *20*
JGQD.dic, *3*, 13
JGQD.estimates, 14
JGQD.mcmc, *2*, *3*, *9*, *12*, *13*, *15*, 15, *19*
JGQD.plot, 18
JGQD.remove, *3*, *9*, *17*, *18*, 20
JSDEsim1, 21
JSDEsim2, 21
JSDEsim3, 22
junkfunction2 (RcppArmadillo-Functions), 23

RcppArmadillo-Functions, 23