DiffusionRgqd: An R Package for Performing Inference and Analysis on Time-Inhomogeneous Quadratic Diffusion Processes

Etienne A.D. Pienaar
University of Cape Town

Melvin M. Varughese
University of Cape Town

Abstract

Diffusion processes are useful tools for quantifying the dynamics of real world processes. By formulating a model in terms of a stochastic differential equation it is possible to gain insight into the dynamics of continuously evolving processes from discretely sampled trajectories using a compact set of equations. As such diffusion models are extremely flexible and have found application in numerous fields of science. Perhaps one of the principle limitations in the use of diffusion models is the relatively sparse ecosystem of models with analytically intractable dynamics. As such, analysis has often focused on linear, time-homogeneous models. In this paper, we introduce the **DiffusionRgqd** package: A collection of tools for performing inference and analysis on scalar and bivariate time-inhomogeneous diffusion processes with quadratic drift and diffusion terms in R. The package focuses on likelihood based inference and model selection for discretely observed diffusion processes and analysis based on quantities such as the transitional density and first passage time densities for scalar diffusion processes. We illustrate various features of the package using practical examples by analysing a number non-linear diffusion processes and the analysis of a real-world dataset.

Keywords: stochastic differential equation, diffusion process, R, first passage time density, C++.

1. Introduction

Stochastic calculus has been used in numerous scientific disciplines to characterize the dynamics of stochastic systems. Diffusion processes can be seen as continuous time dynamical systems that incur both deterministic and random fluctuations. Indeed, diffusion processes can be interpreted as the stochastic counterparts to systems of ordinary differential equations (ODEs) that are so ubiquitous in the sciences. As such diffusion processes provide a means of modelling both the local behaviour (at small time scales) and global dynamics (on large time scales) of a process using a single system of equations, whilst simultaneously accounting for random deviations in the process trajectory. Although the analysis of diffusion processes have centred around diffusion models with analytically tractable dynamics, such models often fall short in describing real-world processes with more complex underlying behaviour driven by non-linear spatial and/or time dependencies. Indeed, the problem of performing inference on non-linear diffusion processes has lead to the development of many innovative strategies for extracting model parameters from discretely observed time series. Although these algorithms lend access to wide array of diffusion models, the mathematics that underpin these methods can be daunting and often require a good understanding of technical material from outside the discipline of pure statistics or the context of the desired application. As such the use of these strategies have been limited to fields that overlap with stochastic calculus such as for example mathematical finance and physics. Whilst the application of non-linear diffusion models has propagated in recent years, the growth has been somewhat stunted by the computational complexity of existing methodologies - at

least from the perspective of researchers in non-statistical or mathematical fields. Consequently demand has arisen for software that makes non-linear diffusion models more accessible in less mathematically focused sciences. In response to this demand we have endeavoured to develop a series of packages for the ubiquitous statistical software language, R, that aims to collect methods for performing inference and analysis on diffusion processes.

In the present paper we develop the **DiffusionRgqd** package - a collection of tools for performing inference and analysis on a class of quadratic diffusion processes. The routines are centred around a computationally efficient numerical method for calculating accurate approximations to the transitional density of polynomial diffusion processes. By using dynamic algorithm construction techniques, these routines construct solutions tailored to the model specification without requiring any mathematical input from the user over and above the model definition. As such we are able to optimize the computational efficiency of routines in the package whilst providing minimal constraints on the model specification within the applicable class of diffusions. By separating the user from the underlying mathematical technicalities, the package provides access to a suitably general class of diffusions whilst demanding only basic programming skills and a graduate level understanding of likelihood based inference procedures. For the R package developed in the present paper we focus on two problems: Firstly, we develop modules for performing likelihood inference on quadratic scalar and bivariate diffusion models for discretely observed time-series. Secondly, we develop a module for computing the first passage time density of a scalar diffusion transiting through a fixed barrier.

The paper is organised as follows: Section 2 introduces some theoretical concepts relating to diffusion processes that are relevant to the methodology of the paper. In Section 3 we introduce the modelling interface and outline the methodology that underpins the the package. This is achieved by defining a suitably general class of quadratic diffusion processes that can be easily communicated in the software environment. In Section 4 we subsequently develop the mathematical underpinnings of the package based on a computationally efficient method for calculating the transition density of a diffusion process. Section 5 focuses on the mechanical aspects of routines in the package wherein we outline how mathematical redundancies are combined with the C++ language in order to maximize computational efficiency of the package within R. Section 6 outlines the routines contained package and Section 7 illustrates the capabilities of the package via practical examples and – where applicable – comparisons to existing packages. Finally in Section 8 we give some concluding remarks to the paper.

2. Diffusion processes

Diffusion processes are defined as continuous-time continuous-state Markov processes, governed by stochastic differential equations (SDEs). The dynamics of the k-dimensional vector $\mathbf{X}_t = \{X_t^1, X_t^2, \ldots, X_t^k\}'$ is given by the SDE:

$$\mathbf{dX_t} = \boldsymbol{\mu}(\mathbf{X}_t, t)dt + \boldsymbol{\sigma}(\mathbf{X}_t, t)\mathbf{dB}_t, \tag{1}$$

where $\boldsymbol{\mu}(\mathbf{X}_t, t) = (\mu_i(\mathbf{X}_t, t))_{i=1,2,\ldots,k}$ is the k-dimensional drift vector of the process and $\boldsymbol{\sigma}(\mathbf{X}_t, t) = (\sigma_{i,j}(\mathbf{X}_t, t))_{i,j=1,2,\ldots,k}$ gives the $k \times k$ diffusion matrix of the process. The system in eqn 1 is driven by a k-dimensional vector of independent Brownian motions $\mathbf{B}_t = \{B_t^1, B_t^2, \ldots, B_t^k\}'$ with the fundamental properties:

- for all $j = 1, \ldots, k$, $B_0^j = 0$,
- non-overlapping increments $B_t^j - B_s^j$ with $s < t$ are independent and
- $B_t^j - B_s^j \sim N(0, t-s)$ where $N(0, \phi)$ is the Normal distribution with standard deviation $\sqrt{\phi}$ and mean 0.

Furthermore, define $\mathbf{\Gamma}(\mathbf{X}_t, t) = \boldsymbol{\sigma}(\mathbf{X}_t, t)\boldsymbol{\sigma}'(\mathbf{X}_t, t) = (\gamma_{i,j}(\mathbf{X}_t, t))_{i,j=1,2,\ldots,k}$ then we have:

$$\lim_{t \downarrow s} \frac{E\big[X_t^{(i)} - X_s^{(i)} | X_s^{(i)}\big]}{t - s} = \mu_i(\mathbf{X}_t, t), \quad i = 1, 2, \ldots, k \tag{2}$$

and

$$\lim_{t \downarrow s} \frac{E\big[(X_t^{(i)} - X_s^{(i)})(X_t^{(j)} - X_s^{(j)}) | X_s^{(i)}\big]}{t - s} = \gamma_{i,j}(\mathbf{X}_t, t), \quad i, j = 1, 2, \ldots, k \tag{3}$$

where $\gamma_{i,j}(\mathbf{X}_t, t) = \sum_{n=1}^{k} \sigma_{i,n}(\mathbf{X}_t, t)\sigma_{j,n}(\mathbf{X}_t, t)$.

A pivotal quantity in the analysis of diffusion processes is the transition probability density function. The transitional density, $f(\mathbf{X}_t|\mathbf{X}_s)$, of the process moving from state $\mathbf{X}_s$ at time s to state $\mathbf{X}_t$ at time t, is given by the Kolmogorov forward equation (Aït-Sahalia *et al.* 2008):

$$\frac{\partial f(\mathbf{X}_t|\mathbf{X}_s)}{\partial t} = -\sum_{i=1}^{k} \frac{\partial}{\partial X^i}\big[\mu_i(\mathbf{X}_t, t) f(\mathbf{X}_t|\mathbf{X}_s)\big] + \frac{1}{2}\sum_{i=1}^{k}\sum_{j=1}^{k} \frac{\partial^2}{\partial X^i \partial X^j}\big[\gamma_{i,j}(\mathbf{X}_t, t) f(\mathbf{X}_t|\mathbf{X}_s)\big], \tag{4}$$

with the initial condition – that is the transitional density when $t \downarrow s$ – given by the multivariate Dirac delta function $f(\mathbf{x}|\mathbf{X}_s) = \delta(\mathbf{x} - \mathbf{X}_s)$ where

$$\delta(\mathbf{y}) = \begin{cases} \infty & \text{if } \mathbf{y} = \mathbf{0}, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

The intuition behind eqn 4 is that, the transition density at time s starts from an infinite point mass at the initial condition $\mathbf{X}_s$ (since this point is occupied with certainty) and starts propagating mass over the state space as time increases. The behaviour of this probability flow is dictated by the drift and diffusion differential terms on the right hand side of eqn 4. When the drift and/or diffusion terms are time dependent the probability current in the state space may increase, contract or oscillate with time. Figure 1 illustrates the evolution of the transitional density of a time-inhomogeneous scalar diffusion model, whereby both the drift and diffusion terms are sinusoidal. The solution of eqn 4 thus gives the probabilistic evolution of eqn 1 and forms the starting point of probability based analysis of a given diffusion process.

Although partial differential equations (PDEs) with singular initial conditions such as eqn 4 present various difficulties from a computational perspective, the notion of a continuously evolving probability density function means that diffusion models can capture the dynamics of continuously evolving real-world phenomena in a very natural way. Although analytical solutions to eqn 4 can rarely be found, excellent methods for calculating analytical approximations of the transition density do exist. For example, Aït-Sahalia *et al.* (2008) derive accurate short-horizon approximations to eqn 4 based on a Hermite series that can accommodate non-linearities in the drift and diffusion of the model. Huang (2011) applies Wagner-Platen expansions (see Platen (1999)) to the conditional moments of a target diffusion which can then be plugged into a surrogate density in order to yield a short-transition horizon approximation to the transition density. In order to push beyond the threshold of short-horizon approximations one has to resort to numerical methods. However, although numerical schemes such as the Crank-Nicolson method (Crank and Nicolson 1996) or the method of lines (MOL) (Hamdi, Schiesser, and Griffiths 2007) can be used to solve eqn 4 directly, the computational burden of such schemes may magnify significantly in the present context. This follows since calculating quantities of interest such as the likelihood for a discretely observed trajectory relies on numerous evaluations of the transitional density, possibly over varying time-scales. In combination these methods produce desirable results for a wide variety of non-linear diffusions, however, the aim of the present paper is to develop software that can handle inference at sample resolutions that may be too sparse for analytical short-horizon expansions whilst still remaining computationally feasible in

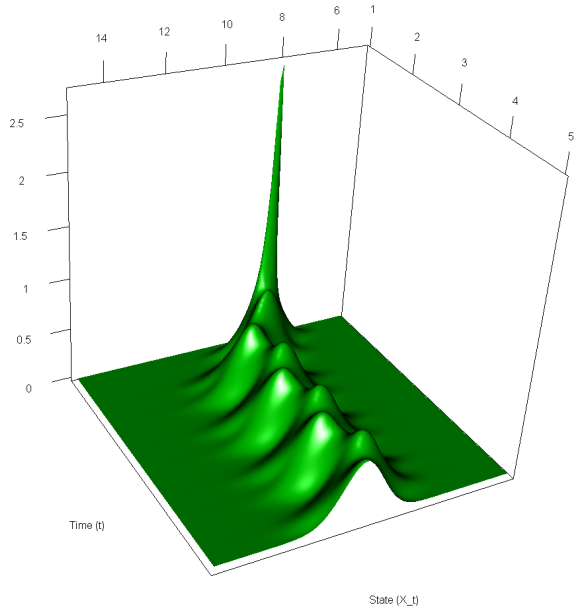

Figure 1: The approximate transitional density of a scalar diffusion process with drift $\mu(X_t, t) = 2(10+\sin(2\pi(t-0.5))-X_t)$ and diffusion $\sigma(X_t, t) = \sqrt{0.25(1+0.75\sin(4\pi t))X_t}$ with initial value $X_1 = 8$. This surface was generated using the `GQD.density()` function in the **DiffusionRgqd** package.

a non-parallel computing environment. As such we adopt the cumulant truncation procedure developed by Varughese (2013), whereby the transition density can be approximated accurately and efficiently over arbitrarily large transition horizons for a suitably general class of non-linear diffusion models.

For purposes of inference we shall assume that the drift and diffusion terms - and consequently the transitional density - are dependent on a vector of parameters, $\boldsymbol{\theta}$. When a finite number of data points of the continuous process $\mathbf{X}_t$ is observed at discrete time epochs $\{t_1, t_2, \ldots, t_N\}$, resulting in the set of observations $\{X_{t_i} : i = 1, \ldots, N\}$, the Markov property of the diffusion X_t can be used in order to represent the likelihood as a product of the transitional densities of the process (Aït-Sahalia *et al.* 2008):

$$L(\boldsymbol{\theta}|\{X_{t_i} : i = 1, \ldots, N\}) \propto \prod_{i=1}^{N-1} f(\mathbf{X}_{t_{i+1}}|\mathbf{X}_{t_i}). \tag{6}$$

Thus, by approximating the transition density for a given diffusion model and plugging it into eqn 6, we may estimate the parameter vector $\boldsymbol{\theta}$ using standard likelihood-based inference techniques.

Although most of the literature on diffusion processes are concerned with inferring dynamics from discretely observed trajectories, the application of diffusion models extend far beyond formulating a compact description of the dynamics of a given process. Often the objective of an analysis is to gain insight into the behaviour of events contingent on the trajectory of the process rather than the trajectory of the process itself. In a predictive context, one may be interested in the probability of a particular event occurring by a given time in the future. As such, a quantity that has been explored at length throughout the history of diffusion processes is the distribution of the time elapsed until the diffusion crosses a predefined threshold. The distribution of the

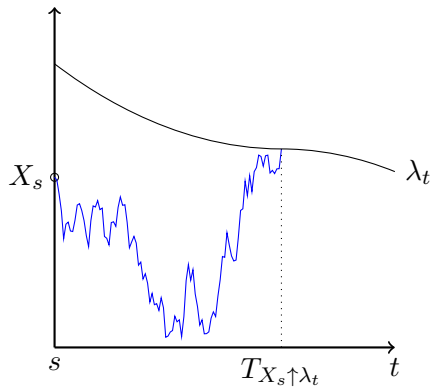

Figure 2: First passage time $T_{X_s\uparrow\lambda_t}$ of a scalar diffusion (solid blue), starting in state X_s, to crossing a time-varying barrier (solid black).

first passage time

$$T_{X_s\uparrow\lambda_t} = \inf\{t > s : X_t > \lambda_t\} \tag{7}$$

of a scalar diffusion process X_t crossing a fixed barrier λ_t is given by the first kind Volterra equation

$$f(\lambda_t|X_s) = \int_s^t f(\lambda_t|\lambda_u)g(\lambda_u|X_s)du, \tag{8}$$

where $g(\lambda_t|X_s)$ denotes the first passage time density evaluated at time t. Figure 2 illustrates the problem graphically.

Again, difficulties arise when attempting to solve first passage time problems analytically. It is clear from eqn 8 that these difficulties stem from both the intractability of the transition density $f(X_t|X_s)$ and the encapsulating integral equation. Even when the transition density is available analytically, solutions to eqn 8 are not guaranteed to exist. As such, the analysis of first passage time problems has been limited to problems with features that aid the calculation of analytical solutions. Consequently, when first passage time densities are used in practice, researchers often have to resort to using diffusions with analytically tractable first passage time densities. In such cases both the dynamics of the process and the shape of the barrier are dictated *a priori* to conducting the analysis. However, by combining theoretical modifications of eqn 8 and numerical techniques for the evaluation of the resulting equation with the transitional density approximation under the cumulant truncation procedure, we may calculate first passage time densities for a class of non-linear diffusions transiting through time dependent barriers.

3. Generalized quadratic diffusions

A common problem that arises when developing mathematical software is designing a mechanism for translating the abstract lexical structures of mathematics into usable syntax for the programming environment. Traversing this language barrier has become a science in itself, resulting in numerous schools of thought on what constitutes optimal interface mechanics. Indeed, in statistical fields such as generalized linear modelling, a very natural structural interface has evolved in the underlying mathematics that has been emulated in software design, with models being defined syntactically in much the same way as the written language. Although research papers on diffusion processes usually reiterate some mathematical grammar like the ubiquitous shorthand for SDEs (see for example eqn 1), a unified framework for specifying a diffusion model in the context of inference is less established. This is due in part to the diversity of methods and the classes of diffusions to which they apply. These classes are often defined by

precluding assumptions such as time-homogeneity and/or reducibility. In the absence of such universally accepted 'language' we have adopted a simple design suitable for the methodology we have chosen whereby the analysis is constrained to diffusions that have, at most, second order polynomial terms in the drift and diffusion coefficients of the model process. We term these the generalized quadratic diffusions (GQDs) which are characterized by the SDEs

$$dX_t = [g_0(t) + g_1(t)X_t + g_2(t)X_t^2]dt + \sqrt{q_0(t) + q_1(t)X_t + q_2(t)X_t^2}dB_t \tag{9}$$

and

$$d\begin{bmatrix} X_t \\ Y_t \end{bmatrix} = \begin{bmatrix} \sum_{i+j\leq 2} a_{ij}(t)X_t^i Y_t^j \\ \sum_{i+j\leq 2} b_{ij}(t)X_t^i Y_t^j \end{bmatrix} dt + \begin{bmatrix} \sigma_{11}(X_t, Y_t, t) & \sigma_{12}(X_t, Y_t, t) \\ \sigma_{21}(X_t, Y_t, t) & \sigma_{22}(X_t, Y_t, t) \end{bmatrix} d\begin{bmatrix} B_t^1 \\ B_t^2 \end{bmatrix} \tag{10}$$

with

$$\boldsymbol{\sigma}(X_t, Y_t, t)\boldsymbol{\sigma}'(X_t, Y_t, t) = \begin{bmatrix} \sum_{i+j\leq 2} c_{ij}(t)X_t^i Y_t^j & \sum_{i+j\leq 2} d_{ij}(t)X_t^i Y_t^j \\ \sum_{i+j\leq 2} e_{ij}(t)X_t^i Y_t^j & \sum_{i+j\leq 2} f_{ij}(t)X_t^i Y_t^j \end{bmatrix}, \tag{11}$$

for the scalar and bivariate case respectively. For both scalar and bivariate GQDs the indices of the coefficients are formulated to reflect powers of the diffusion process in the model, thus providing a grammatical link between the coefficients and variables contained in the model. Within this framework, a model can thus be specified simply by including the desired coefficients of the relevant GQD. For example a time-inhomogeneous stochastic volatility model with SDE

$$d\begin{bmatrix} X_t \\ Y_t \end{bmatrix} = \begin{bmatrix} \theta_1(\theta_2 - X_t) \\ \theta_4(\theta_5 + \theta_6 \sin(2\pi t) - Y_t) \end{bmatrix} dt + \begin{bmatrix} \theta_3^2 Y_t & 0 \\ 0 & \theta_7^2 Y_t \end{bmatrix}^{1/2} d\begin{bmatrix} B_t^1 \\ B_t^2 \end{bmatrix}, \tag{12}$$

can easily be be identified within the GQD framework by matching the coefficients of terms in the target model to the those in the general model:

$$\begin{aligned} a_{00}(t) &= \theta_1\theta_2 \\ a_{10}(t) &= -\theta_1 \\ b_{00}(t) &= \theta_4(\theta_5 + \theta_6 \sin(2\pi t)) \\ b_{01}(t) &= -\theta_4 \\ c_{01}(t) &= \theta_3^2 \\ f_{01}(t) &= \theta_7^2. \end{aligned} \tag{13}$$

In R, the lexical structure of equations 9 and 10 can then easily be replicated by defining functions with names that reflect those of the coefficients of the target model. For example, using coefficients in eqn 13:

```
# Give the parameter vector some values.
 theta  <- c(1,5,1,0.2,1,1,0.2)

# Define the model
 a00 <- function(t){theta[1]*theta[2]}
 a10 <- function(t){-theta[1]}
 c01 <- function(t){theta[3]*theta[3]}

 b00 <- function(t){theta[4]*(theta[5]+theta[6]*sin(2*pi*t))}
 b01 <- function(t){-theta[4]}
 f01 <- function(t){theta[7]*theta[7]}

# Now call some function from the DiffusionRgqd package.
# BiGQD.density(Xs, Ys, Xt, Yt, s, t, delt)
```

By replicating this in the syntax of the **DiffusionRgqd** package one can easily define and distinguish between various models in a single workspace by using the subscripts of the coefficients as visual cues as opposed to having to identify coefficients visually in a single long expression for each drift and diffusion term. Indeed, given that we have allowed the models to have arbitrary time dependences, such expressions would most likely run over multiple lines making them even more illegible. Interface considerations aside, placing constraints on the spatial complexity of the models circumvents a number of mathematical subtleties that arise when attempting to approximate the transitional densities of models with higher order non-linear drift and diffusion terms. The generalized quadratic framework thus enables us to set up a modelling 'sandbox' wherein robust and accurate numerical approximations of the transition density can be calculated whilst providing an intuitive interface for the software environment.

4. Approximating the transitional density of a GQD

Varughese (2013) introduces a computationally efficient method to approximate the transitional density of a diffusion process. The scheme aims to encapsulate information about the trajectory of the transitional density in a finite system of ordinary differential equations that govern the evolution of the cumulants of the process as opposed to dealing directly with the Kolmogorov equation directly. Assuming that a sufficient amount of information is contained within these statistics we may subsequently obtain accurate approximations of the transitional density by way of a surrogate density. We begin by outlining the scheme for scalar diffusions and then expand to the bivariate case.

4.1. Deriving cumulant equations for GQDs

For a scalar diffusion process with SDE

$$dX_t = \mu(X_t, t)dt + \sigma(X_t, t)dB_t, \tag{14}$$

let $M(X_t, t) = E[\exp(\alpha X_t)]$ denote the moment generating function (MGF) of X_t. Then it can be shown that $M(x, t)$ satisfies the partial differential equation:

$$\frac{\partial}{\partial t}M(\alpha, t) = \alpha\mu\left(\frac{\partial}{\partial\alpha}, t\right)M(\alpha, t) + \frac{1}{2}\alpha^2\sigma^2\left(\frac{\partial}{\partial\alpha}, t\right)M(\alpha, t), \tag{15}$$

where $\mu(\frac{\partial}{\partial\alpha}, t)$ and $\sigma^2(\frac{\partial}{\partial\alpha}, t)$ are differential operators on $M(\alpha, t)$. That is, when $\mu(X_t, t)$ and $\sigma^2(X_t, t)$ contain integer powers of X_t we may find a PDE for the MGF in terms of derivatives with respect to α. For example, let $\mu(x, t) = A + Bx + Cx^2$ and $\sigma^2(x, t) = D^2$ then:

$$\frac{\partial}{\partial t}M(\alpha, t) = \alpha\left[A + B\frac{\partial}{\partial\alpha} + C\frac{\partial^2}{\partial\alpha^2}\right]M(\alpha, t) + \frac{1}{2}\alpha^2D^2M(\alpha, t). \tag{16}$$

Now, let $u_j(t) = E(X_t^j)$ and

$$M(\alpha, t) = \sum_{j=0}^{\infty}\frac{\alpha^j u_j(t)}{j!}. \tag{17}$$

Then we may easily derive a system of ODEs for the non-central moments of a diffusion process by plugging eqn 17 into eqn 16 and equating the α coefficients on the LHS and RHS of the resulting equations. In our example problem we would arrive at the system:

$$\frac{\partial}{\partial t}u_j(t) = j(A + Bu_j(t) + Cu_{j+1}(t)) + D^2\frac{j(j-1)}{2}u_{j-2}(t)\mathbb{1}_{j\geq 2}, \quad \forall j \geq 1; \tag{18}$$

where $\mathbb{1}_A = \text{Ind}(A)$. Note however that eqn 18 implies that an infinite dimensional system of non-central moments are required in order for the system to be determinate. The dimensionality

of the system is caused by the inclusion of the quadratic term in the drift. That is, in order to evaluate the j-th non-central moment, we require knowledge of the trajectory of the $(j+1)$-th and subsequent non-central moments. Thus, when we look at a finite set, say the first m, non-central moments of a non-linear polynomial diffusion we observe a substantial amount of *leakage* of information into the higher-order moments. In order to deal with this leakage we may alternatively consider the behaviour of the cumulants of such diffusion processes. Let

$$K(\alpha, t) = \sum_{j=1}^{\infty} \frac{\alpha^j \kappa_j(t)}{j!} \tag{19}$$

be the CGF where $\kappa_j(t)$ denotes the j-th cumulant of the process at time t. The cumulants $\kappa_j(t)$ are then defined in relation to the non-central moments by:

$$\sum_{j=1}^{\infty} \frac{\alpha^j \kappa_j(t)}{j!} = \log\Big(1 + \sum_{j=1}^{\infty} \frac{\alpha^j u_j(t)}{j!}\Big). \tag{20}$$

By making use of the relationship between the MGF and the cumulant generating function (CGF)

$$\frac{1}{M(\alpha, t)} \frac{\partial M(\alpha, t)}{\partial v} = \frac{\partial K(\alpha, t)}{\partial v} \tag{21}$$

and the recursive relation

$$\frac{1}{M}\left(\frac{\partial^{(r+1)} M}{\partial v^{(r+1)}}\right) = \left[\frac{\partial K}{\partial v}\right]\left[\frac{1}{M}\left(\frac{\partial^{(r)} M}{\partial v^{(r)}}\right)\right] + \frac{\partial}{\partial v}\left[\frac{1}{M}\left(\frac{\partial^{(r)} M}{\partial v^{(r)}}\right)\right] \tag{22}$$

for $r = 1, 2, \ldots$, we may derive a similar system of ODEs for the cumulants of a diffusion process. That is by dividing both sides of eqn 15 by $M(\alpha, t)$ we may use eqn 22 in order to derive a PDE for the cumulant generating function. Thus for scalar GQDs with drift and diffusion coefficients $\{g_i(t) : i = 0, 1, 2\}$ and $\{q_i(t) : i = 0, 1, 2\}$ respectively, we have:

$$\frac{\partial}{\partial t} K(\alpha, t) = \alpha \sum_{i=0}^{2} g_i(t) \frac{1}{M}\left(\frac{\partial^{(i)} M}{\partial u^{(i)}}\right) + \frac{1}{2}\alpha^2 \sum_{i=0}^{2} q_i(t) \frac{1}{M}\left(\frac{\partial^{(i)} M}{\partial u^{(i)}}\right), \tag{23}$$

where the terms $\frac{1}{M}(\frac{\partial^{(i)} M}{\partial u^{(i)}})$ may be expressed in terms of $K(\alpha, t)$ via eqns 21 and 22.
Then by replacing $K(\alpha, t)$ by the m-th order truncated series

$$K^{(m)}(\alpha, t) \approx \sum_{j=1}^{m} \frac{\alpha^j \kappa_j(t)}{j!}, \tag{24}$$

plugging the relevant derivatives of eqn 24 into eqn 23 and equating coefficients on the LHS and RHS, one can derive an m-dimensional system ODEs that describe the approximate evolution of the cumulants $\{\kappa_j(t) : j = 1, ..., m; t > s\}$ over time. For scalar GQDs we may derive a closed form expression for the system of ODEs that govern the evolution of the cumulants. For a m-th order truncation of a scalar GQD the ODEs that govern the evolution of the truncated

cumulants is given by the system:

$$\begin{aligned}\frac{\partial}{\partial t}\kappa_j(t) =& g_0(t)\mathbb{1}_{j=1} \\ &+ g_1(t)j\kappa_j(t) \\ &+ g_2(t)\left(j\kappa_{j+1}(t) + \sum_{r=1}^{j} r\binom{j}{r}\kappa_r(t)\kappa_{j-r+1}(t)\right) \\ &+ q_0(t)\mathbb{1}_{j=2} \\ &+ q_1(t)\frac{j(j-1)}{2}\kappa_{j-1}(t)\mathbb{1}_{j\geq 2} \\ &+ q_2(t)\left(\frac{j(j-1)}{2}\kappa_j(t) + \frac{j}{2}\sum_{r=1}^{j-1} r\binom{j-1}{r}\kappa_r(t)\kappa_{j-1-r+1}(t)\right)\mathbb{1}_{j\geq 2},\end{aligned} \tag{25}$$

with initial conditions $\kappa_1(s) = X_s$ and $\kappa_j(s) = 0\ \forall j > 1$. The initial conditions follow from the fact that, at time s, the state X_s is occupied with certainty, hence the first moment of the process reflects this position and the zero higher order moments reflect the absolute certainty.

The advantage of this route is twofold: Firstly, we can manage the 'leakage' that occurs in the cumulants by assuming that cumulants above a certain order negligible and subsequently set them equal to zero - an assumption that is valid for diffusions of the quadratic class assumed in the present paper. Secondly, the non-central moments usually assume values that are orders of magnitude larger than the cumulants, which may lead to instabilities under numerical evaluation.

Consider now a bivariate diffusion process with SDE:

$$d\begin{bmatrix}X_t \\ Y_t\end{bmatrix} = \begin{bmatrix}\mu_1(X_t, Y_t, t) \\ \mu_2(X_t, Y_t, t)\end{bmatrix} dt + \begin{bmatrix}\sigma_{1,1}(X_t, Y_t, t) & \sigma_{1,2}(X_t, Y_t, t) \\ \sigma_{2,1}(X_t, Y_t, t) & \sigma_{2,2}(X_t, Y_t, t)\end{bmatrix} d\begin{bmatrix}B_t^1 \\ B_t^2\end{bmatrix}. \tag{26}$$

By imposing the drift and diffusion structure of eqn 10 the corresponding PDE for the bivariate MGF, $M(X_t, Y_t, t) = E[\exp(\alpha X_t + \beta Y_t)]$, becomes:

$$\begin{aligned}\frac{\partial}{\partial t}M_2(\alpha,\beta,t) =& \left[\alpha\sum_{i+j\leq 2} a_{ij}(t)\frac{\partial^i}{\partial\alpha^i}\frac{\partial^j}{\partial\beta^j} + \beta\sum_{i+j\leq 2} b_{ij}(t)\frac{\partial^i}{\partial\alpha^i}\frac{\partial^j}{\partial\beta^j}\right]M_2(\alpha,\beta,t) \\ &+ \frac{1}{2}\left[\alpha^2\sum_{i+j\leq 2} c_{ij}(t)\frac{\partial^i}{\partial\alpha^i}\frac{\partial^j}{\partial\beta^j} + \alpha\beta\sum_{i+j\leq 2} d_{ij}(t)\frac{\partial^i}{\partial\alpha^i}\frac{\partial^j}{\partial\beta^j}\right. \\ &\left. + \alpha\beta\sum_{i+j\leq 2} e_{ij}(t)\frac{\partial^i}{\partial\alpha^i}\frac{\partial^j}{\partial\beta^j} + \beta^2\sum_{i+j\leq 2} f_{ij}(t)\frac{\partial^i}{\partial\alpha^i}\frac{\partial^j}{\partial\beta^j}\right]M_2(\alpha,\beta,t).\end{aligned} \tag{27}$$

By applying the cumulant truncation procedure to the bivariate GQD we can derive in similar fashion a general expression for the system of ODEs that govern the evolution of the cumulants, truncated up to an arbitrary order m. That is by assuming that $\kappa_{ij}(t) = 0 \quad \forall \quad i + j > m$ the truncated CGF is given by:

$$K_2^{(m)}(\alpha,\beta,t) = \sum_{i+j\leq m}\frac{\alpha^i\beta^j}{i!j!}\kappa_{ij}(t). \tag{28}$$

Now define the retention operator, J_{ij}^{vw} that operates on α and β:

$$J_{ij}^{vw}\alpha^r\beta^s = \mathbb{1}_{r+v=i,s+w=j}. \tag{29}$$

Furthermore, let

$$L_{xy} = \sum_{i=0}^{m}\sum_{j=0}^{m}\alpha^{i-x}\beta^{j-y}i^{\underline{x}}j^{\underline{y}}/(i!j!)\kappa_{ij}(t) \tag{30}$$

where $i^{\underline{x}} = i(i-1)\dots(i-x+1)$ and $j^{\underline{y}} = j(j-1)\dots(j-y+1)$, $\Theta = \{a, b, c, d, e, f\}'$ and define a set of multi-indexes $\lambda = \{(00), (10), (20), (01), (02), (11)\}'$. Let the subscript q denote the q-th entry of the vectors λ and Θ. Then the system of ODEs that govern the evolution of the cumulants of eqn 10 is given by:

$$\begin{aligned}\frac{\partial}{\partial t}\kappa_{ij}(t) = \sum_q J_{ij}^{\lambda_q}[(\Theta_q)_{00}\mathbb{1}_{\lambda_q} + (\Theta_q)_{10}L_{10} + (\Theta_q)_{01}L_{01} + (\Theta_q)_{02}[L_{11} - L_{10}L_{01}] \\ + (\Theta_q)_{20}[L_{20} - L_{10}L_{10}] + (\Theta_q)_{02}[L_{02} - L_{01}L_{01}],\end{aligned} \tag{31}$$

with initial conditions $\kappa_{10}(s) = X_s$, $\kappa_{01}(s) = Y_s$ and $\kappa_{ij}(s) = 0$ otherwise. As in the scalar case, the initial conditions for the cumulant equations reflects the fact that the position of the process at the starting coordinate (X_s, Y_s) is known. Consequently, all second order and higher cumulants have initial conditions equal to zero.

4.2. Surrogate densities

By applying the cumulant truncation procedure to SDEs of the generalized quadratic form we may accurately approximate the evolution of the cumulants over arbitrarily large time horizons. However, in order to approximate the transitional density at any given point in the support of the process, we need to carry these cumulants into a probability density function. If the first two or three cumulants contain sufficient information about the probabilistic evolution of the process, standard densities such as the Normal or Gamma distribution may be used. However, for diffusion processes it often occurs that higher order cumulants are indeed informational. As such we employ a number of density approximations suitable for carrying higher order cumulants into the approximate transitional density. One such approach is to use the saddlepoint approximation. For scalar GQDs under truncation order m, the univariate saddlepoint approximation is given by:

$$\tilde{f}_{sdl}^{(m)}(x_t|X_s) = \frac{1}{\sqrt{2\pi\frac{\partial^2 K^{(m)}}{\partial\alpha^2}(\alpha_0, t)}}\exp\left(\frac{\partial K^{(m)}}{\partial\alpha}(\alpha_0, t) - \alpha_0 x_t\right), \tag{32}$$

where α_0 solves

$$\frac{\partial^2 K^{(m)}}{\partial\alpha^2}(\alpha, t) = x_t. \tag{33}$$

As an alternative to the saddlepoint approximation we may employ suitable members of the multimodal Pearson system developed by Cobb, Koppstein, and Chen (1983). The system consists of a set of kernel functions that depend on a predetermined number of non-central moments. These kernels serve to extend four principal classes of density, namely the Normal, Gamma, Inverse Gamma and Beta distributions. In addition, each distribution class is characterized by a system of equations which relate the non-central moments to the parameters of each respective kernel. Thus, given a finite set of non-central moments we may evaluate any density nested within the system by calculating the kernel parameters and plugging them into a the appropriate kernel expression. In the present context we may thus, in similar fashion to the saddlepoint approximation, calculate the transitional density based on the trajectories of the non-central moments of a given diffusion. Formally, given an even number of m non-central moments $\{u_i(t) : i = 1 : m\}$, we re-iterate the expressions of Cobb *et al.* (1983) under the notation of the present paper:

Let

$$\mathbf{A} = \begin{pmatrix} 1 & u_1(t) & \dots & u_{m/2}(t) \\ u_1(t) & u_2(t) & \dots & u_{m/2+1}(t) \\ \vdots & \vdots & \ddots & \vdots \\ u_{m/2}(t) & u_{m/2+1}(t) & \dots & u_m(t) \end{pmatrix}, \tag{34}$$

and $\boldsymbol{\beta} = \{\beta_1, \beta_2, \ldots, \beta_{m/2+1}\}$.
Then the multimodal Normal class is given by:

$$\tilde{f}_N^{(m)}(x|X_s) \propto \exp\Big(-\sum_{i=1}^{m/2+1} \beta_i x^i/i\Big) \quad \forall x \in (-\infty, \infty), \tag{35}$$

where $\boldsymbol{\beta} = \mathbf{A}\mathbf{v}$ and $\mathbf{v} = \{v_i = (i-1)u_{i-2}(t) : i = 1, ..., m/2+1\}'$.
The multimodal Gamma class is given by

$$\tilde{f}_G^{(m)}(x|X_s) \propto x^{-\beta_1} \exp\Big(-\sum_{i=2}^{m/2+1} \beta_i x^{i-1}/(i-1)\Big) \quad \forall x \in [0, \infty), \tag{36}$$

where $\boldsymbol{\beta} = \mathbf{A}\mathbf{v}$ and $\mathbf{v} = \{v_i = iu_{i-1}(t) : i = 1, ..., m/2+1\}'$.
The multimodal Inverse Gamma class is given by

$$\tilde{f}_{IG}^{(m)}(x|X_s) \propto x^{-\beta_2} \exp\Big(-\beta_1/x - \sum_{i=3}^{m/2+1} \beta_i x^{i-2}/(i-2)\Big) \quad \forall x \in [0, \infty), \tag{37}$$

where $\boldsymbol{\beta} = \mathbf{A}\mathbf{v}$ and $\mathbf{v} = \{v_i = (i+1)u_i(t) : i = 1, ..., m/2+1\}'$.
The multimodal Beta class is given by

$$\tilde{f}_B^{(m)}(x|X_s) \propto x^{-\beta_1}(1-x)^{-\sum_{i=1}^{m/2+1}\beta_i} \exp\Big(-\sum_{i=1}^{m/2-1}\Big(\sum_{j=i+1}^{m/2} \beta_j\Big)x^i/i\Big) \quad \forall x \in [0, 1], \tag{38}$$

where $\boldsymbol{\beta} = \mathbf{A}\mathbf{v}$ and $\mathbf{v} = \{v_i = iu_{i-1}(t) - (i+1)u_i(t) : i = 1, ..., m/2+1\}'$. By applying the cumulant truncation procedure we may recover the approximate trajectories of the non-central moments through the well known relation:

$$\begin{aligned} u_1(t) &= \kappa_1(t), \\ u_i(t) &= \kappa_i(t) + \sum_{j=1}^{i-1} \binom{i-1}{j-1} \kappa_j(t) u_{i-j}(t) \quad \forall i = 2, \ldots, m. \end{aligned} \tag{39}$$

Subsequently, by plugging the $u_i(t)$ into any of the above densities we may approximate the transitional density of a scalar GQD.

For bivariate GQDs under an (even) m-th order truncation, the saddlepoint approximation (Renshaw 2000) is given by:

$$\tilde{f}_{sdl}^{(m)}(\{x_t, y_t\}|\{X_s, Y_s\}) = \frac{\exp\big(K_2^{(m)}(\alpha_0, \beta_0) - \alpha_0 x_t - \beta_0 y_t\big)}{2\pi\sqrt{\frac{\partial^2 K_2^{(m)}}{\partial \alpha^2}\frac{\partial^2 K_2^{(m)}}{\partial \beta^2} - \big(\frac{\partial K_2^{(m)}}{\partial \alpha \partial \beta}\big)^2}}, \tag{40}$$

for

$$\begin{aligned} K_2^{(m)}(\alpha, \beta) = {} & \alpha\kappa_{10}(t) + \frac{\alpha^2}{2}\kappa_{20}(t) + \frac{\alpha^3}{6}\kappa_{30}(t) + \frac{\alpha^4}{24}\kappa_{40}(t) + \ldots + \frac{\alpha^m}{m!}\kappa_{m0}(t) \\ & + \beta\kappa_{01}(t) + \frac{\beta^2}{2}\kappa_{02}(t) + \frac{\beta^3}{6}\kappa_{03}(t) + \frac{\beta^4}{24}\kappa_{04}(t) + \ldots + \frac{\alpha^m}{m!}\kappa_{0m}(t) \\ & + \alpha\beta\kappa_{11}(t) + \frac{\alpha^2\beta}{2}\kappa_{21}(t) + \frac{\alpha\beta^2}{2}\kappa_{12}(t) + \ldots + \frac{\alpha^{m/2}\beta^{m/2}}{(m/2)!)^2}\kappa_{((m/2)(m/2)}(t) + \ldots \\ & + \frac{\alpha^{m-1}\beta}{(m-1)!}\kappa_{(m-1)1}(t) + \frac{\alpha\beta^{m-1}}{(m-1)!}\kappa_{1(m-1)}(t), \end{aligned} \tag{41}$$

where α_0 and β_0 solves the system:

$$\begin{aligned}\frac{\partial K_2^{(m)}}{\partial \alpha}(\alpha,\beta) &= x_t,\\ \frac{\partial K_2^{(m)}}{\partial \beta}(\alpha,\beta) &= y_t.\end{aligned} \tag{42}$$

Since eqn 42 requires numerical evaluation, it is possible, in certain circumstances, that numerical instabilities arise when using the bivariate saddlepoint approximation. For this reason we include, as an alternative to the bivariate saddlepoint approximation, the bivariate Edgeworth expansion (Barndorff-Nielsen and Cox 1979): Let $f_N(\{x_t, y_t\}|\{X_s, Y_s\})$ denote the bivariate Normal distribution

$$\begin{aligned}&\tilde{f}_N(\{x_t, y_t\}|\{X_s, Y_s\}) = \\ &\frac{1}{2\pi\sqrt{\kappa_{20}\kappa_{02}(1-\rho_{11}^2)}}\exp\left(-\frac{(x_t-\kappa_{10})^2}{2\kappa_{20}(1-\rho_{11}^2)} + \frac{2\rho_{11}(x_t-\kappa_{10})(y_t-\kappa_{01})}{2\sqrt{\kappa_{20}\kappa_{02}}(1-\rho_{11}^2)} - \frac{(y-\kappa_{01})^2}{2\kappa_{02}(1-\rho_{11}^2)}\right),\end{aligned} \tag{43}$$

$H_i(u)$ the i-th scalar Hermite polynomial (see Andrews, Askey, and Roy (1999))

$$H_i(u) = (-1)^i e^{\frac{u^2}{2}}\frac{\partial^i}{\partial u^i}e^{-\frac{u^2}{2}} \tag{44}$$

with $H_0(u) = 1$ and $\rho_{ij} = \frac{\kappa_{ij}}{\sqrt{\kappa_{20}^i\kappa_{02}^j}}$. Then the fourth-order bivariate Edgeworth expansion for the transitional density is given by:

$$\tilde{f}_{edg}(\{x_t, y_t\}|\{X_s, Y_s\}) = \tilde{f}_N(\{x_t, y_t\}|\{X_s, Y_s\}) \times \left(1 + \frac{1}{6}A(\dot{x}_t, \dot{y}_t) + \frac{1}{24}B(\dot{x}_t, \dot{y}_t) + \frac{1}{72}C(\dot{x}_t, \dot{y}_t)\right) \tag{45}$$

where $\dot{x}_t = (x_t - \kappa_{10})/\sqrt{\kappa_{20}}$, $\dot{y}_t = (y_t - \kappa_{01})/\sqrt{\kappa_{02}}$ and the terms A, B and C are given by:

$$\begin{aligned}A(u,v) &= H_3(u)\rho_{30} + 3H_2(u)H_1(v)\rho_{21} + 3H_1(u)H_2(v)\rho_{12} + H_3(v)\rho_{03},\\ B(u,v) &= H_4(u)\rho_{40} + 4H_3(u)H_1(v)\rho_{31} + 6H_2(u)H_2(v)\rho_{22} + 4H_1(u)H_3(v)\rho_{13} + H_4(v)\rho_{04},\end{aligned} \tag{46}$$

and

$$\begin{aligned}C(u,v) = {} & H_6(u)\rho_{30}^2 + 6H_5(u)H_1(v)\rho_{21}\rho_{30} + 6H_4(u)H_2(v)\rho_{12}\rho_{30} + 2H_3(u)H_3(v)\rho_{03}\rho_{30}\\ & + 9H_4(u)H_2(v)\rho_{21}^2 + 18H_3(u)H_3(v)\rho_{21}\rho_{12} + 6H_2(u)H_4(v)\rho_{21}\rho_{03} + 9H_2(u)H_4(v)\rho_{12}^2\\ & + 6H_1(u)H_5(v)\rho_{12}\rho_{03} + H_1(u)H_6(v)\rho_{03}^2.\end{aligned} \tag{47}$$

We end off with some remarks on the cumulant truncation procedure as applied to GQDs. Due to the inter-dependence between the cumulants according to the systems in eqns 25 and 31 one requires a sufficiently large truncation order (m) in order for the systems to accurately replicate the time-evolution of a given diffusion process' cumulants. That is, unless it happens that the transitional density turns out to be Normal - in which case a second order truncation ($m = 2$) will describe the cumulants exactly – a sufficient number of higher order cumulants need to be included in the cumulant system in order for the cumulant trajectories to be accurate. Fortunately under the GQD framework, truncation orders of $m = 4$ produce sufficiently accurate approximations for use in practical applications for both scalar and bivariate GQDs. Furthermore, we find that the saddlepoint approximations perform well as surrogate densities and prove to be quite reliable. Thus, as a default, we recommend using a 4-th order truncation in conjunction with the corresponding saddlepoint approximation in order to approximate the transitional density of a GQD. For scalar GQDs where the saddlepoint approximation breaks down, for

example when the dynamics of the diffusion dictate that the diffusion has finite support, one may employ an appropriate member of the multimodal Pearson system of densities or Normal distribution. Likewise should the saddlepoint approximation break down for whatever reason in the bivariate case, we provide the option of selecting either the bivariate Edgeworth expansion or bivariate Normal distribution. Although we fix the maximum truncation order for bivariate GQDs to $m = 4$, the **DiffusionRgqd** package supports cumulant and density truncations for even orders up to and including $m = 8$ for scalar GQDs.

4.3. Approximating the first passage time density of a scalar GQD

It is well known that the distribution for the first passage time of a scalar diffusion process starting in X_s transiting through a constant threshold $\lambda_t = \phi$ is given by the solution to the Volterra integral equation of the first kind:

$$f(\lambda_t|X_s) = \int_s^t f(\lambda_t|\lambda_u)g(\lambda_u|X_s)du. \tag{48}$$

where $g(\lambda_t|X_s)$ denotes the first passage time density at time t.

Unfortunately, even when $f(X_t|X_s)$ is known, eqn 48 may not be analytically tractable. By applying a Euler approximation to eqn 48 one may derive an iterative updating equation as follows: Split the time domain into $N + 1$ equispaced time nodes $t_0 = s, t_1 = s + \Delta, \ldots, t_N = s + N\Delta$ and set $g(\lambda_{t_1}|X_{t_0}) = f(\lambda_{t_1}|X_{t_0})$. Then for $i = 2, \ldots, N$ evaluate:

$$g(\lambda_{t_i}|X_{t_0}) = \frac{f(\lambda_{t_i}|X_s) - \sum_{k=0}^{i} f(\lambda_{t_i}|\lambda_{t_k})g(\lambda_{t_k}|X_{t_0})\Delta}{f(\lambda_{t_i}|\lambda_{t_{i-1}})\Delta}. \tag{49}$$

However, due to the singular integrand of eqn 48 at s (Ricciardi and Sacerdote 1979), numerical solutions such as eqn 49 may be subject to systematic error. Although these singularities occur at an infinitesimal scale, the effects are not inconsequential on a finite scale (i.e. under discritisation of the time domain). In order to deal with this, Buonocore, Nobile, and Ricciardi (1987) developed an alternative integral equation wherein the kernel of the integral equation can be modified in order to remove the singularities prior to the calculation of the first passage time density. The second kind Volterra integral equation for the first passage time density under Buonocore *et al.* (1987) is given by:

$$g(\lambda_t|X_s) = 2\psi(\lambda_t|X_s) - 2\int_s^t g(\lambda_v|X_s)\psi(\lambda_t|\lambda_v)dv \tag{50}$$

for $X_s < B_s$, where

$$\psi(x_t|y_t) = \frac{\partial}{\partial t}F(x_t|y_t) + k(t)f(x_t|y_s) + r(t)[1 - F(x_t|y_s)]. \tag{51}$$

and $k(t)$ and $r(t)$ are chosen in such a way so as to ensure 'regularity' of the kernel. It is important to make the distinction that this revised integral equation not only deals with the irregularity of the kernel but also permits the barrier λ_t to be time dependent (note the assumption that $\lambda_t = \phi$ for all t in eqn 48). Although the revised equation rather elegantly circumvents the drawbacks of eqn 8, it does introduce some minor mathematical complications. Note that $\psi(x_t|y_s)$ depends on both the transition p.d.f. and c.d.f. of X_t. In order to make the revised equation usable within the practical confines of a software package, we circumvent both integration of the p.d.f., i.e. remove $F(x_t|y_s)$, and find a systematic way of choosing $r(t)$ and $k(t)$. Fortunately the literature provides us with a solution to both problems: Note that eqn 51 can be related to the probability current

$$c(x_t|y_s) = [\mu(X_t, t) - \sigma(x_t, t)\sigma'(x_t, t)]f(x_t|y_s) - \frac{1}{2}\sigma^2(x_t, t)f'(x_t|y_s) \tag{52}$$

where $\sigma'(x_t,t) = \frac{\partial}{\partial u}\sigma(u,t)|_{u=x_t}$ and $f'(x_t,t) = \frac{\partial}{\partial u}f(u,t)|_{u=x_t}$. This expression may then be simplified further by observing that

$$\frac{\partial}{\partial t}f(x_t|y_s) = -\frac{\partial}{\partial x_t}c(x_t|y_s) \tag{53}$$

and thus

$$\frac{\partial}{\partial t}F(x_t|y_s) = -c(x_t|y_s). \tag{54}$$

Consequently one may simplify eqn 51 by removing the derivative of the cumulative density function:

$$\psi(x_t|y_s) = -c(x_t|y_s) + k(t)f(x_t|y_s) + r(t)[1 - F(x_t|y_s)]. \tag{55}$$

Then, following corollary 1 in Jáimez, Román, and Ruiz (1995): Set

$$r(t) = 0 \tag{56}$$

and

$$k(t) = \frac{1}{2}\left[\mu(\lambda_t,t) - \frac{\partial}{\partial t}\lambda_t - \frac{1}{2}\sigma(\lambda_t,t)\sigma'(\lambda_t,t)\right]. \tag{57}$$

Subsequently eqn 55 may be evaluated in terms of the transition probability density function and its first derivative alone

$$\psi(x_t|x_s) = -\left[\frac{1}{2}\mu(x_t,t) - \frac{1}{2}\frac{\partial}{\partial t}\lambda_t - \frac{3}{4}\sigma(x_t,t)\sigma'(x_t,t)\right]f(x_t|x_s) + \frac{1}{2}\sigma^2(x_t,t)f'(x_t|x_s), \tag{58}$$

thus avoiding the need to evaluate $F(x_t|y_s)$ whilst ensuring regularity of the kernel of eqn 50. Since (50) is still not analytically tractable Buonocore *et al.* (1987) gives the discretised counterpart to the revised second kind Volterra equation in the form of the iterative updating equation:

$$\tilde{g}(\lambda_{t_i}|X_{t_0}) = -2\psi(\lambda_{t_i}|X_{t_0}) + \sum_{k=0}^{i-1} 2\psi(\lambda_{t_i}|\lambda_{t_k})\tilde{g}(\lambda_{t_k}|X_{t_0})\Delta, \tag{59}$$

for $i = 2,\ldots,N$ where again $t_0 = s, t_1 = s+\Delta, \ldots, t_N = s+N\Delta$.

Given the transitional density of a diffusion process, eqn 59 may then be used to accurately approximate the first passage time density of a scalar diffusion transiting through a single barrier. Previously, Varughese and Pienaar (2015) developed an updating equation for evaluating the first passage time density of time-homogeneous diffusion processes. Specifically, by incorporating the cumulant truncation procedure in the evaluation of the transitional densities on which the updating equation relies, it is possible to evaluate first passage times for polynomial diffusion processes. Similarly, by combining eqn 59 with the cumulant truncation procedure we can extend the analysis to time-inhomogeneous non-linear diffusions by plugging the appropriate density approximations into eqn 58 and applying eqn 59. That is let $\tilde{f}_{sdl}^{(m)}(x_t|x_s)$ and $\tilde{f}_{sdl}^{\prime(m)}(x_t|x_s)$ denote the density approximation and its first derivative under the cumulant truncation procedure. Then the revised probability current under an m-th order truncation may be approximated by:

$$\tilde{\psi}^{(m)}(x_t|x_s) = -\left[\frac{1}{2}\mu(x_t,t) - \frac{1}{2}\frac{\partial}{\partial t}\lambda_t - \frac{3}{4}\sigma(x_t,t)\sigma'(x_t,t)\right]\tilde{f}_{sdl}^{(m)}(x_t|x_s) + \frac{1}{2}\sigma^2(x_t,t)\tilde{f}_{sdl}^{\prime(m)}(x_t|x_s). \tag{60}$$

In order to calculate the revised probability current we require evaluation of the derivative of the density approximation $\tilde{f}_{sdl}^{(m)}(x_t,x_s)$. For example, under the saddlepoint approximation this can be achieved as follows: Since α_0 in eqn 32 depends on x_t through eqn 33 it follows that:

$$\frac{\partial}{\partial x_t}\alpha_0 = \left(\sum_{i=2}^{m}\frac{\alpha_0^{i-2}}{(i-2)!}\kappa_i(t)\right)^{-1}. \tag{61}$$

Furthermore, let $\alpha_0' = \frac{\partial}{\partial x_t}\alpha_0$ then the first derivative of the saddlepoint approximation can be verified as:

$$\tilde{f}_{sdl}^{\prime(m)}(x_t|x_s) = \left[-\frac{1}{2\frac{\partial^2 K^{(m)}}{\partial \alpha^2}(\alpha_0,t)}\left(\sum_{i=3}^{m}\frac{\alpha_0^{i-3}\alpha_0'}{(i-3)!}\kappa_i(t)\right) + \left(\sum_{i=3}^{m}\frac{\alpha_0^{i-3}\alpha_0'}{(i-3)!}\kappa_i(t) - \alpha_0' x_t - \alpha_0\right)\right] \times \tilde{f}_{sdl}^{(m)}(x_t|x_s). \tag{62}$$

Thus, using eqn 62 in conjunction with eqns 60 and 59 we can numerically evaluate the first passage time density of scalar GQDs transiting through time dependent barriers.

For the initial version of the **DiffusionRgqd** package, first passage time densities are calculated using $m = 4$ in conjunction with the saddlepoint approximation. Furthermore, first passage time calculations for time-homogeneous and time-inhomogeneous are carried out using separate routines. This is done so as to take advantage of computational redundancies that manifest in the time-homogeneous case, meaning that computations can be carried out with greater efficiency than for time-inhomogeneous first passage time problems.

5. Building computationally optimized solutions in R with C++

Although the cumulant truncation procedure provides a computationally efficient means of approximating the transition density of a non-linear diffusion, the application thereof in the context of likelihood inference may be computationally demanding if not coded efficiently. In general, given N observations of a process, a single evaluation of the likelihood demands $N-1$ approximations of the transitional density. This overhead is magnified when evaluations of the likelihood are made iteratively in MCMC procedures and/or when transition densities are approximated over relatively large time-horizons. For example, when evaluating the likelihood of 500 observations of a bivariate process for 100000 iterations of an MCMC we are required to numerically solve (thus incurring further iteration) a multidimensional, non-linear system of ODEs 499×100000 times - a tedious task even under ideal computing conditions. Thus care needs to be taken when setting up code within interpretive languages such as R.

Given that inference procedures within the **DiffusionRgqd** package will be applied to datasets of varying complexity (i.e. size, time dependence, linearity etc.) we are prompted to maximize the computational efficiency of relevant routines. As the first step in improving computational efficiency we circumvent the interpretive overhead of the R language by making use of the C++ language which may be interfaced with the R environment through the **Rcpp** package. This is further facilitated by the **RcppArmadillo** package (Eddelbuettel and Sanderson 2014) - a linear algebra library written for use with the C++ language in R. By combining the C++ language with the **RcppArmadillo** libraries we can efficiently manipulate large matrices, thus making it possible to vectorise all numerical elements of the cumulant truncation procedure within the C++ language.

A natural consequence of adopting the GQD framework is that various forms of mathematical redundancies surface that may be exploited for further computational gains. As mentioned in Section 4.1, depending on the coefficients that are included or excluded for a given GQD, components of the cumulant truncation procedure may be simplified in order to avoid unnecessary calculations. For example, when a model has redundant higher order cumulants, the dimension of the cumulant system from eqns 25 or 31 may be reduced. Where the analysis permits a second order truncation the Normal distribution may be used in place of the saddlepoint approximation. Furthermore when the data is of homogeneous resolution (equispaced observations) we circumvent the need to keep track of each transition horizon individually. Within the GQD framework we may identify these redundancies prior to conducting the analysis and adapt a given routine accordingly. We emulate this in the GQD package source code by identifying reducible components from coefficients supplied by the user in real time and subsequently *stitch* together an optimal solution from predefined blocks of code.

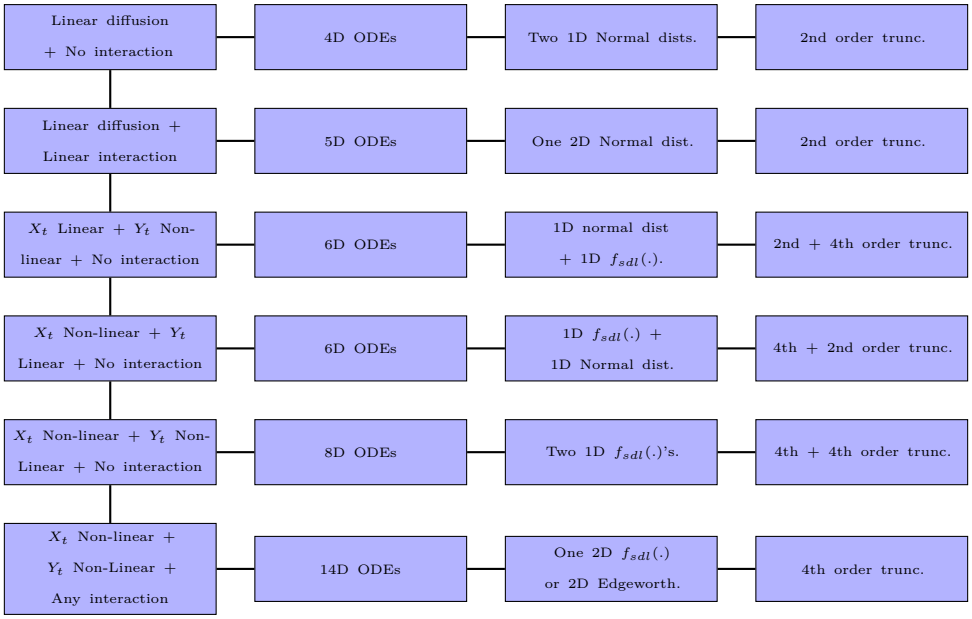

Figure 3: Default solution types for various bivariate GQD: Depending on the coefficients of a given model, the moment equations may become reducible due to redundant dimensions. As such computational efficieny may be improved in certain cases by evaluating a smaller set of moment equations as well as a simpler density approximation. Although the dimensions of the cumulant equations (second column) are always determined by the diffusion, the package does allow one to override the type of density approximation used manually should the need arise.

Consider for example applying the cumulant truncation procedure to the bivariate GQD. By checking which of the 36 coefficients are included in the model specification we can identify a optimum algorithmic solution. When the dimensions of the process are independent one of four outcomes are possible: either both dimensions are normally distributed, the X_t dimension is normally distributed and the Y_t dimension is non-normal, *vice versa* or both are non-normal. Normality of either dimension can be established by checking whether the coefficients of non-linear terms in the drift or diffusion of the relevant dimension are absent. Furthermore since the cross-cumulants of a bivariate density become redundant when no interaction terms are included, the dimensions of the cumulant system resulting from eqn 31 can be reduced to 4,6 or 8 dimensions depending on which of the aforementioned outcomes apply. Finally and perhaps the most important reducible component for independent dimensions is the fact that we may evaluate the surrogate density analytically, as opposed to eqn 40 which requires numerical evaluation of the roots of eqn 42.

Whenever interaction terms are included in the model, one of 2 outcomes are possible: either the model has a bivariate Normal transition density or not, in which case we employ either the bivariate saddlepoint approximation or the bivariate Edgeworth expansion. The corresponding cumulant system is then 5-dimensional if bivariate Normal or 14-dimensional whenever non-linear terms are included. Thus, for a bivariate Normal density we may evaluate the surrogate density analytically whilst the bivariate saddlepoint requires an additional layer of numerical overhead. Figure 3 summarizes how various particular solutions of the cumulant truncation procedure for the bivariate GQD arise.

As an example of how model specification effects computational complexity and the structure of a 'coded' solution, consider evaluating the likelihood of an SDE:

$$\begin{aligned} dX_t &= \theta_1(\theta_2 + \theta_3 \sin(2\pi(t)) - X_t)dt + \theta_4\sqrt{X_t}dW_t^{(1)} \\ dY_t &= \theta_5 Y_t(\theta_6 - Y_t)dt + \theta_7 Y_t dW_t^{(2)}. \end{aligned} \tag{63}$$

By applying the cumulant truncation procedure, we first derive the system of ODEs that govern the evolution of the cumulants. Following Figure 3 we apply a 4-th order truncation on each dimension which results in the cumulant system:

$$\begin{aligned}
\kappa'_{10}(t) &= \theta_1(\theta_2 + \theta_3 \sin(2\pi t)) - \theta_1\kappa_{10}(t) \\
\kappa'_{20}(t) &= -\theta_1 2\kappa_{20}(t) + \theta_4^2\kappa_{10}(t) \\
\kappa'_{30}(t) &= -\theta_1 3\kappa_{30}(t) + \theta_4^2 3\kappa_{20}(t) \\
\kappa'_{40}(t) &= -\theta_1 4\kappa_{40}(t) + \theta_4^2 6\kappa_{30}(t) \\
\kappa'_{01}(t) &= \theta_5\theta_6\kappa_{01} - \theta_5(\kappa_{01}\kappa_{01} + 1\kappa_{02}) \\
\kappa'_{02}(t) &= \theta_5\theta_6 2\kappa_{02} - \theta_5(4\kappa_{02}\kappa_{01} + 2\kappa_{03}) + \theta_7^2(\kappa_{02} + \kappa_{01}\kappa_{01}) \\
\kappa'_{03}(t) &= \theta_5\theta_6 3\kappa_{03} - \theta_5(6\kappa_{01}\kappa_{03} + 6\kappa_{02}\kappa_{02} + 3\kappa_{04}) + \theta_7^2(3\kappa_{03} + 6\kappa_{01}\kappa_{02}) \\
\kappa'_{04}(t) &= \theta_5\theta_6 4\kappa_{04} - \theta_5(8\kappa_{01}\kappa_{04} + 12\kappa_{02}\kappa_{03} + 12\kappa_{03}\kappa_{02}) + \theta_7^2(6\kappa_{04} + 12\kappa_{01}\kappa_{03} + 12\kappa_{02}\kappa_{02})
\end{aligned} \tag{64}$$

with initial conditions $\kappa_{10}(s) = x_s$, $\kappa_{01}(s) = y_s$ and $\kappa_{\cdot}(s) = 0$ otherwise.

In order to solve eqn 64 we may employ single-step Runge-Kutta methods: A process whereby a system of ODEs is evaluated numerically at fixed points over a desired transition horizon $[s, t]$. Let

$$\frac{d}{dt}\boldsymbol{\kappa}(t) = h(\boldsymbol{\kappa}(t), t) \tag{65}$$

denote the system of cumulant equations for a given diffusion model (i.e., for eqn 63 $h(\boldsymbol{\kappa}(t), t)$ is given by the right hand side of eqn 64). Then, subject to the initial condition $\boldsymbol{\kappa}(s) = \boldsymbol{\kappa}_0(s)$, we can evaluate the cumulant trajectories at consecutive time points $t_0 = s, t_1, t_2, \ldots, t_{M+1} = t$ via updating equations of the form:

$$\boldsymbol{\kappa}_n(t_{i+1}) = \boldsymbol{\kappa}_0(t_i) + \sum_{j=0}^{n-1} c_j h(\boldsymbol{\kappa}_j(t_i + a_j\Delta), t_i + a_j\Delta)\Delta \tag{66}$$

for $i = 0, 1, \ldots M$, with

$$\boldsymbol{\kappa}_j(t_i + a_j\Delta) = \boldsymbol{\kappa}_0(t_i) + \sum_{k=0}^{j-1} b_{ij} h(\boldsymbol{\kappa}_k(t_i + a_k\Delta), t_i + a_k\Delta)\Delta \tag{67}$$

for $j = 1, 2, \ldots, n-1$, $a_{\cdot}$, $b_{\cdot\cdot}$ and $c_{\cdot}$ all real valued and integer n. The order of the approximation and its accuracy depend on both n (the number of stages used during a single update) and the values chosen for the coefficients $a_{\cdot}$, $b_{\cdot\cdot}$ and $c_{\cdot}$. For introductory material on Runge-Kutta methods see for example Butcher (2007) or Atkinson (2008). For purposes of the **DiffusionRgqd** package we employ either a 4(5)-th order Runge-Kutta method using the coefficients of the so-called Runge-Kutta-Fehlberg method (Fehlberg 1970), or the 8(10)-th order Runge-Kutta scheme of Feagin (2007) where the term in brackets indicates the embedded order used in order to evaluate the local truncation error estimate. Note that in the case of the 4(5) method we use the 4-th order result in order to update the approximation and in the case of the 8(10) method we use the 10-th order result. Throughout the package we opt to use fixed step sizes, say $t_0 = s, t_1 = s + \Delta, ..., t_i = s + i\Delta, ..., t - \Delta, t$ with $\Delta = (t - s)/M$, as opposed to adaptive step sizes. In addition to providing more control over the quality of the approximate cumulant systems within a given application we find that fixed step-size schemes are significantly less prone to breaking down in comparison to adaptive step sizes.

As the final step in evaluating the likelihood we need to evaluate the transitional density. Note that since both dimensions are non-linear and no interaction terms are present we may approximate the transitional density at t by plugging the approximate solution to the cumulants

into the product of two scalar saddlepoint approximations (see eqn 32):

$$d_2(\{x_t, y_t\}|\{x_s, y_s\}) = \frac{\exp\left(\frac{\partial}{\partial\alpha}K_x^{(4)}(\alpha_0, t) - \alpha_0 x_t + \frac{\partial}{\partial\beta}K_y^{(4)}(\beta_0, t) - \beta_0 y_t\right)}{2\pi\sqrt{\frac{\partial^2}{\partial\alpha^2}K_x^{(4)}(\alpha_0, t)\frac{\partial^2}{\partial\beta^2}K_y^{(4)}(\beta_0, t)}}, \tag{68}$$

where

$$K_x^{(4)}(\alpha, t) = \kappa_{10}(t)\alpha + \frac{\kappa_{20}(t)\alpha^2}{2} + \frac{\kappa_{30}(t)\alpha^3}{6} + \frac{\kappa_{40}(t)\alpha^4}{24} \tag{69}$$

and

$$\alpha_0 = -\frac{\kappa_{30}(t)}{\kappa_{40}(t)} + \left(-\frac{q}{2} + \sqrt{C}\right)^{1/3} - \left(\frac{q}{2} + \sqrt{C}\right)^{1/3}, \tag{70}$$

with

$$\begin{aligned} p &= \frac{1}{3}\left(\frac{1}{2}\kappa_{40}(t)\kappa_{20}(t) - \frac{1}{4}\kappa_{30}(t)^2\right)/\left(\frac{1}{6}\kappa_{40}(t)\right)^2, \\ q &= \frac{1}{27}\left(\frac{27}{36}\kappa_{40}(t)^2(\kappa_{10}(t) - x_t) - \frac{3}{4}\kappa_{40}(t)\kappa_{30}(t)\kappa_{20}(t) + \frac{1}{4}\kappa_{30}(t)^3\right)/\left(\frac{1}{6}\kappa_{40}(t)\right)^3, \\ C &= \frac{q^2}{4} + \frac{p^3}{27} \end{aligned} \tag{71}$$

and β_0 can be evaluated similarly for $K_y^{(4)}(\beta, t)$.

Finally, by combining these elements we can evaluate the log-likelihood of eqn 63 numerically for any number of transitions. This can be achieved by using the `solver()` function in the C++ script given in Appendix C. The code thus represents an efficient numerical solution of the likelihood of a bivariate time-series under model 63 using the cumulant truncation procedure. The script can then compiled by the `sourceCpp()` function from the **Rcpp** package after which `solver()` may then be called as a function in the **R** environment. In this way, by delegating computationally expensive parts of the cumulant truncation procedure to C++, we can significantly speed up calculations and simply manage the application thereof by wrapping it in an R function.

Now, altering the model only slightly by for example including an interaction term, say $\theta_8 X_t Y_t$ in the drift of X_t, would result in a much more complex solution, which in turn would require similarly more complex code. As such, for any given model we would have to go through the same process as above in order to identify the most computationally efficient solution. This remains true for the scalar case as well. For example, depending on what density and truncation order that is used, a vast array of approximations may be constructed. As such we have developed the **DiffusionRgqd** package so as to construct the most efficient C++ solution for a given model specification with minimal mathematical input from the user. The C++ code is then handled by whichever **DiffusionRgqd** routine is called by the user and thus the analysis can commence without burdening the user with any mathematical or algorithmic intricacies. In this way we are able to generate computationally efficient routines for performing inference and analysis on diffusion processes that fit the generalized quadratic framework within the R environment.

6. The DiffusionRgqd package

6.1. Outline of the package

DiffusionRgqd consists of a set of functions that allow the user to perform inference and analysis on generalized quadratic diffusions. The main routines that appear in the package are (main functions that do not use C++ are indicated with an asterisk):

`BiGQD.density`*: Calculate the transition density of a bivariate GQD with time-inhomogeneous coefficients over a specified time interval.

`BiGQD.mcmc` : Use a MCMC algorithm to draw parameters of a bivariate GQD model with time-inhomogeneous coefficients.

`BiGQD.mle` : Calculate the maximum likelihood estimates for the parameters of a bivariate GQD model with time-inhomogeneous coefficients.

`GQD.density`* : Calculate the transition density of a scalar GQD with time-inhomogeneous coefficients over a specified time interval.

`GQD.mcmc` : Use a MCMC algorithm to draw the parameters of a scalar GQD model with time-inhomogeneous coefficients.

`GQD.mle` : Calculate the maximum likelihood estimates for the parameters of a GQD model with time-inhomogeneous coefficients.

`GQD.passage` : Approximate the first passage time density of a time-homogeneous GQD to a fixed boundary.

`GQD.TIpassage` : Approximate the first passage time density of a GQD with time-inhomogeneous coefficients to a fixed boundary.

In addition to the main routines, some supporting functions have been created to make the package more user friendly. These include:

`GQD.remove` : Removes the coefficients of an existing GQD model from the current workspace.

`BiGQD.dic` : Summarizes DIC values from a list of `GQD.mcmc` and `BiGQD.mcmc` objects.

`BiGQD.aic` : Summarizes AIC values from a list of `GQD.mle` and `BiGQD.mle` objects .

`GQD.plot` : Plot routines for various classes of objects in the **DiffusionRgqd** package.

6.2. `GQD.mcmc()` details

Before commencing with concrete examples of the package we detail some of the input parameters and how they relate to the theory developed in sections 4.1 and 4.3 by focusing on the `GQD.mcmc()` function. `GQD.mcmc()` is perhaps the most involved function in the package with respect to input, however most of the parametric input of `GQD.mcmc()` is shared by other routines in the package. The function call consists of (see *usage* in the package help files):

```
GQD.mcmc(X,time,mesh,theta,sds,...)
```

The input parameters for `GQD.mcmc()` are

`X`:

A vector containing discretely observed data points of a time series to be modelled.

`time`:

A vector containing the time points at which the observations in `X` were made. Although `time` values are restricted to be numeric e.g. $0, 0.1, 0.2, \ldots$, dates can easily be converted to numerical values prior to input using the standard `as.Date` functions in R. It is important to distinguish the unit of measurement used relative to the time signature e.g. monthly data using a yearly unit interval may result in a time signature `seq(0,1,by=1/12)` for each unit of time.

`mesh`:

`mesh` gives number of updates used in the evaluation of the chosen Runge-Kutta scheme (see eqns 66 and 67) for all transition horizons. The number of updates imply the time discritization of each transition horizon. Each transition horizon `time[i+1]-time[i]` is subdivided into `mesh` subintervals, i.e. `mesh+1` mesh points, over which the cumulant equations are evaluated numerically. Note that the same number of points are used regardless of the individual magnitudes of `time[i+1]-time[i]`. Thus if the data are not equispaced, `mesh` should be chosen so as to be large enough to ensure a sufficiently fine mesh on the maximal transition horizon `max(diff(time))`.

`theta`:

The parameter vector of the process. The values given for the vector `theta` are used as the starting values for parameter chains generated using a Random Walk Metropolis Hastings scheme (RWMH). Note that the estimation routines in **DiffusionRgqd** use C++. Thus, in order to ensure the syntactical compatibility with the underlying C++ code, `theta` is used as a reserved name for parameters in the coefficient functions. As such any unrecognised variables will result in the execution of the algorithm being halted. In any event, where needed, routines will conduct a number of basic syntax checks in order to guide the user in the right direction.

`sds`:

Proposal distribution standard deviations under the RWMH scheme. 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/discard. When calculating parameter estimates this value may be changed externally.

`Trunc`:

A vector indicating the truncation order to be used on the cumulant equations and the density approximation respectively. Possible values are `c(4,4)`, `c(6,4)`, `c(8,4)`, `c(6,6)`, `c(8,6)` and `c(8,8)`. This follows since the number of coordinates in the cumulant equations (eqn 25) preclude those in the corresponding density approximations.

`Dtype`:

The density approximation to be used. Possible values are `'Saddlepoint'`, `'Normal'`, `'Gamma'`, `'InvGamma'` and `'Beta'` corresponding to the saddlepoint approximation (eqn 32) and the respective classes of the Pearson system (eqns 35, 36, 37 and 38).

`P`:

The number of mesh points used in the normalization of the Pearson system (see Appendix A).

`alpha`:

A 'spread' parameter controlling mesh spacing used in the normalization of the Pearson system (see Appendix A).

`lower, upper`:

Upper and lower bounds used in the normalization of the Pearson system (see Appendix A).

`plot.chain`:

If = `TRUE` (default), a trace plot of the MCMC chain will be made along with a trace of the acceptance rate.

`Tag`:

A text tag that can be assigned to a given model for easy identification when calling summary functions such as `GQD.dic()`.

7. Example applications

In the examples that follow we demonstrate how **DiffusionRgqd** package is used in practice. The latest version of the package along with full scripts of the examples given here can be found on GitHub at https://github.com/eta21.

7.1. Generate the transition density of a time-inhomogeneous GQD

As an introductory example to the **DiffusionRgqd** package we approximate the transitional density of a scalar diffusion over an arbitrarily chosen transition horizon. This can be achieved using the `GQD.density()` function. `GQD.density()` generates the transitional density of a GQD for a given initial value using the cumulant truncation procedure outlined in Section 4.1. The function serves as a good starting point for any analysis being conducted using the **DiffusionRgqd** package as it allows one to check whether a proposed model does not exhibit nonsensical dynamics with respect to the problem at hand. Perhaps more importantly, it can be used to check the validity of the density approximation and/or an appropriate truncation order for the cumulants.

Consider a Cox-Ingersoll-Ross (CIR) (Cox, Ingersoll, and Ross 1985) process with time-dependent coefficients:

$$dX_t = 2(10 + \sin(2\pi(t - 0.5)) - X_t)dt + \sqrt{0.25(1 + 0.75\sin(4\pi t))X_t}dB_t. \tag{72}$$

In order to analyse eqn 72 in R we need to define the model within the workspace. Since **DiffusionRgqd** uses a functional input interface which relies on declaring the functional form of a given model's coefficients, we need to make sure that the workspace doesn't contain any object names that might clash with those of the model coefficients. We can do this by simply running the `GQD.remove()` command:

```
library(DiffusionRgqd)
GQD.remove()
```

If any objects are recognized with names that may clash with names reserved for use with the `GQD.density()` function they will subsequently be removed. In this case we used a *vanilla* R session so the function will simply indicate that no clashes were detected and removed.

```
## [1] "Removed :  NA "
```

The purpose of the `GQD.remove()` function is to act as a *model-eraser* in situations where multiple models are being defined in a single R session. The next step is to write eqn 72 in terms of its coefficients. We can define eqn 72 in the current R session by declaring the coefficient functions:

```
G0 <- function(t){2*(10+sin(2*pi*(t-0.5)))}
G1 <- function(t){-2}
Q1 <- function(t){0.25*(1+0.75*(sin(4*pi*t)))}
```

The functions `G0`, `G1` and `Q1` together constitute the R-language counterpart of eqn 72 under the framework of Section 3. Note that for scalar GQDs we have capitalized the coefficient names given in Section 3 in order to avoid the possibility of calling `q()` accidentally, in which case the R console will prompt to quit.

In order to approximate the transitional density using the methodology of Section 4.1 we need to define the peripheral parameters of the problem. This consists of giving the initial condition of the SDE (the starting value of the diffusion), the starting time of the diffusion and the geometry of the transitional horizon, i.e. spatial values where the density is to be evaluated and the corresponding time mesh. These elements can be defined by assigning values to the arguments of `GQD.density()`: `Xs`, `Xt`,`s`, `t` and `delt`. Respectively, these parameters represent the initial value, the values at which to evaluate the transition density, ther starting time, the final time and the step size for the time mesh. That is, for a diffusion process starting at time `s` in state `Xs`, the transition density is approximated at times `s` + `delt`, `s` + 2`delt` ... upto and including `t` at all values of the vector `Xt`. In R:

```
states     <-  seq(5,15,1/10)
initial    <-  8
Tmax       <-  5
Tstart     <-  1
increment  <-  1/100

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

`GQD.density()` creates a list containing a matrix of density values, spatial coordinates at which the density was evaluated, corresponding time coordinates for the time mesh and finally a matrix of trajectories for the cumulants and moments that were used in evaluating the density approximation. In this case we have assigned the output to an object called `M` for further use in the workspace. Since the density approximation is evaluated over a space-time lattice it can best be visualized with a perspective plot. For example, using the **rgl** package (Adler, Murdoch *et al.* 2014):

```
library(rgl)
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=F,xlab='State (X_t)',ylab=
        'Time(t)',zlab='Density f(X_t|X_s)')
```

we can recreate the surface in Figure 1. Alternatively, one can simply use `GQD.plot(M)` in order to visualize a given density approximation in a similar way.

Note that we have not directly specified what truncation order to be used in the calculation of the density approximation. As mentioned in Section 4.2, the default is to use a 4-th order

truncation in conjunction with a 4-th order truncated saddlepoint approximation. In most cases it suffices to use the default settings although, as will be shown in the following example, little effort is required to extend the analysis to higher order truncations and/or alternate density approximations. Although this example operates well within the limits of the capabilities of the methodology it serves to illustrate the simplicity of the interface of the **DiffusionRgqd** package - often producing desired results in less than 10 lines of code with minimal mathematical input.

7.2. Time-inhomogeneous Jacobi diffusion: A scalar diffusion with finite support

Although most practical applications of diffusion processes result in the use of diffusions with uni-modal transitional densities it is still possible to conceive of a diffusion within the GQD framework that exhibits atypical dynamics. For example, a particularly interesting phenomena occurs when considering GQDs for which the diffusion term is of the form:

$$\sigma(X_t, t) = c\sqrt{X_t(1 - X_t)}. \tag{73}$$

Assuming the process starts within the interval $[0, 1]$, whenever the diffusion approaches the bounds 0 or 1, the dynamics of the process is dominated by the behaviour of the drift function, $\mu(X_t, t)$. Provided $\partial/\partial x \mu(x, t)|_{x=0} > 0, \partial/\partial x \mu(x, t)|_{x=1} < 0 \quad \forall t$ are sufficiently large, the process will reflect from the boundaries and remain within the interval $[0, 1]$. A special case of this behaviour can be seen with the so-called Jacobi diffusion (Gouriéroux and Valéry 2004), whereby $\mu(X_t, t)$ is linear in X_t. For purposes of the illustration we shall assume a time-inhomogeneous Jacobi diffusion:

$$dX_t = a(b(1 + 0.25\sin(2\pi t)) - X_t)dt + c\sqrt{X_t(1 - X_t)}dB_t, \tag{74}$$

with $X_0, b \in [0, 1]$ and $a > 0$. The diffusion exhibits oscillating drift dynamics whereby the process is pulled towards a point fluctuating between $0.75b$ and $1.25b$. If and when the trajectory of the process hits 0 or 1 it is reflected toward the interior of the domain. By applying the cumulant truncation procedure to eqn 74 we approximate the evolution of the cumulants for various orders of truncation under the Beta-type density from the multimodal Pearson system (eqn 38). This may be achieved in R using:

```
# Set some parameter values:
a <- 0.5; b <-  0.6; cc <- 1; X0 <- 0.5;

# Give the coefficient form of the diffusion:
G0 <- function(t){a*(b*(1+0.25*sin(pi*t)))}
G1 <- function(t){-a}
Q1 <- function(t){cc}
Q2 <- function(t){-cc}

# What states should the density be evaluated at:
states=seq(1/1000,1-1/1000,1/1000)

# Generate the transitional density for truncation orders 4,6 and 8:
res1 <- GQD.density(X0,states,0,2,1/100,Dtype='Beta',Trunc=c(4,4))
res2 <- GQD.density(X0,states,0,2,1/100,Dtype='Beta',Trunc=c(6,6))
res3 <- GQD.density(X0,states,0,2,1/100,Dtype='Beta',Trunc=c(8,6))
```

The objects `res1`, `res2` and `res3` now contain the transitional density approximations for truncation order 4, 6 and 8. The additional parameter Dtype = 'Beta' sets the density

approximation to eqn 38 while the variable `Trunc = c(6,6)` sets the cumulant truncation (first item in `c()`) to $m = 6$ (thus evaluating eqn 25 for all $i = 1, 2, \ldots, 6$) whilst setting $m = 6$ for eqn 38 (second item in `c()`). Note that we separate the cumulant truncation order from that of the density in order to allow lower order density truncations under a given cumulant truncation order, for example `Trunc=c(6,4)`.

In order to check the accuracy of the approximation we may compare the approximate transition density to a simulated transition density of the Jacobi diffusion. This can be achieved by simulating a number of sample paths for eqn 74 and subsequently plotting a histogram of the resulting trajectories. For a diffusion process with SDE:

$$dX_t = \mu(X_t, t)dt + \sigma(X_t, t)dB_t \tag{75}$$

we may simulate a trajectory at discrete time points by applying a Euler-Maruyama scheme to 75 and evaluate the resulting difference equation iteratively. That is given an initial value X_{t_0}, we evaluate for all $i = 1, 2, \ldots$:

$$X_{t_i} = X_{t_{i-1}} + \mu(X_{t_{i-1}}, t_{i-1})\Delta + \sigma(X_{t_{i-1}}, t_{i-1})r_\Delta \tag{76}$$

where $r_\Delta \sim \mathrm{N}(0, \sigma^2 = \Delta)$ and $t_i - t_{i-1} = \Delta$. By plugging the drift and diffusion terms of eqn 74 into this updating equation we may simulate a number of trajectories for the restricted diffusion. However, since eqn 76 is not exact the simulated paths may actually exit the $[0, 1]$ region. As such we employ the simple modification that at each iteration X_{t_i} is set to $\min(\max(0, X_{t_{i-1}} + \mu(X_{t_{i-1}}, t_{i-1})\Delta + \sigma(X_{t_{i-1}}, t_{i-1})r_\Delta), 1)$. By simulating a large number of trajectories we can compare each respective density approximation by superimposing the approximation over a histogram of the simulated trajectories at any given time point. This can easily be achieved using the code:

```
N      <- 100000  # Number of trajectories to simulate.
smdelt <- 1/1000  # Simulation stepsize.
d      <- 0       # A variable for tracking time.
X      <- rep(X0,N)

# Take snapshots of the density at these times:
when <- c(0,0.25,0.5,1,1.75)

for(i in 2:(2/smdelt))
{
    # A Euler-Maruyama type approximation:
    X <- pmax(pmin(X + (G0(d)+G1(d)*X)*smdelt
                +sqrt(Q1(d)*X+Q2(d)*X^2)*rnorm(length(X),sd=sqrt(smdelt)),1),0)
    d <- d+smdelt
    if(any(when==round(d,3)))
    {
       index <- which(res1$time==round(d,3))
      hist(X,col='gray85',freq=F,breaks=30,
           main=paste0('Transitional Density at t = ',round(d,3)),ylim=c(0,3))
      lines(res1$density[,index]~res1$Xt,col='red',lty='dotdash',lwd=2)
      lines(res2$density[,index]~res2$Xt,col='green',lty='solid',lwd=2)
      lines(res3$density[,index]~res3$Xt,col='blue',lty='dashed',lwd=2)
      legend('top', legend=c('Truncation: (4,4).','Truncation: (6,6).',
             'Truncation: (8,8).'), col = c('red','green','blue'),
              lty=c('dotdash','solid','dashed'),lwd=2,bg='white')
```

```
    }
}
```

The resulting output is given in Figure 4. The figures suggest that the approximation remains accurate for truncation orders 6 and 8 whilst $m = 4$ produces less desirable results. Interestingly, as time progresses it seems that all orders of approximation produce a reasonable approximation of the transition density. This may be due to the long-run dynamics of the process being somewhat simpler than in early phases in the transition horizon. Figure 5 shows a perspective plot of the entire trajectory of the transitional density using the 8-th order truncation. As in the previous example this can be done by using the `GQD.plot()` command i.e., `GQD.plot(res3)`. As is typical for diffusion processes we see that the transitional density starts out as an infinite point mass and shortly exhibits a normal-like distribution. As time progresses, however, the density transits into a 'U-shape' with oscillating peakedness in conjunction with the sinusoidal term of the drift function.

7.3. Bivariate non-linear dynamics: The stochastic Lotka-Volterra eqns

Although the **DiffusionRgqd** package allows one to perform comprehensive analysis on scalar quadratic diffusions, similar analysis can be conducted for interesting bivariate diffusions. A model that is often used to illustrate non-linear dynamics in the analysis of ODEs is that of the Lotka-Volterra model. The corresponding ODEs are typically given as:

$$\begin{aligned} \frac{\partial x_t}{\partial t} &= ax_t - bx_t y_t \\ \frac{\partial y_t}{\partial t} &= -cy_t + dx_t y_t \end{aligned} \tag{77}$$

for some positive a, b, c and d. The equations are often used to describe the dynamics of two interacting populations wherein the population growth rate of the populations are mutually influenced by the current level of the opposing population. As such the model has been used to explain oscillatory behaviour in predator-prey relationships (Hoppensteadt 2006) where x_t denotes the prey population and y_t the predator population at time t. Continuing with the predator-prey metaphor, perhaps one deficiency of the model, one might argue, is the absence of random input and subsequent effects on population levels. Indeed, under the ODE formulation the predicted population behaviour (given fixed parameters) are completely deterministic. Another deficiency might be the absence of growth inhibiting factors such as disease or over-grazing. For these purposes we may define an example of a stochastic counterpart to the Lotka-Volterra equations as:

$$\begin{aligned} dX_t &= (aX_t - bX_tY_t)dt + f\sqrt{X_t}dB_t^{(1)} \\ dY_t &= (-cY_t + dX_tY_t - eY_t^2)dt + g\sqrt{Y_t}dB_t^{(2)} \end{aligned} \tag{78}$$

The intuition behind the example eqn 78 is that the drift terms replicate eqn 77 but with the addition of the $-eY_t^2$ term. The additional term represents the effect of overpopulation for the predator when, for example, harvesting of prey becomes ever more inefficient as the predator population grows. Furthermore, the population volatilities are assumed to increase with population size meaning that random fluctuations are less severe when populations are small and *vice versa*.

In order to analyse eqn 78, we may consider the evolution of its transitional density. This can be achieved in similar fashion to the previous examples through the `BiGQD.density()` function.

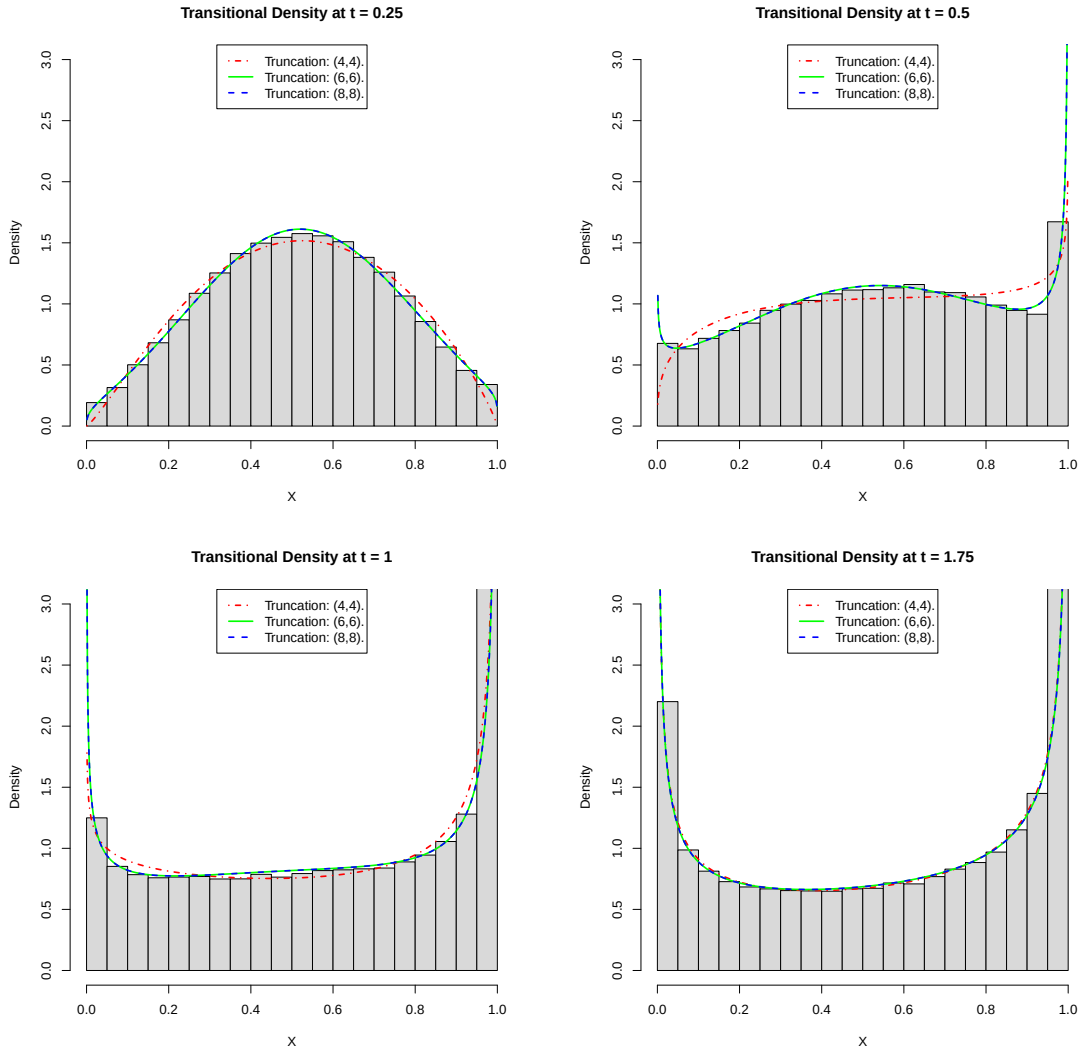

Figure 4: Histograms of simulated trajectories for eqn 74 and transitional density approximations for various truncation orders at indicated times. Truncation orders 6 (green) and 8 (blue, dashed) prove accurate while a 4th order truncation (red, dot-dashed) fails to reproduce the desired shape for the transitional density at $t = 0.5$.

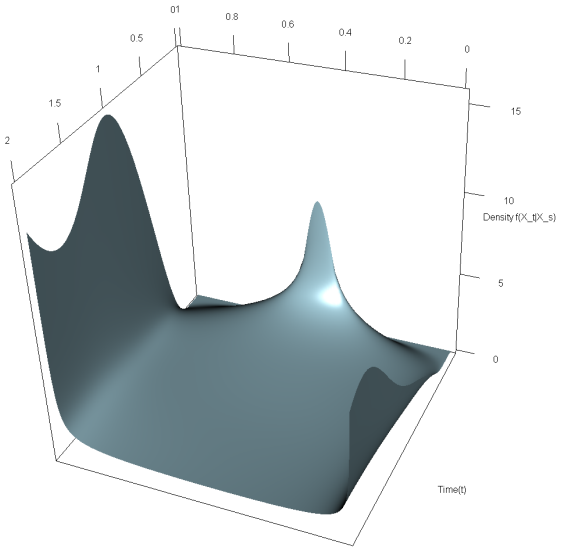

Figure 5: Evolution of the transition density of eqn 74 over time using the 8-th order cumulant truncation in conjunction with 8-th order truncated Pearson Beta density (see eqn 38).

For purposes of this example, consider the SDE:

$$\begin{aligned} dX_t &= (1.5X_t - 0.4X_tY_t)dt + \sqrt{0.05X_t}dB_t^{(1)} \\ dY_t &= (-1.5Y_t + 0.4dX_tY_t - 0.2Y_t^2)dt + \sqrt{0.1Y_t}dB_t^{(2)} \end{aligned} \tag{79}$$

with initial values $X_0 = 5$ and $Y_0 = 5$. In R we may generate the transitional density over the transition horizon $t \in [0, 10]$ using the code:

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

# Define the X dimension 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(Xs=5,Ys=5,Xt=seq(3,8,length=50),Yt=seq(2,6,length=50),s=0,
t=10,delt=1/100)
```

As in the scalar case, a model is coded by defining the coefficients of the diffusion as functions with names that reflect the powers of each terms' X_t and Y_t components. As with `GQD.density()`, `BiGQD.density()` approximates the transition density at times `s`, `s+delt` ... on a lattice of grid points given by the user (`Xt=seq(3,8,length=50)` and `Xt=seq(2,6,length=50)`) for a

given pair of initial values (`Xs =5,Ys=5`). Since the transition density of a bivariate diffusion at a fixed time point is given by a surface in three dimensions, `BiGQD.density()` returns a three dimensional array wherein the transition density approximation is given as slices of this array corresponding to each time point `s`, `s+delt` etc. We can visualize the evolution of the transition density over time by making contour plots of the transition density at different time points. Incidentally the mean trajectory of eqn 78 corresponds exactly to the trajectory of eqn 77 with the addition of the term $-0.2Y_t^2$. Consequently, in order to monitor the trajectory of the approximation it would be useful to compare the coordinates of the first moments of the process $(E(X_t), E(Y_t))$ under the cumulant truncation procedure to those resulting from simulated trajectories of eqn 79. For brevity we have included the simulated moments as a dataset, `SDEsim3`, giving simulated values for $E(X_t)$ and $E(Y_t)$ at times $0, 0.01, 0.02, \ldots, 10$. In order to visualize the evolution of the transition density we may make use of contour plots, for example in R:

```
# Load simulated trajectory of the joint expectation:
data(SDEsim3)
attach(SDEsim3)

# Record graphs at time points along the trajectory:
time.index <- c(10,300,750,1000) +1

for(i in time.index)
{
# Now illustrate the density using a contour plot:
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='Predator',plot.axes=
{
# Add trajectory of simulated expectation:
lines(my~mx,col='black',lty='dashed',lwd=2)
# 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('dashed',NA),pch=c(NA,21),lwd=c(2,NA),
legend=c('Simulated Expectation','Predicted Expectation'))
})
}
```

Figure 6 shows the resulting graphical output. As illustrated, the predicted expectation of eqn 79 under the cumulant truncation procedure matches very closely the simulated mean trajectory. However, what the transition density demonstrates is that, in contrast to the deterministic nature of eqn 77, the inclusion of the Brownian motion terms in eqn 78 implies vastly different population dynamics: In the deterministic case the trajectories are predicted to move ever closer to an equilibrium point (attractor), whilst with the stochastic equations it is likely for either of the two populations to move far away from such an equilibrium point due to the presence of random fluctuations - even as the expected trajectory stabilizes. Thus, where the deterministic equations may predict both populations to survive with certainty, as is with the current parameter set, under the stochastic equations both extinction and explosion events are probable while on average the population coordinates will tend to some equilibrium position as time increases.

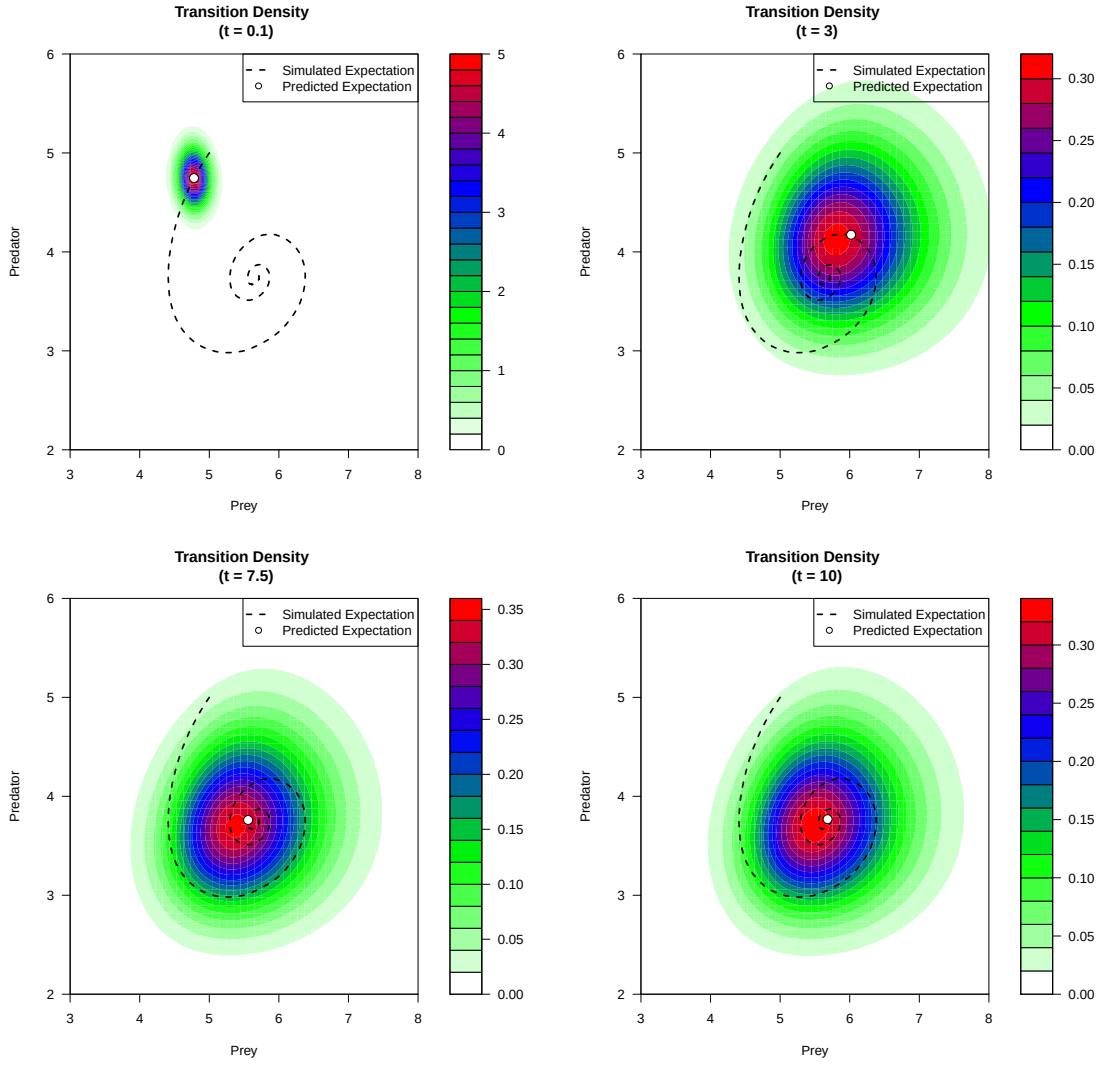

Figure 6: Coloured contourplots of the transition density at fixed time points $t \in \{0.1, 3, 7.5, 10\}$. Superimposed on the simulated mean trajectory of the process (dashed line) is the predicted expectation under the cumulant truncation procedure (white circle).

7.4. Maximum likelihood estimation: Stochastic volatility models

Diffusion processes are often used in financial applications to model the trajectories of asset prices or financial instruments. Although a great deal of literature is dedicated to the fitting and calibration of well known diffusion models to financial data, less time is spent assessing how well the dynamics of such models represent that which is observed in a given real-world dataset. Thus, for the present example we analyse a dataset that is often used to demonstrate the application of stochastic volatility models in finance, namely the Standard and Poor's 500 index (SPX)[1]. Although the term 'stochastic volatility' may be used to refer to any model with randomly varying higher order moments, in the context of diffusion models it usually refers to the process of treating the volatility terms of a scalar diffusion model as being driven by a stochastic process itself. Perhaps the most famous stochastic volatility model is the Heston model (Heston 1993), a bivariate SDE of the form:

$$\begin{aligned} dS_t &= \theta_1 S_t dt + \theta_2 S_t \sqrt{V_t} dB_t^{(1)} \\ dV_t &= (\theta_3 - \theta_4 V_t)dt + \theta_5 \sqrt{V_t} dB_t^{(2)} \end{aligned} \tag{80}$$

where S_t denotes the spot price process and V_t denotes the corresponding variance process. Thus, in the Heston model the spot price is modelled by a geometric Brownian motion with the addition that the volatility coefficient for the spot price is itself driven by a diffusion process – in this case, a CIR process. Furthermore, it is assumed that the Brownian motions $dB_t^{(1)}$ and $dB_t^{(2)}$ are correlated i.e., $\text{corr}(B_t^{(1)}, B_t^{(2)}) = \theta_6$. For the SPX, although data for the spot process S_t is readily available, obtaining the trajectory of the volatility process is less trivial. In order to conduct the analysis we shall assume a suitable proxy for the volatility of the index by making use if the Chicago Board Options Exchange (CBOE) Market Volatility Index (VIX), which is based on the implied volatility of options written on the SPX. We source data for the index by using the **Quandl** package as follows:

```
library(Quandl)
library(DiffusionRgqd)
# Source data for the S&P500 index (SPX).
quandldata1 <-  Quandl("YAHOO/INDEX_GSPC", collapse="weekly",
start_date="1990-01-01",end_date="2015-01-01", type="raw")

St <- rev(quandldata1[,names(quandldata1)=='Close'])
time1 <-rev(quandldata1[,names(quandldata1)=='Date'])

# Source data for the volatility index (VIX).
quandldata2 <-  Quandl("YAHOO/INDEX_VIX", collapse="weekly",
start_date="1990-01-01",end_date="2015-01-01", type="raw")

Vt <- rev(quandldata2[,names(quandldata2)=='Close'])
time2 <- rev(quandldata2[,names(quandldata2)=='Date'])
```

Figure 7 plots the resulting weekly closing prices for the SPX and VIX for the period 1990-01-01 to 2015-01-01. Visual inspection of the time series suggests that the volatility of the index does indeed vary significantly over time as is evidenced by the VIX which was observed to be as low as 9.5% during December of 1993 and high as 79% during October of 2008.

In order to conduct the analysis we consider the log-returns of the index. Let $R_t = \log(S_t)$ then

[1]The corresponding ticker symbols for Google finance and Yahoo finance are INX and ĜSPC respectively

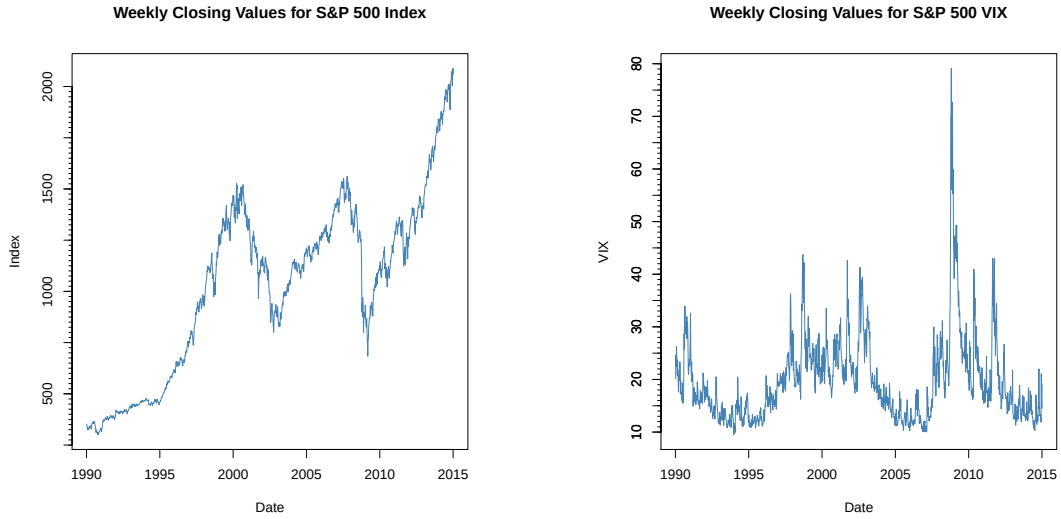

Figure 7: Weekly closing values for the Standard and Poor's 500 index (SPX) and Standard and Poor's 500 volatility index (VIX) for the period 1990-01-01 to 2015-01-01. Note that the volatility index is expressed in terms of volatility whilst models such as the Heston model are expressed in terms of the variance process (i.e., volatility squared).

by Ito's lemma, the corresponding Heston model under the log-transform is given by:

$$\begin{aligned} dR_t &= (\theta_1 - \frac{1}{2}\theta_2^2 V_t)dt + \theta_2\sqrt{V_t}dB_t^{(1)} \\ dV_t &= (\theta_3 - \theta_4 V_t)dt + \theta_5\sqrt{V_t}dB_t^{(2)} \end{aligned} \tag{81}$$

with $\text{corr}(B_t^{(1)}, B_t^{(2)}) = \theta_6$. In order to incorporate the correlation coefficient of the Brownian motions into the SDE it is useful to write the SDE in terms of independent Brownian motions: Let $W_t^{(1)}$ and $W_t^{(2)}$ be independent Brownian motions then we can construct correlated Brownian motions $B_t^{(1)}$ and $B_t^{(2)}$ with $\text{corr}(B_t^{(1)}, B_t^{(2)}) = \theta_6$ through the standard translation:

$$\begin{bmatrix} dB_t^{(1)} \\ dB_t^{(2)} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \theta_6 & \sqrt{1-\theta_6^2} \end{bmatrix} \begin{bmatrix} dW_t^{(1)} \\ dW_t^{(2)} \end{bmatrix}. \tag{82}$$

Thus the appropriate diffusion tensor for the Heston model is given by:

$$\begin{bmatrix} \theta_2\sqrt{V_t} & 0 \\ 0 & \theta_5\sqrt{V_t} \end{bmatrix} \begin{bmatrix} 1 & \theta_6 \\ \theta_6 & 1 \end{bmatrix} \begin{bmatrix} \theta_2\sqrt{V_t} & 0 \\ 0 & \theta_5\sqrt{V_t} \end{bmatrix}' = \begin{bmatrix} \theta_2^2 V_t & \theta_2\theta_5\theta_6 V_t \\ \theta_2\theta_5\theta_6 V_t & \theta_5^2 V_t \end{bmatrix}. \tag{83}$$

Under the GQD framework the Heston model can be defined in R using the code:

```
# R_t coefficients:
a00 <- function(t){theta[1]}
a01 <- function(t){-0.5*theta[2]*theta[2]}
c01 <- function(t){theta[2]*theta[2]}
d01 <- function(t){theta[2]*theta[5]*theta[6]}
# V_t coefficients:
b00 <- function(t){theta[3]}
b01 <- function(t){-theta[4]}
e01 <- function(t){theta[2]*theta[5]*theta[6]}
f01 <- function(t){theta[5]*theta[5]}
```

Note the inclusion of the vector `theta` in the coefficients: In the **DiffusionRgqd** package the variable name `theta` is preserved for the parameterization of coefficient functions. This is done so as to facilitate the modelling process in which case all components apart from the vector `theta` remain fixed.

The next step is to prepare the time series input for the analysis. In order to execute any of the estimation routines we need to supply a column matrix with the observed trajectories as well as a numerical vector containing the time points at which the observations were made. For the SPX data we shall assume that we are working on an annual time scale (i.e. one unit of time is equivalent to one year) with a 365 day time convention:

```
# Create data matrix and numerical time vector :
X <- cbind(log(St),(Vt/100)^2)
time <- cumsum(c(0,diff(as.Date(time1))*(1/365)))
```

Once the data is prepared we call the `BiGQD.mle()` routine in order to calculate maximum likelihood estimates of the parameter vector `theta`. As in the previous examples, the `Bi`-prefix to the GQD-function is used to distinguish between scalar and bivariate diffusions, whilst the suffix `.mle` refers to the purpose of the routine. In order to perform maximum likelihood estimation we supply `BiGQD.mle()` with the data matrix `X`, the time-vector `time` and some starting parameters:

```
# Some starting parameters for the optimization routine:
theta.start <- c(0,1,1,0.5,1,0)
# Calculate MLEs of the parameter vector:
model_1 <- BiGQD.mle(X,time,mesh=10,theta=theta.start)
```

An important distinction to make between estimation routines in the package such as `BiGQD.mle()` and routines like `GQD.density()` used in the previous examples is the use of C++ code. As outlined in Section 5, where multiple evaluations of the likelihood are to be carried out using the cumulant truncation procedure, the estimation routines will set up and compile a C++ routine tailored to the current diffusion model. As such, when an estimation routine is called for the first time for a given set of model coefficients, there might be some initial delay before the estimation routine commences. Naturally, provided the model coefficients remain the same, a second run would not be subject to this delay. For the `BiGQD.mle()` routine, once the applicable C++ code is compiled, a summary of the model is printed and maximization of the likelihood function commences using R's built in optimization routine `optim()`. After execution of the optimization routine some additional information about the model is appended to the summary. The resulting output looks something like:

```
##  ================================================================
##                      GENERALIZED QUADRATIC DIFFUSON
##  ================================================================
##  ____________________ Drift Coefficients ________________________
##  a00 : theta[1]
##  a01 : -0.5*theta[2]*theta[2]
##  ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...
##  b00 : theta[3]
##  b01 : -theta[4]
##  ____________________ Diffusion Coefficients ____________________
##  c01 : theta[2]*theta[2]
##  ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...
##  d01 : theta[2]*theta[5]*theta[6]
```

```
##  ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...
##  e01 : theta[2]*theta[5]*theta[6]
##  ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...
##  f01 : theta[5]*theta[5]
##
##  ___________________________ Model Info ___________________________
##  Time Homogeneous    : Yes
##  Data Resolution     : Homogeneous: dt=0.0192
##  # Removed Transits. : None
##  Density approx.     : 4th Ord. Truncation, Bivariate-Saddlepoint
##  Elapsed time        : 00:00:09
##  ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...
##  dim(theta)          : 6
##  ------------------------------------------------------------------
```

The textual output serves both as a visual buffer between models in the workspace as well as a visual check on whether the model that has been recognized by the `BiGQD.mle()` function corresponds to what the user intended it to be. In the workspace itself, `BiGQD.mle()` returns a list object containing information about the optimization run as well as peripheral information about the model. In order to extract the resulting parameter estimates one may use the `GQD.estimates()` function, which will return a summary of the parameter estimates resulting from the estimation procedure:

```
# Retreve parameter estimates and appr. 95% CIs:
GQD.estimates(model_1)
```

```
##          Estimate Lower_95 Upper_95
## theta[1]    0.083    0.031    0.135
## theta[2]    0.770    0.740    0.799
## theta[3]    0.168    0.128    0.207
## theta[4]    3.822    2.816    4.827
## theta[5]    0.431    0.414    0.447
## theta[6]   -0.671   -0.700   -0.641
```

The Heston model thus provides insight into the dynamics of the SPX and its corresponding variance dynamics. Indeed, the resulting parameter estimates confirms well documented features such as strong negative correlation in the noise of the SPX and VIX series as evidenced by the correlation parameter θ_6. By modifying the Heston model we can easily perform a more detailed analysis by including other peripheral data such as risk free rates and dividends in order to formulate the model in the market context (see for example Hurn, Lindsay, and McClelland (2014)). However, we defer such analysis in favour of answering the more general question of whether the the Heston model can be improved upon for the data at hand. For example, one might argue that the variance of both R_t and V_t should scale quadratically ($\theta_2^2 V_t^2$ and $\theta_5^2 V_t^2$) as opposed to linearly ($\theta_2^2 V_t$ and $\theta_5^2 V_t$) with respect to the state of the variance process, as in the case of the Heston model. To test this we can fit the model:

$$\begin{aligned} dR_t &= (\theta_1 - \frac{1}{2}\theta_2^2 V_t^2)dt + \theta_2 \sqrt{V_t^2} dB_t^{(1)} \\ dV_t &= (\theta_3 - \theta_4 V_t)dt + \theta_5 \sqrt{V_t^2} dB_t^{(2)}. \end{aligned} \tag{84}$$

Note, that apart from the terms θ_1, θ_3 and $-\theta_4 V_t$ there are no common terms between the previous model and the new model. Consequently, in order for `BiGQD.mle()` to ignore the

previous model coefficients we need to remove the previous model coefficients from the workspace. This be achieved by making use of the `GQD.remove()` function:

```
GQD.remove()  # Remove the previous model coefficients
```

```
## [1] "Removed :  a00 a01 b00 b01 c01 d01 e01 f01"
```

```
# R_t coefficients:
a00 <- function(t){theta[1]}
a02 <- function(t){-0.5*theta[2]*theta[2]}
c02 <- function(t){theta[2]*theta[2]}
d02 <- function(t){theta[2]*theta[5]*theta[6]}
# V_t coefficients:
b00 <- function(t){theta[3]}
b01 <- function(t){-theta[4]}
e02 <- function(t){theta[2]*theta[5]*theta[6]}
f02 <- function(t){theta[5]*theta[5]}
theta.start <- c(0,1,1,1,1,0)
model_2 <- BiGQD.mle(X,time,mesh=10,theta=theta.start)
```

In order to compare model fit for the various models we may make use of the `GQD.aic()` function. This can be achieved by compiling all of the model outputs `model_1`, `model_2` etc. into a list and passing the result as an argument to `GQD.aic()`. `GQD.aic()` will then produce a summary of AIC, BIC and likelihood statistics for all the listed models along with the number of parameters, the number of observations and a convergence diagnostic from the `optim()` output.

```
# Compare AIC and BIC vlaues for models 1 and 2:
GQD.aic(list(model_1,model_2))
```

```
##         Convergence p min.likelihood             AIC             BIC    N
## Model 1           0 6      -7852.212      -15692.424      -15661.381 1305
## Model 2           0 6      -7965.957  [=] -15919.914  [=] -15888.871 1305
```

Based on the AIC and BIC statistics, the revised model does indeed improve upon the Heston model with respect to model fit. The result suggests that the volatility structure of the index and variance process is more sensitive to changes in volatility than is predicted by the Heston model. Indeed, the volatility structure of revised model need not be the only improvement that can be made. We may also postulate various forms of the drift interaction and compare AIC statistics accordingly. For these purposes we consider three additional models:

$$\begin{aligned} dR_t &= -\frac{1}{2}\theta_1^2 V_t^2 dt + \theta_1\sqrt{V_t^2}dB_t^{(1)} \\ dV_t &= (\theta_2 - \theta_3 V_t)dt + \theta_4\sqrt{V_t^2}dB_t^{(2)}, \end{aligned} \tag{85}$$

which removes the constant term from the drift of eqn 84 resulting in a slightly simpler model that presents a drift free index on the original SPX scale (S_t).

$$\begin{aligned} dR_t &= (\theta_1 + \theta_7 R_t - \frac{1}{2}\theta_2^2 V_t^2)dt + \theta_2\sqrt{V_t^2}dB_t^{(1)} \\ dV_t &= (\theta_3 - \theta_4 V_t)dt + \theta_5\sqrt{V_t^2}dB_t^{(2)}, \end{aligned} \tag{86}$$

which modifies the drift of eqn 84 to include an R_t-term. Finally, we include drift feedback of the SPX into the VIX series using the model:

$$\begin{aligned} dR_t &= (\theta_1 - \frac{1}{2}\theta_2^2 V_t^2)dt + \theta_2\sqrt{V_t^2}dB_t^{(1)} \\ dV_t &= (\theta_3 - \theta_4 V_t + \theta_7 R_t)dt + \theta_5\sqrt{V_t^2}dB_t^{(2)}. \end{aligned} \tag{87}$$

Using the GQD framework these models can then be fitted using the R-code:

```
GQD.remove()
# R_t coefficients:
a02 <- function(t){-0.5*theta[1]*theta[1]}
c02 <- function(t){theta[1]*theta[1]}
d02 <- function(t){theta[1]*theta[4]*theta[5]}
# V_t coefficients:
b00 <- function(t){theta[2]}
b01 <- function(t){-theta[3]}
e02 <- function(t){theta[1]*theta[4]*theta[5]}
f02 <- function(t){theta[4]*theta[4]}

theta.start <- c(1,1,1,1,0)
model_3 <- BiGQD.mle(X,time,mesh=10,theta=theta.start)
```

```
GQD.remove()
# R_t coefficients:
a00 <- function(t){theta[1]}
a10 <- function(t){theta[7]}
a02 <- function(t){-0.5*theta[2]*theta[2]}
c02 <- function(t){theta[2]*theta[2]}
d02 <- function(t){theta[2]*theta[5]*theta[6]}
# V_t coefficients:
b00 <- function(t){theta[3]}
b01 <- function(t){-theta[4]}
e02 <- function(t){theta[2]*theta[5]*theta[6]}
f02 <- function(t){theta[5]*theta[5]}

theta.start <- c(0,1,1,1,1,0,0)
model_4 <- BiGQD.mle(X,time,mesh=10,theta=theta.start)
```

```
GQD.remove()
# R_t coefficients:
a00 <- function(t){theta[1]}
a02 <- function(t){-0.5*theta[2]*theta[2]}
c02 <- function(t){theta[2]*theta[2]}
d02 <- function(t){theta[2]*theta[5]*theta[6]}
# V_t coefficients:
b00 <- function(t){theta[3]}
b10 <- function(t){theta[7]}
b01 <- function(t){-theta[4]}
e02 <- function(t){theta[2]*theta[5]*theta[6]}
f02 <- function(t){theta[5]*theta[5]}
```

```
theta.start <- c(0,1,1,1,1,0,0)
model_5 <- BiGQD.mle(X,time,mesh=10,theta=theta.start)
```

Comparing the AIC and BIC values of eqn 81 and eqns 84 through 87, we conclude that the revised model still fits the data best whilst eqn 86 produces very similar AIC and BIC statistics.

```
GQD.aic(list(model_1,model_2,model_3,model_4,model_5))
```

```
##         Convergence p min.likelihood             AIC             BIC    N
## Model 1           0 6      -7852.212      -15692.424      -15661.381 1305
## Model 2           0 6      -7965.957  [=] -15919.914  [=] -15888.871 1305
## Model 3           0 5      -7948.554      -15887.109      -15861.239 1305
## Model 4           0 7      -7966.154      -15918.308      -15882.090 1305
## Model 5           0 7      -7936.321      -15858.642      -15822.424 1305
```

Looking at the parameter estimates of eqn 86, this can possibly be explained by the similarity of the fitted drift of eqns 84 and 86 since the parameter estimate of θ_7 appears to be not significantly different from 0.

```
GQD.estimates(model_4)
```

```
##          Estimate Lower_95 Upper_95
## theta[1]    0.014   -0.394    0.422
## theta[2]    4.254    4.094    4.414
## theta[3]    0.094    0.066    0.122
## theta[4]    3.042    1.950    4.133
## theta[5]    1.746    1.683    1.809
## theta[6]   -0.647   -0.677   -0.616
## theta[7]    0.017   -0.043    0.077
```

7.5. Model selection for 2D diffusions via DIC

Although stochastic volatility models have been used with great success in financial contexts, the application of non-linear diffusion models extend to other fields of science as well. Naturally, the context in which such models are applied present their own challenges. Indeed, finance presents a somewhat ideal problem set in the sense that data can be sourced at very high resolution for regularly spaced observations over long periods of time. In fields such as ecology, the data is often recorded irregularly and rarely exceed monthly observation frequencies (see for example Varughese (2011) and Varughese and Pienaar (2013)). That said, with the rapidly changing landscape of ecological monitoring, such datasets are likely to improve over time. Perhaps the principal difference between financial data and *naturally* occurring processes is the presence of dominating seasonal effects such as changes in temperature and/or long term growth trends. For these purposes we have developed the **DiffusionRgqd** package to accommodate time-inhomogeneous coefficients for generalized quadratic diffusion processes.

In order to demonstrate the ability of the package to identify and distinguish between temporal patterns for time inhomogeneous models we set up experimental data by simulating trajectories from a time-inhomogeneous bivariate GQD with stochastic volatility. For purposes of the experiment we shall simulate a target SDE:

$$\begin{aligned} dX_t &= (1.0(7.5 - X_t) + 1.5Y_t)dt + 0.5\sqrt{X_t Y_t}dB_t^{(1)} \\ dY_t &= (1.5(5 - Y_t) + 3\sin(0.25\pi t))dt + 0.25\sqrt{Y_t}dB_t^{(2)}. \end{aligned} \tag{88}$$

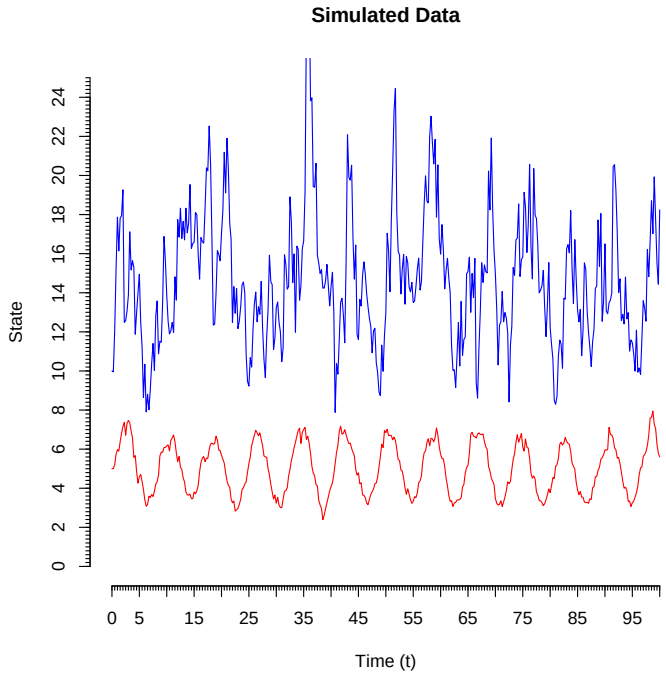

Figure 8: Simulated trajectory of eqn 88. The X_t trajectory (blue) exhibits periodic increases in state and volatility in conjunction with Y_t (red). The trajectory of Y_t exhibits dominating sinusoidal dynamics with little volatility.

We simulate eqn 88 by performing Euler-Maruyama updates of the target diffusion and recording the trajectory at the desired time points. For the current example updates were made at a resolution of $\Delta = 1/2000$ after which every 500-th update was recorded for a total of 401 observations, thus resulting in equidistant transition horizons of $1/4$ time units. For convenience we have included a simulated trajectory of the target model in the package's datasets which may be called into the workspace in standard fashion using the `data()` command:

```
# Call a simulated dataset into the workspace:
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=F)
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)
```

Figure 8 shows a plot of the resulting time-series. For purposes of the experiment we shall fit three competing models of which one is the true data generating process. The competing models are chosen to have superficially similar dynamics. For brevity we shall assume that we know from the outset that the process exhibits sinusoidal dynamics with known periodicity. This is not unrealistic as in practice it is usually clear from the context what the periodicity of the process is relative to the time scale on which the data were observed. For the first model,

we shall assume sinusoidal time-dependence for the drift of the y-dimension with interaction occurring only through the diffusion terms:

$$\begin{aligned} dX_t &= (\theta_1\theta_2 - \theta_1 X_t)dt + \theta_3\sqrt{X_t Y_t}dB_t^{(1)} \\ dY_t &= (\theta_4\theta_5 - \theta_4 Y_t + \theta_7 \sin(0.25\pi t))dt + \theta_6\sqrt{Y_t}dB_t^{(2)}. \end{aligned} \tag{89}$$

For the second model we assume that the X_t and Y_t trajectories arise independently with the oscillatory dynamics resulting purely from time-inhomogeneity:

$$\begin{aligned} dX_t &= (\theta_1\theta_2 - \theta_1 X_t)dt + \sqrt{\theta_7(1+\sin(0.25\pi t)) + \theta_3^2 X_t}dB_t^{(1)} \\ dY_t &= (\theta_4\theta_5 - \theta_4 Y_t + \theta_8 \sin(0.25\pi t))dt + \theta_6\sqrt{Y_t}dB_t^{(2)}. \end{aligned} \tag{90}$$

Finally, for the third candidate we assume the true model:

$$\begin{aligned} dX_t &= (\theta_1\theta_2 - \theta_1 X_t + \theta_7 Y_t)dt + \theta_3\sqrt{X_t Y_t}dB_t^{(1)} \\ dY_t &= (\theta_4\theta_5 - \theta_4 Y_t + \theta_8 \sin(0.25\pi t))dt + \theta_6\sqrt{Y_t}dB_t^{(2)}. \end{aligned} \tag{91}$$

In R we can specify the first candidate model as per the usual GQD syntax:

```
# Define the coefficients of a proposed model:
a00 <- function(t){theta[1]*theta[2]}
a10 <- function(t){-theta[1]}
c11 <- function(t){theta[3]*theta[3]}

b00 <- function(t){theta[4]*theta[5]+theta[7]*sin(0.25*pi*t)}
b01 <- function(t){-theta[4]}
f01 <- function(t){theta[6]*theta[6]}
```

After the model has been specified we may infer parameter estimates using the `BiGQD.mcmc()` routine. As outlined in Section 5, `BiGQD.mcmc()` sets up a computationally efficient likelihood evaluation routine in `C++` code which is subsequently called within a Metropolis-Hastings algorithm in order to produce sample chain of the model parameters under a given model specification. The algorithm as implemented by `BiGQD.mcmc()` follows:

1. Given some starting parameter vectors $\theta_{p\times 1}$ and $\sigma^{\theta}_{p\times 1}$, set $\theta^{old} = \theta$ and
2. propose an update for the parameter vector by setting

$$\theta_i^{new} = \theta_i^{old} + Z_{\sigma_i^\theta} \tag{92}$$

   for all $i = 1, 2, \ldots p$, where $Z_{\sigma_i^\theta}$ are $\mathrm{N}(0, (\sigma_i^\theta)^2)$ random deviates.
3. Subsequently, evaluate the ratio:

$$R = \frac{\prod_{i=1}^{N-1} f_{\theta^{new}}(\mathbf{X}_{t_{i+1}}|\mathbf{X}_{t_i})\pi(\theta^{new})}{\prod_{i=1}^{N-1} f_{\theta^{old}}(\mathbf{X}_{t_{i+1}}|\mathbf{X}_{t_i})\pi(\theta^{old})} \tag{93}$$

   where $\pi(\theta)$ denotes a prior density on the parameter vector.
4. Then accept the proposed move with probability $\max(R, 0)$. That is set

$$\theta^{old} = \begin{cases} \theta^{new} & \text{if } \max(R,0) > u \\ \theta^{old} & \text{otherwise,} \end{cases} \tag{94}$$

   where u is a $\mathrm{U}(0, 1)$ random deviate.

5. Return to step 1.

In R, the parameters of the algorithm are passed as arguments to `BiGQD.mcmc()`. For example, the starting parameters corresponding to the vector θ may be set using the `theta` argument, while the proposal standard deviations σ^{θ} may be set using the `sds` argument. Other arguments pertaining the density approximation such as the time-mesh step size and density type may be specified as with `BiGQD.mle()` in the previous example. The algorithm is then repeated a specified number of times as per the argument `updates`.

Apart from the parametric nuances introduced by the Metropolis-Hastings algorithm, another point that warrants attention is the inclusion of prior densities. For all of the models above we shall assume prior densities on the parameters θ_1 and θ_4 with $\theta_1 \sim \mathrm{N}(1, 5^2)$ and $\theta_4 \sim \mathrm{N}(1, 5^2)$. These densities can be included in the R environment by defining a function named `priors()` (a function name that will be recognized by `BiGQD.mcmc()` in similar fashion to the model coefficients) taking the vector `theta` as an argument. The function body can then be written as a product of the desired prior densities using standard R density functions such as `dnorm()`, `dgamma()` etc. or any other user defined density function:

```
# Place some prior distributions on theta[1] and theta[4]:
priors <- function(theta){dnorm(theta[1],1,5)*dnorm(theta[4],1,5)}
```

After defining a model and the prior densities to be used in the analysis we can make an initial run of the M-H algorithm by providing a starting parameter vector, standard deviation vector for the proposal densities and the number of M-H updates to perform:

```
# Some starting parameters for the MCMC procedure:
mesh    <- 10                          # Number of mesh points per transition.
updates <- 150000                      # Number of updates for each chain.
burns   <- 50000                       # Number of discarded updates.
X <- cbind(Xt,Yt)                      # Combine the data into a column matrix .
th <- c(5,5,5,5,5,5,5)                # Some starting values for the parameters.

# Define some proposal std. devs. and run the MH-algorithm:
par.sds <- c(0.22,0.30,0.02,0.11,0.04,0.01,0.21)
m1 <- BiGQD.mcmc(X,time,mesh,th,par.sds,updates,burns)
```

When the calculations are complete, `BiGQD.mcmc()` will return a list with the resulting MCMC chain, the history of acceptance rates and other information pertaining to the model such as DIC statistics. By default, `BiGQD.mcmc()` will also make a trace plot of the resulting Markov chain as illustrated in Figure 9.

If the resulting trace-plot is satisfactory estimates for the model can be calculated by passing the model object to the `GQD.estimates()` function as in the previous example. `GQD.estimates()` will recognize the model as MCMC output and calculate parameter estimates by discarding the first `burns` iterations and thinning the chain by a specified amount (using the argument `thin`). Subsequently, the resulting parameter estimates, 90% credibility intervals and correlation matrix is printed to the console. For diagnostic purposes an ACF-plot for each element of the parameter chain is plotted using the same colour-coding as the trace-plot produced by `BiGQD.mcmc()`:

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

```
##          Estimate Lower_90 Upper_90 | cor[,1] cor[,2] cor[,3] cor[,4] cor[,5] cor[,6] cor[,7]
```

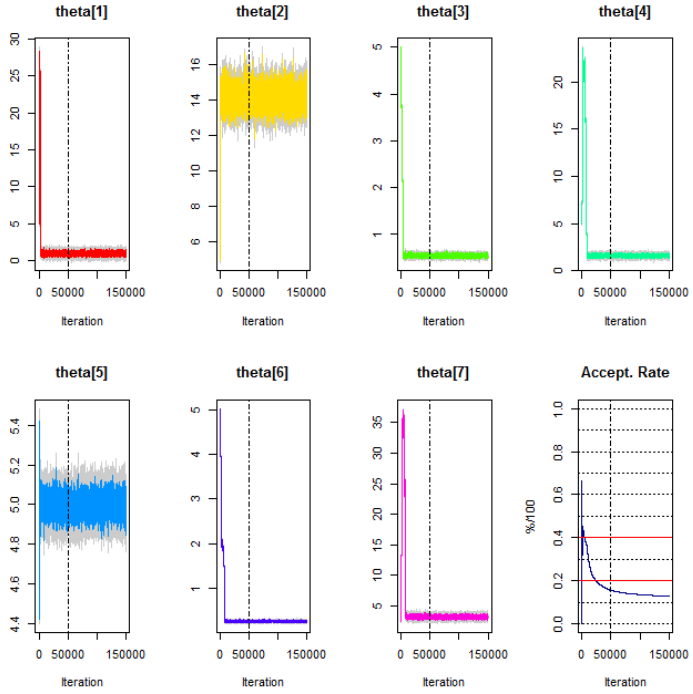

Figure 9: A trace plot of the parameter updates for model 89 generated by `BiGQD.mcmc()`. A trace plot is drawn for each parameter of the target model in conjunction with regected proposals (light gray). The vertical dashed lines indicate the end of the burn in period. In addition, a trace plot of the acceptance rate of the RWMH algorithm is drawn with guide lines ranging from 0% to 100% in increments of 10% (20% and 40% highlighted in red).

```
## theta[1]    0.955    0.707    1.218 |      1   -0.14    0.37    0.04   -0.08    0.08    0.05
## theta[2]    14.15   13.315   14.981 |  -0.14       1   -0.06   -0.01    0.07    0.07   -0.01
## theta[3]    0.539    0.507    0.574 |   0.37   -0.06       1    0.01    -0.1       0   -0.01
## theta[4]    1.591    1.447     1.74 |   0.04   -0.01    0.01       1   -0.05    0.17    0.84
## theta[5]        5    4.938    5.064 |  -0.08    0.07    -0.1   -0.05       1    0.04    0.04
## theta[6]    0.264     0.25    0.281 |   0.08    0.07       0    0.17    0.04       1    0.17
## theta[7]    3.222     2.97    3.475 |   0.05   -0.01   -0.01    0.84    0.04    0.17       1
```

Should a more detailed analysis of the parameter chains be required, this can easily done using using packages such as **coda** (Plummer, Best, Cowles, and Vines 2006) which provide a comprehensive set of MCMC analysis tools. For example, `mcmc.coda <- mcmc(m1$par.matrix)` creates an `mcmc` object that will be recognized by routines in the **coda** library.

In order to assess convergence of the MH-algorithm it is standard practice to run multiple chains from different starting points. Although this is typically achieved by manually running the MH-algorithm multiple times the practice warrants at least some degree of automation. For these purposes we have included a fail-over variable in the output of the `BiGQD.mcmc()` function. That is should `BiGQD.mcmc()` fail for any reason, whether it be unrealistic initial values or some numerical failure, a variable is returned in the output list indicating that failure has occurred. This variable may thus be monitored within a loop, making it easy to make repeated MCMC runs automatically. In addition to the fail-over variable we have included a tagging argument which can be used to mark an instance of a call to `BiGQD.mcmc()`. In the current example each model is assigned the tag '`Model A_run_()`' with the convention that `()` keeps track of which run the object pertains to. Using randomly chosen starting points one can perform a number of MCMC runs and store the output in a list object:

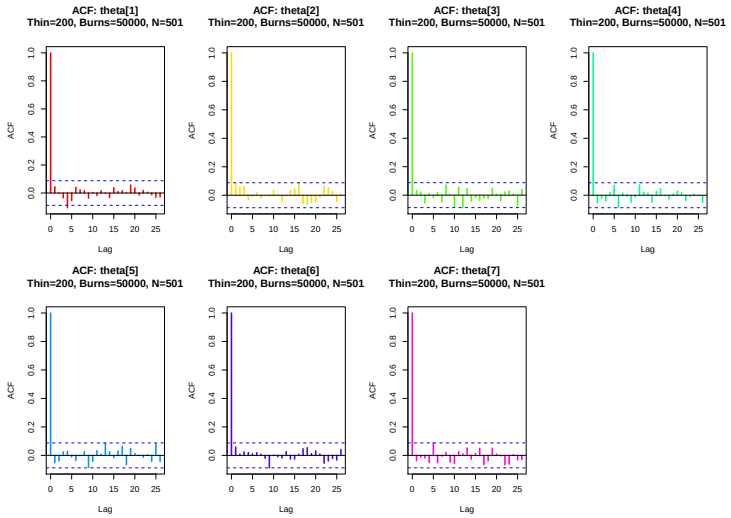

Figure 10: Autocorrelation function (ACF) plot for the thinned parameter chain of eqn 89. The ACF-plot is produced when `GQD.estimates()` is called.

```
# Run the MCMC procedure M=4 times for each model:
M  <- 4
SaveOutput  <- list() # Create some  storage for model objects.
M.counter   <- 1      # Create a count variable.
Tot.counter <- 1      # An overall count variable.
kill.count  <- 1      # Set fail-over counter.

while((M.counter<=M)&(kill.count<20))
{
  # Make a new run from a random starting point each time:
  th <- runif(7,1,10)
  m1 <- BiGQD.mcmc(X,time,mesh,th,par.sds,updates,burns,
                   Tag=paste0('Model_A_run_',M.counter))

  # If the chain does not fail, store the output. If the chain fails,
  # keep track of the number of failures:
  if(!m1$failed.chain)
  {
    SaveOutput[[Tot.counter]]  <-  m1
    Tot.counter <- Tot.counter+1
    M.counter   <- M.counter +1
    kill.count  <- 1
  }else
  {
    kill.count <- kill.count+1
  }
}
```

By repeating this process we can conduct inference on the second model and third model as well, and subsequently fill the `SaveOutput` list with consecutive MCMC runs for ach model. Once

all the MCMC runs are concluded we may summarise the output via the `GQD.dic()` function. `GQD.dic()` collates DIC statistics and other relevant information from a list of `BiGQD.mcmc()` output objects in a concise form. For example, when the routine is applied to the `SaveOutput` variable. The minimum DIC model is then indicated by a `[=]` marker.

```
GQD.dic(SaveOutput)
```

```
##                Elapsed_Time Time_Homogeneous       p           DIC      pD   N
## Model_A_run_1      00:09:14                No    7.00       1640.18    7.10 401
## Model_A_run_2      00:08:49                No    7.00       1640.04    7.04 401
## Model_A_run_3      00:08:34                No    7.00       1640.04    7.04 401
## Model_A_run_4      00:08:04                No    7.00       1640.11    7.07 401
## Model_B_run_1      00:09:42                No    8.00       1649.66    7.86 401
## Model_B_run_2      00:09:21                No    8.00       1649.65    7.83 401
## Model_B_run_3      00:09:21                No    8.00       1650.00    8.03 401
## Model_B_run_4      00:09:21                No    8.00       1650.10    8.07 401
## Model_C_run_1      00:08:21                No    8.00  [=] 1618.95    7.78 401
## Model_C_run_2      00:08:28                No    8.00       1619.48    8.05 401
## Model_C_run_3      00:08:23                No    8.00       1619.64    8.12 401
## Model_C_run_4      00:08:34                No    8.00       1619.67    8.14 401
```

From the DIC calculations the algorithm does indeed narrow down the correct model. Finally, we can compare the estimated parameters of eqn 91 to the true model parameters by applying the `GQD.estimates()` function to the final MCMC run:

```
GQD.estimates(SaveOutput[[12]],thin=200)
```

```
         Estimate Lower_90 Upper_90 | cor[,1] cor[,2] cor[,3] cor[,4] cor[,5] cor[,6] cor[,7] cor[,8]
theta[1]    1.162    0.885    1.477 |       1    0.31    0.44    0.05   -0.08    0.04    0.33    0.03
theta[2]    6.873    4.176    9.752 |    0.31       1    0.12    0.04    0.04    0.04   -0.75    0.01
theta[3]     0.54    0.507    0.578 |    0.44    0.12       1    0.09   -0.08    0.08    0.17    0.06
theta[4]    1.584    1.434     1.75 |    0.05    0.04    0.09       1   -0.03     0.2   -0.01    0.87
theta[5]    4.997     4.94    5.059 |   -0.08    0.04   -0.08   -0.03       1    0.01   -0.08    0.05
theta[6]    0.264    0.247    0.281 |    0.04    0.04    0.08     0.2    0.01       1       0    0.18
theta[7]    1.787    1.124    2.422 |    0.33   -0.75    0.17   -0.01   -0.08       0       1       0
theta[8]    3.206    2.958    3.507 |    0.03    0.01    0.06    0.87    0.05    0.18       0       1
```

Given the relatively low sample resolution of the simulated dataset, the parameter estimates compare favourably with the true parameter set. Interestingly, the credibility intervals for the estimates of θ_2 and θ_7 are quite wide, whilst those for the diffusion parameters θ_3 and θ_6 are relatively narrow. This suggests that for a diffusion process with dynamics governed by eqn 88 the current observed trajectory is relatively short, making it difficult to accurately measure the drift dynamics. Indeed, this is an important practical consideration when conducting inference on diffusion models: Since one can only partially observe a single trajectory of the process over a finite horizon, the quality of inference that can be made on any given model is necessarily dictated by the regularity with which observations are made and the length of the observation horizon. Thus, for simple models such as eqn 89, there is often more certainty about what parameter values could have likely resulted in the observed trajectory than for more complicated models such as eqns 90 and 91.

7.6. Scalar first passage time problems

Another useful application of **DiffusionRgqd** package is the approximation of first passage time densities via the `GQD.TIpassge()` function. The `GQD.TIpassage()` function uses the

same GQD interface as other functions in the package and operates in similar fashion to the `GQD.density()` function. However, since `GQD.TIpassage()` relies on calculating transitional density approximations for a large number of initial values in combination with the recursive updating algorithm of eqn 59, its internal workings have more in common with the `GQD.mle()` and `GQD.mcmc()` functions, where computationally optimized solutions with respect to the cumulant truncation procedure are constructed in C++ which is subsequently executed in R. As an introduction to the function we first compare the `GQD.TIpassage()` function to an existing R package for calculating first passage time densities for Gaussian diffusions and subsequently investigate the properties of the first passage time density of a non-linear diffusion transiting through a moving barrier.

An excellent package for the analysis of first passage time problems is the **fptdApprox** (Román-Román, Serrano-Pérez, and Torres-Ruiz 2014) package. Since **fptdApprox** can very effectively handle first passage time problems for diffusions with analytically tractable transitional densities we use it to compare some of the results from the **DiffusionRgqd** package. Consider for example a diffusion process with SDE:

$$dX_t = 0.5(5 - X_t)dt + dB_t, \tag{95}$$

with $X_0 = 3$. For purposes of calculating a first passage time consider then also a continuous barrier

$$\lambda_t = 5 + 0.25\sin(2\pi t). \tag{96}$$

Under the **fptdApprox** package we may use the `Approx.fpt.density()` function in order to approximate the first passage time density. The interface requires that one define a diffusion process by configuring an object that consists of expressions giving the drift, diffusion and transitional density of the model at hand. The resulting object is then used by the `Approx.fpt.density()` function to approximate the first passage time density. More formally:

```
library(fptdApprox)
# Under `fptdApprox':
# Define the diffusion process and give its transitional density:
OU <- diffproc(c("alpha*x + beta","sigma^2",
"dnorm((x-(y*exp(alpha*(t-s)) - beta*(1 - exp(alpha*(t-s)))/alpha))/
(sigma*sqrt((exp(2*alpha*(t-s)) - 1)/(2*alpha))),0,1)/
(sigma*sqrt((exp(2*alpha*(t-s)) - 1)/(2*alpha)))",
"pnorm(x, y*exp(alpha*(t-s)) - beta*(1 - exp(alpha*(t-s)))/alpha,
sigma*sqrt((exp(2*alpha*(t-s)) - 1)/(2*alpha)))"))
# Approximate the first passgage time density for OU, starting in X_0 = 3
# passing through 5+0.25*sin(2*pi*t) on the time interval [0,10]:
res1 <- Approx.fpt.density(OU, 0, 10, 3,
"5+0.25*sin(2*pi*t)", list(alpha=-0.5,beta=0.5*5,sigma=1))
```

Using the **DiffusionRgqd** package we begin by defining the model as per usual according to the GQD framework. Then we call the `GQD.TIpassage()` function and provide it with the parameters of the first passage time problem. Note that `GQD.TIpassage()` circumvents the need to specify the transitional density as it will use the model coefficients to recognize and construct the appropriate numerical approximation of the required transitional densities. In present form, for computational purposes, the `GQD.TIpassage()` function evaluates eqn 59 for constant barriers only. However, this is not a limiting constraint as it can be shown that for any barrier function that can be decomposed as

$$\lambda_t = \phi + f(t) \tag{97}$$

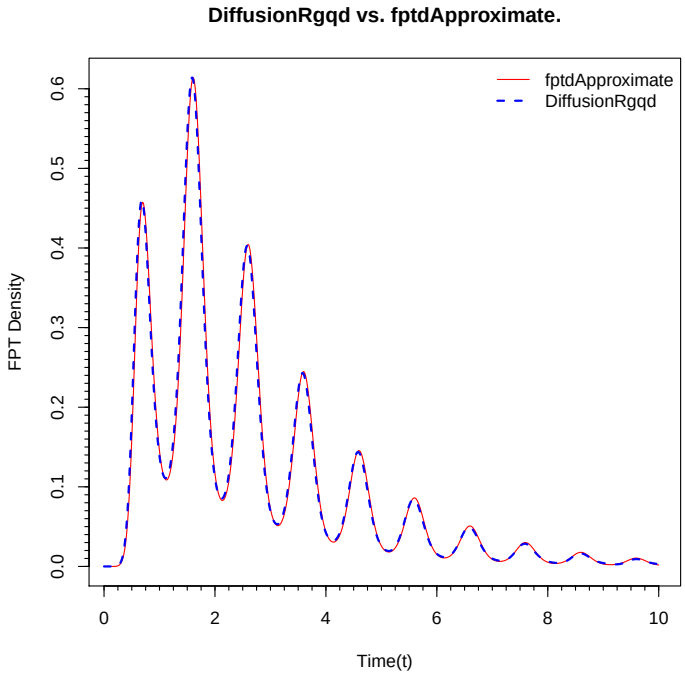

Figure 11: Approxiate first passage times for eqn 95 transiting through a sinusoidal barrier calculated using the **fptdApprox** package (red) and **DiffusionRgqd** package (blue, dashed). The approximate solutions produce nearly identical results.

where $f(t)$ is continuous, the first passage time problem $\{Y_t = X_t - f(t) \uparrow \phi | X_s - f(s)\}$ is equivalent to that of $\{X_t \uparrow \lambda_t | X_s\}$. All that remains is to calculate the dynamics of Y_t under Ito's lemma.

```
library(DiffusionRgqd)
# Under `DiffusionRgqd':
# Define the diffusion process
G0=function(t){0.5*5-0.5*pi*cos(2*pi*t)-0.5*0.25*sin(2*pi*t)}
G1=function(t){-0.5}
Q0=function(t){1}
# Approximate the first passgage time density for OU, starting in X_0 = 3
# passing through 5+0.25*sin(2*pi*t) on the time interval [0,10]:
res2 <- GQD.TIpassage(Xs=3,B=5,s=0,t=10,delt=1/200)
```

By plotting the subsequent approximations we infer from figure 11 that the approximations are near identical.

```
 # Let's compare the resulting densities:
plot(res1$y~res1$x,type='l',col='red',xlab='Time(t)',ylab='FPT Density',
main= 'DiffusionRgqd vs. fptdApproximate.')
lines(res2$density~res2$time,col='blue',lwd=2,lty='dashed')
legend('topright',lty=c('solid','dashed'),col=c('red','blue'),legend=
c('fptdApproximate','DiffusionRgqd'),lwd=c(1,2),bty='n')
axis(2,at=seq(0,0.6,by=1/10/10),tcl=-0.2,labels=NA)
axis(1,at=seq(0,10,by=1/4),tcl=-0.2,labels=NA)
```

Note that since functions in the **DiffusionRgqd** always assumes that a numerical solution is required, it is possible for this example, where the transition density is available analytically, to produce a solution that is more efficiently calculable. However, the aim of the package is tackle problems with intractable dynamics under the GQD framework, where transition densities are rarely analytically available. Furthermore, as with the inference procedures, we again have a great deal of freedom for specifying a first passage time problem and subsequently calculating an accurate approximate solution. Consider for example a diffusion with SDE:

$$dX_t = \theta_1 X_t(10 + 0.2\sin(2\pi t) + 0.3\sqrt{t}(1 + \cos(3\pi t)) - X_t)dt + 0.05X_t dB_t, \tag{98}$$

with $X_1 = 8$, θ_1 a fixed parameter, and a constant barrier $\lambda_t = 12$. Note that for eqn 98, no analytical solution for the transition density exists. Consequently, we can no longer make use of the **fptdApprox** package in order to generate the desired first passage time density, albeit for comparative purposes. As such we compare the resulting approximation to a simulated first passage time density for eqn 98 transiting through $\lambda_t = 12$. Perhaps the simplest algorithm for achieving this is to simulate numerous trajectories of eqn 98 and record the times at which these trajectories cross λ_t. In R this can be achieved for example by:

```
# Simulate the first passage time density:
theta <- 0.5
mu <- function(X,t){
theta[1]*(10+0.2*sin(2*pi*t)+0.3*sqrt(t)*(1+cos(3*pi*t)))*X-theta[1]*X^2}
sigma <- function(X,t){sqrt(0.1)*X}
simulate.fpt=function(N,S,B,delt)
{
X=rep(S,N)
Ndim=N
time.vector=rep(0,N)
t=1
k1=0
k2=0
pb=txtProgressBar(min=0,max=Ndim,initial=0,char = "=",width = 59, style = 3)
while(Ndim>=1)
{
k1=k1+1
setTxtProgressBar(pb,N-Ndim)
X=X+mu(X,t)*delt+sigma(X,t)*rnorm(Ndim,sd=sqrt(delt))
# Check if the barrier is crossed and keep trajectories that
# have survived:
I0=X<B
X=X[I0]
count=sum(!I0)
Ndim=length(X)
t=t+delt
if(count>0)
{
time.vector[k2+1:count]=t
k2=k2+count
}
}
close(pb)
return(time.vector)
```

```
}

res.sim <- simulate.fpt(500000,8,12,1/2000)
```

where we have set $\theta_1 = 0.5$. It is important to note that this scheme is subject to bias since the finite step size used in the simulation implies that the scheme fails to account for trajectories that may have already crossed λ_t yet are recorded below λ_t subsequent to each update (Giraudo and Sacerdote 1999). However this is a somewhat technical matter which falls outside of the scope of the present paper. As such we have chosen the parameters of the diffusion in such a way that the scheme suffices for visual comparison. Using `GQD.TIpassage()` we can again analyse the first passage time problem by defining the model in terms of GQD-coefficients:

```
GQD.remove()
```

```
## [1] "Removed :  G0 G1 Q0"
```

```
# 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))
```

Note that we have parametrised the coefficients using the reserved variable `theta` in similar fashion to `BiGQD.mcmc()`. This allows one to calculate the first passage time density for various parameter values without having to re-compile C++ code repeatedly. For example, we can evaluate the first passage time problem for eqn 98 for θ_1 running from 0.1 to 0.5:

```
# Comapre the numerical solution to the simulated density:
hst=hist(res.sim,freq=F,plot=T,breaks=100)
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)
lines(hst$density~c(hst$mids-diff(hst$mids)[1]/2),type='s',lty='solid',lwd=2)
# Change the parameter and see the effect on the f.p.t. density.
th.seq=seq(0.1,0.5,1/20)
for(i in 2:length(th.seq))
{
  res3=GQD.TIpassage(8,12,1,4,1/100,,theta=c(th.seq[i]))
  lines(res3$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]))
```

Figure 12 illustrates the effect of varying θ_1: As the value of the parameter decreases the time taken to reach and exceed the barrier increases and the effect of the time dependent terms become less prominent. This makes sense since θ_1 in some sense dictates the 'speed' at which the process drifts toward the equilibrium line $10 + 0.2\sin(2\pi t) + 0.3\sqrt{t}(1 + \cos(3\pi t))$. Since this line

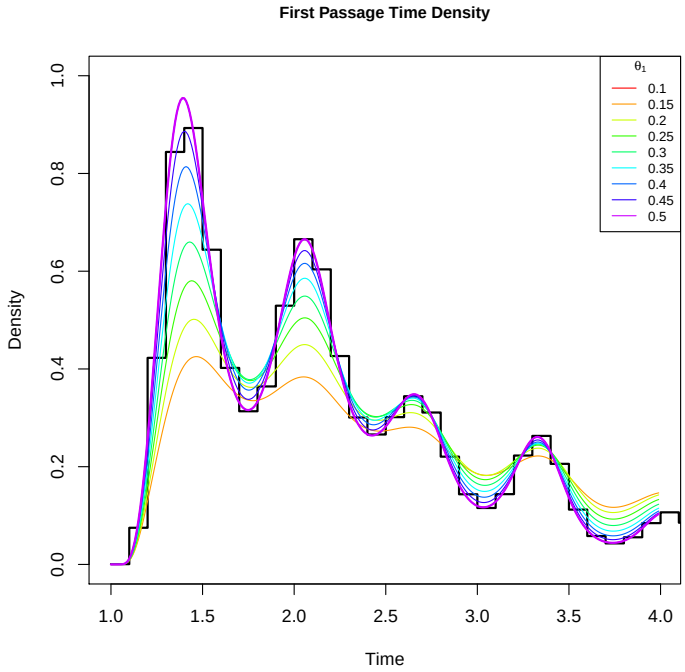

Figure 12: First passage time density of eqn 98 through $\lambda_t = 11$ for various values of θ_1. Superimposed (black outline) is a histogram of simulated first passage times for $\theta_1 = 0.5$, calculated from 500000 simulated trajectories using a step size increment of $1/2000$ time units.

starts out above the starting point $X_1 = 8$, θ_1 will dictate how intensely the drift of the process pulls it towards the barrier. Finally, comparing the approximate first passage time density to that of the simulated first passage time, it can be seen that the approximation is indeed valid.

8. Summary

Diffusion processes are a flexible tool for modelling phenomena from various fields of science. By making use of a computationally efficient algorithm for calculating the transition density of a diffusion process we develop a class of scalar and bivariate quadratic diffusion processes termed the generalized quadratic diffusion processes for which we can calculate accurate approximations to the likelihood of a discretely observed processes. Using this framework we develop the **DiffusionRgqd** package: A collection of tools for performing inference and analysis on time-inhomogeneous quadratic diffusion processes. By identifying computational redundancies within the generalized quadratic class it is possible to define an algorithm whereby computational overhead of calculating the likelihood of a given model can be minimized. Combining this with the speed of the C++ language within R, it is possible to perform likelihood based inference on discretely observed processes very efficiently in a non-parallel computing environment. In addition to the inference procedures, we develop accompanying routines for calculating approximations to the transition density of a given diffusion as well as the first passage time density of a scalar transition density transiting through a barrier. These peripheral functions can be used to investigate the dynamics of a given model directly and may aid in the formulation of appropriate diffusion models prior to conducting inference on an observed process or the analysis of a model under fitted parameters.

We demonstrate how the package may be used to analyse various time-inhomogeneous non-linear diffusion models and how one may conduct model selection for discretely observed diffusions by

comparing standard goodness of fit statistics such as the AIC, BIC and DIC for a plethora of models. Using the **DiffusionRgqd** package it is possible analyse real world data using non-linear, time-inhomogeneous diffusion models with minimal mathematical input over and above defining defining a model of interest. By providing a simple interface to the relatively complicated mathematical objects which are stochastic differential equations, it is hoped that the package will make diffusion models more accessible to areas of science where diffusion processes are less frequently used, and perhaps make it possible to conduct analysis that has previously been limited by the analytical intractability of non-linear diffusion processes.

Acknowledgements

Etienne Pienaar and Melvin Varughese were both funded by the National Research Foundation of South Africa.

References

Adler D, Murdoch D, *et al.* (2014). ***rgl**: 3D Visualization Device System (OpenGL).* R package version 0.93.996, URL http://CRAN.R-project.org/package=rgl.

Aït-Sahalia Y, *et al.* (2008). "Closed-form Likelihood Expansions for Multivariate Diffusions." *The Annals of Statistics*, **36**(2), 906–937.

Andrews GE, Askey R, Roy R (1999). *Special Functions*, volume 71. Cambridge University Press.

Atkinson KE (2008). *An Introduction to Numerical Analysis*. John Wiley & Sons.

Barndorff-Nielsen O, Cox DR (1979). "Edgeworth and Saddle-Point Approximations With Statistical Applications." *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 279–312.

Buonocore A, Nobile A, Ricciardi L (1987). "A New Integral Equation for the Evaluation of First-Passage-Time Probability Densities." *Advances in Applied Probability.*, **19**, 784–800.

Butcher J (2007). "Runge-Kutta Methods." *Scholarpedia*, **2**(9), 3147. revision #91735.

Cobb L, Koppstein P, Chen NH (1983). "Estimation and Moment Recursion Relations for Multimodal Distributions of the Exponential Family." *Journal of the American Statistical Association*, **78**(381), 124–130.

Cox J, Ingersoll J, Ross S (1985). "A Theory of the Term Structure of Interest Rates." *Econometrica*, **53**, 385–407.

Crank J, Nicolson P (1996). "A Practical Method for Numerical Evaluation of Solutions of Partial Differential Equations of the Heat-Conduction Type." *Advances in Computational Mathematics*, **6**(1), 207–226.

Eddelbuettel D, 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 Conference on Scientific Computing*.

Fehlberg E (1970). "Classical Fourth- and Lower Order Runge-Kutta Formulas With Stepsize Control and Their Application to Heat Transfer Problems." *Computing*, **6**(1), 61–71.

Giraudo MT, Sacerdote L (1999). "An Improved Technique for the Simulation of First Passage Times for Diffusion Processes." *Communications in Statistics-Simulation and Computation*, **28**(4), 1135–1163.

Gouriéroux C, Valéry P (2004). "Estimation of a Jacobi Process." *Preprint*, p. 116.

Hamdi S, Schiesser WE, Griffiths GW (2007). "Method of Lines." *Scholarpedia*, **2**(7), 2859.

Heston SL (1993). "A Closed-Form Solution for Options with Stochastic Volatility With Applications to Bond and Currency Options." *Review of Financial Studies*, **6**(2), 327–343.

Hoppensteadt F (2006). "Predator-Prey Model." *Scholarpedia*, **1**(10), 1563. revision #91666.

Huang X (2011). "Quasi-Maximum Likelihood Estimation of Discretely Observed Diffusions." *The Econometrics Journal*, **14**(2), 241–256.

Hurn A, Lindsay K, McClelland A (2014). "Estimating the Parameters of Stochastic Volatility Models Using Option Price Data." *Journal of Business & Economic Statistics*, (just-accepted), 00–00.

Jáimez RG, Román PR, Ruiz FT (1995). "A Note on the Volterra Integral Equation for the First-Passage-Time Probability Density." *Journal of applied probability*, pp. 635–648.

Platen E (1999). "An Introduction to Numerical Methods for Stochastic Differential Equations." *Acta Numerica*, **8**, 197–246.

Plummer M, Best N, Cowles K, Vines K (2006). "**CODA**: Convergence Diagnosis and Output Analysis for MCMC." *R News*, **6**(1), 7–11. URL http://CRAN.R-project.org/doc/Rnews/.

Renshaw E (2000). "Applying the Saddlepoint Approximation to Bivariate Stochastic Processes." *Mathematical Biosciences*, **168**(1), 57–75.

Ricciardi LM, Sacerdote L (1979). "The Ornstein-Uhlenbeck Process as a Model for Neuronal Activity." *Biological Cybernetics*, **35**(1), 1–9.

Román-Román P, Serrano-Pérez J, Torres-Ruiz F (2014). "More General Problems on First-Passage Times for Diffusion Processes: A New Version of the **fptdApprox** R Package." *Applied Mathematics and Computation*, **244**, 432–446.

Varughese MM (2011). "A Framework for Modelling Ecological Communities and Their Interactions With the Environment." *Ecological Complexity*, **8**(1), 105–112.

Varughese MM (2013). "Parameter Estimation for Multivariate Diffusion Systems." *Computational Statistics & Data Analysis*, **57**(1), 417–428.

Varughese MM, Pienaar EAD (2013). "Statistical Inference for a Multivariate Diffusion Model of an Ecological Time Series." *Ecosphere*, **4**(8), art104.

Varughese MM, Pienaar EAD (2015). "Computing the First Passage Time Density for Time-Homogeneous, Polynomial Diffusion Processes." *Technical report*.

Appendix

A. Normalization of the Pearson-system densities

Although the Pearson system offers a considerable amount of flexibility with respect to carrying moments into a valid density approximation, this flexibility comes at the cost of having to compute normalizing constants. Unfortunately analytical expressions for integrals of the kernels in eqns 35 - 38 over their respective support cannot in general be found for $m > 2$. As such we have to resort to numerical methods in order to normalize the Pearson densities. For example, for the Normal class we may employ the trapezoidal approximation:

$$\int_{-\infty}^{\infty} N(x|X_s)dx \approx \sum_{i=1}^{P-1} N(\tau_i|X_s)\rho_i\Delta \tag{99}$$

with the modifications $\tau_i = \frac{y_i}{1-(y_i)^2}e^{\alpha} + u_1$, $\rho_i = \frac{1+y_i^2}{(1-(y_i)^2)^2}e^{\alpha}$, where $P > 1$ is a positive integer with $y_i = -\mathcal{L} + 2i\Delta$ for $i = 0, 1, \ldots, P$.

Note that, in this particular instance, evaluation starts at (τ_1, ρ_1) and ends at (τ_{P-1}, ρ_{P-1}) since by definition $N(-\infty|X_s) = N(\infty|X_s) = 0$, thus negating the endpoints under the trapezoidal rule. The parameters $\mathcal{L}$ and Δ are in turn determined by the relations:

$$\mathcal{L} = -\frac{1}{2(u_1 - x^-)}(\sqrt{e^{2\alpha} + 4(u_1 - x^-)^2} - e^{\alpha}) \tag{100}$$

and

$$\Delta = \Big(-\frac{1}{2(u_1 - x^+)}(\sqrt{e^{2\alpha} + 4(u_1 - x^+)^2} - e^{\alpha}) - \mathcal{L}\Big)/P. \tag{101}$$

The parameters x^- and x^+ represent pre-defined lower and upper bounds on the integration range in the original coordinate system, i.e. the support of the diffusion. By evaluating the limits $\lim_{x^- \to -\infty} \tau$ and $\lim_{x^- \to \infty} \tau$, the behaviour of the infinite integral is preserved. However, in practice these limits will typically be made finite in order to avoid numerical underflow at the extremes of a given distribution. The role of the parameters α and P is to control the mesh spacing within the limits $[x^-, x^+]$. For a given number of mesh points P, increasing α will lower the concentration of mesh points around u_1, whilst increasing P increases the mesh resolution. Finally we note the important distinction that the resulting mesh under this scheme is in fact time dependent. This is due to the u_1 being included in the mesh translation. This, in conjunction with the exponential mesh spacing ensures that the numerical integration accumulates more information within 'dense' regions of the support and less information where there is little density, provided that it is unimodal. To see the effect of varying P and α parameters we follow up the example of Section 7.1 and use two sets of parameters for α and P:

```
# Normalization regime no. 1:
M1 <- GQD.density(Xs=initial,Xt=states,s=Tstart,t=Tmax,delt=increment,
                  Dtype='Normal', P = 100,alpha=1,lower = 1,upper = 20)

# Normalization regime no. 2:
M2 <- GQD.density(Xs=initial,Xt=states,s=Tstart,t=Tmax,delt=increment,
                  Dtype='Normal', P = 200,alpha=3,lower = 1,upper = 20)
```

Subsequently, we can extract the mesh used for the normalization in each case and plot the mesh points as they vary over time. The resulting plots are given in figure 13.

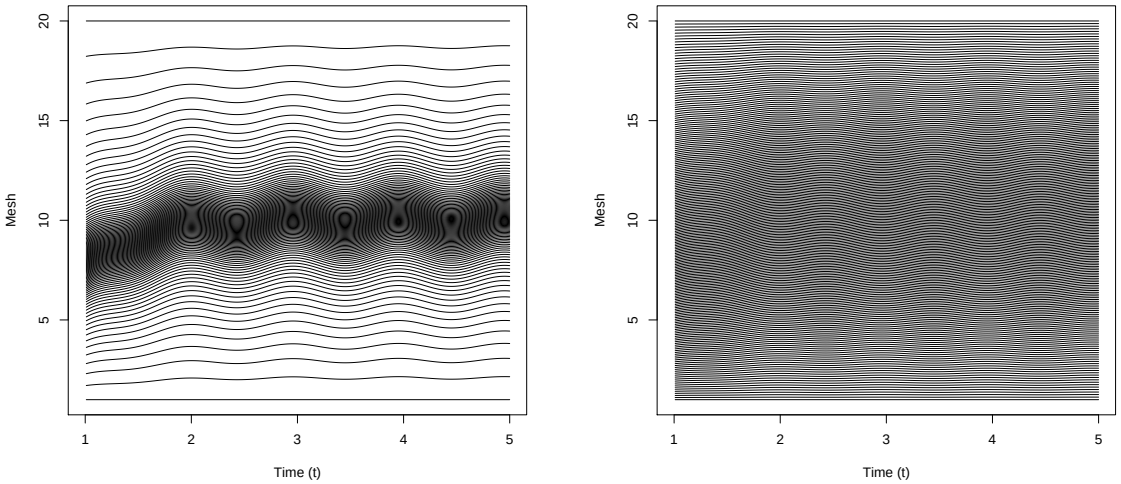

Figure 13: Various mesh structures used for normalising the 4-th order Noramal type Pearson density. Using $\alpha = 1$ for $P = 100$ points over the interval $[1, 20]$ (left) results in a mesh that is concentrated arounf the mean trajectory of the process and sparse closer to the extrmes whilst setting $\alpha = 3$ for $P = 200$ (right) gives a more uniform mesh.

```
plot(1,1,type= 'n',xlim=c(1,5),ylim=c(1,20),xlab='Time (t)',ylab = 'Mesh')
for(i in 1:100)
{
  lines(M1$mesh[i,]~M1$time)
}

plot(1,1,type= 'n',xlim=c(1,5),ylim=c(1,20),xlab='Time (t)',ylab = 'Mesh')
for(i in 1:200)
{
  lines(M2$mesh[i,]~M1$time)
}
```

	a_{00}	a_{10}	a_{20}	a_{01}	a_{02}	a_{11}
$\dot{\kappa}_{10}$	1	$+1\kappa_{10}$	$+1\kappa_{10}\kappa_{10} + 1\kappa_{20}$	$+1\kappa_{01}$	$+1\kappa_{01}\kappa_{01} + 1\kappa_{02}$	$+1\kappa_{11} + 1\kappa_{10}\kappa_{01}$
$\dot{\kappa}_{20}$	.	$+2\kappa_{20}$	$+2\kappa_{10}\kappa_{20} + 2\kappa_{20}\kappa_{10} + 2\kappa_{30}$	$+2\kappa_{11}$	$+2\kappa_{01}\kappa_{11} + 2\kappa_{11}\kappa_{01} + 2\kappa_{12}$	$+2\kappa_{21} + 2\kappa_{10}\kappa_{11} + 2\kappa_{20}\kappa_{01}$
$\dot{\kappa}_{30}$	.	$+3\kappa_{30}$	$+3\kappa_{10}\kappa_{30} + 6\kappa_{20}\kappa_{20} + 3\kappa_{30}\kappa_{10} + 3\kappa_{40}$	$+3\kappa_{21}$	$+3\kappa_{01}\kappa_{21} + 6\kappa_{11}\kappa_{11} + 3\kappa_{21}\kappa_{01} + 3\kappa_{22}$	$+3\kappa_{31} + 3\kappa_{10}\kappa_{21} + 6\kappa_{20}\kappa_{11} + 3\kappa_{30}\kappa_{01}$
$\dot{\kappa}_{40}$	.	$+4\kappa_{40}$	$+4\kappa_{10}\kappa_{40} + 12\kappa_{20}\kappa_{30} + 12\kappa_{30}\kappa_{20} + 4\kappa_{40}\kappa_{10}$	$+4\kappa_{31}$	$+4\kappa_{01}\kappa_{31} + 12\kappa_{11}\kappa_{21} + 12\kappa_{21}\kappa_{11} + 4\kappa_{31}\kappa_{01}$	$+4\kappa_{10}\kappa_{31} + 12\kappa_{20}\kappa_{21} + 12\kappa_{30}\kappa_{11} + 4\kappa_{40}\kappa_{01}$
$\dot{\kappa}_{01}$	.	.	.	.	.	.
$\dot{\kappa}_{02}$	.	.	.	.	.	.
$\dot{\kappa}_{03}$	.	.	.	.	.	.
$\dot{\kappa}_{04}$	.	.	.	.	.	.
$\dot{\kappa}_{11}$	.	$+1\kappa_{11}$	$+1\kappa_{10}\kappa_{11} + 1\kappa_{11}\kappa_{10} + 1\kappa_{21}$	$+1\kappa_{02}$	$+1\kappa_{01}\kappa_{02} + 1\kappa_{02}\kappa_{01} + 1\kappa_{03}$	$+1\kappa_{12} + 1\kappa_{10}\kappa_{02} + 1\kappa_{11}\kappa_{01}$
$\dot{\kappa}_{12}$	.	$+1\kappa_{12}$	$+1\kappa_{10}\kappa_{12} + 2\kappa_{11}\kappa_{11} + 1\kappa_{12}\kappa_{10} + 1\kappa_{22}$	$+1\kappa_{03}$	$+1\kappa_{01}\kappa_{03} + 2\kappa_{02}\kappa_{02} + 1\kappa_{03}\kappa_{01} + 1\kappa_{04}$	$+1\kappa_{13} + 1\kappa_{10}\kappa_{03} + 2\kappa_{11}\kappa_{02} + 1\kappa_{12}\kappa_{01}$
$\dot{\kappa}_{21}$	.	$+2\kappa_{21}$	$+2\kappa_{10}\kappa_{21} + 2\kappa_{11}\kappa_{20} + 2\kappa_{20}\kappa_{11} + 2\kappa_{21}\kappa_{10} + 2\kappa_{31}$	$+2\kappa_{12}$	$+2\kappa_{01}\kappa_{12} + 2\kappa_{02}\kappa_{11} + 2\kappa_{11}\kappa_{02} + 2\kappa_{12}\kappa_{01} + 2\kappa_{13}$	$+2\kappa_{22} + 2\kappa_{10}\kappa_{12} + 2\kappa_{11}\kappa_{11} + 2\kappa_{20}\kappa_{02} + 2\kappa_{21}\kappa_{01}$
$\dot{\kappa}_{22}$	.	$+2\kappa_{22}$	$+2\kappa_{10}\kappa_{22} + 4\kappa_{11}\kappa_{21} + 2\kappa_{12}\kappa_{20} + 2\kappa_{20}\kappa_{12} + 4\kappa_{21}\kappa_{11} + 2\kappa_{22}\kappa_{10}$	$+2\kappa_{13}$	$+2\kappa_{01}\kappa_{13} + 4\kappa_{02}\kappa_{12} + 2\kappa_{03}\kappa_{11} + 2\kappa_{11}\kappa_{03} + 4\kappa_{12}\kappa_{02} + 2\kappa_{13}\kappa_{01}$	$+2\kappa_{10}\kappa_{13} + 4\kappa_{11}\kappa_{12} + 2\kappa_{12}\kappa_{11} + 2\kappa_{20}\kappa_{03} + 4\kappa_{21}\kappa_{02} + 2\kappa_{22}\kappa_{01}$
$\dot{\kappa}_{13}$	.	$+1\kappa_{13}$	$+1\kappa_{10}\kappa_{13} + 3\kappa_{11}\kappa_{12} + 3\kappa_{12}\kappa_{11} + 1\kappa_{13}\kappa_{10}$	$+1\kappa_{04}$	$+1\kappa_{01}\kappa_{04} + 3\kappa_{02}\kappa_{03} + 3\kappa_{03}\kappa_{02} + 1\kappa_{04}\kappa_{01}$	$+1\kappa_{10}\kappa_{04} + 3\kappa_{11}\kappa_{03} + 3\kappa_{12}\kappa_{02} + 1\kappa_{13}\kappa_{01}$
$\dot{\kappa}_{31}$	.	$+3\kappa_{31}$	$+3\kappa_{10}\kappa_{31} + 3\kappa_{11}\kappa_{30} + 6\kappa_{20}\kappa_{21} + 6\kappa_{21}\kappa_{20} + 3\kappa_{30}\kappa_{11} + 3\kappa_{31}\kappa_{10}$	$+3\kappa_{22}$	$+3\kappa_{01}\kappa_{22} + 3\kappa_{02}\kappa_{21} + 6\kappa_{11}\kappa_{12} + 6\kappa_{12}\kappa_{11} + 3\kappa_{21}\kappa_{02} + 3\kappa_{22}\kappa_{01}$	$+3\kappa_{10}\kappa_{22} + 3\kappa_{11}\kappa_{21} + 6\kappa_{20}\kappa_{12} + 6\kappa_{21}\kappa_{11} + 3\kappa_{30}\kappa_{02} + 3\kappa_{31}\kappa_{01}$

Table 1: Cumulant equation terms for coefficients a_{00} to a_{11} of eqn 10.

B. Cumulant equations for bivariate GQDs

By expanding eqn 31, one may derive a system of ODEs that approximates the evolution of the cumulants of a bivariate GQD over time. Tables 1 to 6 give terms to include on the RHS of the cumulant system for each coefficient of eqn 10. That is, for each dimension on the left of 31 (first column), include the terms in the column of every non-zero coefficient. Note that we have dropped the explicit dependence of κ_{ij} on t (i.e $\kappa_{ij}(t) = \kappa_{ij}$) for compactness. Also $\dot{\kappa}_{ij} = \frac{\partial}{\partial t}\kappa_{ij}$.

	b_{00}	b_{10}	b_{20}	b_{01}	b_{02}	b_{11}
$\dot{\kappa}_{10}$	.	.	.	.	.	.
$\dot{\kappa}_{20}$	.	.	.	.	.	.
$\dot{\kappa}_{30}$	.	.	.	.	.	.
$\dot{\kappa}_{40}$	.	.	.	.	.	.
$\dot{\kappa}_{01}$	1	$+1\kappa_{10}$	$+1\kappa_{10}\kappa_{10} + 1\kappa_{20}$	$+1\kappa_{01}$	$+1\kappa_{01}\kappa_{01} + 1\kappa_{02}$	$+1\kappa_{11} + 1\kappa_{01}\kappa_{10}$
$\dot{\kappa}_{02}$	.	$+2\kappa_{11}$	$+2\kappa_{10}\kappa_{11} + 2\kappa_{11}\kappa_{10} + 2\kappa_{21}$	$+2\kappa_{02}$	$+2\kappa_{01}\kappa_{02} + 2\kappa_{02}\kappa_{01} + 2\kappa_{03}$	$+2\kappa_{12} + 2\kappa_{01}\kappa_{11} + 2\kappa_{02}\kappa_{10}$
$\dot{\kappa}_{03}$	.	$+3\kappa_{12}$	$+3\kappa_{10}\kappa_{12} + 6\kappa_{11}\kappa_{11} + 3\kappa_{12}\kappa_{10} + 3\kappa_{22}$	$+3\kappa_{03}$	$+3\kappa_{01}\kappa_{03} + 6\kappa_{02}\kappa_{02} + 3\kappa_{03}\kappa_{01} + 3\kappa_{04}$	$+3\kappa_{13} + 3\kappa_{01}\kappa_{12} + 6\kappa_{02}\kappa_{11} + 3\kappa_{03}\kappa_{10}$
$\dot{\kappa}_{04}$	.	$+4\kappa_{13}$	$+4\kappa_{10}\kappa_{13} + 12\kappa_{11}\kappa_{12} + 12\kappa_{12}\kappa_{11} + 4\kappa_{13}\kappa_{10}$	$+4\kappa_{04}$	$+4\kappa_{01}\kappa_{04} + 12\kappa_{02}\kappa_{03} + 12\kappa_{03}\kappa_{02} + 4\kappa_{04}\kappa_{01}$	$+4\kappa_{01}\kappa_{13} + 12\kappa_{02}\kappa_{12} + 12\kappa_{03}\kappa_{11} + 4\kappa_{04}\kappa_{10}$
$\dot{\kappa}_{11}$	.	$+1\kappa_{20}$	$+1\kappa_{10}\kappa_{20} + 1\kappa_{20}\kappa_{10} + 1\kappa_{30}$	$+1\kappa_{11}$	$+1\kappa_{01}\kappa_{11} + 1\kappa_{11}\kappa_{01} + 1\kappa_{12}$	$+1\kappa_{21} + 1\kappa_{01}\kappa_{20} + 1\kappa_{11}\kappa_{10}$
$\dot{\kappa}_{12}$	.	$+2\kappa_{21}$	$+2\kappa_{10}\kappa_{21} + 2\kappa_{11}\kappa_{20} + 2\kappa_{20}\kappa_{11} + 2\kappa_{21}\kappa_{10} + 2\kappa_{31}$	$+2\kappa_{12}$	$+2\kappa_{01}\kappa_{12} + 2\kappa_{02}\kappa_{11} + 2\kappa_{11}\kappa_{02} + 2\kappa_{12}\kappa_{01} + 2\kappa_{13}$	$+2\kappa_{22} + 2\kappa_{01}\kappa_{21} + 2\kappa_{02}\kappa_{20} + 2\kappa_{11}\kappa_{11} + 2\kappa_{12}\kappa_{10}$
$\dot{\kappa}_{21}$	.	$+1\kappa_{30}$	$+1\kappa_{10}\kappa_{30} + 2\kappa_{20}\kappa_{20} + 1\kappa_{30}\kappa_{10} + 1\kappa_{40}$	$+1\kappa_{21}$	$+1\kappa_{01}\kappa_{21} + 2\kappa_{11}\kappa_{11} + 1\kappa_{21}\kappa_{01} + 1\kappa_{22}$	$+1\kappa_{31} + 1\kappa_{01}\kappa_{30} + 2\kappa_{11}\kappa_{20} + 1\kappa_{21}\kappa_{10}$
$\dot{\kappa}_{22}$	.	$+2\kappa_{31}$	$+2\kappa_{10}\kappa_{31} + 2\kappa_{11}\kappa_{30} + 4\kappa_{20}\kappa_{21} + 4\kappa_{21}\kappa_{20} + 2\kappa_{30}\kappa_{11} + 2\kappa_{31}\kappa_{10}$	$+2\kappa_{22}$	$+2\kappa_{01}\kappa_{22} + 2\kappa_{02}\kappa_{21} + 4\kappa_{11}\kappa_{12} + 4\kappa_{12}\kappa_{11} + 2\kappa_{21}\kappa_{02} + 2\kappa_{22}\kappa_{01}$	$+2\kappa_{01}\kappa_{31} + 2\kappa_{02}\kappa_{30} + 4\kappa_{11}\kappa_{21} + 4\kappa_{12}\kappa_{20} + 2\kappa_{21}\kappa_{11} + 2\kappa_{22}\kappa_{10}$
$\dot{\kappa}_{13}$	.	$+3\kappa_{22}$	$+3\kappa_{10}\kappa_{22} + 6\kappa_{11}\kappa_{21} + 3\kappa_{12}\kappa_{20} + 3\kappa_{20}\kappa_{12} + 6\kappa_{21}\kappa_{11} + 3\kappa_{22}\kappa_{10}$	$+3\kappa_{13}$	$+3\kappa_{01}\kappa_{13} + 6\kappa_{02}\kappa_{12} + 3\kappa_{03}\kappa_{11} + 3\kappa_{11}\kappa_{03} + 6\kappa_{12}\kappa_{02} + 3\kappa_{13}\kappa_{01}$	$+3\kappa_{01}\kappa_{22} + 6\kappa_{02}\kappa_{21} + 3\kappa_{03}\kappa_{20} + 3\kappa_{11}\kappa_{12} + 6\kappa_{12}\kappa_{11} + 3\kappa_{13}\kappa_{10}$
$\dot{\kappa}_{31}$	.	$+1\kappa_{40}$	$+1\kappa_{10}\kappa_{40} + 3\kappa_{20}\kappa_{30} + 3\kappa_{30}\kappa_{20} + 1\kappa_{40}\kappa_{10}$	$+1\kappa_{31}$	$+1\kappa_{01}\kappa_{31} + 3\kappa_{11}\kappa_{21} + 3\kappa_{21}\kappa_{11} + 1\kappa_{31}\kappa_{01}$	$+1\kappa_{01}\kappa_{40} + 3\kappa_{11}\kappa_{30} + 3\kappa_{21}\kappa_{20} + 1\kappa_{31}\kappa_{10}$

Table 2: Cumulant equation terms for coefficients b_{00} to b_{11} of eqn 10.

	c_{00}	c_{10}	c_{20}	c_{01}	c_{02}	c_{11}
$\dot{\kappa}_{10}$	.	.	.	.	.	.
$\dot{\kappa}_{20}$	1	$+1\kappa_{10}$	$+1\kappa_{20} + 1\kappa_{10}\kappa_{10}$	$+1\kappa_{01}$	$+1\kappa_{02} + 1\kappa_{01}\kappa_{01}$	$+1\kappa_{11} + 1\kappa_{01}\kappa_{10}$
$\dot{\kappa}_{30}$	.	$+3\kappa_{20}$	$+3\kappa_{30} + 3\kappa_{10}\kappa_{20} + 3\kappa_{20}\kappa_{10}$	$+3\kappa_{11}$	$+3\kappa_{12} + 3\kappa_{01}\kappa_{11} + 3\kappa_{11}\kappa_{01}$	$+3\kappa_{21} + 3\kappa_{01}\kappa_{20} + 3\kappa_{11}\kappa_{10}$
$\dot{\kappa}_{40}$	.	$+6\kappa_{30}$	$+6\kappa_{40} + 6\kappa_{10}\kappa_{30} + 12\kappa_{20}\kappa_{20} + 6\kappa_{30}\kappa_{10}$	$+6\kappa_{21}$	$+6\kappa_{22} + 6\kappa_{01}\kappa_{21} + 12\kappa_{11}\kappa_{11} + 6\kappa_{21}\kappa_{01}$	$+6\kappa_{31} + 6\kappa_{01}\kappa_{30} + 12\kappa_{11}\kappa_{20} + 6\kappa_{21}\kappa_{10}$
$\dot{\kappa}_{01}$	.	.	.	.	.	.
$\dot{\kappa}_{02}$	.	.	.	.	.	.
$\dot{\kappa}_{03}$	.	.	.	.	.	.
$\dot{\kappa}_{04}$	.	.	.	.	.	.
$\dot{\kappa}_{11}$	.	.	.	.	.	.
$\dot{\kappa}_{12}$	.	.	.	.	.	.
$\dot{\kappa}_{21}$	.	$+1\kappa_{11}$	$+1\kappa_{21} + 1\kappa_{10}\kappa_{11} + 1\kappa_{11}\kappa_{10}$	$+1\kappa_{02}$	$+1\kappa_{03} + 1\kappa_{01}\kappa_{02} + 1\kappa_{02}\kappa_{01}$	$+1\kappa_{12} + 1\kappa_{01}\kappa_{11} + 1\kappa_{02}\kappa_{10}$
$\dot{\kappa}_{22}$	.	$+1\kappa_{12}$	$+1\kappa_{22} + 1\kappa_{10}\kappa_{12} + 2\kappa_{11}\kappa_{11} + 1\kappa_{12}\kappa_{10}$	$+1\kappa_{03}$	$+1\kappa_{04} + 1\kappa_{01}\kappa_{03} + 2\kappa_{02}\kappa_{02} + 1\kappa_{03}\kappa_{01}$	$+1\kappa_{13} + 1\kappa_{01}\kappa_{12} + 2\kappa_{02}\kappa_{11} + 1\kappa_{03}\kappa_{10}$
$\dot{\kappa}_{13}$	.	.	.	.	.	.
$\dot{\kappa}_{31}$	.	$+3\kappa_{21}$	$+3\kappa_{31} + 3\kappa_{10}\kappa_{21} + 3\kappa_{11}\kappa_{20} + 3\kappa_{20}\kappa_{11} + 3\kappa_{21}\kappa_{10}$	$+3\kappa_{12}$	$+3\kappa_{13} + 3\kappa_{01}\kappa_{12} + 3\kappa_{02}\kappa_{11} + 3\kappa_{11}\kappa_{02} + 3\kappa_{12}\kappa_{01}$	$+3\kappa_{22} + 3\kappa_{01}\kappa_{21} + 3\kappa_{02}\kappa_{20} + 3\kappa_{11}\kappa_{11} + 3\kappa_{12}\kappa_{10}$

Table 3: Cumulant equation terms for coefficients c_{00} to c_{11} of eqn 10.

	d_{00}	d_{10}	d_{20}	d_{01}	d_{02}	d_{11}
$\dot{\kappa}_{10}$	.	.	.	.	.	.
$\dot{\kappa}_{20}$	.	.	.	.	.	.
$\dot{\kappa}_{30}$	.	.	.	.	.	.
$\dot{\kappa}_{40}$	.	.	.	.	.	.
$\dot{\kappa}_{01}$	.	.	.	.	.	.
$\dot{\kappa}_{02}$	.	.	.	.	.	.
$\dot{\kappa}_{03}$	.	.	.	.	.	.
$\dot{\kappa}_{04}$	.	.	.	.	.	.
$\dot{\kappa}_{11}$	1	$+0.5\kappa_{10}$	$+0.5\kappa_{20}$ + $0.5\kappa_{10}\kappa_{10}$	$+0.5\kappa_{01}$	$+0.5\kappa_{02}$ + $0.5\kappa_{01}\kappa_{01}$	$+0.5\kappa_{11}$ + $0.5\kappa_{01}\kappa_{10}$
$\dot{\kappa}_{12}$	.	$+1\kappa_{11}$	$+1\kappa_{21} + 1\kappa_{10}\kappa_{11} + 1\kappa_{11}\kappa_{10}$	$+1\kappa_{02}$	$+1\kappa_{03} + 1\kappa_{01}\kappa_{02} + 1\kappa_{02}\kappa_{01}$	$+1\kappa_{12} + 1\kappa_{01}\kappa_{11} + 1\kappa_{02}\kappa_{10}$
$\dot{\kappa}_{21}$	.	$+1\kappa_{20}$	$+1\kappa_{30} + 1\kappa_{10}\kappa_{20} + 1\kappa_{20}\kappa_{10}$	$+1\kappa_{11}$	$+1\kappa_{12} + 1\kappa_{01}\kappa_{11} + 1\kappa_{11}\kappa_{01}$	$+1\kappa_{21} + 1\kappa_{01}\kappa_{20} + 1\kappa_{11}\kappa_{10}$
$\dot{\kappa}_{22}$	.	$+2\kappa_{21}$	$+2\kappa_{31} + 2\kappa_{10}\kappa_{21} + 2\kappa_{11}\kappa_{20}+2\kappa_{20}\kappa_{11}+ 2\kappa_{21}\kappa_{10}$	$+2\kappa_{12}$	$+2\kappa_{13} + 2\kappa_{01}\kappa_{12} + 2\kappa_{02}\kappa_{11}+2\kappa_{11}\kappa_{02}+ 2\kappa_{12}\kappa_{01}$	$+2\kappa_{22} + 2\kappa_{01}\kappa_{21} + 2\kappa_{02}\kappa_{20}+2\kappa_{11}\kappa_{11}+ 2\kappa_{12}\kappa_{10}$
$\dot{\kappa}_{13}$	.	$+1.5\kappa_{12}$	$+1.5\kappa_{22}$ + $1.5\kappa_{10}\kappa_{12}$ + $3\kappa_{11}\kappa_{11}$ + $1.5\kappa_{12}\kappa_{10}$	$+1.5\kappa_{03}$	$+1.5\kappa_{04}$ + $1.5\kappa_{01}\kappa_{03}$ + $3\kappa_{02}\kappa_{02}$ + $1.5\kappa_{03}\kappa_{01}$	$+1.5\kappa_{13}$ + $1.5\kappa_{01}\kappa_{12}$ + $3\kappa_{02}\kappa_{11}$ + $1.5\kappa_{03}\kappa_{10}$
$\dot{\kappa}_{31}$	.	$+1.5\kappa_{30}$	$+1.5\kappa_{40}$ + $1.5\kappa_{10}\kappa_{30}$ + $3\kappa_{20}\kappa_{20}$ + $1.5\kappa_{30}\kappa_{10}$	$+1.5\kappa_{21}$	$+1.5\kappa_{22}$ + $1.5\kappa_{01}\kappa_{21}$ + $3\kappa_{11}\kappa_{11}$ + $1.5\kappa_{21}\kappa_{01}$	$+1.5\kappa_{31}$ + $1.5\kappa_{01}\kappa_{30}$ + $3\kappa_{11}\kappa_{20}$ + $1.5\kappa_{21}\kappa_{10}$

Table 4: Cumulant equation terms for coefficients d_{00} to d_{11} of eqn 10.

	e_{00}	e_{10}	e_{20}	e_{01}	e_{02}	e_{11}
$\dot{\kappa}_{10}$	.	.	.	.	.	.
$\dot{\kappa}_{20}$	.	.	.	.	.	.
$\dot{\kappa}_{30}$	.	.	.	.	.	.
$\dot{\kappa}_{40}$	.	.	.	.	.	.
$\dot{\kappa}_{01}$	.	.	.	.	.	.
$\dot{\kappa}_{02}$	.	.	.	.	.	.
$\dot{\kappa}_{03}$	.	.	.	.	.	.
$\dot{\kappa}_{04}$	.	.	.	.	.	.
$\dot{\kappa}_{11}$	1	$+0.5\kappa_{10}$	$+0.5\kappa_{20}$ + $0.5\kappa_{10}\kappa_{10}$	$+0.5\kappa_{01}$	$+0.5\kappa_{02}$ + $0.5\kappa_{01}\kappa_{01}$	$+0.5\kappa_{11}$ + $0.5\kappa_{01}\kappa_{10}$
$\dot{\kappa}_{12}$	.	$+1\kappa_{11}$	$+1\kappa_{21} + 1\kappa_{10}\kappa_{11} + 1\kappa_{11}\kappa_{10}$	$+1\kappa_{02}$	$+1\kappa_{03} + 1\kappa_{01}\kappa_{02} + 1\kappa_{02}\kappa_{01}$	$+1\kappa_{12} + 1\kappa_{01}\kappa_{11} + 1\kappa_{02}\kappa_{10}$
$\dot{\kappa}_{21}$	.	$+1\kappa_{20}$	$+1\kappa_{30} + 1\kappa_{10}\kappa_{20} + 1\kappa_{20}\kappa_{10}$	$+1\kappa_{11}$	$+1\kappa_{12} + 1\kappa_{01}\kappa_{11} + 1\kappa_{11}\kappa_{01}$	$+1\kappa_{21} + 1\kappa_{01}\kappa_{20} + 1\kappa_{11}\kappa_{10}$
$\dot{\kappa}_{22}$	.	$+2\kappa_{21}$	$+2\kappa_{31} + 2\kappa_{10}\kappa_{21} + 2\kappa_{11}\kappa_{20}+2\kappa_{20}\kappa_{11}+ 2\kappa_{21}\kappa_{10}$	$+2\kappa_{12}$	$+2\kappa_{13} + 2\kappa_{01}\kappa_{12} + 2\kappa_{02}\kappa_{11}+2\kappa_{11}\kappa_{02}+ 2\kappa_{12}\kappa_{01}$	$+2\kappa_{22} + 2\kappa_{01}\kappa_{21} + 2\kappa_{02}\kappa_{20}+2\kappa_{11}\kappa_{11}+ 2\kappa_{12}\kappa_{10}$
$\dot{\kappa}_{13}$	.	$+1.5\kappa_{12}$	$+1.5\kappa_{22}$ + $1.5\kappa_{10}\kappa_{12}$ + $3\kappa_{11}\kappa_{11}$ + $1.5\kappa_{12}\kappa_{10}$	$+1.5\kappa_{03}$	$+1.5\kappa_{04}$ + $1.5\kappa_{01}\kappa_{03}$ + $3\kappa_{02}\kappa_{02}$ + $1.5\kappa_{03}\kappa_{01}$	$+1.5\kappa_{13}$ + $1.5\kappa_{01}\kappa_{12}$ + $3\kappa_{02}\kappa_{11}$ + $1.5\kappa_{03}\kappa_{10}$
$\dot{\kappa}_{31}$	.	$+1.5\kappa_{30}$	$+1.5\kappa_{40}$ + $1.5\kappa_{10}\kappa_{30}$ + $3\kappa_{20}\kappa_{20}$ + $1.5\kappa_{30}\kappa_{10}$	$+1.5\kappa_{21}$	$+1.5\kappa_{22}$ + $1.5\kappa_{01}\kappa_{21}$ + $3\kappa_{11}\kappa_{11}$ + $1.5\kappa_{21}\kappa_{01}$	$+1.5\kappa_{31}$ + $1.5\kappa_{01}\kappa_{30}$ + $3\kappa_{11}\kappa_{20}$ + $1.5\kappa_{21}\kappa_{10}$

Table 5: Cumulant equation terms for coefficients e_{00} to e_{11} of eqn 10.

	f_{00}	f_{10}	f_{20}	f_{01}	f_{02}	f_{11}
$\dot{\kappa}_{10}$	.	.	.	.	.	.
$\dot{\kappa}_{20}$	.	.	.	.	.	.
$\dot{\kappa}_{30}$	.	.	.	.	.	.
$\dot{\kappa}_{40}$	.	.	.	.	.	.
$\dot{\kappa}_{01}$	.	.	.	.	.	.
$\dot{\kappa}_{02}$	1	$+1\kappa_{10}$	$+1\kappa_{20} + 1\kappa_{10}\kappa_{10}$	$+1\kappa_{01}$	$+1\kappa_{02} + 1\kappa_{01}\kappa_{01}$	$+1\kappa_{11} + 1\kappa_{01}\kappa_{10}$
$\dot{\kappa}_{03}$	.	$+3\kappa_{11}$	$+3\kappa_{21} + 3\kappa_{10}\kappa_{11} + 3\kappa_{11}\kappa_{10}$	$+3\kappa_{02}$	$+3\kappa_{03} + 3\kappa_{01}\kappa_{02} + 3\kappa_{02}\kappa_{01}$	$+3\kappa_{12} + 3\kappa_{01}\kappa_{11} + 3\kappa_{02}\kappa_{10}$
$\dot{\kappa}_{04}$	.	$+6\kappa_{12}$	$+6\kappa_{22} + 6\kappa_{10}\kappa_{12} + 12\kappa_{11}\kappa_{11} + 6\kappa_{12}\kappa_{10}$	$+6\kappa_{03}$	$+6\kappa_{04} + 6\kappa_{01}\kappa_{03} + 12\kappa_{02}\kappa_{02} + 6\kappa_{03}\kappa_{01}$	$+6\kappa_{13} + 6\kappa_{01}\kappa_{12} + 12\kappa_{02}\kappa_{11} + 6\kappa_{03}\kappa_{10}$
$\dot{\kappa}_{11}$	.	.	.	.	.	.
$\dot{\kappa}_{12}$	.	$+1\kappa_{20}$	$+1\kappa_{30} + 1\kappa_{10}\kappa_{20} + 1\kappa_{20}\kappa_{10}$	$+1\kappa_{11}$	$+1\kappa_{12} + 1\kappa_{01}\kappa_{11} + 1\kappa_{11}\kappa_{01}$	$+1\kappa_{21} + 1\kappa_{01}\kappa_{20} + 1\kappa_{11}\kappa_{10}$
$\dot{\kappa}_{21}$	.	.	.	.	.	.
$\dot{\kappa}_{22}$	.	$+1\kappa_{30}$	$+1\kappa_{40} + 1\kappa_{10}\kappa_{30} + 2\kappa_{20}\kappa_{20} + 1\kappa_{30}\kappa_{10}$	$+1\kappa_{21}$	$+1\kappa_{22} + 1\kappa_{01}\kappa_{21} + 2\kappa_{11}\kappa_{11} + 1\kappa_{21}\kappa_{01}$	$+1\kappa_{31} + 1\kappa_{01}\kappa_{30} + 2\kappa_{11}\kappa_{20} + 1\kappa_{21}\kappa_{10}$
$\dot{\kappa}_{13}$	.	$+3\kappa_{21}$	$+3\kappa_{31} + 3\kappa_{10}\kappa_{21} + 3\kappa_{11}\kappa_{20} + 3\kappa_{20}\kappa_{11} + 3\kappa_{21}\kappa_{10}$	$+3\kappa_{12}$	$+3\kappa_{13} + 3\kappa_{01}\kappa_{12} + 3\kappa_{02}\kappa_{11} + 3\kappa_{11}\kappa_{02} + 3\kappa_{12}\kappa_{01}$	$+3\kappa_{22} + 3\kappa_{01}\kappa_{21} + 3\kappa_{02}\kappa_{20} + 3\kappa_{11}\kappa_{11} + 3\kappa_{12}\kappa_{10}$
$\dot{\kappa}_{31}$	.	.	.	.	.	.

Table 6: Cumulant equation terms for coefficients f_{00} to f_{11} of eqn 10.

C. Example C++ code

The script below gives an example of C++ code constructed by the `BiGQD.mcmc()` function. The `solver()` function can be used to evaluate the log-likelihood of eqn 63.

```
#include <RcppArmadillo.h>
#include <math.h>
#define pi           3.14159265358979323846  /* pi */
using namespace arma;
using namespace Rcpp;
using namespace R;

// [[Rcpp::depends("RcppArmadillo")]]
// [[Rcpp::export]]
  mat f(mat a,vec theta,vec t,int N2)
{

  mat atemp(N2,8);
    atemp.col(0)=(+(theta[1] * (theta[2] + theta[3] * sin(2 * pi * t)))*1+(-theta[1])*(+1*a.col(0)));
    atemp.col(1)=(+(-theta[1])*(+2*a.col(1))+(theta[4] * theta[4])*(+1*a.col(0)));
    atemp.col(2)=(+(-theta[1])*(+3*a.col(2))+(theta[4] * theta[4])*(+3*a.col(1)));
    atemp.col(3)=(+(-theta[1])*(+4*a.col(3))+(theta[4] * theta[4])*(+6*a.col(2)));
    atemp.col(4)=(+(theta[5] * theta[6])*(+1*a.col(4))+(-theta[5])*(+1*a.col(4)%a.col(4)+1*a.col(5)))↩
        ;
    atemp.col(5)=(+(theta[5] * theta[6])*(+2*a.col(5))+(-theta[5])*(+2*a.col(4)%a.col(5)+2*a.col(5)%a↩
        .col(4)+2*a.col(6))+(theta[7] * theta[7])*(+1*a.col(5)+1*a.col(4)%a.col(4)));
    atemp.col(6)=(+(theta[5] * theta[6])*(+3*a.col(6))+(-theta[5])*(+3*a.col(4)%a.col(6)+6*a.col(5)%a↩
        .col(5)+3*a.col(6)%a.col(4)+3*a.col(7))+(theta[7] * theta[7])*(+3*a.col(6)+3*a.col(4)%a.col↩
        (5)+3*a.col(5)%a.col(4)));
    atemp.col(7)=(+(theta[5] * theta[6])*(+4*a.col(7))+(-theta[5])*(+4*a.col(4)%a.col(7)+12*a.col(5)%↩
        a.col(6)+12*a.col(6)%a.col(5)+4*a.col(7)%a.col(4))+(theta[7] * theta[7])*(+6*a.col(7)+6*a.↩
        col(4)%a.col(6)+12*a.col(5)%a.col(5)+6*a.col(6)%a.col(4)));
  return atemp;
}

  // [[Rcpp::export]]
  mat  solver(vec Xs,vec Ys,vec Xt,vec Yt,vec theta,int N,double delt,int N2,vec tt,mat starts)
{
  mat resss(N2,3);
  mat fx0(N2,8);
  mat fx1(N2,8);
  mat fx2(N2,8);
  mat fx3(N2,8);
  mat fx4(N2,8);
  mat fx5(N2,8);
  mat fx6(N2,8);
  mat fx7(N2,8);
  mat fx8(N2,8);
  mat fx9(N2,8);
  mat fx10(N2,8);
  mat fx11(N2,8);
  mat fx12(N2,8);
  mat fx13(N2,8);
  mat fx14(N2,8);
  mat fx15(N2,8);
  mat fx16(N2,8);
  mat x0(N2,8);
  mat x1(N2,8);
  mat x2(N2,8);
  mat x3(N2,8);
```

```
  mat x4(N2,8);
  mat x5(N2,8);
  mat x6(N2,8);
  mat x7(N2,8);
  mat x8(N2,8);
  mat x9(N2,8);
  mat x10(N2,8);
  mat x11(N2,8);
  mat x12(N2,8);
  mat x13(N2,8);
  mat x14(N2,8);
  mat x15(N2,8);
  mat x16(N2,8);
  x0.fill(0);
  x0.col(0)=Xs;
  x0.col(4)=Ys;
  vec d=tt;
  for (int i = 1; i < N; i++)
{
 fx0=f(x0,theta,d,N2);
  x1=x0+delt*(0.1*fx0);
  fx1=f(x1,theta,d+0.10000000000000000000000000000000000000000000000000000000000000000000*delt,N2);
  x2=x0+delt*(-0.9151765613752 91*fx0+1.45453440217827*fx1);
  fx2=f(x2,theta,d+0.53935784080298178753248519788130243685727344970100901550550 0*delt,N2);
  x3=x0+delt*(0.202259190301118*fx0+0.6067775709033 54*fx2);
  fx3=f(x3,theta,d+0.80903676120447268129872779682195365528591017455151352325825 0*delt,N2);
  x4=x0+delt*(0.184024714708644*fx0+0.197966831227192*fx2-0.0729547847313633*fx3);
  fx4=f(x4,theta,d+0.30903676120447268129872779682195365528591017455151352325825 0*delt,N2);
  x5=x0+delt*(0.0879007340206681*fx0+0.4104597025202 61*fx3+0.4827137536788 66*fx4);
  fx5=f(x5,theta,d+0.98107419021979526825487954831056208048905674611872488202780 5*delt,N2);
  x6=x0+delt*(0.085970050490246*fx0+0.330885963040722*fx3+0.48966295730945*fx4-0.0731856375070851*fx5↩
      );
  fx6=f(x6,theta,d+0.83333333333333333333333333333333333333333333333333333333333333333333*delt,N2);
  x7=x0+delt*(0.120930449125334*fx0+0.260124675758296*fx4+0.0325402621549091*fx5-0.0595780211817361*↩
      fx6);
  fx7=f(x7,theta,d+0.35401736585680237632926418594879674211582405380737396832418 4*delt,N2);
  x8=x0+delt*(0.110854379580391*fx0-0.0605761488255006*fx5+0.321763705601778*fx6+0.510485725608063*↩
      fx7);
  fx8=f(x8,theta,d+0.88252766196473234642550148697966907518286784426805211966379 1*delt,N2);
  x9=x0+delt*(0.112054414752879*fx0-0.144942775902866*fx5-0.333269719096257*fx6+0.49926922955688*fx7↩
      +0.509504608929686*fx8);
  fx9=f(x9,theta,d+0.64261575824032254815707549702043953595950173636321269590987 5*delt,N2);
  x10=x0+delt*(0.113976783964186*fx0-0.0768813364203357*fx5+0.239527360324391*fx6+0.397774662368095*↩
      fx7+0.0107558956873607*fx8-0.327769124164019*fx9);
  fx10=f(x10,theta,d+0.35738424175967745184292450297956046404049826363678730409012 5*delt,N2);
  x11=x0+delt*(0.0798314528280196*fx0-0.0520329686800603*fx5-0.0576954146168549*fx6↩
      +0.194781915712104*fx7+0.1453849231883 25*fx8-0.0782942710351671*fx9-0.114503299361099*fx10);
  fx11=f(x11,theta,d+0.11747233803526765357449851302033092481713215573194788033620 9*delt,N2);
  x12=x0+delt*(0.985115610164857*fx0+0.330885963040722*fx3+0.489662957309 45*fx4-1.37896486574844*fx5↩
      -0.861164195027636*fx6+5.784288136375 37*fx7+3.28807761985104*fx8-2.38633905093136*fx9↩
      -3.254793424836 44*fx10-2.16343541686423*fx11);
  fx12=f(x12,theta,d+0.83333333333333333333333333333333333333333333333333333333333333333333*delt,N2);
  x13=x0+delt*(0.895080295771633*fx0+0.197966831227192*fx2-0.0729547847313633*fx3-0.8512362396620 08*↩
      fx5+0.398320112318533*fx6+3.639372631810 36*fx7+1.5482287703983*fx8-2.12221714704054*fx9↩
      -1.583503985453 26*fx10-1.71561608285936*fx11-0.0244036405750127*fx12);
  fx13=f(x13,theta,d+0.30903676120447268129872779682195365528591017455151352325825 0*delt,N2);
  x14=x0+delt*(-0.915176561375291*fx0+1.454534402178 27*fx1+0*fx2+0*fx3-0.777333643644968*fx4+0*fx5↩
      -0.0910895662155176*fx6+0.0910895662155176*fx12+0.777333643644968*fx13);
  fx14=f(x14,theta,d+0.53935784080298178753248519788130243685727344970100901550550 0*delt,N2);
  x15=x0+delt*(0.1*fx0-0.157178665799771*fx2+0.157178665799771*fx14);
  fx15=f(x15,theta,d+0.10000000000000000000000000000000000000000000000000000000000000000000*delt,N2);
  x16=x0+delt*(0.1817813007000 95*fx0+0.675*fx1+0.342758159847 19*fx2+0*fx3+0.259111214548323*fx4↩
      -0.358278966717952*fx5-1.045948959408 83*fx6+0.930327845415627*fx7+1.779509594317 08*fx8+0.1*fx9↩
      -0.282547569539044*fx10-0.159327350119973*fx11-0.145515894647002*fx12-0.259111214548323*fx13↩
      -0.342758159847 19*fx14-0.675*fx15);
  fx16=f(x16,theta,d+delt,N2);

  x0=x0+(0.033333333333333333333333333333333333333333333333333333333333333333333*fx0
         +0.025000000000000000000000000000000000000000000000000000000000000000000*fx1
         +0.033333333333333333333333333333333333333333333333333333333333333333333*fx2
         +0.000000000000000000000000000000000000000000000000000000000000000000000*fx3
         +0.050000000000000000000000000000000000000000000000000000000000000000000*fx4
         +0.000000000000000000000000000000000000000000000000000000000000000000000*fx5
         +0.040000000000000000000000000000000000000000000000000000000000000000000*fx6
         +0.000000000000000000000000000000000000000000000000000000000000000000000*fx7
         +0.189237478148923490158306404106012326238162346948625830327194*fx8
         +0.277429188517743176508360262560654340428504319718040836339472*fx9
         +0.277429188517743176508360262560654340428504319718040836339472*fx10
         +0.189237478148923490158306404106012326238162346948625830327194*fx11
         -0.040000000000000000000000000000000000000000000000000000000000000000000*fx12
         -0.050000000000000000000000000000000000000000000000000000000000000000000*fx13
         -0.033333333333333333333333333333333333333333333333333333333333333333333*fx14
         -0.025000000000000000000000000000000000000000000000000000000000000000000*fx15
         +0.033333333333333333333333333333333333333333333333333333333333333333333*fx16)*delt;
 d=d+delt;
}

  vec p=(1.0/3.0) *(3*(x0.col(3)/6.0)%x0.col(1) - pow(x0.col(2)/2.0,2))/pow(x0.col(3)/6.0,2);
  vec q=(1.0/27.0)*(27*pow(x0.col(3)/6.0,2)%(x0.col(0)-Xt) - 9*(x0.col(3)/6.0)%(x0.col(2)/2.0)%x0.col↩
      (1) + 2*pow(x0.col(2)/2.0,3))/pow(x0.col(3)/6.0,3);
  vec chk=pow(q,2)/4.0 + pow(p,3)/27.0;
  vec th=-(x0.col(2)/2.0)/(3*(x0.col(3)/6.0))+pow(-q/2.0+sqrt(chk),(1.0/3.0))-pow(q/2.0+sqrt(chk)↩
      ,(1.0/3.0));

  vec K =x0.col(0)%th+(x0.col(1)%th%th)/2.0+(x0.col(2)%th%th%th)/6.0 +(x0.col(3)%th%th%th%th)/24.0;
  vec K1=x0.col(0)    +(x0.col(1)%th)       +(x0.col(2)%th%th)/2.0    +(x0.col(3)%th%th%th)/6.0;
  vec K2=x0.col(1)    +(x0.col(2)%th)       +(x0.col(3)%th%th)/2.0;
  vec val=-0.5*log(2*3.141592653589793*K2)+(K-th%K1);
```

```
  p=(1.0/3.0) *(3*(x0.col(7)/6.0)%x0.col(5) - pow(x0.col(6)/2.0,2))/pow(x0.col(7)/6.0,2);
  q=(1.0/27.0)*(27*pow(x0.col(7)/6.0,2)%(x0.col(4)-Yt) - 9*(x0.col(7)/6.0)%(x0.col(6)/2.0)%x0.col(5) ↩
       + 2*pow(x0.col(6)/2.0,3))/pow(x0.col(7)/6.0,3);
  chk=pow(q,2)/4.0 + pow(p,3)/27.0;
  th=-(x0.col(6)/2.0)/(3*(x0.col(7)/6.0))+pow(-q/2.0+sqrt(chk),(1.0/3.0))-pow(q/2.0+sqrt(chk)↩
       ,(1.0/3.0));

  K =x0.col(4)%th+(x0.col(5)%th%th)/2.0+(x0.col(6)%th%th%th)/6.0 +(x0.col(7)%th%th%th%th)/24.0;
  K1=x0.col(4)   +(x0.col(5)%th)       +(x0.col(6)%th%th)/2.0    +(x0.col(7)%th%th%th)/6.0;
  K2=x0.col(5)   +(x0.col(6)%th)       +(x0.col(7)%th%th)/2.0;
  val=val-0.5*log(2*3.141592653589793*K2)+(K-th%K1);
  resss.col(0)=val;
  return(resss);
}
```

Affiliation:

Etienne A.D. Pienaar
Department of Statistical Sciences
University of Cape Town
Rondebosch, Cape Town 7707
South Africa
E-mail: etiennead@gmail.com

Melvin M. Varughese
Department of Statistical Sciences
University of Cape Town
Rondebosch, Cape Town 7707
South Africa
E-mail: melvin.varughese@uct.ac.za
Telephone: +27/21/650-5230
Fax: +27/21/650-4773