The YUIMA Project: A Computational Framework for Simulation and Inference of Stochastic Differential Equations

Alexandre Brouste  Masaaki Fukasawa  Hideitsu Hino  Stefano M. Iacus
University of Le Mans  Osaka University  University of Tsukuba  University of Milan

Kengo Kamatani  Yuta Koike  Hiroki Masuda  Ryosuke Nomura
University of Tokyo  University of Tokyo  Kyushu University  University of Tokyo

Teppei Ogihara  Yasutaka Shimuzu  Masayuki Uchida  Nakahiro Yoshida
Osaka University  Osaka University  Osaka University  University of Tokyo

Abstract

The YUIMA Project is an open source and collaborative effort aimed at developing the R package yuima for simulation and inference of stochastic differential equations. In the yuima package stochastic differential equations can be of very abstract type, multidimensional, driven by Wiener process or fractional Brownian motion with general Hurst parameter, with or without jumps specified as Lévy noise. The yuima package is intended to offer the basic infrastructure on which complex models and inference procedures can be built on. This paper explains the design of the yuima package and provides some examples of applications.

Keywords: inference for stochastic processes, simulation, stochastic differential equations.

1. Introduction

The plan of the YUIMA Project is to construct the bases for a complete environment for

1YUIMA is both the acronym for Yoshida-Uchida-Iacus-Masuda-Andothers but also the name of an important character in Buddhism religion (http://www.kyohaku.go.jp/eng/nyuzou/meihin/kaiga/chuugoku/item01.html) whose approach to problems fits this project well.
simulation and inference for stochastic processes via an R (R Core Team 2013) package called yuima. The yuima package adopts the S4 system of classes and methods (Chambers 1998).

Under this framework, the yuima package also supplies various functions to carry out simulation and statistical analysis. Both categories of procedures may depend on each other. Statistical inference often requires a simulation technique as a subroutine, and a certain simulation method needs to fix a tuning parameter by applying a statistical methodology. It is especially in the case of stochastic processes because most of the expected values involved do not admit an explicit expression. The yuima package facilitates comprehensive, systematic approaches to the solution.

Stochastic differential equations are commonly used to model random evolution along continuous or practically continuous time, such as the random movements of stock prices. Theory of statistical inference for stochastic differential equations already has a fairly long history, more than three decades, but it is still developing quickly new methodologies and expanding the area. The formulas produced by the theory are usually very sophisticated, which makes it difficult for standard users not necessarily familiar with this field to enjoy benefits. For example, by taking advantage of the analytic approach, the asymptotic expansion method for computing option prices (i.e., expectation of an irregular functional of a stochastic process) provides precise approximation values instantaneously. The expansion formula, which has a long expression involving more than 900 terms of multiple integrals, is already coded in the yuima package for generic diffusion processes.

The yuima package delivers up-to-date methods as a package onto the desk of the user working with simulation and/or statistics for stochastic differential equations. In the yuima package, stochastic differential equations can be of very abstract type, multidimensional, driven by Wiener process or fractional Brownian motion with general Hurst parameter, with or without jumps specified as Lévy noise.

The yuima package is intended to offer the basic infrastructure on which complex models and inference procedures can be built on. This paper explains the design of the yuima package and illustrates some examples of applications. The paper is organized as follows. Section 2 is an overview about the package. Section 3 describes the way in which models are specified in yuima. Section 4 describes how to simulate yuima models. Section 5 explains asymptotic expansion methods. Section 6 is a review of basic inference procedures. Finally, Section 7 gives additional details and the roadmap of the YUIMA Project.

Although we assume that the reader of this paper has a basic knowledge of the R language, most of the examples are easy to understand if he/she knows stochastic differential equations intuitively or symbolically.

2. The yuima package

The package yuima depends on some other packages, like zoo (Zeileis and Grothendieck 2005), which can be installed separately. The package zoo is used internally to store time series data. This dependence may change in the future adopting a more flexible class for internal storage of time series.
2.1. How to obtain the package

The **yuima** package is hosted on R-Forge and the web page of the project is [http://R-Forge.R-project.org/projects/yuima](http://R-Forge.R-project.org/projects/yuima). The R-Forge page contains the latest development version, and stable version of the package are available through the Comprehensive R Archive Network (CRAN) at [http://CRAN.R-project.org/package=yuima](http://CRAN.R-project.org/package=yuima). Development versions of the package are not supposed to be stable or functional, thus the occasional user should consider to install the stable version first.

2.2. The main objects and classes

Before discussing the methods for simulation and inference for stochastic process solutions to stochastic differential equations, we give an overview of the main classes in the package. As mentioned there are different classes of objects defined in the **yuima** package and the main class ‘**yuima**’. This class is composed of several slots. Figure 1 represents the different classes and their slots. The different slots do not need to be all present at the same time.

---

**Figure 1:** The main classes in the **yuima** package.
For example, in case one wants to simulate a stochastic process, only the slots `model` and `sampling` should be present, while the slot `data` will be filled by the simulator. We discuss in detail the different objects separately in the next sections.

The general idea of the `yuima` package is to separate into different subclass objects the statistical model, the data and the statistical methods. As will be explained with several examples, the user may give a mathematical description of the statistical model with `setModel` which prepares a `yuima.model` object by filling the appropriate slots. If the aim is the simulation of the solution of the stochastic differential equation specified in the `yuima.model` object then, using the method `simulate`, it is possible to obtain one trajectory of the process. As an output, a `yuima` object (i.e., a possibly incomplete, object of class `yuima`) is created which contains the original model specified in the `yuima.model` object in the slot named `model` and two additional slots named `data`, for the simulated data, and `sampling` which contains the description of the simulation scheme used as well as other informations. The details of `simulate` will be explained in Section 4 along with the use of method `setSampling` which allows to specify the type of sampling scheme to be used by the `simulate` method. But a `yuima` object may contain the slot `data` not only as the outcome of `simulate` but also because the user decides to analyse its own data. In this case the method `setData` is used to transform most types of R time series objects into a proper `yuima.data` object. When the slots `data` and `model` are available, many other methods can be used to perform statistical analysis on these SDE (stochastic differential equation) models. These methods will be discussed in Section 6. Further, functionals of SDE’s can be defined using the `setFunctional` method and evaluated using asymptotic expansion methods as explained in Section 5. The `setFunctional` method creates a `yuima.functional` object which is included along with a `yuima.model` in a `yuima` object in order to be used for the evaluation of its asymptotic expansion.

2.3. The ‘yuima.model’ class

At present, in `yuima` three main classes of SDE’s can be easily specified. Here we present a brief overview of these models as they will be described in detail in Section 3, but this allows to introduce an overall view of the slots of the ‘yuima.model’ class.

In `yuima` one can describe three main families of stochastic processes at present. These models can be one or multidimensional and eventually described as parametric models. Let \( X_0 = x_0 \) be the initial value of the process, then, the main classes are as follows:

- **Diffusion models** described as
  \[
  dX_t = a(t, X_t, \theta)dt + b(t, X_t, \theta)dW_t, \quad X_0 = x_0,
  \]
  where \( W_t \) is a standard Brownian motion.

- **SDE’s driven by fractional Gaussian noise**, with \( H \) the Hurst parameter, described as
  \[
  dX_t = a(t, X_t, \theta)dt + b(t, X_t, \theta)dW^{H}_t, \quad X_0 = x_0.
  \]
Diffusion processes with jumps and Lévy processes solution to
\[
dX_t = a(t, X_t, \theta)dt + b(t, X_t, \theta)dW_t + \int_{|z|>1} c(X_{t-}, z)\mu(dt, dz) \\
+ \int_{0<|z|\leq 1} c(X_{t-}, z)\{\mu(dt, dz) - \nu(dz)dt\} 
\]
\[= X_0 ,
\]
The functions \(a(\cdot), b(\cdot)\) and \(c(\cdot)\) may have a different number of arguments. For example, if the model is homogeneous in time and drift and diffusion coefficients are entirely specified, then we will use the notation \(a(x)\) and \(b(x)\) and describe the diffusion model simply as \(dX_t = a(X_t)dt + b(X_t)dW_t\). And so forth. Detailed hypotheses and regularity conditions on the coefficients \(a(\cdot), b(\cdot)\) and \(c(\cdot)\) for each class of models will be given in the next sections. Nevertheless, it is important to remark that these notations only matter to the mathematical description of the model while each coefficient is passed to \texttt{yuima} methods as \texttt{R} mathematical expressions. This means that, for example, \(a(t, X_t, \theta) = t \cdot \sqrt{\theta} X_t\) will be passed as \(t \ast \texttt{sqrt(x \ast theta)}\), therefore, the order of the arguments is not relevant to \texttt{R} as well as its mathematical description as long as it is consistent through each specific section. Further, \texttt{yuima} is able to accept any user-specified notation for the state variable \(x\) (for \(X_t\)) and the time variable \(t\) and the remaining term of an \texttt{R} expression will be interpreted as parameter as will be explained in Section \ref{sect:examples}. We are now able to give an overview of the main slots of the most important class of the \texttt{yuima} package.

The \texttt{‘yuima.model’} class contains information about the SDE of interest. The constructor function \texttt{setModel} is used to give a mathematical description of the SDE. All functions in the package are assumed to get as much information as possible from the model instead of replicating the same code everywhere. If there are missing pieces of information, we may change or extend the description of the model.

An object of class \texttt{‘yuima.model’} contains several slots listed below. To see inside its structure, one can use the \texttt{R} command \texttt{str} on a \texttt{‘yuima’} object.

- \texttt{drift} is an \texttt{R} vector of expressions which contains the drift specification;
- \texttt{diffusion} is itself a list of 1 slot which describes the diffusion coefficient matrix relative to first noise;
- \texttt{hurst} is the Hurst index of the fractional Brownian motion, by default \(0.5\) meaning a standard Brownian motion. More details will be given in Section \ref{sect:hurst};
- \texttt{parameter}, which is a short name for “parameters”, is a list with the following entries (more details in Section \ref{sect:parameters}):
  - \texttt{all} contains the names of all the parameters found in the diffusion and drift coefficient;
  - \texttt{common} contains the names of the parameters in common between the drift and diffusion coefficients;
  - \texttt{diffusion} contains the parameters belonging to the diffusion coefficient;
  - \texttt{drift} contains the parameters belonging to the drift coefficient;
jump contains the parameters belonging to the coefficient of the Lévy noise;

measure contains the parameters describing the Lévy measure (explained details in Section 3.6);

- measure specifies the measure of the Lévy noise (see Section 3.6);

- measure.type is a switch to specify how the Lévy measure is described (see Section 3.6);

- state.variable and time.variable, by default, are assumed to be x and t but the user can freely choose them and they matter to the right-hand side of the equation of the SDE. The ‘yuima.model’ class assumes that the user either uses default names for state.variable and time.variable variables or specifies his own names. All other symbols are considered parameters and distributed accordingly in the parameter slot. Example of use will be given in Section 3.1;

- jump.variable indicates the name of the variable used in the description of the Lévy component (see Section 3.6);

- solve.variable contains a vector of variable names, each element corresponds to the name of the solution variable (left-hand side) of each equation in the model, in the corresponding order. An example of use can be found in Section 3.4;

- noise.number indicates the number of sources of noise;

- xinit is the initial value of the SDE;

- equation.number represents the number of equations, i.e., the number of one-dimensional SDE’s;

- dimension reports the dimensions of the parameter space. It is a list of the same length of parameter with the same names;

- J.flag is for internal use only.

As seen in the above, the parameter space is accurately described internally in a ‘yuima’ object because in inference for SDE’s, estimators of different parameters have different properties. Usually, the rate of convergence for estimators in the diffusion coefficient are similar to the ones in the i.i.d. (independent and identically distributed) sampling while estimators of parameters in the drift coefficient are slower or, in some cases, not even consistent. The yuima package always tries to implement the best statistical inference for the given model under the observed sampling scheme.

### 3. Model specification

In order to show how general the approach of the yuima package is, we present some examples. Throughout this section we assume that all the SDE’s exist while in Section 6 we will give regularity conditions needed to have a properly defined statistical model.
3.1. One-dimensional diffusion processes

Assume that we want to describe the following SDE

\[ dX_t = -3X_t \, dt + \frac{1}{1 + X_t^2} \, dW_t. \]

In the above \( a(x) = -3x \) and \( b(x) = \frac{1}{1 + x^2} \) according to the notation of the previous section and \( W_t \) is a standard Wiener process. This can be described in \texttt{yuima} by specifying the drift and diffusion coefficients as plain R expressions passed as strings

\[ R> mod1 <- \texttt{setModel}(\text{drift} = "-3 \times x", \text{diffusion} = "1/(1 + x^2)") \]

By default, \texttt{yuima} assumes that the state variable is \( x \) and the time variable is \( t \) and the solve variable is again \( x \). Notice that the left-hand side of the equation is implicit, this is why \texttt{’yuima.model’} has the slot \texttt{solve.variable}. The user should not be worried about the warning raised by \texttt{yuima} at this stage, as this is just to inform her on the implicit assumption on the solution variable. At this point, the package fills the proper slots of the \texttt{’yuima’} object

\[ R> \text{str(mod1)} \]

```
Formal class 'yuima.model' [package "yuima"] with 16 slots
 ..@ drift : expression((-3 * x))
 ..@ diffusion :List of 1
     ..$ : expression(1/(1 + x^2))
 ..@ hurst : num 0.5
 ..@ jump.coeff : expression()
 ..@ measure : list()
 ..@ measure.type : chr(0)
 ..@ parameter :Formal class 'model.parameter' [package "yuima"] ...
     .. .. ..@ all : chr(0)
     .. .. ..@ common : chr(0)
     .. .. ..@ diffusion: chr(0)
     .. .. ..@ drift : chr(0)
     .. .. ..@ jump : chr(0)
     .. .. ..@ measure : chr(0)
 ..@ state.variable : chr "x"
 ..@ jump.variable : chr(0)
 ..@ time.variable : chr "t"
 ..@ noise.number : num 1
 ..@ equation.number: int 1
 ..@ dimension : int [1:6] 0 0 0 0 0 0
 ..@ solve.variable : chr "x"
 ..@ xinit : num 0
 ..@ J.flag : logi FALSE
```

From the above, it is possible to see that the jump coefficient is void and the Hurst parameter is set to 0.5, because this is a model where the driving process is the standard Brownian
Figure 2: The \texttt{plot} function is used to draw a trajectory of a simulated ‘yuima’ object.

motion, i.e., a fractional Brownian motion if Hurst index $H = \frac{1}{2}$. Now, with \texttt{mod1} in hands, it is extremely easy to simulate a trajectory by the Euler-Maruyama scheme of the process as follows

\begin{verbatim}
R> set.seed(123)
R> X <- simulate(mod1)
\end{verbatim}

which can be plotted using the command \texttt{plot}

\begin{verbatim}
R> plot(X)
\end{verbatim}

and the result is shown in Figure 2.

The \texttt{simulate} function fills in addition the two slots \texttt{data} and \texttt{sampling} of the ‘yuima’ object.

\begin{verbatim}
R> str(X, vec.len = 2)
\end{verbatim}

Formal class 'yuima' [package "yuima"] with 5 slots

.. @ data : Formal class 'yuima.data' [package "yuima"] ...
  ... @ original.data: ts [1:101, 1] 0 -0.056 ...
  ... @ attr(*, "dimnames")=List of 2
  ... @ $ : NULL
  ... @ $ : chr "Series 1"
  ... @ attr(*, "tsp")= num [1:3] 0 1 100
  ... @ zoo.data : List of 1
  ... @ $ Series 1:‘zooreg’ series from 0 to 1
Data: num [1:101] 0 -0.056 ...
Index: num [1:101] 0 0.01 0.02 0.03 0.04 ...
Frequency: 100

.. @ model : Formal class 'yuima.model' [package "yuima"] ...
More details on how to change the default sampling scheme for the `simulate` method and how to perform subsampling will be given in Section 4.
3.2. User specified state and time variables

Suppose now that the user wants to specify her own model using a prescribed notation, e.g., some SDE’s like

\[ dY_s = -3sY_s ds + \frac{1}{1+Y_s^2}dW_s, \]

where \( a(s, y) = -3sy \) and \( b(y) = 1/(1 + y^2) \). Then this model can be described in yuima as follows

\[ R> mod1b <- setModel(drift = "-3 * s * y", diffusion = "1/(1 + y^2)", \]
\[ + state.var = "y", time.var = "s") \]

In this case the solution variable is the same as the state variable. Indeed, the ‘yuima.model’ object appears as follows

\[ R> str(mod1b) \]

Once again, the user may use the simulate method to perform simulation.

3.3. Specification of parametric models

Assume now that we want to specify a parametric model like this

\[ dX_t = -\theta X_t dt + \frac{1}{1 + X_t^2}dW_t \]
where \( a(x, \theta) = -\theta x \) and \( b(x, \gamma) = 1/(1 + x^\gamma) \). Then, \texttt{yuima} attempts to distinguish the parameters’ names from the ones of the state and time variables

\[
R> \texttt{mod2 <- setModel(drift = "-theta * x", diffusion = "1/(1 + x^gamma)")}
\]

so, in this case, \texttt{theta} and \texttt{gamma}, which are different from \( x \) and \( t \), are assumed to be parameters. Notice that in the above notation \( \theta \) and \( \gamma \) are generic names for the components of a parameters’ vector \( \theta \) in the notation of Section 2.3.

\[
R> \texttt{str(mod2)}
\]

\texttt{Formal class 'yuima.model' [package "yuima"] with 16 slots}

.. @ drift : expression((-theta * x))
.. @ diffusion : List of 1
... $ : expression(1/(1 + x^gamma))
.. @ hurst : num 0.5
.. @ jump.coeff : expression()
.. @ measure : list()
.. @ measure.type : chr(0)
.. @ parameter : Formal class 'model.parameter' ...
... @ all : chr [1:2] "theta" "gamma"
... @ common : chr(0)
... @ diffusion : chr "gamma"
... @ drift : chr "theta"
... @ jump : chr(0)
... @ measure : chr(0)
.. @ state.variable : chr "x"
.. @ jump.variable : chr(0)
In order to simulate the parametric model it is necessary to specify the values of the parameters \( \theta \) and \( \gamma \) as shown in the next code chunk

```r
R> set.seed(123)
R> X <- simulate(mod2, true.param = list(theta = 1, gamma = 3))
R> plot(X)
```

and the trajectory can be seen in Figure 3.

### 3.4. Multidimensional processes

Next is an example of a system of two SDE's for the couple \((X_{1,t}, X_{2,t})\) driven by three independent Brownian motions \((W_{1,t}, W_{2,t}, W_{3,t})\)

\[
\begin{align*}
\mathrm{d}X_{1,t} &= -3X_{1,t}\mathrm{d}t + dW_{1,t} + X_{2,t}dW_{3,t} \\
\mathrm{d}X_{2,t} &= -(X_{1,t} + 2X_{2,t})\mathrm{d}t + X_{1,t}dW_{1,t} + 3dW_{2,t}
\end{align*}
\]

but this has to be organized into matrix form with a vector of drift expressions and a diffusion matrix

\[
\begin{pmatrix}
\mathrm{d}X_{1,t} \\
\mathrm{d}X_{2,t}
\end{pmatrix} =
\begin{pmatrix}
-3X_{1,t} \\
-X_{1,t} - 2X_{2,t}
\end{pmatrix} \mathrm{d}t +
\begin{pmatrix}
1 & 0 & X_{2,t} \\
X_{1,t} & 3 & 0
\end{pmatrix}
\begin{pmatrix}
dW_{1,t} \\
W_{2,t} \\
W_{3,t}
\end{pmatrix}
\]

For this system it is now necessary to instruct `yuima` about the state variables on both the left-hand side of the equation and the right-hand side of the equation, i.e., there is the need to specify also the `solve.variable` for the left-hand side of the SDE: A vector containing the variable names for the numerical solution, the drift vector and the diffusion matrix are constructed and used as inputs for `setModel`

```r
R> sol <- c("x1", "x2")
R> a <- c("-3 * x1", "-x1 - 2 * x2")
R> b <- matrix(c("1", "x1", "0", "3", "x2", "0"), 2, 3)
R> mod3 <- setModel(drift = a, diffusion = b, solve.variable = sol)
```

Looking at the structure of the `noise.number` slot in `mod3`, one can see that this is now set to 3 which is taken as the number of columns of the diffusion matrix. Again, this model can be easily simulated and the trajectory can be seen in Figure 4.

```r
R> set.seed(123)
R> X <- simulate(mod3)
R> plot(X, plot.type = "single", lty = 1:2)
```
Figure 4: A trajectory of the multidimensional SDE described in mod3.

Notice that the role of `solve.variable` is essential because it allows to specify the left-hand side of an SDE equation. For example, if one wants to specify this model instead of the previous one

\[
\begin{align*}
    dX_{2,t} &= -3X_{1,t}dt + dW_{1,t} + X_{2,t}dW_{3,t}, \\
    dX_{1,t} &= -(X_{1,t} + 2X_{2,t})dt + X_{1,t}dW_{1,t} + 3dW_{2,t},
\end{align*}
\]

the `solve.variable` argument should be specified as `solve.variable = c("x2", "x1")` in place of `solve.variable = c("x1", "x2")`, all the rest being the same as in model `mod3`.

But it is also possible to specify more complex models like the following

\[
\begin{align*}
    dX_{1,t} &= X_{2,t} \left| X_{1,t} \right|^{2/3} dW_{1,t}, \\
    dX_{2,t} &= g(t)dX_{3,t}, \\
    dX_{3,t} &= X_{3,t}(\mu dt + \sigma(\rho dW_{1,t} + \sqrt{1-\rho^2}dW_{2,t})),
\end{align*}
\]

\[
(X_{1,0}, X_{2,0}, X_{3,0}) = (1.0, 0.1, 1.0),
\]

with \( \mu = 0.1, \sigma = 0.2, \rho = -0.7 \) and \( g(t) = 0.4 + (0.1 + 0.2t)e^{-2t} \), for example, and where \( W = (W_1, W_2) \) is a 2-dimensional standard Brownian motion. In order to pass this model to `yuima` we need to rewrite it in matrix form. The solution \( X = (X_1, X_2, X_3) \) takes values on \( \mathbb{R}_+^3 \). This is a stochastic integral equation defined as

\[
X_t = X_0 + \int_0^t a(s, X_s)ds + \int_0^t b(s, X_s)dW_s
\]

with

\[
a(s, x) = \begin{pmatrix}
    0 \\
g(s)\mu x_3 \\
\mu x_3
\end{pmatrix}, \quad b(s, x) = \begin{pmatrix}
x_2|x_1|^{2/3} \\
g(s)\sigma \rho x_3 \\
g(s)\sqrt{1-\rho^2}x_3
\end{pmatrix},
\]

for \( x = (x_1, x_2, x_3) \).
The functions \texttt{f1}, \texttt{f2} and \texttt{f3} are defined in a way that, when the trajectory of the processes crosses zero from above, the trajectory is stopped at zero. Notice that in this way the only visible parameter for \texttt{yuima} is \texttt{mu} as \texttt{rho} and \texttt{sig} are inside the bodies of the functions \texttt{f2} and \texttt{f3}. If we want to instruct \texttt{yuima} about these parameters, they should appear explicitly as arguments of the functions as shown in the following \texttt{R} code:

\begin{verbatim}
R> f2 <- function(t, x1, x2, x3, rho, sig) {
+ ret <- 0
+ if(x3 > 0) ret <- rho * sig * x3
+ return(ret)
+ }
R> f3 <- function(t, x1, x2, x3, rho, sig) {
+ ret <- 0
+ if(x3 > 0) ret <- sqrt(1 - rho^2) * sig * x3
+ return(ret)
+ }
R> diff.coef.matrix <- matrix(c("f1(t, x1, x2, x3)", "f2(t, x1, x2, x3) * g(t)", "f2(t, x1, x2, x3) * g(t)", "0", "f3(t, x1, x2, x3) * g(t)", "f3(t, x1, x2, x3)"), 3, 2)
\end{verbatim}

The \texttt{R} code sets up the model with the specified drift and diffusion coefficients and solves for the state variables as follows:

\begin{verbatim}
R> str(sabr.mod@parameter)
\end{verbatim}
R> sabr.mod <- setModel(drift = c("0", "mu * g(t) * x3", "mu * x3"),
+     diffusion = diff.coeffic_matrix, state.var = c("x1", "x2", "x3"),
+     solve.var = c("x1", "x2", "x3"))
R> str(sabr.mod@parameter)

3.5. Fractional Gaussian noise

The yuima allows for the description of SDE’s driven by fractional Brownian motion of the following type
\[ dX_t = a(X_t)dt + b(X_t)dW_t^H, \]
where \( W^H = (W^H_t, t \geq 0) \) is a normalized fractional Brownian motion (fBM), i.e., zero mean Gaussian processes with covariance function
\[ \text{cov}(W^H_s, W^H_t) = \frac{1}{2} (|s|^{2H} + |t|^{2H} - |t-s|^{2H}) \]
with Hurst exponent \( H \in (0, 1) \). The fractional Brownian motion process is neither Markovian nor a semimartingale for \( H \neq \frac{1}{2} \) but remains Gaussian. For \( H > \frac{1}{2} \), the solution \( X_t \) above presents the long-range dependence property that makes it useful for different applications in biology, physics, ethernet traffic or finance. In order to specify a SDE driven by fractional Gaussian noise it is necessary to specify the value of the Hurst parameter. For example, if we want to specify the following fractional Ornstein-Uhlenbeck model
\[ dY_t = 3Y_t dt + dW_t^H \]
we can proceed as follows

R> mod4A <- setModel(drift = "3 * y", diffusion = 1, hurst = 0.3,
+     solve.var = "y")
R> mod4B <- setModel(drift = "3 * y", diffusion = 1, hurst = 0.7,
+     solve.var = "y")
R> set.seed(123)
R> X1 <- simulate(mod4A, sampling = setSampling(n = 1000))
R> X2 <- simulate(mod4B, sampling = setSampling(n = 1000))
R> par(mfrow = c(2, 1), mar = c(2, 3, 1, 1))
R> plot(X1, main = "H = 0.3")
R> plot(X2, main = "H = 0.7")

and the two trajectories can be seen in Figure 5. In this case, the appropriate slot is now filled

R> str(mod4A)

Formal class 'yuima.model' [package "yuima"] with 16 slots
..@ drift : expression((3 * y))
..@ diffusion : List of 1
...$ : expression(1)
..@ hurst : num 0.3
Figure 5: Trajectories of the fractional Ornstein-Uhlenbeck process for different values of the Hurst parameter.
The user can choose between two simulation schemes, namely the Cholesky method and the Wood and Chan (1994) method. This is done via the argument \texttt{methodfGn} in the \texttt{simulate} method. The default simulation scheme is Wood and Chan (1994) and it is chosen by setting \texttt{methodfGn = "WoodChan"}, the other simply by setting \texttt{methodfGn} to "Cholesky".

3.6. Lév\'e processes

Jump processes can be specified in different ways in mathematics and hence in the \texttt{yuima} package. Let $Z_t$ be a compound Poisson process (i.e., jump sizes follow some distribution, like the Gaussian law, and jumps occur at Poisson times). Then it is possible to consider the simple following SDE which involves jumps

$$dX_t = a(t, X_t, \theta)dt + b(t, X_t, \theta)dW_t + dZ_t,$$

or the more general representation

$$dX_t = a(t, X_t, \theta)dt + b(t, X_t, \theta)dW_t + \int_{|z|>1} c(X_{t-}, z)\mu(dt, dz) + \int_{0<|z|\leq 1} c(X_{t-}, z)\{\mu(dt, dz) - \nu(dz)dt\}$$

with $\mu$ a random measure associated with the jumps of $X$, that is,

$$\mu(dt, dz) = \sum_{s>0} 1_{\{\Delta Z_s \neq 0\}} \delta_{(s, \Delta Z_s)}(dt, dz),$$

where $\delta$ denotes the Dirac measure and $Z$ the driving pure-jump Lévy process of the form

$$Z_t = \int_0^t \int_{|z|\leq 1} z\{\mu(ds, dz) - \nu(dz)ds\} + \int_0^t \int_{|z|>1} z\mu(ds, dz).$$

The Lévy measure $\nu$ is any measure satisfying $\nu(\{0\}) = 0$ and $\int (1 \wedge |z|^2)\nu(dz) < \infty$. In the next example we consider a compound Poisson process with intensity $\lambda = 10$ with Gaussian jumps as driving Lévy process. This model can be specified in \texttt{setModel} using the argument \texttt{measure.type = "CP"} (for compound Poisson). A simple Ornstein-Uhlenbeck process with Gaussian jumps like this

$$dX_t = -\theta X_t dt + \sigma dW_t + dZ_t$$

is then specified as follows (the trajectory is shown in Figure 6).

\begin{verbatim}
R> mod5 <- setModel(drift = c("-theta * x"), diffusion = "sigma",
+   jump.coeff = "1", measure = list(intensity = "10",
+   df = list("dnorm(z, 0, 1)"), measure.type = "CP",
+   solve.variable = "x"))
R> set.seed(123)
R> X <- simulate(mod5, true.p = list(theta = 1, sigma = 3),
+                sampling = setSampling(n = 1000))
R> plot(X)
\end{verbatim}
Another possibility is to specify the jump component via the Lévy measure $\nu(\cdot)$. Without going into too much details, here is an example of specification of a simple Ornstein-Uhlenbeck process with IG (inverse Gaussian) Lévy measure $\nu(\cdot)$ and no Poisson component (see Figure 7)

$$dX_t = -x dt + dZ_t$$

R> mod6 <- setModel(drift = "-x", xinit = 1, jump.coeff = "1", + measure.type = "code", measure = list(df = "rIG(z, 1, 0.1)"))
R> set.seed(123)
R> X <- simulate(mod6, sampling = setSampling(Terminal = 10, n = 10000))
R> plot(X)

The argument measure.type = "code" stands for user defined code.

3.7. Specification of generic models in yuima

In general, the yuima package allows to specify a large family of models which are solutions to

$$dX_t = a(t, X_t, \theta) dt + b(t, X_t, \theta) dW_t + c(t, X_t, \theta) dZ_t$$

using the following interface

```
setModel(drift, diffusion, hurst = 0.5, jump.coeff, measure, measure.type, 
state.variable = "x", jump.variable = "z", time.variable = "t", 
solve.variable, xinit)
```

The coefficient $c(\cdot)$ may also depend on the process $Z$, in this case the admissible form for the coefficient is $c(t, x, \theta, z) = c(t, x, \theta) \cdot z$. In the integral representation corresponding
to formula (1) the coefficient $c(\cdot)$ is not allowed to depend on time though. The \texttt{yuima} package implements many multivariate random number generators (RNG’s) which are needed to simulate Lévy paths including \texttt{rIG} (inverse Gaussian), \texttt{rNIG} (normal inverse Gaussian), \texttt{rbgamma} (bilateral gamma), \texttt{rngamma} (normal gamma) and \texttt{rstable} (stable laws). Other user-defined RNG’s can be used freely.

4. Simulation, sampling and subsampling

The \texttt{simulate} function simulates \textquoteleft yuima' models according to the Euler-Maruyama scheme in the presence of non-fractional diffusion noise and Lévy jumps and uses the Cholesky or the Wood and Chan (1994) methods for the fractional Gaussian noise. For diffusion models without jumps, \texttt{yuima} also implements the space discretized Euler-Maruyama method, which is more efficient, in the sense of mean squared error approximation, than the usual time-discretized Euler-Maruyama scheme. The (time-discretized) Euler-Maruyama simulation scheme defines a grid of times $0 = \tau_0 < \tau_1 < \cdots < \tau_j < \tau_{j+1} < \cdots$, and constructs an approximative solution \{$\tilde{X}_t, t \geq 0$\} of the original process \{$X_t, t \geq 0$\} given by

$$\tilde{X}_{\tau_{j+1}} = \tilde{X}_{\tau_j} + b(\tau_j, \tilde{X}_{\tau_j})(\tau_{j+1} - \tau_j) + c(\tau_j, \tilde{X}_{\tau_j})(W_{\tau_{j+1}} - W_{\tau_j}), \quad j \geq 0.$$  

As the interval $\tau_{j+1} - \tau_j$ is independent to \{$W_t\}_{t \geq \tau_j}$ for each $j$, $W_{\tau_{j+1}} - W_{\tau_j}$ has the same distribution as $\sqrt{\tau_{j+1} - \tau_j} N_j$, where $N_j$ is an i.i.d. sequence of standard normal variables. The basic discretization scheme is equidistant, i.e., $\tau_{j+1} - \tau_j = T/n = \Delta_n$, for all $j \geq 0$. But the \texttt{simulate} method provides also the space-discretized Euler-Maruyama method to solve SDE’s. This is at the moment designed for 1 factor SDE’s only, i.e., the case where the driving Brownian motion $W$ is 1-dimensional. In this case, the discretization scheme \{$\tau_j$\} is defined as

$$\tau_0 = 0, \quad \tau_{j+1} = \inf\{t > \tau_j; |W_t - W_{\tau_j}| = \epsilon\}$$
for each $j \geq 0$. Internally, simulate takes $\epsilon^2 = T/n = \Delta_n$ which coincides with the mean of the interval $\tau_{j+1} - \tau_j$. This space-discretizing scheme is known to be 3 times more efficient than the usual time-discretized scheme one in the sense of the mean squared error (Fukasawa 2011). To make use of the space-discretized Euler-Maruyama scheme, one should use the argument space.discretized = TRUE which, by default, is set to FALSE. Other simulation schemes for 1-dimensional diffusion processes as explained in Iacus (2008) will be implemented in the near future.

The simulate function accepts several arguments including the description of the sampling structure, which is an object of type ‘yuima.sampling’. The setSampling allows for the specification of different sampling parameters including random sampling. Further, the subsampling allows to subsample a trajectory of a simulated SDE or a given time series in the yuima.data slot of a ‘yuima’ object. Sampling and subsampling can be specified jointly as arguments to the simulate function. This is convenient if one wants to simulate data at very high frequency but then return only low frequency data for inference or other applications. In what follows we explain how to specify arguments of these yuima functions. Complete details can be found in the manual pages of the yuima package.

Assume that we want to simulate the following model

$$\begin{align*}
    dX_{1,t} &= -\theta X_{1,t}dt + dW_{1,t} + X_{2,t}dW_{3,t} \\
    dX_{2,t} &= -(X_{1,t} + \gamma X_{2,t})dt + X_{1,t}dW_{1,t} + \delta dW_{2,t}
\end{align*}$$

Now we prepare the model using the setModel constructor function where the variables for the numerical solution sol, the drift vector b and the diffusion matrix s are defined before calling setModel.

```r
R> sol <- c("x1", "x2")
R> b <- c("-theta * x1", "-x1 - gamma * x2")
R> s <- matrix(c("1", "x1", "0", "delta", "x2", "0"), 2, 3)
R> mymod <- setModel(drift = b, diffusion = s, solve.variable = sol)
```

Suppose now that we want to simulate the process on a regular grid on the interval $[0, 3]$ and $n = 3000$ observations. We can prepare the sampling structure as follows

```r
R> samp <- setSampling(Terminal = 3, n = 3000)
```

and let us analyze its content

```r
R> str(samp)
```

Formal class 'yuima.sampling' [package "yuima"] with 11 slots

- ..@ Initial : num 0
- ..@ Terminal : num 3
- ..@ n : num 3000
- ..@ delta : num 0.001
- ..@ grid :List of 1
- ..$ : num [1:3001] 0 0.001 0.002 0.003 0.004 0.005 0.006 0.007 ...
- ..@ random : logi FALSE
As seen from the output, the sampling structure is quite rich and we will show how to specify some of the slots later on. We simulate this process by specifying the `sampling` argument in the `simulate` method.

```r
R> set.seed(123)
R> X2 <- simulate(mymod, sampling = samp)
```

the sampling structure is recorded along with the data in the ‘yuima’ object `X2`

```r
R> str(X2@sampling)
```

Subsampling

The sampling structure can be used to operate subsampling. The next example shows how to perform Poisson random sampling, with two independent Poisson processes, one per coordinate of `X2`.

```r
R> newsamp <- setSampling(random = list(rdist = c(function(x) + rexp(x, rate = 10), function(x) rexp(x, rate = 20))))
R> str(newsamp)
```

```r
Formal class 'yuima.sampling' [package "yuima"] with 11 slots
 ..@ Initial : num 0
 ..@ Terminal : num [1:2] 3 3
 ..@ n : num [1:2] 3000 3000
 ..@ delta : num 0.001
 ..@ grid :List of 1
 ...$ : num [1:3001] 0 0.001 0.002 0.003 0.004 0.005 0.006 0.007 ...
 ..@ random : logi FALSE
 ..@ regular : logi TRUE
 ..@ sdelta : num(0)
 ..@ sgrid : num(0)
 ..@ oindex : num(0)
 ..@ interpolation: chr "pt"
```
Figure 8: An example of Poisson random subsampling: green and red dots are sampled according to two different and independent Poisson processes.

```r
..@ random :List of 1
.. ..$ rdist:List of 2
.. ... ..$ :function (x)
.. ... ..$ :function (x)
..@ regular : logi FALSE
..@ sdelta : num(0)
..@ sgrid : num(0)
..@ oindex : num(0)
..@ interpolation: chr "pt"

In the above we have specified two independent exponential distributions to represent Poisson arrival times. Notice that the slot regular is set to FALSE. We subsample the original trajectory of \( X_2 \) using the subsampling function (see Figure 8)

```r
code
R> newdata <- subsampling(X2, sampling = newsamp)
R> plot(X2,plot.type = "single", lty = c(1, 3), ylab = "X2")
R> points(get.zoo.data(newdata)[[1]], col = "red")
R> points(get.zoo.data(newdata)[[2]], col = "green", pch = 18)
```
Again one can look at the structure of the sampling structure.

Subsampling can be used within the `simulate` function. What is usually done in simulation studies, is to simulate the process at very high frequency but then use data for estimation at a lower frequency (see Figure 10). This can be done in a single step in the following way:

```r
R> set.seed(123)
R> Y.sub <- simulate(mymod, sampling = setSampling(delta = 0.001, n = 1000),
+                     subsampling = setSampling(delta = 0.01, n = 100))
R> set.seed(123)
```
R> Y <- simulate(mymod, sampling = setSampling(delta = 0.001, n = 1000))
R> plot(Y, plot.type = "single")
R> points(get.zoo.data(Y.sub)[[1]], col = "red")
R> points(get.zoo.data(Y.sub)[[2]], col = "green", pch = 18)

In the previous code we have simulated the process twice just to show the effect of the
subsampling, but the reader should use only the line which outputs the simulation to Y.sub
as seen in Figure 11.

R> plot(Y.sub, plot.type = "single")

5. Asymptotic expansion

For numerical computation of the expectation of a random variable, the Monte-Carlo method
gives a universal solution although it is time consuming and involves stochastic errors of a
certain scale depending on the number of replications. An alternative tool is the asymptotic
expansion method that can often give a solution with accuracy comparable or superior to
Monte-Carlo methods. The asymptotic expansion method has an advantage in the computa-
tional time because the approximation is given through an analytic formula.

Let us consider a family of $d$-dimensional diffusion processes $X = (X_t^{(\varepsilon)})_{t \in [0,T]}$ $(\varepsilon \in (0,1])$
specified by the stochastic integral equation

$$X_t^{(\varepsilon)} = x_0 + \int_0^t a(X_s^{(\varepsilon)},\varepsilon)ds + \int_0^t b(X_s^{(\varepsilon)},\varepsilon)dW_s, \quad t \in [0,T]$$

for $\varepsilon \in (0,1]$, where $W_t = (W_{1,t}, \ldots, W_{r,t})$ is an $r$-dimensional Wiener process. The functional
of interest is expressed in the following abstract form
\[ F^{(e)} = \sum_{\alpha=0}^{r} \int_{0}^{T} f_{\alpha}(X^{(e)}_{t}, \varepsilon) dW^{\alpha}_{t} + F(X^{(e)}_{T}, \varepsilon), \quad W^{0}_{t} = t. \] \hspace{1cm} (3)

A typical application is the Asian option pricing. For example, in the Black-Scholes model
\[ dX^{(e)}_{t} = \mu X^{(e)}_{t} dt + \varepsilon X^{(e)}_{t} dW_{t}, \] \hspace{1cm} (4)
the price of the option under zero interest rate is of the form
\[ \mathbb{E} \left[ \max \left( \frac{1}{T} \int_{0}^{T} X^{(e)}_{t} dt - K, 0 \right) \right]. \]

Thus the functional of interest is
\[ F^{(e)} = \frac{1}{T} \int_{0}^{T} X^{(e)}_{t} dt, \quad r = 1 \]
with
\[ f_{0}(x, \varepsilon) = \frac{x}{T}, \quad f_{1}(x, \varepsilon) = 0, \quad F(x, \varepsilon) = 0 \]
in (3).

Similarly, for \( F(x, \varepsilon) = x \), the functional becomes \( F^{(e)} = X^{(e)}_{T} \) and the price of the European call option is \( \mathbb{E}[\max(X^{(e)}_{T} - K, 0)] \). This value has a closed form in the Black-Scholes economy, but it is necessary to apply some numerical method for pricing the Asian option even in this linear case.

Returning to the general system (2)–(3), we will assume that the stochastic system is deterministic in the limit as \( \varepsilon \downarrow 0 \), that is,
\[ b(\cdot, 0) = 0 \quad \text{and} \quad f_{\alpha}(\cdot, 0) = 0 \quad (\alpha = 1, \ldots, r). \]

Since \( X^{(0)}_{t} \) is the deterministic solution to the ordinary differential equation
\[ dX^{(0)}_{t}/dt = a(X^{(0)}_{t}), \quad X^{(0)}_{0} = x_{0}, \]
the functional \( F^{(0)} \) becomes a constant given by
\[ F^{(0)} = \int_{0}^{T} f_{0}(X^{(0)}_{t}, 0) dt + F(X^{(0)}_{T}, 0). \] \hspace{1cm} (5)

Under standard regularity of \( a, b, f_{\alpha} \) and \( F \), it is possible to show \( F^{(e)} \) has a version that is smooth in \( \varepsilon \in [0, 1) \) almost surely, and hence,
\[ \tilde{F}^{(e)} := \varepsilon^{-1}(F^{(e)} - F^{(0)}) \]
admits a stochastic expansion
\[ \tilde{F}^{(e)} \sim \tilde{F}^{[0]} + \varepsilon \tilde{F}^{[1]} + \varepsilon^{2} \tilde{F}^{[2]} + \cdots \] \hspace{1cm} (6)
as $\varepsilon \downarrow 0$. This stochastic expansion makes sense in the Sobolev spaces of the Malliavin calculus. Then the so-called Watanabe's theory \textsuperscript{2} \cite{Watanabe1987} validates the asymptotic expansion of the (generalized) expectation

$$
\mathbb{E}[g(\tilde{F}(\varepsilon))] \sim d_0(g) + \varepsilon d_1(g) + \varepsilon^2 d_2(g) + \cdots
$$

(7)

as $\varepsilon \downarrow 0$ for measurable functions $g$ at most polynomial growth or, more generally, for Schwartz distributions, under the uniform nondegeneracy of the Malliavin covariance of $\tilde{F}(\varepsilon)$\textsuperscript{2}. In the present situation, each $d_i(g)$ is expressed as

$$
d_i(g) = \int g(z)p_i(z)\phi(z;0,v)dz,
$$

where $p_i$ is a polynomial and $\phi(z;0,v)$ is the density of the normal distribution $N(0,v)$ with $v = \text{Cov}[\tilde{F}(0)]$. Polynomials $p_i$ are given by the conditional expectation of multiple Wiener integrals. The expansion (7) holds uniformly in a class of functions $g$.

As mentioned above, Monte-Carlo methods require a huge number of simulations to get the desired accuracy of the calculation of the expectation, while the asymptotic expansion of $F(\varepsilon)$ gives very fast and accurate approximation by analytic formulation. The \texttt{yuima} package provides functions to construct the functional $F(\varepsilon)$ and perform automatic asymptotic expansion based on the Malliavin calculus starting from a ‘\texttt{yuima}’ object. This asymptotic expansion approach to option pricing was proposed in early 1990’s \cite{Yoshida1992a, Takahashi1999, Kunitomo2001}, and several related papers are available today.

Though our method can be applied to the nonlinear system (2)–(3), just as an example, we shall consider the Asian call option of the geometric Brownian motion of Equation 4 with $\mu = 1$ and $x_0 = 1$, and

$$
g(x) = \max\left(F(0) - K + \varepsilon x, 0\right).
$$

(8)

Set the model (2) and the functional (3) as follows:

```
R> model <- setModel(drift = "x", diffusion = matrix("x * e", 1, 1))
R> T <- 1
R> xinit <- 150
R> K <- 100
R> f <- list(expression(x/T), expression(0))
R> F <- 0
R> e <- 0.5
R> yuima <- setYuima(model = model,
+   sampling = setSampling(Terminal = T, n = 1000))
R> yuima <- setFunctional(yuima, f = f, F = F, xinit = xinit, e = e)
```

This time the \texttt{setFunctional} command fills the appropriate slots inside the ‘\texttt{yuima}’ object

```
R> str(yuima@functional)
```

\textsuperscript{2}This condition ensures the smoothness of the distribution of $\tilde{F}(\varepsilon)$. It should be remarked that the Watanabe’s theory is more general than the present use for the variable $\tilde{F}(\varepsilon)$ having a Gaussian principal part $\tilde{F}(0)$.
Formal class 'yuima.functional' [package "yuima"] with 4 slots
  ..@ F : num 0
  ..@ f :List of 2
     .. ..$ : expression(x/T)
     .. ..$ : expression(0)
  ..@ xinit: num 150
  ..@ e : num 0.5

Then the limit $F(0)$ of $F'(\epsilon)$ is easily obtained by calling the function $F0$ on the ‘yuima’ object:

```r
R> F0 <- F0(yuima)
R> F0
[1] 257.6134
```

Set the function $g$ according to (8), where $\epsilon$ is the noise level:

```r
R> rho <- expression(0)
R> epsilon <- e
R> g <- function(x) {
  + tmp <- (F0 - K) + (epsilon * x)
  + tmp[(epsilon * x) < (K - F0)] <- 0
  + tmp
  + }
```

Now we are at the point of computing the coefficients $d_i$ ($i = 0, 1, 2$) in the expansion of the price $E[\max(F'(\epsilon) - K, 0)]$ by applying the function `asymptotic_term`:

```r
R> asymp <- asymptotic_term(yuima, block = 10, rho, g)
R> asymp
```

Then the sums

```r
R> asy1 <- asymp$d0 + e * asymp$d1
R> asy1
[1] 156.608
```

and

```r
R> asy2 <- asymp$d0 + e * asymp$d1 + e^2 * asymp$d2
R> asy2
[,1]
[1,] 157.6082
```
give the first and second order asymptotic expansions of the Asian call price, respectively. We remark that the expansion of $\mathbb{E}[g(\tilde{F}(e))G(e)]$ is also possible by the same method for a functional $G(e)$ having a stochastic expansion like (6). Thus our method works even under the existence of a stochastic discount factor.

One can compare the result of the asymptotic expansion with other well known techniques like Edgeworth series expansion for the log-normal distribution as proposed, e.g., in Levy (1992). This approximation is available through the package fExoticOptions (Wuertz 2012).

```R
R> library("fExoticOptions")
R> levy <- LevyAsianApproxOption(TypeFlag = "c", S = xinit, SA = xinit, + X = K, Time = 1, time = 1, r = 0.0, b = 1, sigma = e)@price
R> levy
[1] 157.7712
```

and the relative difference between the two approximations is $-0.1\%$.

**Asymptotic expansion for general stochastic processes**

Of course, the yuima approach is more general in that the above Lévy approximation only holds when the process $X_t$ is the geometric Brownian motion. We now give an example when the underlying process $X_t$ is the following CIR model

$$dX_t = 0.9X_t \epsilon dt + \sqrt{X_t} \epsilon dW_t, \quad X_0 = 1,$$

and we calculate the asymptotic expansion of a European call option with strike price $K = 10$ for $\epsilon = 0.4$.

```R
R> a <- 0.9
R> e <- 0.4
R> Terminal <- 3
R> xinit <- 1
R> K <- 10
R> drift <- "a * x"
R> diffusion <- "e * sqrt(x)"
R> model <- setModel(drift = drift, diffusion = diffusion)
R> n <- 1000 * Terminal
R> yuima <- setYuima(model = model, + sampling = setSampling(Terminal = Terminal, n = n))
R> f <- list(c(expression(0)), c(expression(0)))
R> F <- expression(x)
R> yuima.ae <- setFunctional(yuima, f = f, F = F, xinit = xinit, e = e)
R> rho <- expression(0)
R> F1 <- F0(yuima.ae)
R> get.ge <- function(x, epsilon, K, F0) {
+ tmp <- (F0 - K) + (epsilon * x[1])
+ tmp[(epsilon * x[1]) > (K - F0)] <- 0
```
R> g <- function(x) {
+   return(get_ge(x,e,K,F1))
+ }
R> time1 <- proc.time()
R> asymp <- asymptotic_term(yuima.ae, block = 100, rho, g)

[1] "compute X.t0"

R> time2 <- proc.time()

We now extract the first and second order terms of the asymptotic expansion from object
asymp

R> ae.value0 <- asymp$d0
R> ae.value0

[1] 0.7219652

R> ae.value1 <- asymp$d0 + e * asymp$d1
R> ae.value1

[1] 0.5787545

R> ae.value2 <- as.numeric(asymp$d0 + e * asymp$d1 + e^2 * asymp$d2)
R> ae.value2

[1] 0.5617722

R> ae.time <- time2 - time1
R> ae.time

user system elapsed
2.437 0.017 2.454

As can be seen, the contribution of the term corresponding to the second order of the asymptotic expansion gives a real contribution to the approximation and the final approximated value 0.56177 can be compared with a Monte-Carlo estimate based on 1000000 replications which is equal to 0.561059, but more demanding in terms of CPU time. The relative difference among the two estimates is 0.1%.

6. Inference for stochastic processes

The yuima package implements several optimal techniques for parametric, semi- and non-parametric estimation of (multidimensional) stochastic differential equations. Most of the
techniques presented below apply to high frequency data, i.e., when \( \Delta_n \), the time lag between two consecutive observations of the process, converges to zero as the number \( n \) of observations increases.

6.1. How to input data into a ‘yuima’ object

Although most of the examples in this section are given on simulated data, the main way to fill up the data slot of a ‘yuima’ object is to use the function setYuima. The function setYuima sets various slots of the ‘yuima’ object. In particular, to estimate a ‘yuima.model’ called \( \text{mod} \) on the data \( X \) one can use a code like this

\[
\text{my.yuima <- setYuima(data = setData(X), model = mod)}
\]

and then pass my.yuima to the inference functions as described in the following.

For example, assuming that an Internet connection is available, the following simple list of commands downloads data from the internet and constructs a ‘yuima’ object with the data slot containing the time series.

\[
\begin{align*}
R & \text{> data <- read.csv("http://chart.yahoo.com/table.csv?s=IBM&g=d&x=.csv")} \\
R & \text{> x <- setYuima(data = setData(data$Close))} \\
R & \text{> str(x@data)}
\end{align*}
\]

6.2. Quasi maximum likelihood estimation

Consider a multidimensional diffusion process

\[
\text{d}X_t = a(X_t, \theta_2)\text{d}t + b(X_t, \theta_1)\text{d}W_t, \quad X_0 = x_0,
\]

where \( W_t \) is an \( r \)-dimensional standard Wiener process independent of the initial variable \( X_0 \). Moreover, \( \theta_1 \in \Theta_1 \subset \mathbb{R}^p, \theta_2 \in \Theta_2 \subset \mathbb{R}^q, a : \mathbb{R}^d \times \Theta_2 \to \mathbb{R}^d \) and \( b : \mathbb{R}^d \times \Theta_1 \to \mathbb{R}^d \times \mathbb{R}^r \). The naming of \( \theta_2 \) and \( \theta_1 \) is theoretically natural in view of the optimal convergence rates of the estimators for these parameters. Given sampled data \( \textbf{X}_n = (X_{t_i})_{i=0,...,n} \) with \( t_i = i\Delta_n \), \( \Delta_n \to 0 \) as \( n \to \infty \), QMLE (quasi maximum likelihood estimator) makes use of the following approximation of the true log-likelihood for multidimensional diffusions

\[
\ell_n(\textbf{X}_n, \theta) = -\frac{1}{2} \sum_{i=1}^{n} \left\{ \log \det(\Sigma_{i-1}(\theta_1)) + \frac{1}{\Delta_n} \sum_{i-1}^{\infty} \Sigma_{i-1}(\theta_1)[(\Delta X_i - \Delta_n a_{i-1}(\theta_2))^\otimes 2] \right\},
\]

where \( \theta = (\theta_1, \theta_2), \Delta X_i = X_{t_i} - X_{t_{i-1}}, \Sigma_{i}(\theta_1) = \Sigma(\theta_1, X_{t_i}), a_i(\theta_2) = a(X_{t_i}, \theta_2), \Sigma = b^\otimes 2, A^\otimes 2 = AA^T \) and \( A^{-1} \) the inverse of \( A, A[B] = \text{tr}(AB) \). Then the QMLE of \( \theta \) is an estimator that satisfies

\[
\hat{\theta} = \arg\max_{\theta} \ell_n(\textbf{X}_n, \theta)
\]

exactly or approximately.

The yuima package implements QMLE via the qmle function. The interface and the output of the qmle function are made as similar as possible to those of the standard mle function in the stats4 package of the basic R system. The main arguments to qmle consist of a ‘yuima’ object and initial values (start) for the optimizer. The ‘yuima’ object must contain the slots model and data. The start argument must be specified as a named list, where the names of the elements of the list correspond to the names of the parameters as they appear in the ‘yuima’
object. Optionally, one can specify named lists of upper and lower bounds to identify the search region of the optimizer. The standard optimizer is BFGS when no bounds are specified. If bounds are specified then L-BFGS-B is used. More optimizers can be added in the future.

**QMLE in practice**

As an example, we consider the simple model

\[
X_t = (2 - \theta_2 X_t)dt + (1 + X_t^2)^{\theta_1}dW_t, \quad X_0 = 1
\]

with \(\theta_1 = 0.2\) and \(\theta_2 = 0.3\). Before calling `qmle`, we generate sampled data \(X_t\), with \(t_i = i \cdot n^{-\frac{2}{3}}\):

```r
R> ymodel <- setModel(drift = "(2 - theta2 * x)",
                   diffusion = "(1 + x^2)^theta1")
R> n <- 750
R> ysamp <- setSampling(Terminal = n^(1/3), n = n)
R> yuima <- setYuima(model = ymodel, sampling = ysamp)
R> set.seed(123)
R> yuima <- simulate(yuima, xinit = 1,
                   true.parameter = list(theta1 = 0.2, theta2 = 0.3))
```

Now the ‘yuima’ object contains both the model and the data slots. We set the initial values for the optimizer as \(\theta_1 = \theta_2 = 0.5\) and we specify them as a named list called `param.init`. We can now use the function `qmle` to estimate the parameters \(\theta_1\) and \(\theta_2\) as follows

```r
R> param.init <- list(theta2 = 0.5, theta1 = 0.5)
R> low.par <- list(theta1 = 0, theta2 = 0)
R> upp.par <- list(theta1 = 1, theta2 = 1)
R> mle1 <- qmle(yuima, start = param.init, lower = low.par, upper = upp.par)
```

where `upp.par` and `low.par` are the upper and lower bounds of the search region used by the optimizer (L-BFGS-B in this case). The estimated coefficients are extracted from the output object `mle1` as follows

```r
R> summary(mle1)
```

Maximum likelihood estimation

Call:
qmle(yuima = yuima, start = param.init, lower = low.par, upper = upp.par)

Coefficients:

<table>
<thead>
<tr>
<th>Estimate</th>
<th>Std. Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>theta1</td>
<td>0.1969182</td>
</tr>
<tr>
<td>theta2</td>
<td>0.008095453</td>
</tr>
</tbody>
</table>

-2 log L: -282.8676
Theoretical remarks on QMLE

Consistency of an estimator is always a required property; otherwise the estimation would lose mathematical backing because the more data the observer obtains, the worse the estimator behaves. For the consistency of \( \hat{\theta}_1 \), we are assuming \( \Delta_n \to 0 \) as \( n \to \infty \). Indeed, under this condition, \( \hat{\theta}_1 \) has asymptotically (mixed) normality (Genon-Catalot and Jacod 1993; Uchida and Yoshida 2012b). On the other hand, one needs \( T = n\Delta_n \to \infty \) for the consistency of \( \hat{\theta}_2 \) since the Fisher information for \( \theta_2 \) is finite for a finite \( T \) and consistent estimation of \( \theta_2 \) is theoretically impossible. When \( T \to \infty \), usually ergodicity is assumed to ensure a law of large numbers and as a result the consistency of \( \hat{\theta}_2 \) is obtained. Moreover, asymptotic normality is also established. We assume the condition \( n\Delta_n^p \to 0 \) for \( p = 2 \) while applying the quasi log-likelihood (10) based on the local Gaussian approximation. In practical applications, reduction of the parameter’s dimension and relaxing the above condition to \( n\Delta_n^p \to 0 \) for \( p \) larger than 2 are extremely important. Adaptive estimation methods were proposed for \( p = 3 \) and for any \( p \) in Yoshida (1992b) and Uchida and Yoshida (2012a), respectively, with the convergence of moments by a large deviation argument. When \( T \) is regarded to be not large, the small sample effect on estimation of \( \theta_2 \) appears, which will be discussed in Section 6.3.2. Modules for QMLE and Bayes estimators are going to be available for jump-diffusions. See Shimizu and Yoshida (2006) and Ogihara and Yoshida (2012).

6.3. Adaptive Bayes estimation

Consider again the diffusion process in (9) and the quasi likelihood defined in (10).

The adaptive Bayes type estimator is defined as follows. First we choose an initial arbitrary value \( \theta_2^* \in \Theta_2 \) and pretend \( \theta_1 \) is the unknown parameter for which we construct the Bayesian type estimator \( \hat{\theta}_1 \) as follows

\[
\hat{\theta}_1 = \left[ \int_{\Theta_1} \exp \{ \ell_n(\mathbf{X}_n, (\theta_1, \theta_2)) \} \pi_1(\theta_1) d\theta_1 \right]^{-1} \int_{\Theta_1} \theta_1 \exp \{ \ell_n(\mathbf{X}_n, (\theta_1, \theta_2^*)) \} \pi_1(\theta_1) d\theta_1 \tag{12}
\]

where \( \pi_1 \) is a prior density on \( \Theta_1 \). According to the asymptotic theory under high frequency samplings, any function \( \pi_1 \) can be used if it is positive on \( \Theta_1 \). For the estimation of \( \theta_2 \), we use \( \hat{\theta}_1 \) to reform the quasi likelihood function. That is, the Bayes type estimator for \( \theta_2 \) is defined by

\[
\hat{\theta}_2 = \left[ \int_{\Theta_2} \exp \{ \ell_n(\mathbf{X}_n, (\hat{\theta}_1, \theta_2)) \} \pi_2(\theta_2) d\theta_2 \right]^{-1} \int_{\Theta_2} \theta_2 \exp \{ \ell_n(\mathbf{X}_n, (\hat{\theta}_1, \theta_2^*)) \} \pi_2(\theta_2) d\theta_2, \tag{13}
\]

where \( \pi_2 \) is a prior density on \( \Theta_2 \). In this way, we obtain the adaptive Bayes type estimator \( \hat{\theta} = (\hat{\theta}_1, \hat{\theta}_2) \) for \( \theta = (\theta_1, \theta_2) \).

Adaptive Bayes estimation is developed in \texttt{yuima} via the method \texttt{adaBayes}. Consider again the model (11) with the same values for the parameters. In order to perform Bayesian estimation, we prepare prior densities for the parameters. For simplicity we use uniform distributions in \([0, 1] \)

\begin{verbatim}
R> prior <- list(theta2 = list(measure.type = "code", + df = "dunif(theta2, 0, 1)")
+ theta1 = list(measure.type = "code", df = "dunif(theta1, 0, 1)")
\end{verbatim}
Then we call `adaBayes`, on the same sample data we used for the `qmle` function, as follows

```r
R> bayes1 <- adaBayes(yuima, start = param.init, prior = prior)
```

and we can compare the adaptive Bayes estimates with the QMLE estimates

```r
R> coef(summary(bayes1))

<table>
<thead>
<tr>
<th></th>
<th>Estimate</th>
<th>Std. Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>theta1</td>
<td>0.1971567</td>
<td>0.008102415</td>
</tr>
<tr>
<td>theta2</td>
<td>0.3071515</td>
<td>0.126514410</td>
</tr>
</tbody>
</table>
```

```r
R> coef(summary(mle1))

<table>
<thead>
<tr>
<th></th>
<th>Estimate</th>
<th>Std. Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>theta1</td>
<td>0.1969182</td>
<td>0.008095453</td>
</tr>
<tr>
<td>theta2</td>
<td>0.2998350</td>
<td>0.126410524</td>
</tr>
</tbody>
</table>
```

The argument `method = "nomcmc"` in `adaBayes` performs numerical integration, otherwise the MCMC method is used.

**Theoretical remarks on adaptive Bayes estimator**

Under the same conditions, the adaptive Bayes estimator $\hat{\theta}_1$ and $\hat{\theta}_2$ perform in the same way as $\theta_1$ and $\theta_2$, respectively. See the remark in Section 6.2 and also Yoshida (1992b) and Uchida and Yoshida (2012b) for details.

**The effect of small sample size on drift estimation**

It is known from the theory that the estimation of the drift in a diffusion process strongly depends on the length of the observation interval $[0,T]$. In our example above, we took $T = n^{\frac{1}{3}}$, with $n = 750$, which is approximately 9.09. Now we reduce the sample size to $n = 500$ and then $T = 7.94$. We then apply both quasi maximum likelihood and adaptive Bayes type estimators to these data

```r
R> n <- 500
R> ysamp <- setSampling(Terminal = n^(1/3), n = n)
R> yuima <- setYuima(model = ymodel, sampling = ysamp)
R> set.seed(123)
R> yuima <- simulate(yuima, xinit = 1,
+    true.parameter = list(theta1 = 0.2, theta2 = 0.3))
R> param.init <- list(theta2 = 0.5, theta1 = 0.5)
R> mle2 <- qmle(yuima, start = param.init,
+    lower = list(theta1 = 0, theta2 = 0),
+    upper = list(theta1 = 1, theta2 = 1))
R> bayes2 <- adaBayes(yuima, start = param.init, prior = prior)
```

and we look at the estimates
\[ R > \text{coef(summary(bayes2))} \]

<table>
<thead>
<tr>
<th>Estimate</th>
<th>Std. Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>theta1</td>
<td>0.1950772</td>
</tr>
<tr>
<td>theta2</td>
<td>0.2467359</td>
</tr>
</tbody>
</table>

\[ R > \text{coef(summary(mle2))} \]

<table>
<thead>
<tr>
<th>Estimate</th>
<th>Std. Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>theta1</td>
<td>0.1947225</td>
</tr>
<tr>
<td>theta2</td>
<td>0.2193002</td>
</tr>
</tbody>
</table>

Compared to the results above, we see that the parameters in the diffusion coefficients are estimated with good quality while for the parameters in the drift the quality of estimation deteriorates. The adaptive Bayes estimator seems to perform a little better though.

### 6.4. Asynchronous covariance estimation

Suppose that two Itô processes are observed only at discrete times in a nonsynchronous manner. We are interested in estimating the covariance of the two processes accurately in such a situation. This type of problem arises typically in high frequency financial time series. Let \( T \in (0, \infty) \) be a terminal time for possible observations. We consider a two-dimensional Itô process \((X_1, X_2)\) satisfying the SDE’s

\[
\begin{align*}
    dX_{l,t} &= \mu_{l,t} dt + \sigma_{l,t} dW_{l,t}, \quad t \in [0, T] \\
    X_{l,0} &= x_{l,0}
\end{align*}
\]

for \( l = 1, 2 \). Here \( W_l \) denote standard Wiener processes with a progressively measurable correlation process \( d\langle W_1, W_2 \rangle_t = \rho_t dt \), \( \mu_{l,t} \) and \( \sigma_{l,t} \) are progressively measurable processes, and \( x_{l,0} \) are initial random variables independent of \((W_1, W_2)\). Diffusion type processes are in the scope but this model can express more sophisticated stochastic structures.

The process \( X_l \) is supposed to be observed over the increasing sequence of times \( T^{l,i} (i \in \mathbb{Z}_{\geq 0}) \) starting at 0, up to time \( T \). Thus, the observables are \((T^{l,i}, X_{l,i})\) with \( T^{l,i} \leq T \). Each \( T^{l,j} \) may be a stopping time, so it possibly depends on the history of \((X_1, X_2)\) as well as on the precedent stopping times. Two sequences of stopping times \( T^{1,i} \) and \( T^{2,j} \) are nonsynchronous, and irregularly spaced, in general. In particular, function \texttt{ccc} from package \texttt{yuima} can be applied to estimate the quadratic variation of a single stochastic process sampled regularly/irregularly.

The parameter of interest is the quadratic covariation between \( X_1 \) and \( X_2 \):

\[
\theta = \langle X_1, X_2 \rangle_T = \int_0^T \sigma_{1,t} \sigma_{2,t} \rho_t dt.
\]  

(14)

The target variable \( \theta \) is random in general and it can be estimated with the nonsynchronous covariance estimator (Hayashi-Yoshida estimator)

\[
U_n = \sum_{i,j; T^{1,i} \leq T, T^{2,j} \leq T} \left( X_{1,T^{1,i}} - X_{1,T^{1,i-1}} \right) \left( X_{2,T^{2,j}} - X_{2,T^{2,j-1}} \right) 1_{\{T^{1,i-1}, T^{1,i}] \cap (T^{2,j-1}, T^{2,j}] \neq \emptyset \}}.
\]  

(15)
That is, the product of any pair of increments \((X_{1,T_1,i}-X_{1,T_1,i-1})\) and \((X_{2,T_2,j}-X_{2,T_2,j-1})\) will make a contribution to the sum only when the respective observation intervals \((T_{i-1}, T_{i}]\) and \((T_{j-1}, T_{j}]\) are overlapping. It is known that \(U_n\) is consistent and has asymptotically mixed normal distribution as \(n \to \infty\) if the maximum length between two consecutive observing times tends to 0. See Hayashi and Yoshida (2005, 2006, 2008a, b) for details.

**Example: Data generation and estimation by yuima package**

We will demonstrate how to apply function `cce` to nonsynchronous high frequency data by simulation. As an example, consider a two dimensional stochastic process \((X_{1,t}, X_{2,t})\) satisfying the SDE

\[
dX_{1,t} = \sigma_{1,t} dB_{1,t}, \quad dX_{2,t} = \sigma_{2,t} dB_{2,t}. \tag{16}
\]

Here \(B_{1,t}\) and \(B_{2,t}\) denote two standard Wiener processes, however they are correlated as

\[
B_{1,t} = W_{1,t}, \quad B_{2,t} = \int_0^t \rho_s dw_{1,s} + \sqrt{1 - \rho_s^2} dw_{2,s}, \tag{17}
\]

where \(W_{1,t}\) and \(W_{2,t}\) are independent Wiener processes, and \(\rho_t\) is the correlation function between \(B_{1,t}\) and \(B_{2,t}\). We consider \(\sigma_{l,t}, l = 1, 2\), and \(\rho_t\) of the following form in this example:

\[
\sigma_{1,t} = \sqrt{1 + t}, \quad \sigma_{2,t} = \sqrt{1 + t^2}, \quad \rho_t = \frac{1}{\sqrt{2}}. \tag{18}
\]

To simulate the stochastic process \((X_{1,t}, X_{2,t})\), we first build the model by `setModel` as before. It should be noted that the method of generating nonsynchronous data can be replaced by a simpler one but we will take a general approach here to demonstrate a usage of `yuima` which shows that it is a comprehensive package for simulation and estimation of stochastic processes.

```R
R> diff.coef.1 <- function(t, x1 = 0, x2 = 0) sqrt(1 + t)
R> diff.coef.2 <- function(t, x1 = 0, x2 = 0) sqrt(1 + t^2)
R> cor.rho <- function(t, x1 = 0, x2 = 0) sqrt(1/2)
R> diff.coef.matrix <- matrix(c("diff.coef.1(t, x1, x2)",
+   "diff.coef.2(t, x1, x2) * cor.rho(t, x1, x2)", 
+   "diff.coef.2(t, x1, x2) * sqrt(1 - cor.rho(t, x1, x2)^2)"), 2, 2)
R> cor.mod <- setModel(drift = c("", ""), diffusion = diff.coef.matrix,
+ solve.variable = c("x1", "x2"))
```

The parameter we want to estimate is the quadratic covariation between \(X_1\) and \(X_2\):

\[
\theta = \langle X_1, X_2 \rangle_T = \int_0^T \sigma_{1,t}\sigma_{2,t}\rho_t dt. \tag{19}
\]

Later, we will compare estimated values with the true value of \(\theta\) given by
The YUIMA Project

```r
R> CC.theta <- function(T, sigma1, sigma2, rho) {
+   tmp <- function(t) return(sigma1(t) * sigma2(t) * rho(t))
+   integrate(tmp, 0, T)
+ }
```

For the sampling scheme, we will consider independent Poisson sampling. That is, each configuration of the sampling times $T_l,i$ is realized as the Poisson random measure with intensity $n p_l$, and the two random measures are independent of each other as well as the stochastic processes. Under this scheme the data become asynchronous. It is known that

$$n^{1/2}(U_n - \theta) \to N(0,c), \quad (20)$$

as $n \to \infty$, where

$$c = \left( \frac{2}{p_1} + \frac{2}{p_2} \right) \int_0^T (\sigma_{1,t} \sigma_{2,t})^2 dt + \left( \frac{2}{p_1} + \frac{2}{p_2} - \frac{2}{p_1 + p_2} \right) \int_0^T (\sigma_{1,t} \rho_{t} \sigma_{2,t} \rho_{t})^2 dt. \quad (21)$$

```r
R> set.seed(123)
R> Terminal <- 1
R> n <- 1000
R> theta <- CC.theta(T = Terminal, sigma1 = diff.coef.1,
+                     sigma2 = diff.coef.2, rho = cor.rho)$value
R> cat(sprintf("theta = %5.3f\n", theta))
```

theta = 1.000

so in our case $\theta = 1$.

```r
R> yuima.samp <- setSampling(Terminal = Terminal, n = n)
R> yuima <- setYuima(model = cor.mod, sampling = yuima.samp)
R> X <- simulate(yuima)
```

cce takes the sample and returns an estimate of the quadratic covariation. For example, for the complete data in Figure 12, we obtain the following estimates

```r
R> cce(X)
```

$\text{covmat}$

```
[,1]       [,2]
[1,] 1.491938  1.086078
[2,] 1.086078  1.474730
```

$\text{cormat}$

```
[,1]       [,2]
[1,] 1.0000000  0.7321992
[2,]  0.7321992  1.0000000
```

```r
R> plot(X, main = "complete data")
```
We now apply random sampling in the following way: we define a new sampling structure via `setSampling` specifying in the argument `random` a list which contains a vector of random distributions. For the $i$th component of $X$ we specify an exponential distribution with rate $n \cdot p_i / T$ for the random times. This will generate Poisson random times with the corresponding rate.

```r
R> p1 <- 0.2
R> p2 <- 0.3
R> newsamp <- setSampling(
+   random = list(rdists = c(function(x) rexp(x, rate = p1 * n / Terminal),
+                  function(x) rexp(x, rate = p1 * n / Terminal)))))
```

Now we use the `subsampling` function to subsample the original data $X$ into new asynchronous data $Y$ (see Figure 13).

```r
R> Y <- subsampling(X, sampling = newsamp)
R> cce(Y)
```

```r
covmat
[,1]      [,2]
[1,] 1.397269  1.070313
[2,] 1.070313  1.338464

cormat
[,1]      [,2]
[1,] 1.0000000 0.7826494
[2,] 0.7826494 1.0000000
```
asynchronous data

Figure 13: The asynchronous data generated from the simulated ones using Poisson random subsampling.

```r
R> plot(Y, main = "asynchronous data")
```

**Asynchronous estimation for nonlinear systems**

Consider now the two-dimensional system with nonlinear feedback

\[
\begin{align*}
\mathrm{d}X_t &= Y_t \mathrm{d}t + \sigma_1(t, X_t, Y_t) \mathrm{d}W_t \\
\mathrm{d}Y_t &= -X_t \mathrm{d}t + \rho(t, X_t, Y_t) \sigma_2(t, X_t, Y_t) \mathrm{d}W_t + \sigma_2(t, X_t, Y_t) \sqrt{1 - \rho^2(t, X_t, Y_t)} \mathrm{d}B_t
\end{align*}
\]

with

\[
\begin{align*}
\sigma_1(t, X_t, Y_t) &= \sqrt{|X_t(1 + t)|}, \\
\sigma_2(t, X_t, Y_t) &= \sqrt{|Y_t|}, \\
\rho(t, X_t, Y_t) &= \frac{1}{1 + X_t^2}
\end{align*}
\]

and \(W_t, B_t\), two independent Brownian motions. We construct the model and we generate data from it by sampling a path of the process

```r
R> b1 <- function(x, y) y
R> b2 <- function(x, y) -x
R> s1 <- function(t, x, y) sqrt(abs(x) * (1 + t))
R> s2 <- function(t, x, y) sqrt(abs(y))
R> cor.rho <- function(t, x, y) 1/(1 + x^2)
R> diff.mat <- matrix(c("s1(t, x, y)", "s2(t, x, y) * cor.rho(t, x, y)", "", "+ s2(t, x, y) * sqrt(1 - cor.rho(t, x, y)^2)"), 2, 2)
R> cor.mod <- setModel(drift = c("b1","b2"), diffusion = diff.mat, + solve.variable = c("x", "y"), state.var = c("x", "y"))
R> set.seed(111)
R> Terminal <- 1
R> n <- 10000
R> yuima.samp <- setSampling(Terminal = Terminal, n = n)
```
We apply the same Poisson random sampling so that the object \( Y \) will contain asynchronous data (see Figure 14).

\[
R> \text{p1} \leftarrow 0.2 \\
R> \text{p2} \leftarrow 0.3 \\
R> \text{newsamp} \leftarrow \text{setSampling}( \\
+ \quad \text{random} = \text{list}( \text{rdist} = \text{c}( \text{function}(x) \ \text{rexp}(x, \ \text{rate} = \text{p1} * \ \text{n}/\text{Terminal}), \ \\
+ \quad \text{function}(x) \ \text{rexp}(x, \ \text{rate} = \text{p1} * \ \text{n}/\text{Terminal}))) \\
R> Y \leftarrow \text{subsampling}(\text{yuima, sampling} = \text{newsamp}) \\
R> \text{plot}(Y, \ \text{main} = "\text{asynchronous data (non linear case)}")
\]

The estimated covariance for the complete trajectory \( \text{yuima} \) is now compared with the one obtained on asynchronous data \( Y \).

\[
R> \text{cce}(\text{yuima})
\]

\[
\text{covmat} \\
[\text{,1}] \ [\text{,2}] \\
[1,] \ 2.7092720 \ 0.7803843 \\
[2,] \ 0.7803843 \ 3.4705059
\]

\[
\text{cormat} \\
[\text{,1}] \ [\text{,2}] \\
[1,] \ 1.0000000 \ 0.2544988 \\
[2,] \ 0.2544988 \ 1.0000000
\]
### 6.5. Change-point analysis

Consider a multidimensional SDE of the form

\[ dY_t = a_t dt + b(X_t, \theta) dW_t, \quad t \in [0, T], \]

where \( W_t \) is a \( r \)-dimensional Wiener process and \( a_t \) and \( X_t \) are multidimensional processes, \( \theta \in \Theta \subset \mathbb{R}^p, b : \mathbb{R}^d \times \Theta \rightarrow \mathbb{R}^d \times \mathbb{R}^r, \) is the diffusion coefficient (volatility) matrix.

When \( Y = X \) the model above is a diffusion model. The process \( a_t \) may have jumps but should not explode and it is treated as a nuisance in this model. The change-point problem for the volatility is formalized as follows

\[
Y_t = \begin{cases} 
Y_0 + \int_0^t a_s ds + \int_0^t b(X_s, \theta_0^*) dW_s & \text{for } t \in [0, \tau^*) \\
Y_{\tau^*} + \int_{\tau^*}^t a_s ds + \int_{\tau^*}^t b(X_s, \theta_1^*) dW_s & \text{for } t \in [\tau^*, T].
\end{cases}
\]

The change-point \( \tau^* \) instant is unknown and is to be estimated, along with \( \theta_0^* \) and \( \theta_1^* \), from the observations sampled from the path of \( (X, Y) \). The \texttt{yuima} package implements the quasi maximum likelihood approach as in Iacus and Yoshida (2012) described in the following. Let \( \Delta_i Y = Y_{t_i} - Y_{t_{i-1}} \) and define

\[
\Phi_n(t; \theta_0, \theta_1) = \sum_{i=1}^{[nt/T]} G_i(\theta_0) + \sum_{i=[nt/T]+1}^{n} G_i(\theta_1),
\]

(22)

with

\[
G_i(\theta) = \log \det S(X_{t_{i-1}}, \theta) + \Delta_n^{-1}(\Delta_i Y) (\Delta_i Y)^{-1} (\Delta_i Y),
\]

(23)

and \( S = \sigma^2 \). Suppose that there exists an estimator \( \hat{\theta}_k \) for each \( \theta_k, \ k = 0, 1 \). In case \( \theta_k^* \) are known, we define \( \hat{\theta}_k \) just as \( \hat{\theta}_k = \theta_k^* \). The change-point estimator of \( \tau^* \) is

\[
\hat{\tau} = \arg \min_{t \in [0,T]} \Phi_n(t; \hat{\theta}_0, \hat{\theta}_1).
\]

### Example of volatility change-point estimation for 2-dimensional SDE’s

One example of a model that can be analyzed by this technique is, for example, the 2-dimensional SDE

\[
\begin{pmatrix}
    dX_{1,t} \\
    dX_{2,t}
\end{pmatrix} = \begin{pmatrix}
    a_1(X_{1,t}) \\
    a_2(X_{2,t})
\end{pmatrix} dt + \begin{pmatrix}
    \theta_{1,k} \cdot X_{1,t} & 0 \cdot X_{1,t} \\
    0 \cdot X_{2,t} & \theta_{2,k} \cdot X_{2,t}
\end{pmatrix} \begin{pmatrix}
    dW_{1,t} \\
    dW_{2,t}
\end{pmatrix}, \quad t \in [0, T],
\]

In R:

```r
> cce(Y)

$\text{covmat}$

```

```
[,1]   [,2]
[1,]  2.7134061 0.7330106
[2,]  0.7330106 3.3945659
```

```
$\text{cormat}$

```

```
[,1]   [,2]
[1,]  1.0000000 0.2415242
[2,]  0.2415242 1.0000000
```

```
where \(a_1(\cdot)\) and \(a_2(\cdot)\) are any functions and \(\theta_{1,k}\) and \(\theta_{2,k}\) the value of the parameters before \((k = 0)\) and after \(k = 1\) the change-point. Just for simplicity and in order to simulate some data, we specify some specific form for \(a_1(\cdot)\) and \(a_2(\cdot)\) but this information will not be used in the change-point analysis. For example, we will simulate the following 2-dimensional SDE

\[
\begin{pmatrix}
\frac{dX_{1,t}}{dt} \\
\frac{dX_{2,t}}{dt}
\end{pmatrix} = \begin{pmatrix}
\sin(X_{1,t}) & 0 \\
3 - X_{2,t} & 0
\end{pmatrix} dt + \begin{pmatrix}
\theta_{1,k} \cdot X_{1,t} & 0 \cdot X_{1,t} \\
0 \cdot X_{2,t} & \theta_{2,k} \cdot X_{2,t}
\end{pmatrix} \begin{pmatrix}
\frac{dW_{1,t}}{dt} \\
\frac{dW_{2,t}}{dt}
\end{pmatrix}, \quad t \in [0, T],
\]

with change-point instant at time \(\tau = 4\) and \(T = 10\). First, we describe the model to be simulated

```r
R> diff.matrix <- matrix(c("theta1.k * x1", "0 * x2", "0 * x1", + "theta2.k * x2"), 2, 2) 
R> drift.c <- c("sin(x1)", "3 - x2") 
R> drift.matrix <- matrix(drift.c, 2, 1) 
R> ymodel <- setModel(drift = drift.matrix, diffusion = diff.matrix, + time.variable = "t", state.variable = c("x1", "x2"), + solve.variable = c("x1", "x2"))
```

and then simulate two trajectories. One up to the change-point \(\tau = 4\) with parameters \(\theta_{1,0} = 0.5\) and \(\theta_{2,0} = 0.3\)

```r
R> n <- 1000 
R> set.seed(123)
R> t0 <- list(theta1.k = 0.5, theta2.k = 0.3)
R> T <- 10
R> tau <- 4
R> pobs <- tau/T 
R> ysamp1 <- setSampling(n = n * pobs, Initial = 0, delta = 0.01) 
R> yuima1 <- setYuima(model = ymodel, sampling = ysamp1) 
R> yuima1 <- simulate(yuima1, xinit = c(3, 3), true.parameter = t0) 
R> x1 <- yuima1@data@zoo.data[[1]]
R> x1 <- as.numeric(x1[length(x1)])
R> x2 <- yuima1@data@zoo.data[[2]]
R> x2 <- as.numeric(x2[length(x2)])
```

now we generate the second trajectory with parameters \(\theta_{1,1} = 0.2\) and \(\theta_{2,1} = 0.4\). For this trajectory, the initial value is set to the last value of the first trajectory stored in \(x1\) and \(x2\) for the two components of the process (see Figure 15)

```r
R> t1 <- list(theta1.k = 0.2, theta2.k = 0.4)
R> ysamp2 <- setSampling(Initial = n * pobs * 0.01, n = n * (1 - pobs), + delta = 0.01) 
R> yuima2 <- setYuima(model = ymodel, sampling = ysamp2) 
R> yuima2 <- simulate(yuima2, xinit = c(x1, x2), true.parameter = t1)
```

finally we collate the two trajectories
As said, the change-point analysis does not consider the information coming from the drift part of the model and \texttt{yuima} ignores this internally. Just to make clear that the information on the drift term is not considered by the function \texttt{CPoint}, we redefine the \texttt{yuima} model removing the information coming from the drift and then adding back the data.

\begin{verbatim}
R> noDriftModel <- setModel(drift = c("0", "0"), diffusion = diff.matrix,
 +     time.variable = "t", state.variable = c("x1", "x2"),
 +     solve.variable = c("x1", "x2"))
R> noDriftModel <- setYuima(noDriftModel, data = yuima@data)
R> noDriftModel@model@drift
expression(((0), (0))
\end{verbatim}

First we show that there is no difference in using the complete model or the model without drift. For simplicity, we assume to know the true values of the parameters for $\theta_{1,k}$ and $\theta_{2,k}$.

\begin{verbatim}
R> t.test <- CPoint(yuima, param1 = t0, param2 = t1)
R> t.test$tau
[1] 3.98
\end{verbatim}

Figure 15: The 2-dimensional trajectory with change-point around $\tau = 4$. 
As can be seen, the above estimates of the change-point are the same for the complete model `yuima` and the model without drift `noDriftModel`.

**An example of two stage estimation**

In practical situations, the initial values of the parameters are not known and there is the need to provide preliminary estimators of them. One possible approach is the two stage change-point estimation approach as explained in [Iacus and Yoshida (2012)](https://link.to/Iacus2012). The idea is to take a small subset of observations at the very beginning and the end of the time series, estimate a change-point and then refine the estimation.

To this aim, the `yuima` package contains two functions which are useful in the framework of change-point or sequential analysis. The function `qmleL` estimates a model by quasi maximum likelihood using observations in the time interval $[0,t]$ where $t$ can be specified by the user. In our example, we set $t = 2$. Similarly for `qmleR`, which uses only observations in the time interval $[t,T]$. In our example, we take $t = 8$.

```r
R> qmleL(noDriftModel, t = 2, start = list(theta1.k = 0.1, theta2.k = 0.1),
    +    lower = list(theta1.k = 0, theta2.k = 0),
    +    upper = list(theta1.k = 1, theta2.k = 1),
    +    method = "L-BFGS-B") -> estL
R> qmleR(noDriftModel, t = 8, start = list(theta1.k = 0.1, theta2.k = 0.1),
    +    lower = list(theta1.k = 0, theta2.k = 0),
    +    upper = list(theta1.k = 1, theta2.k = 1),
    +    method = "L-BFGS-B") -> estR
R> t0.est <- coef(estL)
R> t1.est <- coef(estR)
```

and now we proceed with change-point estimation

```r
R> t.est3 <- CPoint(noDriftModel, param1 = t0.est, param2 = t1.est)
R> t.est3

$tau
[1] 3.99

$param1
  theta1.k theta2.k
0.4723067 0.2899005

$param2
  theta1.k theta2.k
0.2515379 0.5518635
```
Notice that, even if the estimated parameters are not too accurate because we use small subsets of observations, the change-point estimate remains good. We can now refine the estimate of the change-point and the parameters by iterating the above procedure.

\[
R> \text{qmleL(noDriftModel, t = t.est3$tau, start = list(theta1.k = 0.1, theta2.k = 0.1), lower = list(theta1.k = 0, theta2.k = 0), upper = list(theta1.k = 1, theta2.k = 1), method = "L-BFGS-B") -> estL}
\]

\[
R> \text{qmleR(noDriftModel, t = t.est3$tau, start = list(theta1.k = 0.1, theta2.k = 0.1), lower = list(theta1.k = 0, theta2.k = 0), upper = list(theta1.k = 1, theta2.k = 1), method = "L-BFGS-B") -> estR}
\]

\[
R> \text{t02s.est <- coef(estL)}
\]

\[
R> \text{t12s.est <- coef(estR)}
\]

\[
R> \text{t2s.est3 <- CPoint(noDriftModel, param1 = t02s.est, param2 = t12s.est)}
\]

\[
R> \text{t2s.est3}
\]

\[
\begin{array}{c}
\text{tau} \\
[1] 3.98
\end{array}
\]

\[
\begin{array}{c}
\text{param1} \\
\null \text{theta1.k} \text{ theta2.k} \\
0.4863371 0.2991717
\end{array}
\]

\[
\begin{array}{c}
\text{param2} \\
\null \text{theta1.k} \text{ theta2.k} \\
0.2614729 0.5295390
\end{array}
\]

### 6.6. LASSO model selection

The least absolute shrinkage and selection operator (LASSO) is a useful and well studied approach to the problem of model selection and its major advantage is the simultaneous execution of both parameter estimation and variable selection (Tibshirani 1996; Knight and Fu 2000; Efron, Hastie, Johnstone, and Tibshirani 2004).

To simplify the idea: take a full specified regression model

\[
E(Y | X_1, \ldots, X_k) = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \cdots + \theta_k X_k
\]

perform least squares estimation under \(L_1\) constraints, i.e.,

\[
\hat{\theta} = \arg\min_\theta \left\{ (Y - \theta X)^\top (Y - \theta X) + \sum_{i=1}^k |\theta_i| \right\}
\]

Model selection occurs when some of the \(\theta_i\) are estimated as zeros. The same idea can be applied to diffusion processes. Let \(X_t\) be a diffusion process solution to

\[
dX_t = a(X_t, \alpha)dt + b(X_t, \beta)dW_t
\]
\[ \alpha = (\alpha_1, \ldots, \alpha_p) \in \Theta_p \subset \mathbb{R}^p, \quad p \geq 1 \]
\[ \beta = (\beta_1, \ldots, \beta_q) \in \Theta_q \subset \mathbb{R}^q, \quad q \geq 1 \]

with \( b : \Theta_p \times \mathbb{R}^d \rightarrow \mathbb{R}^d, \sigma : \Theta_q \times \mathbb{R}^d \rightarrow \mathbb{R}^d \times \mathbb{R}^m \) and \( W_t, t \in [0, T] \), is a standard Brownian motion in \( \mathbb{R}^m \). We assume that the functions \( a \) and \( b \) are known up to \( \theta \) and \( \theta_0 = (\alpha_0^T, \beta_0^T) \) its unknown true value. Let \( H_n(X_n, \theta) = -\ell_n(X_n, \theta) \) from Equation 10. The QMLE \( \hat{\theta} \) for this model is the solution of the following problem

\[ \hat{\theta} = (\hat{\alpha}^T, \hat{\beta}^T)^T = \arg \min_{\theta} H_n(X_n, \theta) \]

The adaptive LASSO estimator is defined as the solution to the quadratic problem under \( L_1 \) constraints

\[ \hat{\theta} = (\hat{\alpha}^T, \hat{\beta}^T)^T = \arg \min_{\theta} F(\theta). \]

with

\[ F(\theta) = (\theta - \hat{\theta})^T \hat{H}_n(X_n, \hat{\theta})(\theta - \hat{\theta}) + \sum_{j=1}^{p} \lambda_{n,j} |\hat{\alpha}_j| + \sum_{k=1}^{q} \gamma_{n,k} |\hat{\beta}_k| \]

and \( \hat{H}_n \) is the matrix of second partial derivatives of \( H \) with respect to the vector \( \theta \). For more details see Gregorio and Iacus (2012). The tuning parameters should be chosen as in Zou (2006) in the following way

\[ \lambda_{n,j} = \lambda_0 |\hat{\alpha}_j|^{-\delta_1}, \quad \gamma_{n,k} = \gamma_0 |\hat{\beta}_k|^{-\delta_2}, \]

where \( \hat{\alpha}_j \) and \( \hat{\beta}_k \) are the unpenalized QMLE's of \( \alpha_j \) and \( \beta_k \) respectively, \( \delta_1, \delta_2 > 0 \) and usually taken unitary.

An example of model selection for interest rates data

The lasso method is implemented in the \texttt{yuima} package. Let us consider the full CKLS model (Chan, Karolyi, Longstaff, and Sanders 1992)

\[ dX_t = (\alpha + \beta X_t)dt + \sigma X_t^\gamma dW_t \]

and let us try to estimate the parameter on the U.S. Interest Rates monthly data from 06/1964 to 12/1989 (see Figure 16). We prepare the data from package \texttt{Ecdat} (Croissant 2014), the model and the constraints for optimization

\begin{verbatim}
R> data("Irates", package = "Ecdat")
R> rates <- Irates[, "r1"]
R> plot(rates)
R> X <- window(rates, start = 1964.471, end = 1989.333)
R> mod <- setModel(drift = "alpha + beta * x",
+ diffusion = matrix("sigma * x^gamma", 1, 1))
R> yuima <- setYuima(data = setData(X), model = mod)
R> lambda10 <- list(alpha = 10, beta= 10, sigma = 10, gamma = 10)
R> start <- list(alpha = 1, beta = -.1, sigma = .1, gamma = 1)
R> l0v <- list(alpha = -5, beta = -5, sigma = -5, gamma = -5)
R> upp <- list(alpha = 8, beta = 8, sigma = 8, gamma = 8)
\end{verbatim}
where \( \lambda_{10} \) stores the penalty terms \( \lambda_0 \) and \( \gamma_0 \) of (24) of the LASSO method.

Now we apply the \texttt{lasso} function

```r
R> lasso10 <- lasso(yuima, lambda10, start = start, lower = low, upper = upp,
+                     method = "L-BFGS-B")
```

From which we see that, instead of the general model

\[
\mathrm{d}X_t = (\alpha + \beta X_t)\mathrm{d}t + \sigma X_t^\gamma \mathrm{d}W_t
\]

```r
R> round(lasso10$mle, 2)
sigma gamma alpha beta
 0.13  1.44  2.08 -0.26
```

the LASSO method selects the reduced model

\[
\mathrm{d}X_t = 0.6\mathrm{d}t + 0.12X_t^3 \mathrm{d}W_t.
\]

Notice that this model is not an ergodic one, indicating that the LASSO method shows that the real data are indeed not stationary, but still in the family of CKLS models.
7. Miscellanea and roadmap of the YUIMA project

Other statistical techniques have already been implemented in the development version of the *yuima* package although not yet released into the current distribution. Among these we mention: the QMLE approach for SDE’s with Lévy noise ([Shimizu and Yoshida 2006; Ogihara and Yoshida 2011](#)); the parametric estimation for the fractional Ornstein-Uhlenbeck model ([Brouste and Iacus 2013](#)); different simulation schemes as in [Iacus (2008)](#) for multidimensional diffusion processes; lead-lag estimation ([Hoffmann, Rosenbaum, and Yoshida 2013](#)); hypotheses testing via φ-divergence ([Gregorio and Iacus 2013](#)). For example, a nice (yet incomplete) utility is the method toLatex for objects of class ‘*yuima*’ and ‘*yuima.model*’. A simple writing like toLatex(my-yuima-obj) produces the related LaTeX code which can be copied and pasted in a mathematical paper. For example,

```r
R> a <- c("-3 * x1", ":-x1 - 2 * x2")
R> b <- matrix(c("1", "x1", "0", ":3", "x2", "0"), 2, 3)
R> modtex <- setModel(drift = a, diffusion = b, 
+    solve.variable = c("x1", ":x2"))
R> toLatex(modtex)
```

%%% Copy and paste the following output in your LaTeX file

$$
\begin{array}{c}
\begin{array}{c}
dx1
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
dx2
\end{array}
\end{array}
= \\
\begin{array}{l}
\begin{array}{l}
-3 \cdot x1 \\
-x1 - 2 \cdot x2
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
dW1
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
dW2
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
dW3
\end{array}
\end{array}
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
x1(0)=0 \\
x2(0)=0
\end{array}
\end{array}
$$
which can be typesetted with \LaTeX to produce
\[
\begin{pmatrix}
\frac{dx_1}{dt} \\
\frac{dx_2}{dt}
\end{pmatrix} = \begin{pmatrix}
-3 \cdot x_1 \\
-x_1 - 2 \cdot x_2
\end{pmatrix} dt + \begin{bmatrix}
1 & 0 & x_2 \\
x_1 & 3 & 0
\end{bmatrix} \begin{pmatrix}
W_1 \\
W_2 \\
W_3
\end{pmatrix}
\]
and might be used as a quick starting point for more complex editing of a mathematical paper. Another major plan is to open the contribution to the YUIMA project to external developers in the near future as well as to include other estimation procedures for low frequency and/or sparse data and parallelization of the general infrastructure.

Acknowledgments

This work was in part supported by Japan Society for the Promotion of Science Grants-in-Aid for Scientific Research No. 24340015 (Scientific Research), No. 24650148 (Challenging Exploratory Research); the Global COE Program “The Research and Training Center for New Development in Mathematics” of the Graduate School of Mathematical Sciences, University of Tokyo; by a Cooperative Research Program of the Institute of Statistical Mathematics and by the project PRIN 2009JW2STY, Ministero dell’Istruzione dell’Università e della Ricerca. Project YUIMA I was in part supported by JST Basic Research Programs PRESTO. Project YUIMA II is in part supported by NS Solutions Corporation. NS Solutions Corporation and The Graduate School of Mathematical Sciences, University of Tokyo are conducting a joint study (Azzurro project) on “The Application of Advanced Mathematical Statistics Theories in the Financial Industry”.

References


Croissant Y (2014). \textit{Ecdat: Data Sets for Econometrics}. R package version 0.2-4, URL \url{http://CRAN.R-project.org/package=Ecdat}.


**Affiliation:**

Stefano M. Iacus
Department of Economics, Management and Quantitative Methods
University of Milan
Via Conservatorio 7, 20122 Milan, Italy
E-mail: stefano.iacus@unimi.it
URL: [http://twitter.com/iacus](http://twitter.com/iacus)

Nakahiro Yoshida
Graduate School of Mathematical Science