Abstract

The Yuima Project is an open source and collaborative effort aimed at developing the R package named \texttt{yuima} for simulation and inference of stochastic differential equations.

In the \texttt{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 \texttt{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 \texttt{yuima} package and provides some examples of applications.

\textit{Keywords}: inference for stochastic processes, simulation, stochastic differential equations.

1. Introduction

The plan of the YUIMA Project is to define the bases for a complete environment for simulation and inference for stochastic processes via an R package called \texttt{yuima}. The package \texttt{yuima} provides an object-oriented programming environment for simulation and statistical inference for stochastic processes by R. The \texttt{yuima} package adopts the S4 system of classes
and methods (Chambers 1998).

Under this framework, the yuima package also supplies various functions to execute simulation and statistical analysis. Both categories of procedures may depend 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 the case of stochastic processes because most of 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 a stock price. 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 utilities. For example, the asymptotic expansion method for computing option prices (i.e., expectation of an irregular functional of a stochastic process) provides precise approximation values instantaneously, taking advantage of the analytic approach, but the formula has a long expression like more than 900 terms!

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 provides some examples of applications. The paper is organised as follows. Section 2 is an overview about the package. Section 3 describe the way models are specified in yuima. Section 4 explains asymptotic expansion methods. Section 5 is a review of basic inference procedures. Finally, Section 6 explains 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 be understood by anyone.

2. The yuima package

The package yuima depends on some other packages, like zoo, 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. The R-Forge page contains the latest development version, and stable version of the package will also be available through CRAN. 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. The package can be installed from R-Forge using 
\texttt{install.packages("yuima",repos="http://R-Forge.R-project.org").}

\section*{2.2. The main object and classes}

Before discussing the methods for simulation and inference for stochastic processes solutions to stochastic differential equations, here we discuss the main classes in the package. As mentioned there are different classes of objects defined in the \texttt{yuima} package and the main class is called the \texttt{yuima-class}. 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. For example, in case one wants to simulate a stochastic process, only the slots \texttt{model} and \texttt{sampling} should be present, while the slot \texttt{data} will be filled by the simulator. We now discuss in details the different object separately.

\section*{2.3. The \texttt{yuima.model} class}

At present, in \texttt{yuima} three main classes of stochastic differential equations can be easily specified. All multidimensional and eventually as parametric models.

- diffusions $dX_t = a(t, X_t)dt + b(t, X_t)dW_t$, where $W_t$ is a standard Brownian motion;
- fractional Gaussian noise, with $H$ the Hurst parameter
  \[ dX_t = a(t, X_t)dt + b(t, X_t)dW^H_t; \]
- diffusions with jumps and Lévy processes solution to
  \[ dX_t = a(X_t)dt + b(X_t)dW_t + \int_{|z|>1} c(X_{t-}, z)\mu(dt, dz) \]
  \[ + \int_{0<|z|\leq 1} c(X_{t-}, z)\{\mu(dt, dz) - \nu(dz)dt\}. \]

The \texttt{yuima.model} class contains information about the stochastic differential equation of interest. The constructor function \texttt{setModel} is used to give a mathematical description of the stochastic differential equation. 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.

\begin{itemize}
  \item \texttt{drift} is an \texttt{R} vector of expressions which contains the drift specification;
  \item \texttt{diffusion} is itself a list of 1 slot which describes the diffusion coefficient matrix relative to first noise;
  \item \texttt{hurst} is the Hurst index of the fractional Brownian motion, by default 0.5 meaning a standard Brownian motion;
\end{itemize}
The YUIMA Project

- **parameter** which is a short name for “parameters” which is a list with the following entries:
  - `all` contains the names of all the parameters found in the diffusion and drift coefficient;
  - `common` contains the names of the parameters in common between the drift and diffusion coefficients;
  - `diffusion` contains the parameters belonging to the diffusion coefficient;
  - `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 to the Lévy measure;

- **measure** specifies the measure of the Lévy noise;

- **measure.type** a switch to specify how the Lévy measure is described;

- **state.variable** and **time.variable**, by default, are assumed to be `x` and `t` but the user can freely choose them. The `yuima.model` class assumes that the user either use default names for `state.variable` and `time.variable` variables or specify his own names. All the rest of the symbols are considered parameters and distributed accordingly in the `parameter` slot.

- **jump.variable** the name of the variable used in the description of the Lévy component;

- **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.

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

- **xinit** initial value of the stochastic differential equation;

- **equation.number** represents the number of equations, i.e. the number of one dimensional stochastic differential equations.

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

- **J.flag** for internal use only.

As seen in the above, the parameter space is accurately described internally in a `yuima` object because in inference for stochastic differential equations, 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. sampling while estimators of parameters in the drift coefficient are slower or, in some cases, not even consistent. The `yuima` 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 is the approach in the \texttt{yuima} package we present some examples.

### 3.1. Diffusion processes

Assume that we want to describe the following stochastic differential equation

\[
\text{d}X_t = -3X_t \text{d}t + \frac{1}{1 + X_t^2} \text{d}W_t
\]

This is done in \texttt{yuima} specifying the drift and diffusion coefficients as plain mathematical expressions

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

At this point, the package fills the proper slots of the \texttt{yuima} object

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

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

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

For the above, it is possible to see that the jump coefficient is void and the Hurst parameter is set to 0.5, because this corresponds to the standard Brownian motion. Now, with \texttt{mod1} in hands, it is very easy to simulate a trajectory of the process as follows

\[
\texttt{R> set.seed(123)}
\]
\[
\texttt{R> X <- simulate(mod1)}
\]
\[
\texttt{R> plot(X)}
\]
The simulate function fills in addition the two slots data and sampling of the yuima object.

```r
R> str(X, vec.len = 2)
```

Formal class 'yuima' [package "yuima"] with 5 slots
  ..@ data :Formal class 'yuima.data' [package "yuima"] with 2 slots
    .. ..@ 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: A zoo regular 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"] 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"] with 6 slots
    .. .. ..@ 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 ...
    .. ..@ solve.variable : chr "x"
    .. ..@ xinit : num 0
    .. ..@ J.flag : logi FALSE
  ..@ sampling : Formal class 'yuima.sampling' [package "yuima"] with 11 slots
    .. ..@ Initial : num 0
    .. ..@ Terminal : num 1
    .. ..@ n : num 100
    .. ..@ delta : num 0.01
    .. ..@ grid : List of 1
    .. .. ..$: num [1:101] 0 0.01 0.02 0.03 0.04 ...
    .. ..@ random : logi FALSE
    .. ..@ regular : logi TRUE
3.2. Specification of parametric models

When a parametric model like

\[ dX_t = -\theta X_t \, dt + \frac{1}{1 + X_t} \, dW_t \]

is specified, \texttt{yuima} attempts to distinguish the parameters’ names from the ones of the state and time variables

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

R> str(mod2)

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' [package "yuima"] with 6 slots
.. .. @ 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)
.. @ time.variable : chr "t"
.. @ noise.number : num 1
.. @ equation.number: int 1
.. @ dimension : int [1:6] 2 0 1 1 0 0
.. @ solve.variable : chr "x"
.. @ xinit : num 0
.. @ J.flag : logi FALSE
\end{verbatim}

In order to simulate the parametric model it is necessary to specify the values of the parameters as the next code shows

\begin{verbatim}
R> set.seed(123)
R> X <- simulate(mod2, true.param = list(theta = 1, gamma = 3))
R> plot(X)
\end{verbatim}
3.3. Multidimensional processes

Next is an example with two stochastic differential equations driven by three independent Brownian motions

\[
\begin{align*}
\mathrm{d}X_t^1 &= -3X_t^1 \mathrm{d}t + X_t^2 \mathrm{d}W_t^3 \\
\mathrm{d}X_t^2 &= -(X_t^1 + 2X_t^2) \mathrm{d}t + X_t^1 \mathrm{d}W_t^1 + 3 \mathrm{d}W_t^2
\end{align*}
\]

but this has to be organized into matrix form

\[
\begin{pmatrix}
\mathrm{d}X_t^1 \\
\mathrm{d}X_t^2
\end{pmatrix} = 
\begin{pmatrix}
-3X_t^1 \\
-X_t^1 - 2X_t^2
\end{pmatrix} \mathrm{d}t + 
\begin{bmatrix}
1 & 0 & X_t^2 \\
X_t^1 & 3 & 0
\end{bmatrix}
\begin{pmatrix}
\mathrm{d}W_t^1 \\
\mathrm{d}W_t^2 \\
\mathrm{d}W_t^3
\end{pmatrix}
\]

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)

Again, this model can be easily simulated

R> set.seed(123)
R> X <- simulate(mod3)
R> plot(X, plot.type = "single", lty = 1:2)
But it is also possible to specify more complex models like the following

\[
\begin{align*}
\frac{dX^1_t}{dW^1_t} &= X^2_t |X^1_t|^{2/3} dW^1_t, \\
\frac{dX^2_t}{dW^2_t} &= g(t) dX^3_t, \\
\frac{dX^3_t}{dW^3_t} &= X^3_t (\mu d + \sigma (\rho dW^1_t + \sqrt{1-\rho^2} dW^2_t))
\end{align*}
\]

where \( g(t) = 0.4 + (0.1 + 0.2t) e^{-2t} \).

**Fractional Gaussian noise**

In order to specify a stochastic differential equation 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 model

\[dY_t = 3Y_t dt + dW^H_t\]

we proceed as follows

```R
> mod4 <- setModel(drift = "3*y", diffusion = 1, hurst = 0.3, + solve.var = "y")
> set.seed(123)
> X <- simulate(mod4, sampling = setSampling(n = 1000))
> plot(X)
```

In this case, the appropriate slot is now filled
The user can choose between the two simulation schemes, namely the Cholesky method and Wood and Chan (1994) method.

3.4. Lévy processes

Jump processes can be specified in different ways in mathematics and hence in yuima package. Let $Z_t$ be a Compound Poisson Process (i.e. jumps size follow some distribution, like the Gaussian law and jumps occur at Poisson times). Then it is possible to consider the following SDE which involves jumps

\[ dX_t = a(X_t)dt + b(X_t)dW_t + dZ_t \]

In the next example we consider a compound Poisson process with intensity $\lambda = 10$ with Gaussian jumps. This model can be specified in setModel using the argument measure.type="CP"

A simple Ornstein-Uhlenbeck process with Gaussian jumps

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

is specified as

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 = "y")
R> set.seed(123)
R> X <- simulate(mod5, true.p = list(theta = 1, sigma = 3),
+ sampling = setSampling(n = 1000))
R> plot(X)
Another possibility is to specify the jump component via the Lévy measure. Without going into too much details, here is an example of specification of a simple Ornstein-Uhlenbeck process with IG (Inverse Gaussian) Lévy measure

\[ dX_t = -x dt + dZ_t \]

```r
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)
```

3.5. Specification of generic models

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

\[ dX_t = a(X_t) dt + b(X_t) dW_t + c(X_t) dZ_t \]

using the following interface

```r
R> setModel(drift, diffusion, hurst = 0.5, jump.coeff, measure,
+ measure.type, state.variable = "x", jump.variable = "z",
+ time.variable = "t", solve.variable, xinit)
```
The `yuima` package implements many multivariate Random Numbers Generators (RNG) which are needed to simulate Lévy paths including `rIG` (Inverse Gaussian), `rNIG` (Normal Inverse Gaussian), `rbgamma` (Bilateral Gamma), `rngamma` (Gamma) and `rstable` (Stable Laws). Other user-defined RNG can be used freely.

### 3.6. Simulation, sampling and subsampling

The `simulate` function simulates `yuima` models according to Euler-Maruyama scheme in the presence of non-fractional diffusion noise and Lévy jumps and uses the Cholesky or the Wood and Chan methods for the fractional Gaussian noise. The `simulate` function accepts several arguments including the description of the sampling structure, which is an object of type `yuima.sampling`. The `setSampling` allow for the specification of different sampling parameters including random sampling. Further, the `subsampling` allow to subsample a trajectory of a simulated stochastic differential equation 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. We now go through few examples just to describe the use of standard arguments to these functions but the reader is invited to go thorough the man pages of the `yuima` packages for complete details.

Assume that we want to simulate this model

\[
\begin{align*}
\text{d}X^1_t &= -\theta X^1_t \text{d}t + dW^1_t + X^2_t \text{d}W^3_t \\
\text{d}X^2_t &= -(X^1_t + \gamma X^2_t) \text{d}t + X^1_t \text{d}W^1_t + \delta \text{d}W^2_t
\end{align*}
\]

Now we prepare the model using the `setModel` constructor function

```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 look inside it

```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] 0.001 0.002 0.003 0.004 0.005 0.006 0.007 0.008 0.009 ...
As seen from the output, the sampling structure is quite rich and we will show how to specify few of the slots later one. We now simulate this process specifying the `sampling` argument to `simulate`:

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

now 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. 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)
```

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 0.008 0.009 ... 
..@ random : logi FALSE
..@ regular : logi TRUE
..@ sdelta : num(0)
..@ sgrid : num(0)
..@ oindex : num(0)
..@ interpolation: chr "pt"
In the above we have specified two independent exponential distributions to represent Poissonian random times. Notice that the slot `regular` is now set to `FALSE`. Now we subsample the original trajectory of \( X_2 \) using the `subsampling` function

```r
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)
```

Or we can operate a deterministic sampling specifying two different regular frequencies

```r
R> newsamp <- setSampling(delta = c(0.1, 0.2))
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. 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 the next plot.

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

4. Asymptotic expansion

The `yuima` package can handle asymptotic expansion of functionals of $d$-dimensional diffusion process

$$dX_t^\varepsilon = a(X_t^\varepsilon, \varepsilon)dt + b(X_t^\varepsilon, \varepsilon)dW_t, \quad \varepsilon \in (0, 1]$$

with $W_t$ and $r$-dimensional Wiener process, i.e. $W_t = (W_1^t, \ldots, W_r^t)$. The functional is expressed in the following abstract form

$$F^\varepsilon(X_t^\varepsilon) = \sum_{\alpha=0}^{r} \int_0^T f_{\alpha}(X_t^\varepsilon) dW_t^\alpha + F(X_t^\varepsilon, \varepsilon), \quad W_0^t = t. \quad (1)$$

A typical example of application is the case of Asian option pricing. For example, in the Black & Scholes model

$$dX_t^\varepsilon = \mu X_t^\varepsilon dt + \varepsilon X_t^\varepsilon dW_t \quad (2)$$

the price of the option is of the form

$$\mathbb{E} \left\{ \max \left( \frac{1}{T} \int_0^T X_t^\varepsilon dt - K, 0 \right) \right\}.$$  

Thus the functional of interest is

$$F^\varepsilon(X_t^\varepsilon) = \frac{1}{T} \int_0^T X_t^\varepsilon dt, \quad r = 1$$

with

$$f_0(x, \varepsilon) = \frac{x}{T}, \quad f_1(x, \varepsilon) = 0, \quad F(x, \varepsilon) = 0$$
in (1). So, the call option price requires the composition of a smooth functional

\[ F^\varepsilon(X^\varepsilon_t) = \frac{1}{T} \int_0^T X^\varepsilon_t \, dt, \quad r = 1 \]

with the irregular function

\[ \max(x - K, 0) \]

Monte Carlo methods require a huge number of simulations to get the desired accuracy of the calculation of the price, while asymptotic expansion of \( F^\varepsilon \) provides very accurate approximations. The \texttt{yuima} package provides functions to construct the functional \( F^\varepsilon \), and automatic asymptotic expansion based on Malliavin calculus (Yoshida 1992a) starting from a \texttt{yuima} object. Here is an example. Consider a simple geometric Brownian motion of equation (2) with \( \mu = 1 \) and \( X_0 = 1 \). We define the model and the functional below:

\begin{verbatim}
R> model <- setModel(drift = "x", diffusion = matrix("x*e",
+ 1, 1))
R> T <- 1
R> xinit <- 1
R> K <- 1
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)
\end{verbatim}

this time the \texttt{setFunctional} command fills the appropriate slots inside the \texttt{yuima} object

\begin{verbatim}
R> str(yuima@functional)
Formal class 'yuima.functional' [package "yuima"] with 4 slots
..@ F : num 0
..@ f :List of 2
...$ : expression(x/T)
...$ : expression(0)
..@ xinit: num 1
..@ e : num 0.5
\end{verbatim}

Then, to obtain the first term in the asymptotic expansion (i.e. for \( \varepsilon = 0 \)), it is as easy as calling the function \texttt{F0} on the \texttt{yuima} object:

\begin{verbatim}
R> F0 <- F0(yuima)
R> F0
[1] 1.717423
\end{verbatim}

so the option price approximation is
We can go up to the first order approximation adding one term to the expansion

\[
\begin{align*}
R &> \max(F_0 - K, 0) \\
&> \text{rho} <- \text{expression}(0) \\
&> \text{epsilon} <- e \\
&> \text{g} <- \text{function}(x) \{ \\
&\quad \text{tmp} <- (F_0 - K) + (\text{epsilon} * x) \\
&\quad \text{tmp}[(\text{epsilon} * x) < (K - F_0)] <- 0 \\
&\quad \text{tmp} \\
&\} \\
R &> \text{asymp} <- \text{asymptotic\_term(yuima, block = 10, rho, g)} \\
\end{align*}
\]
and the final value is

\[
\begin{align*}
R &> \text{asymp}\$d0 + e * \text{asymp}\$d1 \\
&> 0.7203969
\end{align*}
\]

5. Inference for stochastic processes

The \texttt{yuima} implements several optimal techniques for parametric, semi- and non-parametric estimation of (multidimensional) stochastic differential equations. Although most of the examples in this section are given on simulated data, the main way to fill up the \texttt{data} slot of a \texttt{yuima} object is to use the function \texttt{setYuima}. The function \texttt{setYuima} sets various slots of the \texttt{yuima} object. In particular, to estimate a \texttt{yuima.model} called \texttt{mod} on the data \texttt{X} one can use a code like this \texttt{my.yuima <- setYuima(data=setData(X), model=mod)} and then pass \texttt{my.yuima} to the inference functions as described in what follows.

5.1. Quasi Maximum Likelihood estimation

Consider the multidimensional diffusion process

\[
dX_t = b(\theta_2, X_t)dt + \sigma(\theta_1, X_t)dW_t
\]
where \(W_t\) is an \(r\)-dimensional standard Wiener process independent of the initial value \(X_0 = x_0\). Quasi-MLE assumes the following approximation of the true log-likelihood for multidimensional diffusions

\[
\ell_n(X_n, \theta) = -\frac{1}{2} \sum_{i=1}^n \left\{ \log \det(\Sigma_{i-1}(\theta_i)) + \frac{1}{\Delta_n} \Sigma_{i-1}^{-1}(\theta_i)[\Delta X_i - \Delta_n b_{i-1}(\theta_2)] \otimes^2 \right\}
\]

where \(\theta = (\theta_1, \theta_2), \Delta X_i = X_{t_i} - X_{t_{i-1}}, \Sigma_i(\theta_1) = \Sigma(\theta_1, X_{t_i}), b_i(\theta_2) = b(\theta_2, X_{t_i}), \Sigma = \sigma \otimes^2, A \otimes^2 = A^T A\) and \(A^{-1}\) the inverse of \(A, A[ B ] \otimes^2 = B^T A B\). Then, (see e.g. Yoshida 1992b; Kessler 1997), the QML estimator of \(\theta\) is

\[
\hat{\theta}_n = \arg \min_\theta \ell_n(X_n, \theta)
\]
As an example, we consider the simple model

\[ \text{d}X_t = (2 - \theta_2 X_t)dt + (1 + X_t^2)^{\theta_1}dW_t \]  

(4)

with \( \theta_1 = 0.2 \) and \( \theta_2 = 0.3 \). We generate sampled data \( X_{t_i} \), with \( t_i = i \cdot n^{-\frac{2}{3}} \).

\begin{verbatim}
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))
\end{verbatim}

With the sampled data we can use the function \texttt{qmle} to estimate the parameters as follows

\begin{verbatim}
R> param.init <- list(theta2 = 0.5, theta1 = 0.5)
R> mle1 <- qmle(yuima, start = param.init, lower = list(theta1 = 0, + theta2 = 0), upper = list(theta1 = 1, theta2 = 1))
\end{verbatim}

and the estimated coefficients are extracted from the output object \texttt{mle1} as follows

\begin{verbatim}
R> summary(mle1)
\end{verbatim}

Maximum likelihood estimation

Call:
\texttt{qmle(yuima = yuima, start = param.init, lower = list(theta1 = 0, + theta2 = 0), upper = list(theta1 = 1, theta2 = 1))}

Coefficients:
\begin{verbatim}
         Estimate Std. Error
theta1 0.1969182  0.008095453
theta2 0.2998350  0.126410524
\end{verbatim}

\(-2\) \texttt{log L}: -282.8676

Notice the interface and the output of the \texttt{qmle} is quite similar to the ones of the standard \texttt{mle} function of the \texttt{stats4} package of the base \texttt{R} system.

5.2. Adaptive Bayes estimation

Consider again the diffusion process solution to

\[ \text{d}X_t = b(X_t, \theta_2)dt + \sigma(X_t, \theta_1)dW_t, \]  

(5)

and the quasi likelihood defined in (3).
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 to make the Bayesian type estimator \( \tilde{\theta}_1 \) as

\[
\tilde{\theta}_1 = \left[ \int_{\Theta_1} \ell_n(x_n, (\theta_1, \theta_2^*)) \pi_1(\theta_1) d\theta_1 \right]^{-1} \int_{\Theta_1} \theta_1 \ell_n(x_n, (\theta_1, \theta_2^*)) \pi_1(\theta_1) d\theta_1 \tag{6}
\]

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

\[
\tilde{\theta}_2 = \left[ \int_{\Theta_2} \ell_n(x_n, (\tilde{\theta}_1, \theta_2)) \pi_2(\theta_2) d\theta_2 \right]^{-1} \int_{\Theta_2} \theta_2 \ell_n(x_n, (\tilde{\theta}_1, \theta_2)) \pi_2(\theta_2) d\theta_2 \tag{7}
\]

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

Adaptive Bayes estimation is developed in \texttt{yuima} via the method \texttt{adaBayes}. Consider again the model (4) with the same values for the parameters. In order to perform Bayesian estimation, we need to prepare the prior densities for the parameters. For simplicity we use uniform distributions in \([0, 1] \)

```R
R> prior <- list(theta2 = list(measure.type = "code", df = "dunif(theta2,0,1)")", +                   theta1 = list(measure.type = "code", df = "dunif(theta1,0,1)"))
```

Then we call \texttt{adaBayes}, on the same sample data we used for the \texttt{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(bayes1)
theta1 theta2
0.1971567 0.3071515
```

```R
R> coef(mle1)
theta1 theta2
0.1969182 0.2998350
```

**Small sample size**

It is known from the theory that the estimation of the drift in a diffusion process strongly depend on the length of the observation interval \([0, T]\). In our example above, we took \( T = n(1/3) \), with \( n = 750 \), which is approximatively 9.09. Now we reduce the sample size to \( n = 500 \) and the value of \( T \) is then \( T = 7.94 \). We then apply both quasi-maximum likelihood and adaptive Bayes type estimators to these data
R> n <- 500
dataFrame <- 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> coef(bayes2)

    theta1    theta2
0.1950772 0.2467359

R> coef(mle2)

    theta1    theta2
0.1947225 0.2193002

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.

5.3. 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 stochastic differential equations

$$dX_t^l = \mu_t^l dt + \sigma_t^l dW_t^l, \quad t \in [0, T]$$

$$X_0^l = x_0^l$$

for $l = 1, 2$. Here $W^l$ denote standard Wiener processes with a progressively measurable correlation process $d\langle W_1, W_2 \rangle_t = \rho_t dt$, $\mu_t^l$ and $\sigma_t^l$ are progressively measurable processes, and $x_0^l$ 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 $X_t^l$ is supposed to be observed at over the increasing sequence of times $T_t^{i,i}$ ($i \in \mathbb{Z}_{\geq 0}$) starting at 0, up to time T. Thus, the observables are $(T_t^{i,i}, X_t^{i,i})$ with $T_t^{i,i} \leq T$. Each $T_t^{i,i}$ may be a stopping time, so possibly depends on the history of $(X^1, X^2)$ as well as the precedent stopping times. Two sequences of stopping times $T_t^{1,i}$ and $T_t^{2,j}$ are nonsynchronous,
and irregularly spaced, in general. In particular, cce can apply to estimation of the quadratic variation of a single stochastic process sampled regularly/irregularly.

The parameter of interest is the quadratic covariation between $X_1$ and $X_2$:

$$\theta = \langle X_1, X_2 \rangle_T = \int_0^T \sigma_1^1 \sigma_1^2 \rho_t \, dt.$$  

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

$$U_n = \sum_{i,j:T^{1,i} \leq T, T^{2,j} \leq T} (X_{T^{1,i}}^1 - X_{T^{1,i}-1}^1)(X_{T^{2,j}}^2 - X_{T^{2,j}-1}^2)1\{((T^{1,i}, T^{1,i}) \cap (T^{2,j}-1, T^{2,j}) \neq \emptyset)\}. \quad (9)$$

That is, the product of any pair of increments $(X_{T^{1,i}}^1 - X_{T^{1,i}-1}^1)$ and $(X_{T^{2,j}}^2 - X_{T^{2,j}-1}^2)$ will make a contribution to the sum only when the respective observation intervals $(T^{1,i}, T^{1,i})$ and $(T^{2,j}-1, T^{2,j})$ are overlapping with each other. It is known that $U_n$ is consist 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, 2008a, 2006, 2008b) for details.

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

We will demonstrate how to apply cce function to nonsynchronous high-frequency data by simulation. As an example, consider a two dimensional stochastic process $(X_1^t, X_2^t)$ satisfying the stochastic differential equation

$$\begin{align*}
dX_1^t &= \sigma_1^1 dB_1^t, \\
dX_2^t &= \sigma_2^2 dB_2^t.
\end{align*} \quad (10)$$

Here $B_1^t$ and $B_2^t$ denote two standard Wiener processes, however they are correlated as

$$\begin{align*}
B_1^t &= W_1^t, \\
B_2^t &= \int_0^t \rho_s \, dW_1^s + \int_0^t \sqrt{1 - \rho^2_s} \, dW_2^s, 
\end{align*} \quad (12)$$

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:

$$\begin{align*}
\sigma_{1,t} &= \sqrt{1 + t}, \\
\sigma_{2,t} &= \sqrt{1 + t^2}, \\
\rho_t &= \frac{1}{\sqrt{2}}.
\end{align*}$$

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 the **yuima** comprehensive package for simulation and estimation of stochastic processes.
The parameter we want to estimate is the quadratic covariation between $X_1$ and $X_2$:

$$\theta = \langle X_1, X_2 \rangle_T = \int_0^T \sigma_{1,t}\sigma_{2,t}\rho_t \, dt.$$  \hfill (13)

Later, we will compare estimated values with the true value of $\theta$ given by

$$n^{1/2}(U_n - \theta) \to N(0, c),$$ \hfill (14)

as $n \to \infty$, where

$$c = \left( \frac{2}{p_1} + \frac{2}{p_2} \right) \int_0^T \left( \sigma_{1,t}\sigma_{2,t} \right)^2 \, dt + \left( \frac{2}{p_1} + \frac{2}{p_2} - \frac{2}{p_1 + p_2} \right) \int_0^T \left( \sigma_{1,t}\sigma_{2,t}\rho_t \right)^2 \, dt.$$ \hfill (15)

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 $\theta = 1.$
**The YUIMA Project**

`cce` takes the sample and returns an estimate of the quadratic covariation. For example, for the complete data

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

\[
\text{\$covmat} \\
\text{[,1]} & \text{[,2]} \\
1,1 & 1.491938 & 1.086078 \\
1,2 & 1.086078 & 1.474730 \\
\text{\$cormat} \\
\text{[,1]} & \text{[,2]} \\
1,1 & 1.0000000 & 0.7321992 \\
1,2 & 0.7321992 & 1.0000000 \\
\]

\[ R \text{> plot(X, main = "complete data")} \]

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 specificy an exponential distribution with rate \( n \cdot p_i/T \) for the random times. This will generate Poisson random times with the corresponding rate.

\[ R \text{> p1 <- 0.2} \]
\[ R \text{> p2 <- 0.3} \]
\[ R \text{> newsamp <- setSampling(random = list(rdist = c(function(x) rexp(x, + rate = p1 * n/Terminal), function(x) rexp(x, rate = p1 * + n/Terminal))))} \]

Now we use the `subsampling` function to subsample the original data \( X \) into new asynchronous data \( Y \).
Asynchronous estimation for nonlinear systems

Consider now the two-dimensional system with nonlinear feedback

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

with \(\sigma_1(t, X_t, Y_t) = \sqrt{|X_t| (1 + t)}\), \(\sigma_2(t, X_t, Y_t) = \sqrt{|Y_t|}\), \(\rho(t, X_t, Y_t) = \frac{1}{1 + X_t^2}\) and \(W_t, B_t\), two independent Brownian motions. We construct the model and we generate data from it.
The YUIMA Project

+ "", "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)
R> yuima <- setYuima(model = cor.mod, sampling = yuima.samp)
R> yuima <- simulate(yuima, xinit = c(2, 3))

We apply the same Poisson random sampling so that the object \( Y \) will contain asynchronous data

R> p1 <- 0.2
R> p2 <- 0.3
R> newsamp <- setSampling(random = list(rdist = c(function(x) rexp(x,
+ rate = p1 * n/Terminal), function(x) rexp(x, rate = p1 *
+ n/Terminal))))
R> Y <- subsampling(yuima, sampling = newsamp)
R> plot(Y, main = "asynchronous data (non linear case)")

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

R> cce(yuima)

$covmat
[,1]      [,2]
[1,] 2.7092720 0.7803843
[2,] 0.7803843 3.4705059
5.4. Change point analysis

Consider a multidimensional stochastic differential equation of the form
\[ dY_t = b_t dt + \sigma(X_t, \theta) dW_t, \quad t \in [0, T], \]
where \( W_t \) a \( r \)-dimensional Wiener process and \( b_t \) and \( X_t \) are multidimensional processes and \( \sigma \) is the diffusion coefficient (volatility) matrix. When \( Y = X \) the problem is a diffusion model. The process \( b_t \) 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

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

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} implements the quasi-maximum likelihood approach as in \cite{iacus2009} described in the following. Let \( \Delta_i Y = Y_{t_i} - Y_{t_{i-1}} \) and define

\[ \Phi_n(t; \theta_0, \theta_1) = \sum_{i=1}^{[nt/T]} G_i(\theta_0) + \sum_{i=[nt/T]+1}^{n} G_i(\theta_1), \quad (16) \]

with

\[ G_i(\theta) = \log \det S(X_{t_{i-1}}, \theta) + \Delta_i^{-1} S(X_{t_{i-1}}, \theta) \quad (17) \]

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

\[ \hat{\tau}_n = \arg\min_{t \in [0, T]} \Phi_n(t; \hat{\theta}_0, \hat{\theta}_1). \]
5.5. Example of Volatility Change-Point Estimation

One example of model that can be analyzed by this technique is, for example, the 2-dimensional stochastic differential equation

\[
\begin{pmatrix}
\frac{dX_1}{dt} \\
\frac{dX_2}{dt}
\end{pmatrix} =
\begin{pmatrix}
b_1(X_1) & b_2(X_2) \\
b_1(X_1) & b_2(X_2)
\end{pmatrix}
\begin{pmatrix}
\theta_{1,k} \cdot X_1^t & 0 \cdot X_1^t \\
0 \cdot X_2^t & \theta_{2,k} \cdot X_2^t
\end{pmatrix}
\begin{pmatrix}
\frac{dW_1}{dt} \\
\frac{dW_2}{dt}
\end{pmatrix}
\]

where \(b_1(\cdot)\) and \(b_2(\cdot)\) 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 \(b_1(\cdot)\) and \(b_2(\cdot)\) but this information will not be used in the change point analysis. For example, we will simulate the following 2-dimensional stochastic differential equation

\[
\begin{pmatrix}
\frac{dX_1}{dt} \\
\frac{dX_2}{dt}
\end{pmatrix} =
\begin{pmatrix}
\sin(X_1^t) & 3 - X_2^t \\
3 - X_1^t & 0 \cdot X_2^t
\end{pmatrix}
\begin{pmatrix}
\theta_{1,k} \cdot X_1^t & 0 \cdot X_1^t \\
0 \cdot X_2^t & \theta_{2,k} \cdot X_2^t
\end{pmatrix}
\begin{pmatrix}
\frac{dW_1}{dt} \\
\frac{dW_2}{dt}
\end{pmatrix}
\]

with change point instant at time \(\tau = 0.4\). First, we describe the model to be simulated

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

and then simulate two trajectories. One up to the change point \(\tau = 4\) with parameters \(\theta_{1,0} = 0.1\) and \(\theta_{2,0} = 0.2\)

\[
R> n <- 1000
R> \text{set.seed(123)}
R> \text{t0 <- list(theta1.k = 0.5, theta2.k = 0.3)}
R> \text{tau <- 0.4}
R> \text{ysamp1} <- \text{setSampling(n = tau * n, Initial = 0, delta = 0.01)}
R> \text{yuima1} <- \text{setYuima(model = ymodel, sampling = ysamp1)}
R> \text{yuima1} <- \text{simulate(yuima1, xinit = c(3, 3), true.parameter = t0)}
R> \text{x1} <- \text{yuima1@data@zoo.data[[1]]}
R> \text{x1} <- \text{as.numeric(x1[length(x1)])}
R> \text{x2} <- \text{yuima1@data@zoo.data[[2]]}
R> \text{x2} <- \text{as.numeric(x2[length(x2)])}
\]

now we generate the second trajectory with parameters \(\theta_{1,1} = 0.6\) and \(\theta_{2,1} = 0.6\). For this trajectory, the initial value is set to the last value of the first trajectory stored in \(x1\) and \(x2\) for the two component of the process.

\[
R> \text{t1} <- \text{list(theta1.k = 0.2, theta2.k = 0.4)}
R> \text{ysamp2} <- \text{setSampling(Initial = n * tau * 0.01, n = n *}
+ \text{ysamp2)}
\]
\[ (1 - \tau), \delta = 0.01 \]

```r
R> yuima2 <- setYuima(model = ymodel, sampling = ysamp2)
R> yuima2 <- simulate(yuima2, xinit = c(x1, x2), true.parameter = t1)
```

finally we collate the two trajectories

```r
R> yuima <- yuima1
R> yuima@data@zoo.data[[1]] <- c(yuima1@data@zoo.data[[1]],
+ yuima2@data@zoo.data[[1]][-1])
R> yuima@data@zoo.data[[2]] <- c(yuima1@data@zoo.data[[2]],
+ yuima2@data@zoo.data[[2]][-1])
```

The composed trajectory appears as follows

```r
R> plot(yuima)
```

As said, the change point analysis do not consider the information coming from the drift part of the model and `yuima` ignores this internally. Just to make clear that the information on the drift term is not considered by the function `CPoint`, we redefine the `yuima` model removing the information coming from the drift and then adding back the data.

```r
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))
```

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

```r
R> t.est <- CPoint(yuima, param1 = t0, param2 = t1)
R> t.est$tau
```
The YUIMA Project

[1] 3.98

R> t.est2 <- CPoint(noDriftModel, param1 = t0, param2 = t1)
R> t.est2$tau

[1] 3.98

Now we proceed first with the estimation of the parameters before and after the change point. 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 \(t=0.2\). Similarly for qmleR, which uses only observations in the time interval \([t, T]\). In our example, we take \(t=0.8\).

R> estL <- 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")
R> estR <- 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")
R> t0.est <- coef(estL)
R> t1.est <- coef(estR)

and now we proceed with change point estimation.

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 a small subsets of observations, the change point estimate remains good. A two stage change point estimation approach is also possible as explained in Iacus and Yoshida (2009).

5.6. LASSO model selection

Let \(X_t\) be a diffusion process solution to

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

with \( b : \Theta_p \times \mathbb{R}^d \to \mathbb{R}^d \), \( \sigma : \Theta_q \times \mathbb{R}^d \to \mathbb{R}^d \times \mathbb{R}^m \) and \( W_t, t \in [0, T] \), is a standard Brownian motion in \( \mathbb{R}^m \). We assume that the functions \( b \) and \( \sigma \) are known up to \( \alpha \) and \( \beta \). We denote by \( \theta = (\alpha, \beta) \in \Theta \) the parametric vector and with \( \theta_0 = (\alpha_0, \beta_0) \) its unknown true value. Let \( H_n(X_n, \theta) = \ell_n(X_n, \theta) \) from equation (3). The quasi-MLE \( \hat{\theta}_n \) for this model is the solution of the following problem
\[ \hat{\theta}_n = (\hat{\alpha}_n, \hat{\beta}_n)' = \arg \min_{\theta} H_n(X_n, \theta) \]

The adaptive LASSO estimator is defined as the solution to the quadratic problem under \( L_1 \) constraints
\[ \hat{\theta}_n = (\hat{\alpha}_n, \hat{\beta}_n) = \arg \min_{\theta} F(\theta). \]

with
\[ F(\theta) = (\theta - \hat{\theta}_n)' \frac{1}{2} H_n(X_n, \hat{\theta}_n)(\theta - \hat{\theta}_n)' + \sum_{j=1}^{p} \lambda_n,j |\alpha_j| + \sum_{k=1}^{q} \gamma_n,k |\beta_k| \]

For more details see De Gregorio and Iacus (forthcoming). The tuning parameters should be chosen as in Zou (2006) in the following way
\[ \lambda_n,j = \lambda_0 |\hat{\alpha}_n,j|^{-\delta_1}, \quad \gamma_n,k = \gamma_0 |\hat{\beta}_n,k|^{-\delta_2} \]

where \( \hat{\alpha}_{n,j} \) and \( \hat{\beta}_{n,k} \) are the unpenalized QML estimator of \( \alpha_j \) and \( \beta_k \) respectively, \( \delta_1, \delta_2 > 0 \) and usually taken unitary.

5.7. An example of use

The lasso method is implemented in the yuima package. Let us consider the full CKLS model
\[ dX_t = (\alpha + \beta X_t)dt + \sigma X_t^{\gamma}dW_t \]
and let us try to estimate the parameter on the U.S. Interest Rates monthly data from 06/1964 to 12/1989. We prepare the data, the model and the constraints for optimization

```r
R> library(Ecdat)
R> data(Irates)
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 = -0.1, sigma = 0.1, gamma = 1)
R> low <- list(alpha = -5, beta = -5, sigma = -5, gamma = -5)
R> upp <- list(alpha = 8, beta = 8, sigma = 8, gamma = 8)
```

and now we apply the lasso function
The YUIMA Project

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

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

R> round(lasso10$mle, 2)

sigma gamma alpha beta
0.13 1.44 2.08 -0.26

R> round(lasso10$lasso, 2)

sigma gamma alpha beta
0.12 1.50 0.59 0.00

the LASSO method selects the reduced model

\[ dX_t = 0.6dt + 0.12X_t^3 dW_t. \]

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.

6. Miscellanea and Roadmap of YUIMA Project

Other statistical techniques are already implemented or will be shortly released in the yuima, For example, a nice utility is the function 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 copy and pasted in a mathematical paper.

References


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://www.economia.unimi.it/~iacus/

Nakahiro Yoshida
Graduate School of Mathematical Science
University of Tokyo
3-8-1 Komaba, Meguro-ku, Tokyo 153-8914, Japan
E-mail: nakahiro@ms.u-tokyo.ac.jp
URL: http://www.ms.u-tokyo.ac.jp/~nakahiro/hp-naka-e

Journal of Statistical Software
published by the American Statistical Association
http://www.jstatsoft.org/
http://www.amstat.org/
Volume VV, Issue II
Submitted: yyyy-mm-dd
MMMMMM YYYY
Accepted: yyyy-mm-dd
Figure 1: The main classes in the *yuima* package.