Use R!

Stefano M. Iacus
Nakahiro Yoshida

Simulation and Inference for Stochastic Processes with YUIMA

A Comprehensive R Framework for SDEs and Other Stochastic Processes

Springer

Use R!

Stefano M. Iacus · Nakahiro Yoshida

Simulation and Inference for Stochastic Processes with YUIMA

A Comprehensive R Framework for SDEs and Other Stochastic Processes

Springer

Stefano M. Iacus
Department of Economics, Management and Quantitative Methods
University of Milan
Milan
Italy

Nakahiro Yoshida
Graduate School of Mathematical Sciences
University of Tokyo
Tokyo
Japan

ISSN 2197-5736 ISSN 2197-5744 (electronic)
Use R!
ISBN 978-3-319-55567-6 ISBN 978-3-319-55569-0 (eBook)
https://doi.org/10.1007/978-3-319-55569-0

Library of Congress Control Number: 2017934627

Printed on acid-free paper

This Springer imprint is published by the registered company Springer International Publishing AG part of Springer Nature
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

To Maite, Lucy and Ludo,
to whom I wish to find what gives
their life true meaning and purpose

and to Tizy,
whose heart I met late in my life,
but, thanks God, not too late!

Stefano M. Iacus

Preface

Statistics for stochastic processes is rapidly developing. It forms a branch of mathematical sciences, spreading over theoretical statistics, probability theory, software development and real data analysis. Since a general theoretical framework of statistical inference for stochastic processes was recently established, statistical inference has been applicable to various stochastic systems and its scope is expanding more and more from ergodic to nonergodic processes, from low-frequency regular to high-frequency irregular sampling schemes, from linear to nonlinear models, and so on.

The formulas provided by the theory are often fairly complicated, and it makes it difficult for nonexperts to use them in their own fields. For example, an asymptotic expansion formula derived by the Malliavin calculus involves hundreds of terms, the Bayesian estimator theoretically validated recently needs modern MCMC methods for computation in practice, and some random number generators for simulation of Lévy-driven stochastic differential equations use quite sophisticated algorithms. Software implementation is an issue in such circumstances.

YUIMA is a computational framework for statistical analysis and simulation for stochastic processes, especially objects described in terms of the stochastic analysis. YUIMA is designed to realize a circle of data analysis, modelling, fitting, simulation, and prediction. Through YUIMA, the user enjoys easily, without depending on his/her expertise, the latest developments in theoretical statistics for stochastic processes.

The YUIMA Project was launched by Stefano Maria Iacus and Nakahiro Yoshida, respectively, an R guru and a dreamer, after a three-person discussion around 2005 set by Masayuki Uchida. Supported by Japan Science and Technology Agency PRESTO (2007–2011), the project implemented a basic structure on R and extended by inviting Hideitsu Hino, Hiroki Masuda, Yasutaka Shimuzu, Kengo Kamatani, Alexandre Brouste, Masaaki Fukasawa and Teppei Ogihara. Hino with the Waseda University team was quite active in programming many YUIMA functions. Collaboration with Kenji Kashiwakura and Kentaro Hoshi with their NS Solutions team in 2012–2016 is acknowledged. The YUIMA Project got new members Yuta Koike, Ryosuke Nomura, Lorenzo Mercuri, Yusuke Shimizu, Shoichi Eguchi, Yuma Uehara, Yuto Yoshida, Emanuele Guidotti and many other young people. Most of them are mathematical statisticians, and this is the reason why the functions of YUIMA are structurally designed with rigorous mathematical backing. Presently, the YUIMA Project (YUIMA III) is supported by Japan Science and Technology Agency CREST led by Prof. Takashi Tsuboi. Special thanks go to Prof. Shigeo Kusuoka for his great support for statistics in mathematics. The authors also thank Mrs. Sayako Takehara and Miss Homare Yoshihira for their help as the secretaries of the laboratory.

We also need to thank MIUR—Ministero dell'Istruzione, dell'Università e della Ricerca, Grant: PRIN 2009JW2STY, for supporting the early work of Lorenzo Mercuri, Emanuele Guidotti and the first author on this project.

Milan, Italy Stefano M. Iacus
Tokyo, Japan Nakahiro Yoshida
February 2018

Contents

Part I
The YUIMA Framework

Chapter 1
The YUIMA Package

1.1 Overview of the Project

The YUIMA[1] Project is collaborative effort of several people aimed at providing a comprehensive environment for the simulation and inference for stochastic processes based on the R (R Core Team 2017) language. The main infrastructure is implemented in an R package called **yuima** (Brouste et al. 2014).

Stochastic differential equations are commonly used to model random evolutions along continuous or practically continuous time, such as the random movements of stock prices and the population dynamics. 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 and rarely made available through user-friendly software. This fact makes quite difficult for the casual practitioner, or even researchers in fields other than stochastic analysis, to take full advantage of them.

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 compared to Monte Carlo methods. Unfortunately, the analytic version of the expansion formula involves more than 900 terms which are multiple integrals. In this situation, the hand coding of these formulas is quite challenging but the **yuima** package automatically implements them for the user. These and many other up-to-date methods are ready to be used through the **yuima** package.

[1]YUIMA is both the acronym for *Yoshida-Uchida-Iacus-Masuda-Andothers* and the name of an important character in Buddhism religion (http://www.kyohaku.go.jp/eng/syuzou/meihin/kaiga/chuugoku/item01.html) whose approach to problems fits well this project.

The **yuima** package is intended to offer the basic infrastructure on which complex models and inference procedures can be built on. The present version of the **yuima** package allows to specify stochastic differential equations of very abstract type, including one- or multidimensional diffusion processes driven by Wiener process or fractional Brownian motion with general Hurst parameter, with or without jumps (i.e. driven by Lévy processes). Further, the **yuima** package allows for the specification of other classes of models like the continuous autoregressive moving average models (CARMA) Doob (1944), Brockwell (2001) and the continuous generalized heteroskedastic model (COGARCH) Klüppelberg et al. (2004), Brockwell et al. (2006), Maller et al. (2008).

1.2 Who Should Read This Book?

Although we assume that the reader of this book 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. This book is intended to be a step-by-step introduction to simulation and inference for stochastic processes using the **yuima** package. The content of this book will be useful to practitioners who want to implement in their field of research, abstract models appearing in the specialized literature of stochastic processes. The **yuima** package can also be very useful to scholars in the field of theoretical statistics and stochastic processes, who want to quickly implement their models and test their performance through simulation or empirical analysis. This book contains examples of real data analysis coming from different fields.

1.3 Structure of the Book

This book consists of two parts. The first part gives a brief introduction to the basic infrastructure of the **yuima** package and its building blocks: model specification, simulation, sampling, data input and the basic functions and methods. The second part of the book is devoted to give a detailed description on how to implement, simulate and estimate several classes of models. Namely, Chap. 2 is focused on diffusion processes and includes some advanced topics like asymptotic expansion methods via Malliavin calculus. Chapter 3 considers compound Poisson processes, Chap. 4 discusses Lévy processes while Chap. 5 treats stochastic differential equations driven by fractional Brownian motion. Finally, Chaps. 6 and 7 introduce CARMA and COGARCH models, respectively. Throughout this book, we assume that all regularity conditions for the existence of the stochastic processes are met; although in special cases we remind explicitly which conditions are required. This is due to the fact that the package **yuima** is not able to verify the correctness of the results when the assumptions are not met. In some cases we will also put in evidence what are the implications when these assumptions are not fully satisfied.

1.4 How to Get the R Code for This Book

The complete R code used in the book has been included in the **yuima** package. R code have been collected by chapters. The **yuima** function to access the code is called `ybook` and accepts a single argument `chapter`. So, for example, to access the code of this chapter the reader should type the following command in the R Console

```
ybook(1)
```

For Chap. 3 the command will be

```
ybook(3)
```

The R code in **yuima** package will be updated to keep up with future releases of **yuima** or R. The examples of this book have been run with R version 3.4.1 and **yuima** package version 1.7.4.

1.5 Main Contribution to the Yuima Package

This book about the YUIMA Project would have not been possible without the **yuima** package itself. The Yuima package has several present and past contributors who develop specialized parts of the software. The YUIMA Project team members are given in alphabetical order: Alexandre **Brouste**, Stefano M. **Iacus**, Kengo **Kamatani**, Yuta **Koike**, Hiroki **Masuda**, Lorenzo **Mercuri**, Ryosuke **Nomura**, Masayuki **Uchida**, Yuma **Uehara** and Nakahiro **Yoshida**. Former members include Masaaki Fukasawa and Yasutaka Shimizu, and a special mention goes to Hideitsu **Hino** for the hard work in the early years of this project. We also acknowledge the efforts of Emanuele Guidotti for providing the graphical user interface through the **yuimaGUI** package (see Sect. 1.15.2).

Table 1.1 summarizes very roughly the contributors for each part and/or function of **yuima** package. Most of the times these sets overlap with the theory developed for such progress in **yuima**, but the theoretical papers are mentioned in each section of this book; here, we only mention the coding efforts and the definition of classes and methods. The whole Yuima Core Team took part in the design of the different pieces of software.

Table 1.1 Very rough contribution to the **yuima** package development

Topic	In book	Main authors
`qmle` coding	Sect. 2.4	Hino, Iacus
`qmle` with `ccp`	Sect. 2.4.2	Kamatani
`sampling` and `subsampling`	Sect. 1.11.1	Fukasawa, Hino, Iacus
`setModel`	Sect. 1.9.1	Hino, Iacus, Mercuri, Masuda, Shimizu
`simulate`	Sect. 1.10	Hino, Iacus, Mercuri, Masuda, Shimizu
`simulate` with `space.discretized`	Sect. 1.10	Fukasawa, Hino
`setPoisson` with `qmle`	Chap. 3	Iacus
CARMA(p,q), `setCarma` with `qmle`	Chap. 6	Iacus, Mercuri
COGARCH(p,q), `setCarma` with `qmle`	Chap. 7	Iacus, Mercuri
Random number generators for Lévy	Chap. 4	Masuda, Uehara
Fractional Brownian motion simulation	Sect. 5.2	Brouste
Fractional O-U estimation and `mmfrac`	Sect. 5.4	Brouste, Iacus
Asymptotic expansion	Sect. 2.13	Hino, Nomura, Yoshida
Hypotheses testing	Sect. 2.7	Iacus
Lead–lag estimation	Sect. 2.12	Koike
Asynchronous covariance estimation	Sect. 2.11	Koike, Hino
LASSO estimation	Sect. 2.9	Iacus
`toLatex`	Sect. 1.15.1	Iacus, Mercuri
`yuimaGUI`	Sect. 1.15.2	Guidotti, Iacus, Mercuri

1.6 Further Developments of Yuima Package

The YUIMA Project is an ongoing project. Not all functionalities are described in this book because at the time of this writing new modules are being added. Among these, there is the class of point process regression models, i.e. where the intensity function of the point process depends on time but also on the process itself and several other covariate processes. This class includes Hawkes processes (Hawkes 1971) as a special case.

The concepts of *maps*, *transform* and *integration* of wide classes of stochastic processes are also under development along with a flexible structure to describe probability laws and related quantities.

1.7 Things to Know About R

For the benefit of the reader who approaches R for the first time thanks to this book, we briefly mention how to get his own copy of the software but we also give some information on the concept of ‘classes’ and ‘methods’ as we will use quite frequently these terms in the text. We redirect the user to Dalgaard (2008) for a gentle introduction to R.

1.7.1 How to Get R

R exists for all major platforms (Mac OS X, MS Windows, Linux and the alike) and can be freely downloaded from the main CRAN repository at the URL http://CRAN.R-Project.org or one of its mirrors. MS Windows users can point their browser directly to http://cran.r-project.org/bin/windows/base, Macintosh users to http://cran.r-project.org/bin/macosx/, and Linux users can choose the version for their system at http://cran.r-project.org/bin/linux/ or use commands like `yum`, `apt-get` or similar, depending on the incarnation of Linux installed in their machines. On Mac OS X and MS Windows, the user needs to run the installer which automatically configures R for their machine. Once installed, Linux users can just run R from the terminal window typing `R`; MS Windows users will find the executable named, for example, `R-3.4.1-win32.exe` if the system is 32-bit and the release of R is 3.4.1; Mac OS X user will find the application `R.app` in their *Applications* folder. Different replacement solutions for the default (or non existent) R GUI exist. We mention one of the most popular named RStudio. The reader of this book can refer to the corresponding website for full details: http://www.rstudio.com.

1.7.2 R and S4 Objects

Although the reader is assumed to have a basic knowledge of R, he is not necessarily aware of the object-oriented nature of the R language. In fact, each object in R belongs to some *class*, and for each class, there exist generic functions called *methods* which perform some task on that object. For example, the function `summary` provides summary statistics which are appropriate for each particular object

```r
data(cars)
class(cars)

## [1] "data.frame"
```

The command `class` shows the class of the object `cars` which is a `data.frame`.

```
summary(cars)

##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00

mod <- lm(dist~speed, data=cars)
summary(mod)

## 
## Call:
## lm(formula = dist ~ speed, data = cars)
## 
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -29.069  -9.525  -2.272   9.215  43.201 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -17.5791     6.7584  -2.601   0.0123 *  
## speed         3.9324     0.4155   9.464 1.49e-12 ***
## ---
## Signif. codes:  
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 15.38 on 48 degrees of freedom
## Multiple R-squared:  0.6511, Adjusted R-squared:  0.6438 
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

We can look at the class of the object `mod` resulting from the application of a linear model (`lm`)

```
class(mod)

## [1] "lm"
```

The standard set of classes and methods in R is called `S3`. In this framework, a method for an object of some class is simply an R function named `method.class`; e.g. `summary.lm` is the function which is called by R when the function `summary` is called with an argument which is an object of class `lm`. R methods like `summary` are very generic, and the function `methods` provides a list of specific methods (which apply to specific types of objects) for some particular method. For example,

```
methods(summary)

##   [1] summary,ANY-method
##   [2] summary,cogarch.est-method
##   [3] summary,cogarch.est.incr-method
##   [4] summary,diagonalMatrix-method
##  [5] summary,mle-method
##  [6] summary,sparseMatrix-method
##  [7] summary,yuima.CP.qmle-method
##  [8] summary,yuima.carma.qmle-method
##  [9] summary,yuima.qmle-method
## [10] summary.Date
## [11] summary.PDF_Dictionary*
## [12] summary.PDF_Stream*
## [13] summary.POSIXct
## [14] summary.POSIXlt
## [15] summary.aov
## [16] summary.aovlist*
## [17] summary.aspell*
## [18] summary.check_packages_in_dir*
## [19] summary.connection
## [20] summary.data.frame
## [21] summary.default
## [22] summary.ecdf*
## [23] summary.factor
## [24] summary.glm
## [25] summary.infl*
## [26] summary.lm
## [27] summary.loess*
## [28] summary.manova
## [29] summary.matrix
## [30] summary.mlm*
## [31] summary.nls*
## [32] summary.packageStatus*
## [33] summary.ppr*
## [34] summary.prcomp*
## [35] summary.princomp*
## [36] summary.proc_time
## [37] summary.shingle*
## [38] summary.srcfile
## [39] summary.srcref
## [40] summary.stepfun
## [41] summary.stl*
## [42] summary.table
## [43] summary.trellis*
## [44] summary.tukeysmooth*
## [45] summary.yearmon*
## [46] summary.yearqtr*
## [47] summary.zoo*
## see '?methods' for accessing help and source code
```

The dot ‘`.`’ naming convention is quite unfortunate because one can artificially create functions which are not proper methods; for example, the `t.test` function is not the method `t` for objects of class `test` but it is just an R function which performs ordinary two-samples *t* test. Moreover, as the function `class` is an accessor function, i.e. can get and set data from/into an object, some weird things may happen. For example, we now create a vector and assign it the class `lm` as follows

```
x <- 1:4
x

## [1] 1 2 3 4
```

```
class(x)

## [1] "integer"

class(x) <- "lm"
class(x)

## [1] "lm"
```

But if we now try commands like `summary`, `print`, `plot` or similar for which methods explicitly designed for the class `lm` exist, R will return an error. The new system of classes and methods which is now fully implemented in R is called `S4`. Objects of class `S4` apparently behave like all other objects in R but they possess properties called ‘slots’, which can be accessed differently from other R objects. Moreover, the way they are constructed is more robust and the transition from a class to another is controlled finely or prevented in some cases to avoid the above misfunctioning situations. The next code estimates the maximum likelihood estimator for the mean of a Gaussian law. It uses the function `mle` from the package `stats4` which is an `S4` package as the name suggests. Again, we are not interested in the statistical part of this example just in R code

```
require(stats4)
set.seed(123)
y <- rnorm(100, mean=1.5)
f <- function(theta=0) -sum(dnorm(x=y, mean=theta,log=TRUE))
fit <- mle(f)
fit

##
## Call:
## mle(minuslogl = f)
##
## Coefficients:
##    theta
## 1.590406
```

We now have a look at the object `fit` returned by the `mle` function

```
str(fit)
```

```
## Formal class ’mle’ [package "stats4"] with 9 slots
## ..@ call : language mle(minuslogl = f)
## ..@ coef : Named num 1.59
## .. ..- attr(*, "names")= chr "theta"
## ..@ fullcoef : Named num 1.59
## .. ..- attr(*, "names")= chr "theta"
## ..@ vcov : num [1, 1] 0.01
## .. ..- attr(*, "dimnames")=List of 2
## .. .. ..$ : chr "theta"
## .. .. ..$ : chr "theta"
## ..@ min : num 133
## ..@ details :List of 6
## .. ..$ par : Named num 1.59
## .. .. ..- attr(*, "names")= chr "theta"
## .. ..$ value : num 133
## .. ..$ counts : Named int [1:2] 6 3
## .. .. ..- attr(*, "names")= chr [1:2] "function" "gradient"
## .. ..$ convergence: int 0
## .. ..$ message : NULL
## .. ..$ hessian : num [1, 1] 100
## .. .. ..- attr(*, "dimnames")=List of 2
## .. .. .. ..$ : chr "theta"
## .. .. .. ..$ : chr "theta"
## ..@ minuslogl:function (theta = 0)
## .. ..- attr(*, "srcref")=Class 'srcref' atomic [1:8] 4 6 4
## 60 6 60 4 4
## .. .. .. ..- attr(*, "srcfile")=Classes 'srcfilecopy',
## 'srcfile' <environment: 0x7f8ba6677a00>
## ..@ nobs : int NA
## ..@ method : chr "BFGS"
```

We now see that this is an `S4` object with slots that, as the structure suggests, can be accessed using the symbol `@` instead of `$`. For example,

```
fit@coef

##    theta
## 1.590406
```

To get the list of methods for `S4` objects, one should use the function `showMethods`

```
showMethods(summary)

## Function: summary (package base)
## object="ANY"
## object="cogarch.est"
## object="cogarch.est.incr"
## object="data.frame"
##     (inherited from: object="ANY")
## object="diagonalMatrix"
## object="lm"
##     (inherited from: object="ANY")
## object="mle"
## object="sparseMatrix"
## object="yuima.CP.qmle"
## object="yuima.carma.qmle"
## object="yuima.qmle"
```

1.8 The Yuima Package

1.8.1 How to Obtain the Package

The stable version of the **yuima** package is available through CRAN and can be installed from CRAN, as for any other package, typing the following command on the R console

```
install.packages("yuima")
```

or using the GUI functionality to install packages. The code companion to this book is supposed to work with the CRAN version. There exists also a development version of the **yuima** package which is hosted on R-Forge, and the Web page of the project is http://r-forge.r-project.org/projects/yuima. Development versions of the package are not supposed to be stable or functional; thus, it is for advanced users or future developers of **yuima** only. The development version of **yuima** package can be installed from R-Forge using the following command

```
install.packages("yuima",repos="http://R-Forge.R-project.org")
```

If, for some reason, the R-Forge system does not provide binary builds of the **yuima** package, the user can also try

```
install.packages("yuima",repos="http://R-Forge.R-project.org",
  type="source")
```

The package **yuima** depends on some other packages, such as **zoo** (Zeileis and Grothendieck 2005), which can be installed separately if R does not install fully all the dependencies. 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. Once the package has been installed on your system, before using any of the commands in this book, you should load the package as for any other R package as follows

```
library(yuima)
```

The official Web of the YUIMA Project can be found at https://yuima-project.com.

1.8.2 The Main Object and Classes

The **yuima** package adopts the S4 system of classes and methods (Chambers 1998). Although the discussion on the methods for simulation and inference for stochastic processes will be postponed to the second part of the book, here, we discuss the main classes of objects as well as some generic features and behaviour of the **yuima** package. As mentioned, there are various classes of objects defined in the **yuima** package but the main class is called the `yuima-class`. This class is composed of several slots. Figure 1.1 represents the 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 `model` and `sampling` have to be prepared, while the slot `data` will be filled by the simulator. We discuss in full detail the different objects separately in the following sections.

The general idea of the **yuima** package is to keep separate the information about the statistical model and the data into different objects to be used later by various

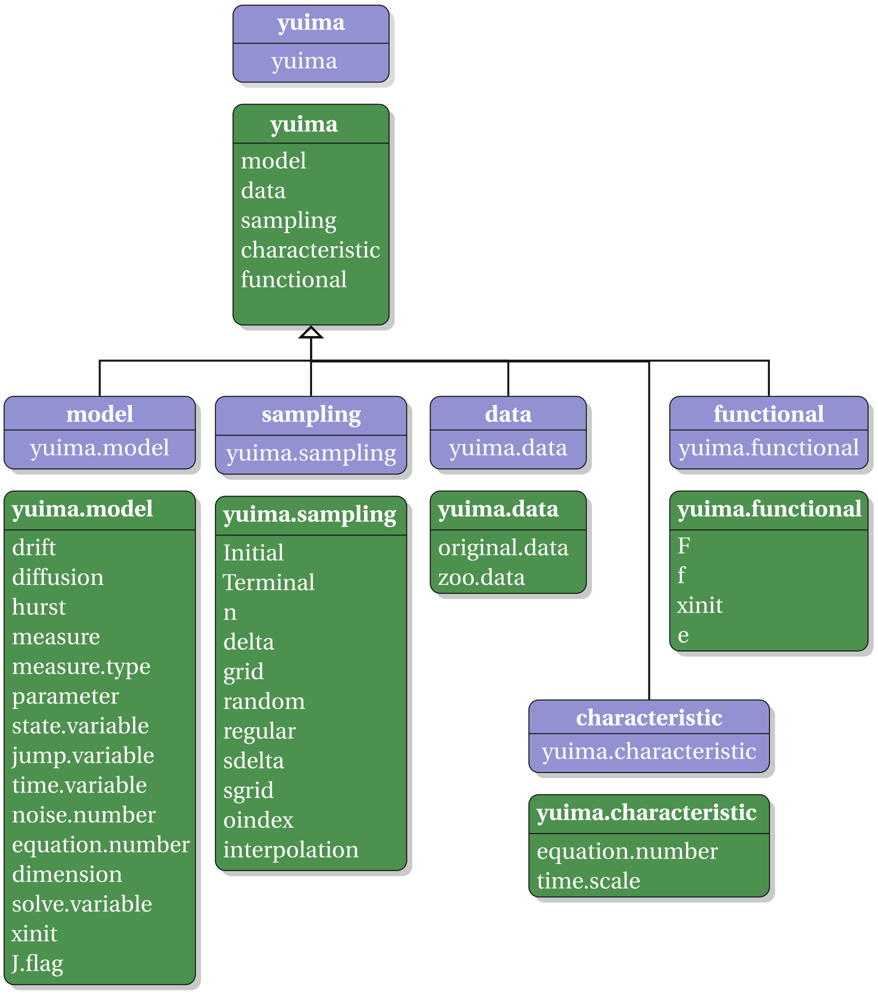


Fig. 1.1 Main classes in the **yuima** package

statistical methods. As it 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 stochastic differential equations 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 (by '`yuima` object' we mean 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 information. The details of `simulate`

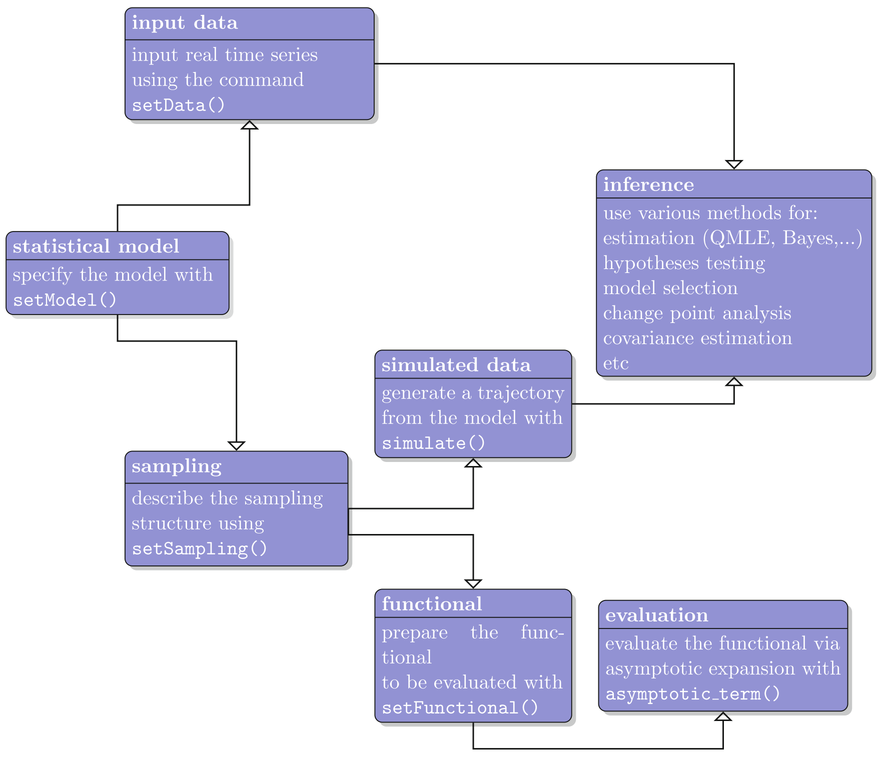

input data
input real time series using the command setData()
inference
use various methods for:
estimation (QMLE, Bayes,...)
hypotheses testing
model selection
change point analysis
covariance estimation
etc
statistical model
specify the model with setModel()
simulated data
generate a trajectory from the model with simulate()
sampling
describe the sampling structure using setSampling()
functional
prepare the functional to be evaluated with setFunctional()
evaluation
evaluate the functional via asymptotic expansion with asymptotic_term()

Fig. 1.2 An example of typical workflow of use of the main functionalities of the **yuima** package

will be explained in Sect. 1.10 along with the use of method `setSampling` which allows to specify a sampling scheme to be used by the `simulate` method.

However, a `yuima` object may contain the slot `data` not only as the outcome of `simulate` but also for own data the user wants to analyse. 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 stochastic models. These methods will be discussed from Sect. 2.4.

Further, functionals of stochastic differential equations can be defined using the `setFunctional` method and evaluated using asymptotic expansion methods as explained in Sect. 2.13. The `setFunctional` method creates a `yuima.functional` object which is included along with a `yuima.model` into a `yuima` object in order to be used for the evaluation of its expected value by asymptotic expansion methods. Figure 1.2 gives an example of the typical use of the functionalities of the **yuima** package.

1.8.3 The `yuima.model` Class

At present, in **yuima** several classes of stochastic differential equations driven by Brownian motion, Lévy processes or fractional Brownian motion can be easily specified as well as CARMA and COGARCH models and some types of point processes. Here, we present a brief overview of a simple one-dimensional diffusion model as part II of this book will consider several other models. This also allows to introduce an overall view of the slots of the `yuima.model` class. In **yuima**, one can describe a great variety of families of stochastic processes. These models can be one- or multidimensional and eventually described as parametric models. Let us consider the stochastic differential equation

$$\mathrm{d}X_t = a(t, X_t, \theta)dt + b(t, X_t, \theta)\mathrm{d}W_t, \quad X_0 = x_0,$$

where W_t is a standard Brownian motion. The three arguments of the functions $a(\cdot, \cdot, \cdot)$ and $b(\cdot, \cdot, \cdot)$ do not need to be specified every time. For example, if the model is homogeneous in time and the drift and diffusion coefficients do not contain the parameter θ, then it is sufficient to use the notation $a(x)$ and $b(x)$ to describe the model $\mathrm{d}X_t = a(X_t)dt + b(X_t)\mathrm{d}W_t$. Detailed hypotheses and regularity conditions on the coefficients $a(\cdot)$ and $b(\cdot)$ for each class of models will be given in the following sections. Nevertheless, it is important to remark that these notations only matter to the mathematical description of the model used in this book because the coefficients are passed to **yuima** methods as R mathematical expressions. This means that, for example, $a(t, X_t, \theta) = t \cdot \sqrt{\theta X_t}$ will be passed as `t*sqrt(x*theta)`; therefore, from the R point of view, the order of the arguments is not relevant as well as the mathematical description used in this text, although it is kept consistent through each section. It is worth to remark that the **yuima** accepts any user-specified notation for the state variable x (for X_t) and the time variable t so that the remaining terms in an R expression will be interpreted as parameters by **yuima** as explained in Sect. 1.9.1. We are now able to give an overview of the main slots of the most important classes of the **yuima** package.

The `yuima.model` class contains information about the stochastic process of interest. The constructor function `setModel` is used to provide a description of the model considered. All functions in the **yuima** package are assumed to extract as much information as possible from the classes to avoid duplications of code and data.

An object of class `yuima.model` may contain several slots, but we will discuss here only the subset which is relevant to this section. Still, the description given here is abstract and can be well understood looking at the examples of Sect. 1.9. The complete structure of a `yuima.model` object can be investigated as usual by using the R command `str` on a `yuima` object or on its slot `yuima.model`. What follows is a partial list of slots of the `yuima.model` class:

- `drift` is a `vector` of R expressions describing the drift coefficient.

- `diffusion` is a `list`-type object which describes the diffusion coefficient matrix; each slot of the list corresponds to one row of the diffusion matrix.
- `parameter`, which is a short name for 'parameters', is a `list`-type object with the following entries (more details in Sect. 1.9.3):
  - `all` contains the names of all the parameters found in the diffusion and drift coefficients.
  - `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.
- `state.variable` and `time.variable`, by default, are assumed to be `x` and `t`, respectively, but the user can freely choose names for them; these names matter to the right-hand side of the equation of the SDE. The `yuima.model` class assumes that the user either keeps 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 Sect. 1.9.1;
- `solve.variable` contains a vector of variable names, and each element corresponds to the name of the solution variable (left-hand side) of each equation in the model, in the respective order. An example of use can be found in Sect. 2.3.
- `noise.number` indicates the number of sources of noise.
- `xinit` is the initial value of the stochastic differential equation or a distribution.
- `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.

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. This will be discussed in more detail in Chap. 2.

1.9 On 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 the solutions of all the stochastic differential equations exist, while in Sect. 2.4, we will give regularity conditions needed to have a properly defined statistical model.

1.9.1 Basic Model Specification

Assume that we want to describe the following stochastic differential equation

$$\mathrm{d}X_t = -3X_t\mathrm{d}t + \frac{1}{1+X_t^2}\mathrm{d}W_t, \quad X_0 = x_0. \tag{1.1}$$

In the above $a(x) = -3x$ and $b(x) = \frac{1}{1+x^2}$ according to the notation of previous section, x_0 is the initial condition and W_t is a standard Wiener process. This model can be described in **yuima** specifying the drift and diffusion coefficients as plain R expressions passed as strings using the `setModel` function:

```
mod1 <- setModel(drift = "-3*x", diffusion = "1/(1+x^2)")
```

By default, the **yuima** package assumes that the state variable (`state.variable` in the `yuima.model` object) for X_t is `x` and the time variable (`time.variable` in the `yuima.model` object) is `t` and the solution variable is the same as the state variable, hence again `x`. Notice that the left-hand side of the equation is implicit, this is why `yuima.model` has the slot `solve.variable` to specify different cases, as we will see in Sect. 2.3. The user should not be worried about the warning raised by **yuima** at this stage, as this is just to inform him or her on the implicit assumption on the solution variable. More precisely, this is how `setModel` thinks about the different arguments:

$$\underbrace{\mathrm{d}X_t}_{\text{LHS =}}\ \texttt{solve.variable} = \underbrace{\overbrace{-3\quad\underbrace{X_t}_{\texttt{state.variable}}}^{\texttt{drift}}\ \overbrace{\mathrm{d}t}^{\texttt{time.variable}} + \overbrace{\frac{1}{1+\underbrace{X_t^2}_{\texttt{state.variable}}}}^{\texttt{diffusion}}\ \overbrace{\mathrm{d}W_t}^{\text{driving noise}}}_{\text{RHS = state and time variables}}.$$

At this point, the **yuima** package fills the proper slots of the `yuima` object

```
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 : list()
## ..@ measure : list()
## ..@ measure.type : chr(0)
## ..@ parameter :Formal class 'model.parameter' [package
## "yuima"] with 7 slots
## .. .. ..@ all : chr(0)
## .. .. ..@ common : chr(0)
## .. .. ..@ diffusion: chr(0)
## .. .. ..@ drift : chr(0)
## .. .. ..@ jump : chr(0)
## .. .. ..@ measure : chr(0)
## .. .. ..@ xinit : 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 : expression((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 motion, i.e. a fractional Brownian motion if Hurst index $H = \frac{1}{2}$. For more details, see Chap. 5.

For a quick look at the type of process being specified, one can simply type the name of the object in the R console or call the method `show` or `print` with the `yuima` object as argument. For example,

```
mod1

##
## Diffusion process
## Number of equations: 1
## Number of Wiener noises: 1
```

gives the same result as `show(mod1)` or `print(mod1)`.

1.9.2 User-Specified State and Time Variables

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

$$\mathrm{d}Y_s = -3sY_s\mathrm{d}s + \frac{1}{1+Y_s^2}\mathrm{d}W_s, \quad Y_0 = y_0, \tag{1.2}$$

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

```
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:

```
tmp <- capture.output(str(mod1b))
writeLines(strwrap(tmp[c(2,3,4,17,19,23)],width=60))

## ..@ drift : expression((-3 * s * y))
## ..@ diffusion :List of 1
## .. ..$ : expression((1/(1 + y^2)))
## ..@ state.variable : chr "y"
## ..@ time.variable : chr "s"
## ..@ solve.variable : chr "y"
```

where we have printed only the relevant information of `str(mod1b)` obtained through `capture.output`.

1.9.3 *Specification of Parametric Models*

Assume now that we want to describe this parametric model:

$$\mathrm{d}X_t = -\mu X_t \mathrm{d}t + \frac{1}{1+X_t^\gamma}\mathrm{d}W_t, \quad X_0 = x_0,$$

where $a(x,\mu) = -\mu x$ and $b(x,\gamma) = 1/(1+x^\gamma)$. This model is specified as follows:

```
mod2 <- setModel(drift = "-mu*x", diffusion = "1/(1+x^gamma)")
```

The **yuima** parser isolates the time and state variables in the expressions of the drift and diffusion coefficients and assumes that the remaining symbols are names of parameters; so, in this case, `mu` and `gamma`, which are different from `x` and `t`, are assumed to be parameters. Notice that in the above notation μ and γ are generic names for the components of a parameters' vector θ in the notation of Sect. 1.8.3.

```
tmp <- capture.output(str(mod2))
writeLines(strwrap(tmp[c(2,3,4,9:13,17,19,23)],width=60))

## ..@ drift : expression((-mu * x))
## ..@ diffusion :List of 1
## .. ..$ : expression((1/(1 + x^gamma)))
## ..@ parameter :Formal class 'model.parameter' [package
## "yuima"] with 7 slots
## .. .. ..@ all : chr [1:2] "mu" "gamma"
## .. .. ..@ common : chr(0)
## .. .. ..@ diffusion: chr "gamma"
## .. .. ..@ drift : chr "mu"
## ..@ state.variable : chr "x"
## ..@ time.variable : chr "t"
## ..@ solve.variable : chr "x"
```

Again, we can have a small summary of the object in the following way

```
mod2

##
## Diffusion process
## Number of equations: 1
## Number of Wiener noises: 1
## Parametric model with 2 parameters
```

1.10 Basic Facts on Simulation

The simulate function simulates yuima models according to the Euler–Maruyama scheme in the presence of nonfractional diffusion noise and Lévy jumps and uses the Cholesky or the Wood and Chan (1994) method for the fractional Gaussian noise. For diffusion models without jumps, **yuima** also implements a space discretized Euler–Maruyama method (see Sect. 1.10). We discuss here a quick way to perform simulation. Consider again the diffusion process of Eq. (1.1) from Sect. 1.9.1

$$\mathrm{d}X_t = -3X_t\mathrm{d}t + \frac{1}{1+X_t^2}\mathrm{d}W_t, \quad X_0 = x_0,$$

which was input into **yuima** as mod1. Now, with mod1 in hands, it is extremely easy to simulate a trajectory of the process as follows:

```
set.seed(123)
X <- simulate(mod1)
```

This trajectory can be plotted using the command plot

```
plot(X)
```

and the result is shown in Fig. 1.3.

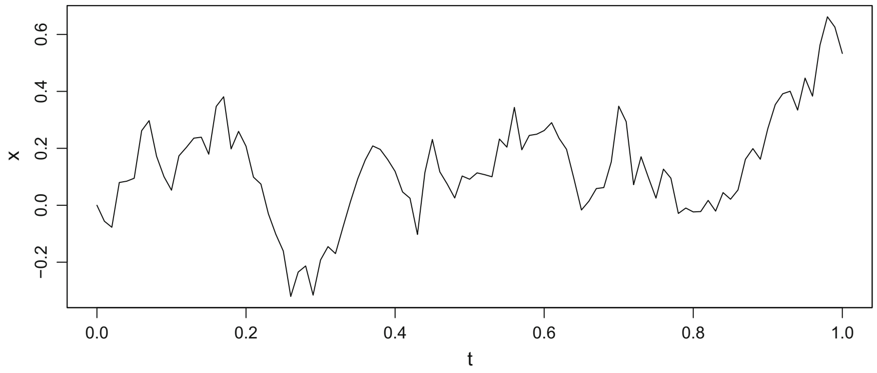


Fig. 1.3 The plot function is used to draw a trajectory of a simulated yuima object

1.10.1 Customization of Simulation Arguments

When no other arguments are passed to the simulate command, the default values are taken into account. In particular, if not specified, the initial value for the simula-

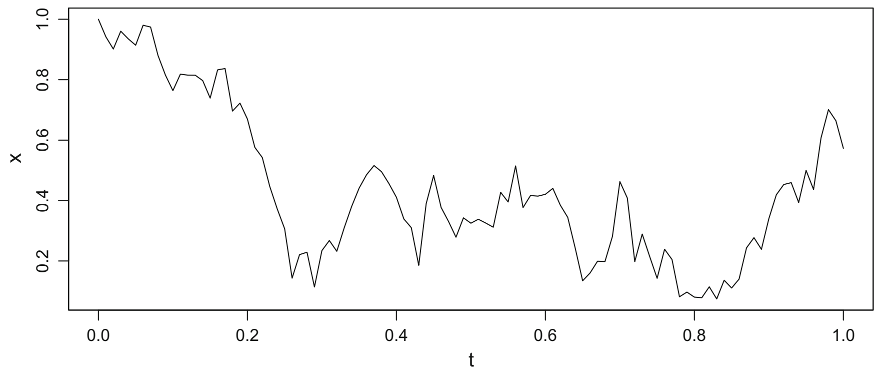


Fig. 1.4 A simulated trajectory with initial value specified with argument `xinit=x0` in the `simulate` command

tion, i.e. X_0, is always set to zero as we can see from Fig. 1.3. For different initial values, one can specify the argument `xinit` as in the next example (see Fig. 1.4),

```
x0 <- 1
set.seed(123)
X <- simulate(mod1, xinit=x0)
plot(X)
```

and the result is shown in Fig. 1.4.

Notice that the output of the `simulate` command is again a **yuima** object which contains, in addition to the model specification, other two slots: the `data` slot:

```
str(X@data,vec.len=2)
```

```
## Formal class 'yuima.data' [package "yuima"] with 2 slots
## ..@ original.data: Time-Series [1:101, 1] from 0 to 1: 1
## 0.942 ...
## .. ..- attr(*, "dimnames")=List of 2
## .. .. ..$ : NULL
## .. .. ..$ : chr "Series 1"
## ..@ zoo.data :List of 1
## .. ..$ Series 1:'zooreg' series from 0 to 1
## Data: num [1:101] 1 0.942 ...
## Index: num [1:101] 0 0.01 0.02 0.03 0.04 ...
## Frequency: 100
```

and the `sampling` slot:

```
str(X@sampling,vec.len=2)

## Formal class 'yuima.sampling' [package "yuima"] with 11 slots
##   ..@ Initial      : num 0
##   ..@ Terminal     : num 1
##   ..@ n            : int 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
##   ..@ sdelta       : num(0)
##   ..@ sgrid        : num(0)
##   ..@ oindex       : num(0)
##   ..@ interpolation: chr "pt"
```

which are filled with proper information. In the above, we have used the argument `vec.len=2` to shorten the output of the command `str` to the first few elements of the numerical vectors. Notice further that the `xinit` slot of the `yuima` object is also properly filled:

```
tmp <- capture.output(str(X))
writeLines(strwrap(tmp[c(14:16,29,31,35,36)],width=60))

## .. .. ..@ diffusion :List of 1
## .. .. .. ..$ : expression((1/(1 + x^2)))
## .. .. ..@ hurst : num 0.5
## .. .. ..@ jump.variable : chr(0)
## .. .. ..@ noise.number : num 1
## .. .. ..@ xinit : expression(1)
## .. .. ..@ J.flag : logi FALSE
```

A small summary of `X` will show both the model and the data structure of this `yuima` object can be obtained as follows:

```
X

##
## Diffusion process
## Number of equations: 1
## Number of Wiener noises: 1
##
## Number of original time series: 1
## length = 101, time range [0 ; 1]
##
## Number of zoo time series: 1
##          length time.min time.max delta
## Series 1    101        0        1  0.01
```

Other arguments which are taken as default values are the `Initial` and `Terminal` time values of the simulation as well as the number of steps in the simulation `n`. Next is an example of a trajectory for process (1.1) from $t_0 = 0.5$ to $T = 1.2$ (see Fig. 1.5)

```
x0 <- 1
set.seed(123)
X <- simulate(mod1, xinit=x0, Initial=0.5, Terminal=1.2)
X
plot(X)
```

The next code shows that `Initial` and `Terminal` are now set as required:

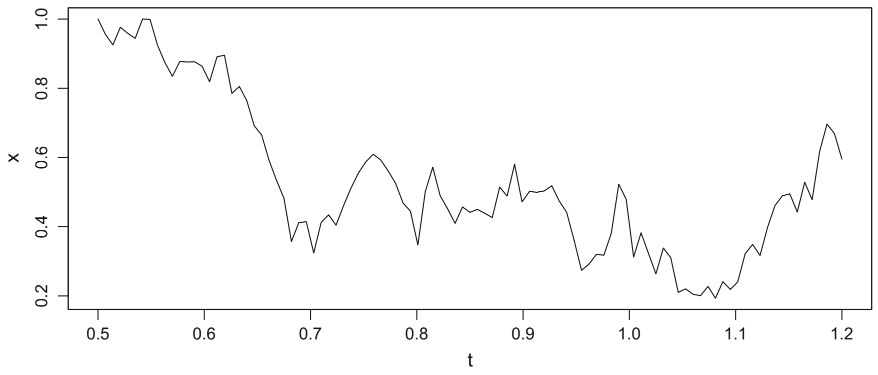


Fig. 1.5 A simulated trajectory with user-specified `Initial` and `Terminal` values

```
str(X@sampling,vec.len=2)

## Formal class 'yuima.sampling' [package "yuima"] with 11 slots
##   ..@ Initial      : num 0.5
##   ..@ Terminal     : num 1.2
##   ..@ n            : int 100
##   ..@ delta        : num 0.007
##   ..@ grid         :List of 1
##   .. ..$ : num [1:101] 0.5 0.507 0.514 0.521 0.528 ...
##   ..@ random       : logi FALSE
##   ..@ regular      : logi TRUE
##   ..@ sdelta       : num(0)
##   ..@ sgrid        : num(0)
##   ..@ oindex       : num(0)
##   ..@ interpolation: chr "pt"
```

Section 1.11 explains in full detail how to have complete control over the sampling scheme for a **yuima** object, and Table 1.2 summarizes some of the default values of the `simulate` method.

1.10.2 Simulation of Models with User-Specified Notation

Suppose now that we want to simulate the model of Eq. (1.2)

$$\mathrm{d}Y_s = -3sY_s\mathrm{d}s + \frac{1}{1+Y_s^2}\mathrm{d}W_s,$$

with user-specified notation as we did in object `mod1b` in Sect. 1.9.2. As the **yuima** package is aware of the user choice for the time and state variables, the situation remains unchanged, and the user just needs to call the `simulate` method on this object,

```
set.seed(123)
X <- simulate(mod1b, xinit=x0)
X
plot(X)
```

and the result is shown in Fig. 1.6.

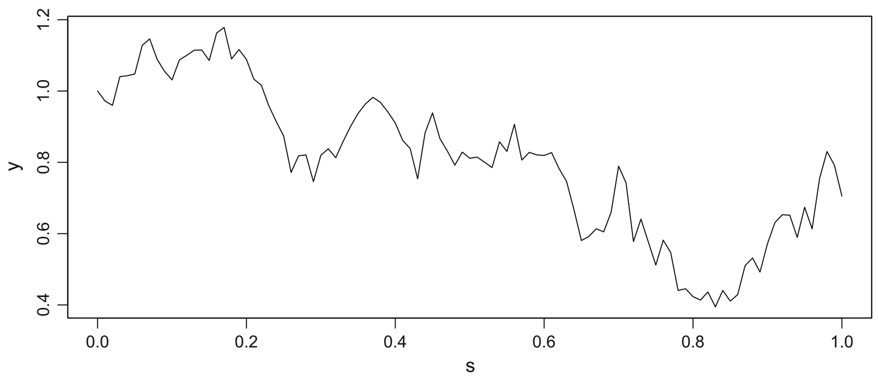


Fig. 1.6 A simulated trajectory of a **yuima** object with user-specified notation as in `mod1b`. Notice that the labels on the horizontal and vertical axes have been set accordingly

1.10.3 Simulation of Parametric Models

In order to simulate a parametric model, it is necessary to specify the values of the parameters via the argument `true.parameter` in the `simulate` command. Consider again the parametric model

$$\mathrm{d}X_t = -\mu X_t \mathrm{d}t + \frac{1}{1+X_t^\gamma}\mathrm{d}W_t$$

which was specified in the **yuima** object `mod2` in Sect. 1.9.3. In order to simulate a trajectory from this model, the `simulate` command needs to know which values of μ and γ have to be used. Next code shows how to specify the couple of values $\mu = 1$ and $\gamma = 3$ using a named `list` in the argument `true.parameter` (shortened to `true.par` in the example):

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

and the trajectory can be seen in Fig. 1.7.

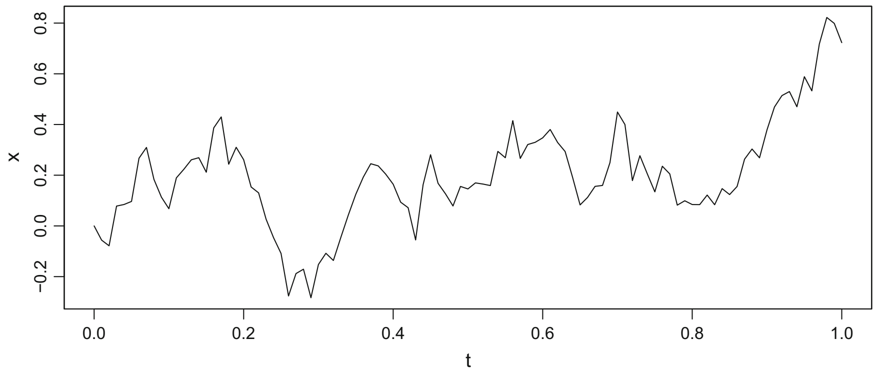


Fig. 1.7 A trajectory simulated from the parametric model `mod2` with user-specified parameters $\mu = 1$ and $\gamma = 3$

Table 1.2 Default values to the `simulate` method. Most options can be controlled using `sampling` and `subsampling` arguments

Description	Argument	Default value
Starting time t_0	`Initial`	0
Ending time T	`Terminal`	`Initial + n*delta` or 1 if `delta` is not specified
Initial value	`xinit`	0
Number of steps	`n`	100
Time mesh	`delta`	If not specified: (`Terminal-Initial`)/`n`
Grid of times	`grid`	If specified, overwrites `Initial`, `Terminal`, `n` and `delta`

1.11 Sampling and Simulate

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 stochastic differential equation or a given time series in the `yuima.data` slot of a `yuima` object. Both sampling and subsampling can be specified 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. Although we will discuss more in detail how to specify multidimensional diffusion processes in Sect. 2.3, let us consider the following two-dimensional model

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

Now we prepare the model using the `setModel` constructor function specifying a vector of drift functions and a matrix of diffusion coefficients (more details in Sect. 2.3) as **yuima** requires a vector representation for the drift coefficient and a matrix representation for the diffusion coefficient. The above model should be passed to **yuima** in the following matrix representation:

$$\begin{pmatrix} dX_{1,t} \\ dX_{2,t} \end{pmatrix} = \begin{pmatrix} -\theta X_{1,t} \\ -X_{1,t} - \gamma \cdot X_{2,t} \end{pmatrix} dt + \begin{bmatrix} 1 & 0 & X_{2,t} \\ X_{1,t} & \beta & 0 \end{bmatrix} \begin{pmatrix} dW_{1,t} \\ dW_{2,t} \\ dW_{3,t} \end{pmatrix}$$

We now prepare a vector of drift expressions in the object `b` and a diffusion matrix in object `s` as well as the description of the state variables we want to use to represent this model in object `sol`:

```r
sol <- c("x1","x2") # variable for numerical solution
b <- c("-theta*x1","-x1-gamma*x2") # drift vector
s <- matrix(c("1","x1","0","beta","x2","0"),2,3) # diff. mat.
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 using the command `setSampling` as follows:

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

and let us analyse its content

```r
str(samp)
```

```
## Formal class 'yuima.sampling' [package "yuima"] with 11
## slots
## ..@ Initial : num 0
## ..@ Terminal : num 3
## ..@ n : int 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"
```

As seen from the output, the sampling structure is quite rich and we will show how to specify some of the slots in the next section. We simulate this process by specifying the `sampling` argument in the `simulate` method

```r
set.seed(123)
X2 <- simulate(mymod, sampling=samp,
 true.param=list(theta=1,gamma=1,beta=1))
X2

##
## Diffusion process
## Number of equations: 2
## Number of Wiener noises: 3
## Parametric model with 3 parameters
##
## Number of original time series: 2
## length = 3001, time range [0 ; 3]
##
## Number of zoo time series: 2
##          length time.min time.max delta
## Series 1   3001        0        3 0.001
## Series 2   3001        0        3 0.001
```

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

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

```
## Formal class 'yuima.sampling' [package "yuima"] with 11
## slots
## ..@ Initial : num 0
## ..@ Terminal : num [1:2] 3 3
## ..@ n : int [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"
```

1.11.1 Sampling and 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
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
## ..@ n : num(0)
## ..@ delta : num(0)
## ..@ grid : NULL
## ..@ random :List of 1
## .. ..$ rdist:List of 2
## .. .. ..$ :function (x)
## .. .. .. ..- attr(*, "srcref")=Class 'srcref' atomic [1:8]
## 2 22 2 49 22 49 2 2
## .. .. .. .. .. ..- attr(*, "srcfile")=Classes
## 'srcfilecopy', 'srcfile' <environment: 0x7f8ba5188b58>
## .. .. ..$ :function (x)
## .. .. .. ..- attr(*, "srcref")=Class 'srcref' atomic [1:8]
## 3 1 3 28 1 28 3 3
## .. .. .. .. .. ..- attr(*, "srcfile")=Classes
## 'srcfilecopy', 'srcfile' <environment: 0x7f8ba5188b58>
## ..@ 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 using the argument `random` in `setSampling`. The argument `random` accepts a named `list`, where the name `rdist` is used to specify the distribution of the random times in the form of a random number generator. In the example, we chose `rexp` to specify exponential random times with some rate. As we have a two-dimensional process, we have specified a vector of random number generators. Looking at the result of the simulation, we can notice that the slot `regular` is now set to `FALSE`. We subsample the original trajectory of `X2` using the `subsampling` function (see Fig. 1.8)

```
newdata <- subsampling(X2, sampling=newsamp)
newdata

##
## Diffusion process
## Number of equations: 2
## Number of Wiener noises: 3
## Parametric model with 3 parameters
##
## Number of original time series: 2
## length = 3001, time range [0 ; 3]
##
## Number of zoo time series: 2
##          length time.min time.max     delta note
## Series 1     31        0    2.951 0.3586586    *
## Series 2     70        0    2.999 0.1496092    *
## ================
## * : maximal mesh

plot(X2,plot.type="single", lty=c(1,3),ylab="X2")
```

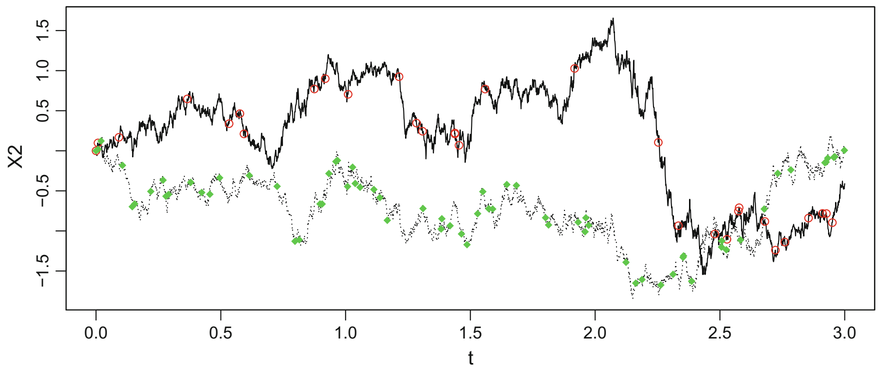


Fig. 1.8 An example of Poisson random subsampling: green and red dots are sampled according to two different and independent Poisson processes

```r
points(get.zoo.data(newdata)[[1]],col="red")
points(get.zoo.data(newdata)[[2]],col="green",pch=18)
```

where we extract the data from the **yuima** object using the method `get.zoo.data` which returns the `zoo.data` slot from the `yuima.data` slot of a `yuima` object. As the `zoo.data` is a `list`-type object where each element contains a single time series, to access the first time series of a multidimensional stochastic process in a `yuima` object we need to type `get.zoo.data(myobj)[[1]]`, where `myobj` is the `yuima` object containing the `data` slot. Notice that, for random sampling, the time series will be irregularly spaced and so the `delta` between the observations is not unique, so the `print` method calculates the largest time lag between the observations and an asterisk is shown to indicate this. Further, the minimal and maximal time span for the observations depends on the random sampling as well.

We can also operate a deterministic sampling by specifying two different regular frequencies (see Fig. 1.9) or, better, two different values for `delta`. In this case, we need to explicitly set `n` to `NULL`; otherwise, the default value of `n` takes precedence over `Terminal`, which is recalculated.

```r
newsamp <- setSampling(Terminal=3, delta=c(0.1,0.2), n=NULL)
newsamp

## Formal class 'yuima.sampling' [package "yuima"] with 11 slots
##   ..@ Initial       : num [1:2] 0 0
##   ..@ Terminal      : num [1:2] 3 3
##   ..@ n             : int [1:2] 30 15
##   ..@ delta         : num [1:2] 0.1 0.2
##   ..@ grid          :List of 2
##   .. ..$ : num [1:31] 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 ...
##   .. ..$ : num [1:16] 0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 ...
##   ..@ random        : logi FALSE
##   ..@ regular       : logi TRUE
##   ..@ sdelta        : num(0)
##   ..@ sgrid         : num(0)
```

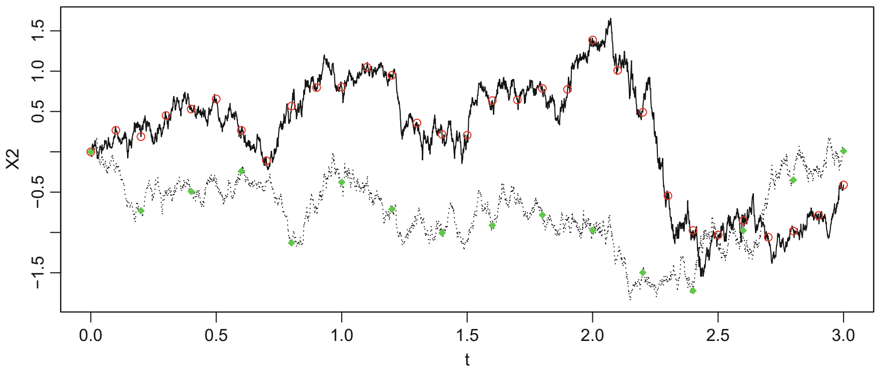


Fig. 1.9 An example of deterministic subsampling: the frequency of red dots is two times the one of the green dots

```
##    ..@ oindex        : num(0)
##    ..@ interpolation: chr "pt"

newdata <- subsampling(X2, sampling=newsamp)
newdata

##
## Diffusion process
## Number of equations: 2
## Number of Wiener noises: 3
## Parametric model with 3 parameters
##
## Number of original time series: 2
## length = 3001, time range [0 ; 3]
##
## Number of zoo time series: 2
##          length time.min time.max delta
## Series 1     31        0        3   0.1
## Series 2     16        0        3   0.2

plot(X2,plot.type="single", lty=c(1,3),ylab="X2")
points(get.zoo.data(newdata)[[1]],col="red")
points(get.zoo.data(newdata)[[2]],col="green", pch=18)
```

Notice that the number of resulting observations is now the result of the subsampling.
Again, one can look at the structure of the sampling structure

```
str(newdata@sampling)

## Formal class 'yuima.sampling' [package "yuima"] with 11 slots
##    ..@ Initial       : num [1:2] 0 0
##    ..@ Terminal      : num [1:2] 3 3
##    ..@ n             : int [1:2] 31 16
##    ..@ delta         : num [1:2] 0.1 0.2
##    ..@ grid          :List of 2
```

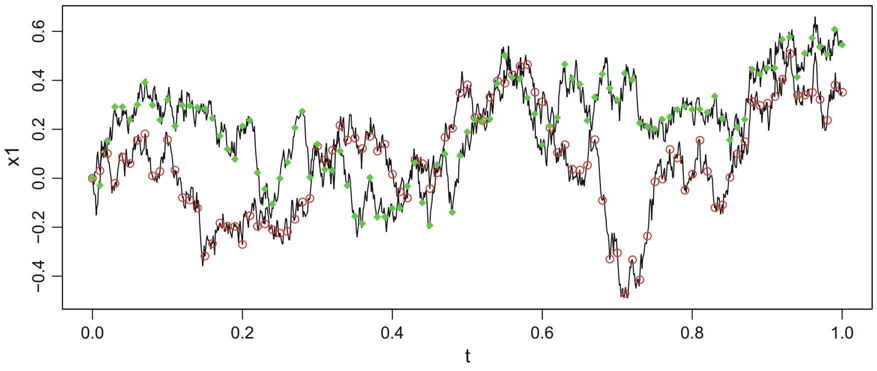


Fig. 1.10 An example of subsampling used within the `simulate` command

```
##    .. ..$ : num [1:31] 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 ...
##    .. ..$ : num [1:16] 0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 ...
##    ..@ random       : logi [1:2] FALSE FALSE
##    ..@ regular      : logi [1:2] TRUE TRUE
##    ..@ sdelta       : num(0)
##    ..@ sgrid        : num(0)
##    ..@ oindex       :List of 2
##    .. ..$ : num [1:31] 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 ...
##    .. ..$ : num [1:16] 0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 ...
##    ..@ interpolation: chr "pt"
```

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 Fig. 1.10). This can be done in a single step in the following way:

```
set.seed(123)
Y.sub <- simulate(mymod,sampling=setSampling(delta=0.001,n=1000),
 subsampling=setSampling(delta=0.01,n=100),
 true.par=list(theta=1,beta=1,gamma=1))
set.seed(123)
Y <- simulate(mymod, sampling=setSampling(delta=0.001,n=1000),
  true.par=list(theta=1,beta=1,gamma=1))
plot(Y, plot.type="single")
points(get.zoo.data(Y.sub)[[1]],col="red")
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 Fig. 1.11.

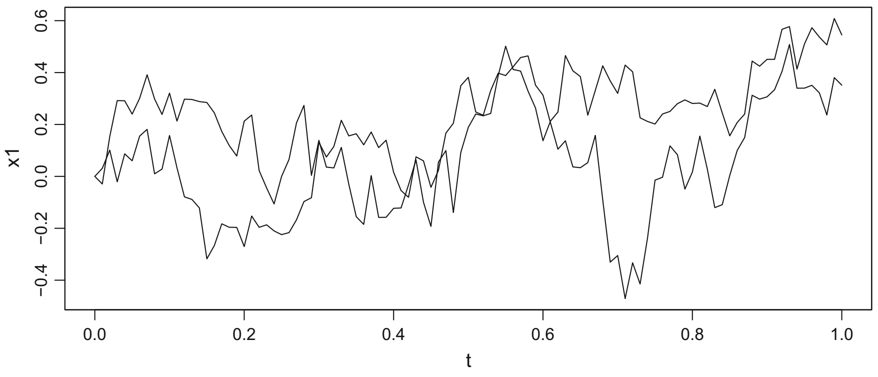


Fig. 1.11 Plotting directly the subsampled trajectory `Y.sub`

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

```
Y

##
## Diffusion process
## Number of equations: 2
## Number of Wiener noises: 3
## Parametric model with 3 parameters
##
## Number of original time series: 2
## length = 1001, time range [0 ; 1]
##
## Number of zoo time series: 2
##           length time.min time.max delta
## Series 1    1001        0        1 0.001
## Series 2    1001        0        1 0.001

Y.sub

##
## Diffusion process
## Number of equations: 2
## Number of Wiener noises: 3
## Parametric model with 3 parameters
##
## Number of original time series: 2
## length = 1001, time range [0 ; 1]
##
## Number of zoo time series: 2
##           length time.min time.max delta
## Series 1     101        0        1  0.01
## Series 2     101        0        1  0.01
```

1.12 How to Make Data Available into a yuima Object

Although most of the examples in this chapter 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 fit a yuima.model called mod on the data X one can use a code like the following:

```
my.yuima <- setYuima(data=setData(X), model=mod)
```

and then pass my.yuima to the inference functions as described in the following. In the previous code, setData transforms time series data in a form that is liked by the **yuima** package. In particular, when data are added to a yuima object into the slot data, the data itself are duplicated: one slot original.data contains the data as passed by the user, and the slot zoo.data contains a zoo version of the data.

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. First of all, we download the data for the IBM stock using the function getSymbols from the **quantmod** package[2] directly from Yahoo Finance:

```
require(quantmod)

## Loading required package: quantmod
## Loading required package: xts
## Loading required package: TTR
## Version 0.4-0 included new data defaults. See ?getSymbols.

getSymbols("IBM", to = "2017-07-31")

## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0. You will still be able to use
## 'loadSymbols' to automatically load data. getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.
##
## WARNING: There have been significant changes to Yahoo Finance data.
## Please see the Warning section of '?getSymbols.yahoo' for details.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.yahoo.warning"=FALSE).
```

The data downloaded from Yahoo Finance are transformed into a xts object with the same name of the symbol, i.e. IBM

```
str(IBM)

## An 'xts' object on 2007-01-03/2017-07-28 containing:
```

[2]In Sect. 1.12.1 we will present different other ways to obtain financial data from the internet.

```
##   Data: num [1:2662, 1:6] 125 125 126 127 128 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:6] "IBM.Open" "IBM.High" "IBM.Low" "IBM.Close" ...
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
## List of 2
##  $ src    : chr "yahoo"
##  $ updated: POSIXct[1:1], format: "2018-02-02 15:44:17"
```

The data downloaded from Yahoo Finance are transformed into a `xts` object with the same name of the symbol, i.e. `IBM`

```
str(IBM)

## An 'xts' object on 2007-01-03/2017-07-28 containing:
##   Data: num [1:2662, 1:6] 125 125 126 127 128 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:6] "IBM.Open" "IBM.High" "IBM.Low" "IBM.Close" ...
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
## List of 2
##  $ src    : chr "yahoo"
##  $ updated: POSIXct[1:1], format: "2018-02-02 15:44:17"
```

which we can inspect with the `head` command just to shorten the output and have a quick preview of this content:

```
head(IBM)

##            IBM.Open IBM.High IBM.Low IBM.Close
## 2007-01-03  125.319  126.892 124.133     97.27
## 2007-01-04  125.409  127.395 124.932     98.31
## 2007-01-05  125.861  126.312 124.971     97.42
## 2007-01-08  127.021  128.311 126.828     98.90
## 2007-01-09  127.769  129.381 127.756    100.07
## 2007-01-10  127.021  127.731 126.286     98.89
##            IBM.Volume IBM.Adjusted
## 2007-01-03    9196800     75.42905
## 2007-01-04   10524500     76.23550
## 2007-01-05    7221300     75.54537
## 2007-01-08   10340000     76.69302
## 2007-01-09   11108200     77.60034
## 2007-01-10    8744800     76.68530
```

Suppose we are interested, for example, in the `Close` column which corresponds to the closing quotations for the IBM in our case. We access these data by pointing to `IBM.Close` column of the object `IBM`. We can now use `setData` to prepare the data for the new `yuima` object created with `setYuima` as follows:

```
x <- setYuima(data=setData(IBM$IBM.Close))
str(x@data)

## Formal class 'yuima.data' [package "yuima"] with 2 slots
##   ..@ original.data:An 'xts' object on 2007-01-03/2017-07-28 containing:
##   Data: num [1:2662, 1] 97.3 98.3 97.4 98.9 100.1 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr "IBM.Close"
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
## List of 2
##   .. ..$ src    : chr "yahoo"
##   .. ..$ updated: POSIXct[1:1], format:  ...
##   ..@ zoo.data     :List of 1
##   .. ..$ IBM.Close:'zoo' series from 2007-01-03 to 2017-07-28
##   Data: num [1:2662] 97.3 98.3 97.4 98.9 100.1 ...
##   Index:  Date[1:2662], format:  ...
```

As can be seen from Fig. 1.12 (top plot), the data in the `yuima` object keep the original time stamps. However, in some cases, one of the crucial information is the time lag between two consecutive observations. As in the case above of daily data, in estimation the data are considered consecutive even if there are holidays or nonworking days between the data (although missing data are always kept). For the above case, the time lag is numerically equivalent to one, i.e. $\Delta = 1$, and this fact may cause problems in the application of the asymptotic high-frequency theory to these data. In this situation, as the chosen value of Δ is completely arbitrary, the best thing to do is to change the time of the data in a way meaningful for the analysis under consideration. Usually, at least in finance, for daily data the reference time horizon T is one year and the time lag between two consecutive data is taken as $\Delta = 1/252 \simeq 0.00397$, where 252 is the average number of working days of financial markets in a year. It is possible in this case to pass the argument `delta` directly into the function `setData`. In this case, the original data will not be altered but only the `zoo.data` slot, which is used later in estimation. Indeed, we can proceed as follows

```
y <- setYuima(data=setData(IBM$IBM.Close, delta=1/252))
str(y@data)

## Formal class 'yuima.data' [package "yuima"] with 2 slots
##   ..@ original.data:An 'xts' object on 2007-01-03/2017-07-28 containing:
##   Data: num [1:2662, 1] 97.3 98.3 97.4 98.9 100.1 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr "IBM.Close"
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
## List of 2
##   .. ..$ src    : chr "yahoo"
##   .. ..$ updated: POSIXct[1:1], format:  ...
##   ..@ zoo.data     :List of 1
##   .. ..$ IBM.Close:'zoo' series from 0 to 10.5595238095238
##   Data: num [1:2662] 97.3 98.3 97.4 98.9 100.1 ...
##   Index:  num [1:2662] 0 0.00397 0.00794 0.0119 0.01587 ...
```

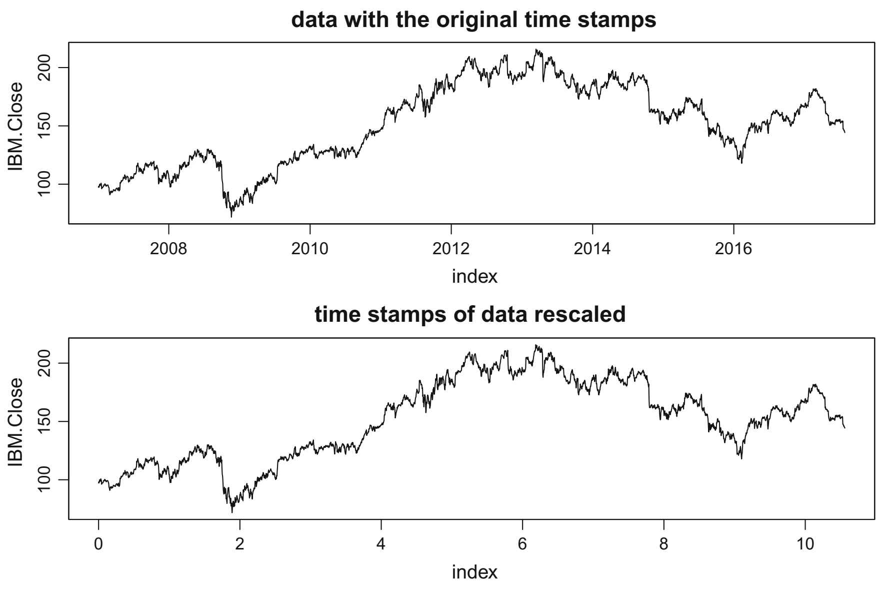


Fig. 1.12 Changing time stamps with `setData` for later inference

Indeed, the `original.data` has not been altered while the `zoo.data` slot has (see also Fig. 1.12 bottom plot).

```
plot(x, main="data with the original time stamps")
plot(y, main="time stamps of data rescaled")
```

From the summary below, we can notice that the time stamps of the original time series are kept for later use and passed to the `zoo` slot unless a different `delta` has been specified at the time of `setData`

```
x

##
##
## Number of original time series: 1
## length = 2662, time range [2007-01-03 ; 2017-07-28]
##
## Number of zoo time series: 1
##             length   time.min   time.max delta note
## IBM.Close   2662 2007-01-03 2017-07-28     5    *
## ================
## * : maximal mesh

y


##
##
```

```
## Number of original time series: 1
## length = 2662, time range [2007-01-03 ; 2017-07-28]
##
## Number of zoo time series: 1
##             length time.min time.max       delta
## IBM.Close    2662        0    10.56 0.003968254
```

1.12.1 Getting Data from Data Providers

Nowadays, there exist several data providers which distribute date via `http` queries or text-based files. Some packages make the process of data acquisition easier. Most package stores the data in their own type of objects. These type of objects will be discussed from Sect. 1.14 on.

The package **quantmod** (Ryan 2013) provides a function which allows to obtain data from Yahoo! Finance,[3] Google Finance,[4] FRED[5] - Federal Reserve Bank of St. Louis or OANDA[6] but also from `MySQL` data bases, `csv` files or plain R data. We have already used the function `getSymbols` to download data. For example, with

```
library(quantmod)
getSymbols("IBM", to = "2017-07-31")

## [1] "IBM"

attr(IBM, "src")


## [1] "yahoo"
```

To get the data from Google Finance, one has to set the argument `src` in this way

```
getSymbols("IBM", to = "2017-07-31",  src="google")

## [1] "IBM"

attr(IBM, "src")

## [1] "google"
```

FRED or OANDA offer exchange rates and currencies data

```
getSymbols("DEXUSEU",src="FRED")

## [1] "DEXUSEU"
```

[3]http://finance.yahoo.com.
[4]http://finance.google.com.
[5]http://research.stlouisfed.org/fred2.
[6]http://www.oanda.com.

```
attr(DEXUSEU, "src")

## [1] "FRED"

getSymbols("EUR/USD",src="oanda")

## [1] "EURUSD"

attr(EURUSD, "src")

## [1] "oanda"

str(EURUSD)

## An 'xts' object on 2017-08-07/2018-02-01 containing:
##   Data: num [1:179, 1] 1.18 1.18 1.17 1.17 1.18 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr "EUR.USD"
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
## List of 2
##  $ src    : chr "oanda"
##  $ updated: POSIXct[1:1], format: "2018-02-02 15:44:20"
```

Once the data have been acquired, `getSymbols` creates an object of class `xts` in the R workspace with the same name of the symbol and containing these data.

The package **fImport** (Wuertz and many others 2013) is similar in its use but returns objects of class `timeSeries` or `fWEBDATA`. Both `yahooSeries` or `yahooImport` can be used to get data from Yahoo! Similar functionalities exist for FRED (`fredSeries`, `fredImport`) and OANDA (`oandaSeries`, `oandaImport`).

The function `get.hist.quote` provided by the package **tseries** (Trapletti and Hornik 2013) downloads data from Yahoo! Finance and creates a `zoo` object

```
library(tseries)
x <- get.hist.quote("IBM")
str(x)
```

```
## 'zoo' series from 1991-01-02 to 2018-01-31
## Data: num [1:6824, 1:4] 81.8 81.4 81.7 80.7 80 ...
## - attr(*, "dimnames")=List of 2
## ..$ : NULL
## ..$ : chr [1:4] "Open" "High" "Low" "Close"
## Index: Date[1:6824], format: "1991-01-02" "1991-01-03"
## "1991-01-04" ...
```

If a licence to Bloomberg is available, the **RBloomberg** package allows for direct interaction with this platform. The function `RBloomberg` makes use of the Desktop COM API which requires the additional **RDCOMClient** or **rcom** package to be installed on the user's workstation.

1.13 How to Extract Data from a `yuima` Object

In Sect. 1.11.1, we have already used the method `get.zoo.data` to extract data from a `yuima` object. The `data` slot contains, as said, both the `original.data` slot, which can store essentially any type of time series object, and the `zoo.data` slot, which reorganizes the time series into a `list` object, where each element of the list is a `zoo` time series. We will discuss in full details many time series classes available in R including the `zoo` class in Sect. 1.14.1. In order to extract the first component of a multidimensional time series (or the one-dimensional time series from the data), we need to use a command like this

```
mydat <- get.zoo.data(y)[[1]]
str(mydat)

## 'zoo' series from 0 to 10.5595238095238
##   Data: num [1:2662] 97.3 98.3 97.4 98.9 100.1 ...
##   Index:  num [1:2662] 0 0.00397 0.00794 0.0119 0.01587 ...
```

To have access to the original data, one should extract the slot `original.data` from the `yuima` object as follows:

```
head(y@data@original.data)

##            IBM.Close
## 2007-01-03     97.27
## 2007-01-04     98.31
## 2007-01-05     97.42
## 2007-01-08     98.90
## 2007-01-09    100.07
## 2007-01-10     98.89
```

```
str(y@data@original.data)
```

```
## An 'xts' object on 2007-01-03/2017-07-28 containing:
## Data: num [1:2662, 1] 97.3 98.3 97.4 98.9 100.1 ...
## - attr(*, "dimnames")=List of 2
## ..$ : NULL
## ..$ : chr "IBM.Close"
## Indexed by objects of class: [Date] TZ: UTC
## xts Attributes:
## List of 2
## $ src : chr "yahoo"
## $ updated: POSIXct[1:1], format: "2018-02-02 15:44:17"
```

1.14 Time Series Classes, Time Data and Time Stamps

The reader expert in the manipulation of time stamps, dates, time series objects and related subjects can skip this section and continue the reading of this book from the next chapter.

1.14.1 Review of Some Time Series Objects in R

This section presents a basic review of some of the time series classes available in R.

1.14.1.1 The `ts` Class

The elementary class of time series object is called `ts`. The multidimensional extension of this class is called `mts`, and they share the same properties. The `ts` structure is designed for handling regularly spaced time series where observations have a given `frequency` (e.g. 12 for monthly data, 7 for daily data) and a given time mesh between observations `deltat`. The arguments `start` date and/or the final date `end` must be specified when a new object is created. Suppose we want to create a time series from this randomly generated data

```
set.seed(123)
some.data <- rnorm(12)
str(some.data)

##  num [1:12] -0.5605 -0.2302 1.5587 0.0705 0.1293 ...
```

To make it to appear as quarterly data starting from the second quarter of 1959, we need to type something like this:

```
X <- ts(some.data, frequency = 4, start = c(1961, 2))
X

##             Qtr1        Qtr2        Qtr3        Qtr4
## 1961             -0.56047565 -0.23017749  1.55870831
## 1962  0.07050839  0.12928774  1.71506499  0.46091621
## 1963 -1.26506123 -0.68685285 -0.44566197  1.22408180
## 1964  0.35981383
```

If we want to create monthly data starting from February 1964, we input it in this way:

```
set.seed(123)
X <- ts(some.data, start = c(1964, 2), frequency = 12)
X

##              Jan         Feb         Mar         Apr
## 1964             -0.56047565 -0.23017749  1.55870831
## 1965  0.35981383
##              May         Jun         Jul         Aug
## 1964  0.07050839  0.12928774  1.71506499  0.46091621
## 1965
##              Sep         Oct         Nov         Dec
## 1964 -1.26506123 -0.68685285 -0.44566197  1.22408180
## 1965
```

where `frequency` describes the number of data points per period; i.e. the time lag is $\Delta = 1/12 \simeq 0.083$. There are several accessory functions to extract information from a `ts` object. In particular, `time` returns the time instant of each observation in the data set; `deltat` extracts the Δt between observations; `end` and `start` return, respectively, the initial and terminal dates, and the frequency of the time series can be obtained with `frequency`:

```
time(X)[1:12]

##  [1] 1964.083 1964.167 1964.250 1964.333 1964.417
##  [6] 1964.500 1964.583 1964.667 1964.750 1964.833
## [11] 1964.917 1965.000

deltat(X)

## [1] 0.08333333

start(X)

## [1] 1964    2

end(X)

## [1] 1965    1

frequency(X)

## [1] 12
```

It is also possible to subset time series using the function `window`. Next code shows how to get quarterly data from `X`

```
window(X, frequency=4)

## Time Series:
## Start = 1964.08333333333
## End = 1964.83333333333
## Frequency = 4
## [1] -0.56047565  0.07050839  0.46091621 -0.44566197
```

1.14.1.2 The zoo Class

The zoo class can host time series in more general way, and it is adopted by **yuima** package to store time series data internally in the slot zoo.data of a yuima.data object. The zoo objects can be indexed by any sequence of real numbers (the name zoo stands for ‘$\mathbb{Z}$-indexed ordered object’). To use zoo, one should load the corresponding **zoo** package (Zeileis and Grothendieck 2005). If no set of indexes is specified, a new zoo object uses an increasing sequence of integers.

```
require(zoo)
X <- zoo( some.data )
X

##           1           2           3           4
## -0.56047565 -0.23017749  1.55870831  0.07050839
##           5           6           7           8
##  0.12928774  1.71506499  0.46091621 -1.26506123
##           9          10          11          12
## -0.68685285 -0.44566197  1.22408180  0.35981383

str(X)

## 'zoo' series from 1 to 12
##   Data: num [1:12] -0.5605 -0.2302 1.5587 0.0705 0.1293 ...
##   Index:  int [1:12] 1 2 3 4 5 6 7 8 9 10 ...
```

To alter or access the index, one can use either time or, better, index

```
index(X)

##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

The object zoo allows also for irregularly spaced time series. For example, let us generate 12 random times from the exponential distribution:

```
rtimes <- cumsum(rexp(12,rate=0.2))
rtimes

##  [1]  4.217286  7.100338 13.745612 13.903499 14.184554
##  [6] 15.767060 17.338196 18.064530 31.695713 31.841480
## [11] 36.865630 39.266704
```

and then create a time series making use of the argument order.by:

```
X <- zoo( rnorm(12), order.by = rtimes)
X

##     4.2173     7.1003    13.7456    13.9035    14.1846
##  0.3598138  0.4007715  0.1106827 -0.5558411  1.7869131
##    15.7671    17.3382    18.0645    31.6957    31.8415
##  0.4978505 -1.9666172  0.7013559 -0.4727914 -1.0678237
##    36.8656    39.2667
## -0.2179749 -1.0260044
```

```
str(X)

## 'zoo' series from 4.21728630529201 to 39.2667037211185
##   Data: num [1:12] 0.36 0.401 0.111 -0.556 1.787 ...
##   Index:  num [1:12] 4.22 7.1 13.75 13.9 14.18 ...
```

To mimic the ts behaviour, one should use explicitly the zooreg (where 'reg' stands for 'regular') function:

```
Xreg <- zooreg(some.data, start = c(1964, 2),  frequency = 12)
time(Xreg)

##  [1] "Feb 1964" "Mar 1964" "Apr 1964" "May 1964"
##  [5] "Jun 1964" "Jul 1964" "Aug 1964" "Sep 1964"
##  [9] "Oct 1964" "Nov 1964" "Dec 1964" "Jan 1965"
```

Any ts object can be converted into a zoo object via as.zoo, but the contrary is possible (via as.ts) only if the time series is regularly spaced, otherwise times are disregarded:

```
Y <- as.ts(X)
time(X)

##  [1]  4.217286  7.100338 13.745612 13.903499 14.184554
##  [6] 15.767060 17.338196 18.064530 31.695713 31.841480
## [11] 36.865630 39.266704

time(Y)

## Time Series:
## Start = 1
## End = 12
## Frequency = 1
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

1.14.1.3 The Class xts

The class xts provided by the package **xts** (Ryan and Ulrich 2014) is specifically designed to handle efficiently dates and time stamps. Here the 'x' means 'extensible'. By default, no index is assigned by the xts function, so it is necessary to specify one:

```
require(xts)
my.time.stamps <- as.Date(rtimes)
my.time.stamps

##  [1] "1970-01-05" "1970-01-08" "1970-01-14"
##  [4] "1970-01-14" "1970-01-15" "1970-01-16"
##  [7] "1970-01-18" "1970-01-19" "1970-02-01"
## [10] "1970-02-01" "1970-02-06" "1970-02-09"
```

```
X <- xts( some.data , order.by = my.time.stamps)
X

##                   [,1]
## 1970-01-05 -0.56047565
## 1970-01-08 -0.23017749
## 1970-01-14  1.55870831
## 1970-01-14  0.07050839
## 1970-01-15  0.12928774
## 1970-01-16  1.71506499
## 1970-01-18  0.46091621
## 1970-01-19 -1.26506123
## 1970-02-01 -0.68685285
## 1970-02-01 -0.44566197
## 1970-02-06  1.22408180
## 1970-02-09  0.35981383

str(X)

## An 'xts' object on 1970-01-05/1970-02-09 containing:
##   Data: num [1:12, 1] -0.5605 -0.2302 1.5587 0.0705 0.1293 ...
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
##  NULL
```

The function `as.Date` was used to transform the above random times into dates.[7] Objects of class `zoo` and `ts` can be converted into objects of class `xts` only if the indexes are true time/class objects or if an additional argument `order.by` is specified appropriately. For example

```
X.ts <- ts(some.data, start = c(1964, 2), frequency = 12)
X.ts

##              Jan         Feb         Mar         Apr
## 1964             -0.56047565 -0.23017749  1.55870831
## 1965  0.35981383
##              May         Jun         Jul         Aug
## 1964  0.07050839  0.12928774  1.71506499  0.46091621
## 1965
##              Sep         Oct         Nov         Dec
## 1964 -1.26506123 -0.68685285 -0.44566197  1.22408180
## 1965

X.zoo <- as.zoo(X.ts)
X.zoo

##    Feb 1964    Mar 1964    Apr 1964    May 1964
## -0.56047565 -0.23017749  1.55870831  0.07050839
##    Jun 1964    Jul 1964    Aug 1964    Sep 1964
##  0.12928774  1.71506499  0.46091621 -1.26506123
##    Oct 1964    Nov 1964    Dec 1964    Jan 1965
## -0.68685285 -0.44566197  1.22408180  0.35981383
```

[7]This way of describing dates is called Unix or POSIX or epoch time. Each date is represented as the number of seconds elapsed since 1 January 1970 at midnight in Coordinated Universal Time (UTC).

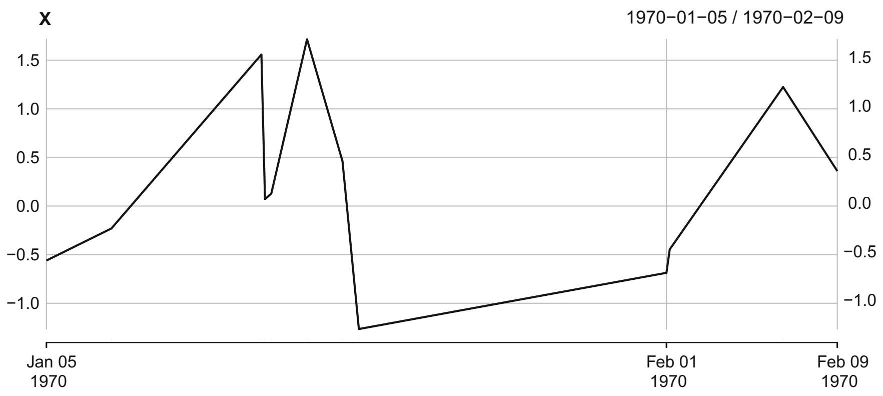


Fig. 1.13 The `plot` method for `xts` objects clearly shows the irregular frequency and missing data of a time series

```
X.xts <- as.xts(X.ts)
X.xts

##                  [,1]
## Feb 1964 -0.56047565
## Mar 1964 -0.23017749
## Apr 1964  1.55870831
## May 1964  0.07050839
## Jun 1964  0.12928774
## Jul 1964  1.71506499
## Aug 1964  0.46091621
## Sep 1964 -1.26506123
## Oct 1964 -0.68685285
## Nov 1964 -0.44566197
## Dec 1964  1.22408180
## Jan 1965  0.35981383
```

and so forth. The package **xts** has its own `plot` method which is designed for real time stamps and irregular time series as shown by the plot in Fig. 1.13.

```
plot(X)
```

1.14.1.4 The Class `irts`

The **tseries** package (Trapletti and Hornik 2013) provides the class `irts`, where 'ir' stands for 'irregular'. Compared to `zoo`, the arguments of `irts` are reversed

```
require(tseries)
X <- irts( rtimes, some.data)
X

## 1970-01-01 00:00:04 GMT -0.5605
## 1970-01-01 00:00:07 GMT -0.2302
## 1970-01-01 00:00:13 GMT 1.559
## 1970-01-01 00:00:13 GMT 0.07051
## 1970-01-01 00:00:14 GMT 0.1293
## 1970-01-01 00:00:15 GMT 1.715
## 1970-01-01 00:00:17 GMT 0.4609
## 1970-01-01 00:00:18 GMT -1.265
## 1970-01-01 00:00:31 GMT -0.6869
## 1970-01-01 00:00:31 GMT -0.4457
## 1970-01-01 00:00:36 GMT 1.224
## 1970-01-01 00:00:39 GMT 0.3598

str(X)

## List of 2
##  $ time : POSIXt[1:12], format: "1970-01-01 01:00:04" ...
##  $ value: num [1:12] -0.5605 -0.2302 1.5587 0.0705 0.1293 ...
##  - attr(*, "class")= chr "irts"
```

The `POSIXt` and other time stamp and date types will be discussed in Sect. 1.14.2.

1.14.1.5 The Class `timeSeries`

The last class of this short review is called `timeSeries` and belongs to the package **timeSeries** (Wuertz and Chalabi 2013) of the suite `Rmetrics`. This package loads the **timeDate** package (Wuertz et al. 2013) discussed later in Sect. 1.14.2.

```
require(timeSeries)

## Loading required package: timeSeries
## Loading required package: timeDate
##
## Attaching package: 'timeSeries'
## The following object is masked from 'package:zoo':
##
##     time<-

X <- timeSeries( some.data,  my.time.stamps)
X

## GMT
##                            TS.1
## 1970-01-05 05:12:53 -0.56047565
## 1970-01-08 02:24:29 -0.23017749
## 1970-01-14 17:53:40  1.55870831
## 1970-01-14 21:41:02  0.07050839
## 1970-01-15 04:25:45  0.12928774
## 1970-01-16 18:24:33  1.71506499
## 1970-01-18 08:07:00  0.46091621
## 1970-01-19 01:32:55 -1.26506123
## 1970-02-01 16:41:49 -0.68685285
## 1970-02-01 20:11:43 -0.44566197
## 1970-02-06 20:46:30  1.22408180
## 1970-02-09 06:24:03  0.35981383
```

```
str(X)

## Time Series:
##   Name:               object
## Data Matrix:
##   Dimension:          12 1
##   Column Names:       TS.1
##   Row Names:          1970-01-05 05:12:53  ...  1970-02-09 06:24:03
## Positions:
##   Start:              1970-01-05 05:12:53
##   End:                1970-02-09 06:24:03
## With:
##   Format:             %Y-%m-%d %H:%M:%S
##   FinCenter:          GMT
##   Units:              TS.1
##   Title:              Time Series Object
##   Documentation:      Fri Feb  2 15:44:21 2018
```

1.14.2 How to Handle Real Time Stamps

Most of the time, the users download data from some providers which specifies time stamp in a proper format. Sometimes, after simulation, there is the need to attach correct time stamps. This section explains the basic concepts necessary to handle different time formats. The Portable Operating System Interface (POSIX) format is an IEEE standard adopted by many UNIX-like operating systems. The function `ISOdate` can be used to create a data object in this way

```
d <- ISOdate(2008,7,3)
d

## [1] "2008-07-03 12:00:00 GMT"
```

The arguments of the `ISOdate` function are as follows

```
args(ISOdate)
```

```
## function (year, month, day, hour = 12, min = 0, sec = 0, tz
## = "GMT")
## NULL
```

and they are self-explanatory. The most important one is the time zone argument `tz` which is set to `GMT` (Greenwich Mean Time) also known as Coordinated Universal Time (UTC). When the time zone is set in this way, then all time stamps `d` are in Greenwich local time. `CET` (Central European Time) corresponds to UTC+1, and it is the time zone for countries in central Europe (e.g. Italy, France, Spain, Germany) We can see that this object `d` is indeed a POSIX time, and in particular, it is of class `POSIXct` where `ct` stands for ‘calendar time’.

```
class(d)

## [1] "POSIXct" "POSIXt"
```

Objects of type `POSIXct` represent in practice number of seconds passed since 1970 in UTC. An alternative representation is called `POSIXlt` which is represented as a list

```
names(as.POSIXlt(d))

## NULL

unlist(as.POSIXlt(d))


##    sec    min   hour   mday    mon   year   wday   yday isdst
##      0      0     12      3      6    108      4    184     0
```

There exist coercing functions to transform dates from one format to the other, such as `as.POSIXlt` and `as.POSIXct`. Calendar information can be represented in different formats through the function `format`

```
format(d,"%a") # week day

## [1] "Thu"

format(d,"%A")

## [1] "Thursday"

format(d,"%b") # month

## [1] "Jul"

format(d,"%B")

## [1] "July"

format(d,"%c") # full date

## [1] "Thu Jul  3 12:00:00 2008"

format(d,"%D") # yy/dd/mm

## [1] "07/03/08"

format(d,"%T") # hh:mm:ss

## [1] "12:00:00"

format(d,"%A %B  %d %H:%M:%S %Y")

## [1] "Thursday July  03 12:00:00 2008"
```

```
format(d,"%A   %d/%m/%Y")

## [1] "Thursday   03/07/2008"

format(d,"%d/%m/%Y (%A)")

## [1] "03/07/2008 (Thursday)"
```

and so forth. For a more information on the date operator %, the reader should refer to the man page of the function `format`. Strings can also be converted into date objects through the function `strptime`

```
x <- c("10jan1962", "2feb1970", "11jul2011", "27jun1968")
strptime(x, "%d%b%Y")

## [1] "1962-01-10 CET"  "1970-02-02 CET"
## [3] "2011-07-11 CEST" "1968-06-27 CEST"
```

In this case, 'jan, feb, jul, jun' are interpreted correctly as January, February, July and June, but in different locales, e.g. Italian, 'jan' and 'jul' will not be understood by the system and hence `strptime` returns a `NA` date. The user should check his own environment before attempting such data manipulations. The `Sys.getlocale` and `Sys.setlocale` functions allow to set and check the current 'locale' setting. The next example temporarily sets the locale settings to Italian and then switches it back to UK English[8]:

```
Sys.getlocale()

## [1] "C/UTF-8/C/C/C/C"

Sys.setlocale("LC_ALL", "it_it")

## [1] "it_it/it_it/it_it/C/it_it/C"

strptime(x, "%d%b%Y")

## [1] "1962-01-10 CET"  "1970-02-02 CET"
## [3] "2011-07-11 CEST" "1968-06-27 CEST"

Sys.setlocale("LC_ALL", "en_GB")

## [1] "en_GB/en_GB/en_GB/C/en_GB/C"

strptime(x, "%d%b%Y")

## [1] "1962-01-10 CET"  "1970-02-02 CET"
## [3] "2011-07-11 CEST" "1968-06-27 CEST"
```

When data are created without any time specification, by default `ISOdate` uses 12am whilst `as.POSIXct` uses 12pm

[8]Note that this example is specific to version of OS X used by the authors of this book. It may give different behaviour on the reader's operating system.

```
format(ISOdate(2006,6,9),"%H:%M:%S")

## [1] "12:00:00"

format(as.POSIXct("2006-06-09"),"%H:%M:%S")

## [1] "00:00:00"
```

1.14.3 Dates Manipulation

The package **timeDate** (Wuertz et al. 2013) provides the function `holiday*` to extract the nonworking days of financial markets:

```
holidayNYSE()

## NewYork
## [1] [2018-01-01] [2018-01-15] [2018-02-19] [2018-03-30]
## [5] [2018-05-28] [2018-07-04] [2018-09-03] [2018-11-22]
## [9] [2018-12-25]

holidayNERC()

## Eastern
## [1] [2018-01-01] [2018-05-28] [2018-07-04] [2018-09-03]
## [5] [2018-11-22] [2018-12-25]
```

It is possible to make calculations with times such as the following:

```
ISOdate(2006,7,10) - ISOdate(2005, 3, 1)

## Time difference of 496 days
```

or, via `timeDate`, we can write

```
my.dates <- timeDate(c("2001-01-09", "2001-02-25"))
diff(my.dates)

## Time difference of 47 days
```

To synchronize data coming from different financial markets, one should take care of time zones. The package **timeDate** provides the function `listFinCenter` which allows to identify financial markets by name:

```
listFinCenter("America*")[1:50]

##  [1] "America/Adak"
##  [2] "America/Anchorage"
##  [3] "America/Anguilla"
##  [4] "America/Antigua"
##  [5] "America/Araguaina"
##  [6] "America/Argentina/Buenos_Aires"
##  [7] "America/Argentina/Catamarca"
##  [8] "America/Argentina/Cordoba"
##  [9] "America/Argentina/Jujuy"
## [10] "America/Argentina/La_Rioja"
## [11] "America/Argentina/Mendoza"
## [12] "America/Argentina/Rio_Gallegos"
## [13] "America/Argentina/San_Juan"
## [14] "America/Argentina/Tucuman"
## [15] "America/Argentina/Ushuaia"
## [16] "America/Aruba"
## [17] "America/Asuncion"
## [18] "America/Atikokan"
## [19] "America/Bahia"
## [20] "America/Barbados"
## [21] "America/Belem"
## [22] "America/Belize"
## [23] "America/Blanc-Sablon"
## [24] "America/Boa_Vista"
## [25] "America/Bogota"
## [26] "America/Boise"
## [27] "America/Cambridge_Bay"
## [28] "America/Campo_Grande"
## [29] "America/Cancun"
## [30] "America/Caracas"
## [31] "America/Cayenne"
## [32] "America/Cayman"
## [33] "America/Chicago"
## [34] "America/Chihuahua"
## [35] "America/Costa_Rica"
## [36] "America/Cuiaba"
## [37] "America/Curacao"
## [38] "America/Danmarkshavn"
## [39] "America/Dawson"
## [40] "America/Dawson_Creek"
## [41] "America/Denver"
## [42] "America/Detroit"
## [43] "America/Dominica"
## [44] "America/Edmonton"
## [45] "America/Eirunepe"
## [46] "America/El_Salvador"
## [47] "America/Fortaleza"
## [48] "America/Glace_Bay"
## [49] "America/Godthab"
## [50] "America/Goose_Bay"
```

and this information can be used to handle dates

```
dA <- timeDate("2011-02-05", Fin="Europe/Zurich")
dB <- timeDate("2016-01-22", Fin="America/Chicago")
dA

## Europe/Zurich
## [1] [2011-02-05 01:00:00]


dB
```

```
## America/Chicago
## [1] [2016-01-21 18:00:00]
```

For further informations about date/time manipulation, a suggested reading is the time/date FAQ (Wuertz et al. 2013) ebook.

1.14.4 Using Dates to Index Time Series

In this section, we will focus only on the classes `zoo`, `xts` and `timeDate`. Let us create some random data and define some string dates[9]:

```
set.seed(123)
mydata <- rnorm(9)
chardata <- sprintf("2010-0%s-01", 9:1)
chardata

## [1] "2010-09-01" "2010-08-01" "2010-07-01" "2010-06-01"
## [5] "2010-05-01" "2010-04-01" "2010-03-01" "2010-02-01"
## [9] "2010-01-01"
```

then generate the corresponding objects with the different classes:

```
X1 <- zoo(mydata, as.Date(chardata))
X2 <- xts(mydata, as.Date(chardata))
X3 <- timeSeries(mydata, chardata)
```

and let us check how these objects look like

```
X1

##  2010-01-01  2010-02-01  2010-03-01  2010-04-01
## -0.68685285 -1.26506123  0.46091621  1.71506499
##  2010-05-01  2010-06-01  2010-07-01  2010-08-01
##  0.12928774  0.07050839  1.55870831 -0.23017749
##  2010-09-01
## -0.56047565


X2

##                   [,1]
## 2010-01-01 -0.68685285
## 2010-02-01 -1.26506123
## 2010-03-01  0.46091621
## 2010-04-01  1.71506499
## 2010-05-01  0.12928774
```

[9]For the use of the `sprintf` command, please check the help page typing `help("sprintf")` in the R Console. Here, we used `sprintf` with `%s` to convert integer numbers like 1, 2 and 3 into string, i.e. ‘1’, ‘2’ and ‘3’.

```
## 2010-06-01  0.07050839
## 2010-07-01  1.55870831
## 2010-08-01 -0.23017749
## 2010-09-01 -0.56047565
```

```
X3
```

```
## GMT
##                   TS.1
## 2010-09-01 -0.56047565
## 2010-08-01 -0.23017749
## 2010-07-01  1.55870831
## 2010-06-01  0.07050839
## 2010-05-01  0.12928774
## 2010-04-01  1.71506499
## 2010-03-01  0.46091621
## 2010-02-01 -1.26506123
## 2010-01-01 -0.68685285
```

Similarly, we should have used the following commands to create a `zoo` object:

```
zA <- zoo(mydata, as.POSIXct(chardata))
zB <- zoo(mydata, ISOdatetime(2016, 9:1, 1, 0,0,0))
zC <- zoo(mydata, ISOdate(2016, 9:1, 1, 0))
zA
```

```
##  2010-01-01  2010-02-01  2010-03-01  2010-04-01
## -0.68685285 -1.26506123  0.46091621  1.71506499
##  2010-05-01  2010-06-01  2010-07-01  2010-08-01
##  0.12928774  0.07050839  1.55870831 -0.23017749
##  2010-09-01
## -0.56047565
```

```
zB
```

```
##  2016-01-01  2016-02-01  2016-03-01  2016-04-01
## -0.68685285 -1.26506123  0.46091621  1.71506499
##  2016-05-01  2016-06-01  2016-07-01  2016-08-01
##  0.12928774  0.07050839  1.55870831 -0.23017749
##  2016-09-01
## -0.56047565
```

```
zC
```

```
##  2016-01-01  2016-02-01  2016-03-01  2016-04-01
## -0.68685285 -1.26506123  0.46091621  1.71506499
##  2016-05-01  2016-06-01  2016-07-01  2016-08-01
##  0.12928774  0.07050839  1.55870831 -0.23017749
##  2016-09-01
## -0.56047565
```

1.14.5 *Joining Two or More Time Series*

Suppose we have two parts of the same time series collected in different periods of time. It is possible to merge them by row, i.e. by date, using the `rbind` function. If the time indexes do not overlap, all classes perform in the same way:

```
set.seed(123)
val1 <- rnorm(9)
val2 <- rnorm(6)
mydate1 <- ISOdate(2016,1:9,1)
mydate2 <- ISOdate(2015,6:11,1)
Z1 <- zoo(val1, mydate1)
Z2 <- zoo(val2, mydate2)
rbind(Z1,Z2)

## 2015-06-01 14:00:00 2015-07-01 14:00:00
##         -0.44566197          1.22408180
## 2015-08-01 14:00:00 2015-09-01 14:00:00
##          0.35981383          0.40077145
## 2015-10-01 14:00:00 2015-11-01 13:00:00
##          0.11068272         -0.55584113
## 2016-01-01 13:00:00 2016-02-01 13:00:00
##         -0.56047565         -0.23017749
## 2016-03-01 13:00:00 2016-04-01 14:00:00
##          1.55870831          0.07050839
## 2016-05-01 14:00:00 2016-06-01 14:00:00
##          0.12928774          1.71506499
## 2016-07-01 14:00:00 2016-08-01 14:00:00
##          0.46091621         -1.26506123
## 2016-09-01 14:00:00
##         -0.68685285

X1 <- xts(val1, mydate1)
X2 <- xts(val2, mydate2)
rbind(X1,X2)

##                            [,1]
## 2015-06-01 12:00:00 -0.44566197
## 2015-07-01 12:00:00  1.22408180
## 2015-08-01 12:00:00  0.35981383
## 2015-09-01 12:00:00  0.40077145
## 2015-10-01 12:00:00  0.11068272
## 2015-11-01 12:00:00 -0.55584113
## 2016-01-01 12:00:00 -0.56047565
## 2016-02-01 12:00:00 -0.23017749
## 2016-03-01 12:00:00  1.55870831
## 2016-04-01 12:00:00  0.07050839
## 2016-05-01 12:00:00  0.12928774
## 2016-06-01 12:00:00  1.71506499
## 2016-07-01 12:00:00  0.46091621
## 2016-08-01 12:00:00 -1.26506123
## 2016-09-01 12:00:00 -0.68685285

W1 <- timeSeries(val1, mydate1)
W2 <- timeSeries(val2, mydate2)
rbind(W1,W2)
```

```
## GMT
##                         TS.1_TS.1
## 2016-01-01 12:00:00 -0.56047565
## 2016-02-01 12:00:00 -0.23017749
## 2016-03-01 12:00:00  1.55870831
## 2016-04-01 12:00:00  0.07050839
## 2016-05-01 12:00:00  0.12928774
## 2016-06-01 12:00:00  1.71506499
## 2016-07-01 12:00:00  0.46091621
## 2016-08-01 12:00:00 -1.26506123
## 2016-09-01 12:00:00 -0.68685285
## 2015-06-01 12:00:00 -0.44566197
## 2015-07-01 12:00:00  1.22408180
## 2015-08-01 12:00:00  0.35981383
## 2015-09-01 12:00:00  0.40077145
## 2015-10-01 12:00:00  0.11068272
## 2015-11-01 12:00:00 -0.55584113
```

Some classes, like `zoo`, will fail to do the binding in case of overlapping dates

```
mydate2 <- ISOdate(2016,4:9,1)
Z2 <- zoo(val2, mydate2)
```

The following code produces an error:

```
rbind(Z1,Z2)
```

```
## Error in rbind(deparse.level, ...) : indexes overlap
```

On the contrary, `timeSeries` and `xts` just duplicate the entries

```
X2 <- xts(val2, mydate2)
rbind(X1,X2)

##                            [,1]
## 2016-01-01 12:00:00 -0.56047565
## 2016-02-01 12:00:00 -0.23017749
## 2016-03-01 12:00:00  1.55870831
## 2016-04-01 12:00:00  0.07050839
## 2016-04-01 12:00:00 -0.44566197
## 2016-05-01 12:00:00  0.12928774
## 2016-05-01 12:00:00  1.22408180
## 2016-06-01 12:00:00  1.71506499
## 2016-06-01 12:00:00  0.35981383
## 2016-07-01 12:00:00  0.46091621
## 2016-07-01 12:00:00  0.40077145
## 2016-08-01 12:00:00 -1.26506123
## 2016-08-01 12:00:00  0.11068272
## 2016-09-01 12:00:00 -0.68685285
## 2016-09-01 12:00:00 -0.55584113

W2 <- timeSeries(val2, mydate2)
rbind(W1,W2)
```

```
## GMT
##                      TS.1_TS.1
## 2016-01-01 12:00:00 -0.56047565
## 2016-02-01 12:00:00 -0.23017749
## 2016-03-01 12:00:00  1.55870831
## 2016-04-01 12:00:00  0.07050839
## 2016-05-01 12:00:00  0.12928774
## 2016-06-01 12:00:00  1.71506499
## 2016-07-01 12:00:00  0.46091621
## 2016-08-01 12:00:00 -1.26506123
## 2016-09-01 12:00:00 -0.68685285
## 2016-04-01 12:00:00 -0.44566197
## 2016-05-01 12:00:00  1.22408180
## 2016-06-01 12:00:00  0.35981383
## 2016-07-01 12:00:00  0.40077145
## 2016-08-01 12:00:00  0.11068272
## 2016-09-01 12:00:00 -0.55584113
```

Merging of time series, similarly to what happens for `data.frame`, is also possible. Again, packages perform differently. Next code shows the different output generated by `zoo`, `xts` and `timeSeries`:

```
merge(Z1,Z2)

##                              Z1         Z2
## 2016-01-01 13:00:00 -0.56047565         NA
## 2016-02-01 13:00:00 -0.23017749         NA
## 2016-03-01 13:00:00  1.55870831         NA
## 2016-04-01 14:00:00  0.07050839 -0.4456620
## 2016-05-01 14:00:00  0.12928774  1.2240818
## 2016-06-01 14:00:00  1.71506499  0.3598138
## 2016-07-01 14:00:00  0.46091621  0.4007715
## 2016-08-01 14:00:00 -1.26506123  0.1106827
## 2016-09-01 14:00:00 -0.68685285 -0.5558411

merge(X1,X2)

##                              X1         X2
## 2016-01-01 12:00:00 -0.56047565         NA
## 2016-02-01 12:00:00 -0.23017749         NA
## 2016-03-01 12:00:00  1.55870831         NA
## 2016-04-01 12:00:00  0.07050839 -0.4456620
## 2016-05-01 12:00:00  0.12928774  1.2240818
## 2016-06-01 12:00:00  1.71506499  0.3598138
## 2016-07-01 12:00:00  0.46091621  0.4007715
## 2016-08-01 12:00:00 -1.26506123  0.1106827
## 2016-09-01 12:00:00 -0.68685285 -0.5558411
```

Generate a two-dimensional time series where `NA` appears when missing observations exist in each time series for a given time stamp. On the other side, `timeSeries` returns a one-dimensional objects with duplicates entries

```
merge(W1,W2)

## GMT
##                               TS.1
## 2016-01-01 12:00:00 -0.56047565
## 2016-02-01 12:00:00 -0.23017749
## 2016-03-01 12:00:00  1.55870831
## 2016-04-01 12:00:00 -0.44566197
## 2016-04-01 12:00:00  0.07050839
## 2016-05-01 12:00:00  0.12928774
## 2016-05-01 12:00:00  1.22408180
## 2016-06-01 12:00:00  0.35981383
## 2016-06-01 12:00:00  1.71506499
## 2016-07-01 12:00:00  0.40077145
## 2016-07-01 12:00:00  0.46091621
## 2016-08-01 12:00:00 -1.26506123
## 2016-08-01 12:00:00  0.11068272
## 2016-09-01 12:00:00 -0.68685285
## 2016-09-01 12:00:00 -0.55584113
```

To mimic the output of `zoo` and `xts`, it is necessary to set explicitly the name of the time series using the argument `units`

```
W2 <- timeSeries(val2, mydate2, units="MyData")
merge(W1,W2)

## GMT
##                               TS.1     MyData
## 2016-01-01 12:00:00 -0.56047565         NA
## 2016-02-01 12:00:00 -0.23017749         NA
## 2016-03-01 12:00:00  1.55870831         NA
## 2016-04-01 12:00:00  0.07050839 -0.4456620
## 2016-05-01 12:00:00  0.12928774  1.2240818
## 2016-06-01 12:00:00  1.71506499  0.3598138
## 2016-07-01 12:00:00  0.46091621  0.4007715
## 2016-08-01 12:00:00 -1.26506123  0.1106827
## 2016-09-01 12:00:00 -0.68685285 -0.5558411
```

Contrary to `xts` and `zoo`, for `timeSeries`, the arguments of `rbind` are not symmetric

```
mydate1 <- ISOdate(2016,1:9,1)
mydate2 <- ISOdate(2015,6:11,1)
W1 <- timeSeries(val1, mydate1)
W2 <- timeSeries(val2, mydate2)
```

The command

```
rbind(W1,W2)

## GMT
##                          TS.1_TS.1
## 2016-01-01 12:00:00 -0.56047565
## 2016-02-01 12:00:00 -0.23017749
## 2016-03-01 12:00:00  1.55870831
## 2016-04-01 12:00:00  0.07050839
## 2016-05-01 12:00:00  0.12928774
## 2016-06-01 12:00:00  1.71506499
## 2016-07-01 12:00:00  0.46091621
## 2016-08-01 12:00:00 -1.26506123
## 2016-09-01 12:00:00 -0.68685285
## 2015-06-01 12:00:00 -0.44566197
## 2015-07-01 12:00:00  1.22408180
## 2015-08-01 12:00:00  0.35981383
## 2015-09-01 12:00:00  0.40077145
## 2015-10-01 12:00:00  0.11068272
## 2015-11-01 12:00:00 -0.55584113
```

produces a different output than

```
rbind(W2,W1)

## GMT
##                       TS.1_TS.1
## 2015-06-01 12:00:00 -0.44566197
## 2015-07-01 12:00:00  1.22408180
## 2015-08-01 12:00:00  0.35981383
## 2015-09-01 12:00:00  0.40077145
## 2015-10-01 12:00:00  0.11068272
## 2015-11-01 12:00:00 -0.55584113
## 2016-01-01 12:00:00 -0.56047565
## 2016-02-01 12:00:00 -0.23017749
## 2016-03-01 12:00:00  1.55870831
## 2016-04-01 12:00:00  0.07050839
## 2016-05-01 12:00:00  0.12928774
## 2016-06-01 12:00:00  1.71506499
## 2016-07-01 12:00:00  0.46091621
## 2016-08-01 12:00:00 -1.26506123
## 2016-09-01 12:00:00 -0.68685285
```

The two functions `sort` and `rev`: can be used to sort the time series according to dates

```
sort( rbind(W2,W1) )

## GMT
##                       TS.1_TS.1
## 2015-06-01 12:00:00 -0.44566197
## 2015-07-01 12:00:00  1.22408180
## 2015-08-01 12:00:00  0.35981383
## 2015-09-01 12:00:00  0.40077145
## 2015-10-01 12:00:00  0.11068272
## 2015-11-01 12:00:00 -0.55584113
## 2016-01-01 12:00:00 -0.56047565
## 2016-02-01 12:00:00 -0.23017749
## 2016-03-01 12:00:00  1.55870831
## 2016-04-01 12:00:00  0.07050839
## 2016-05-01 12:00:00  0.12928774
## 2016-06-01 12:00:00  1.71506499
## 2016-07-01 12:00:00  0.46091621
## 2016-08-01 12:00:00 -1.26506123
## 2016-09-01 12:00:00 -0.68685285

sort( rbind(W2,W1), decr=TRUE)
```

```
## GMT
##                          TS.1_TS.1
## 2016-09-01 12:00:00 -0.68685285
## 2016-08-01 12:00:00 -1.26506123
## 2016-07-01 12:00:00  0.46091621
## 2016-06-01 12:00:00  1.71506499
## 2016-05-01 12:00:00  0.12928774
## 2016-04-01 12:00:00  0.07050839
## 2016-03-01 12:00:00  1.55870831
## 2016-02-01 12:00:00 -0.23017749
## 2016-01-01 12:00:00 -0.56047565
## 2015-11-01 12:00:00 -0.55584113
## 2015-10-01 12:00:00  0.11068272
## 2015-09-01 12:00:00  0.40077145
## 2015-08-01 12:00:00  0.35981383
## 2015-07-01 12:00:00  1.22408180
## 2015-06-01 12:00:00 -0.44566197
```

or to revert the time stamps

```
W2

## GMT
##                              TS.1
## 2015-06-01 12:00:00 -0.4456620
## 2015-07-01 12:00:00  1.2240818
## 2015-08-01 12:00:00  0.3598138
## 2015-09-01 12:00:00  0.4007715
## 2015-10-01 12:00:00  0.1106827
## 2015-11-01 12:00:00 -0.5558411

rev(W2)

## GMT
##                              TS.1
## 2015-11-01 12:00:00 -0.5558411
## 2015-10-01 12:00:00  0.1106827
## 2015-09-01 12:00:00  0.4007715
## 2015-08-01 12:00:00  0.3598138
## 2015-07-01 12:00:00  1.2240818
## 2015-06-01 12:00:00 -0.4456620
```

1.14.6 Subsetting a Time Series

Selection of elements in time series object is similar to subsetting for `matrix`. Let us consider the data set `quotes` available in package **sde** (Iacus 2008)

```
require(sde)
data(quotes)
str(quotes)
```

```
## 'zoo' series from 2006-01-03 to 2007-12-31
## Data: num [1:520, 1:20] 26.8 27 27 26.9 26.9 ...
## - attr(*, "dimnames")=List of 2
## ..$ : chr [1:520] "2006-01-03" "2006-01-04" "2006-01-05"
## "2006-01-06" ...
## ..$ : chr [1:20] "MSOFT" "AMD" "DELL" "INTEL" ...
## Index: Date[1:520], format: "2006-01-03" "2006-01-04"
## "2006-01-05" ...
```

We can see that the `Data` slot consists of a matrix with attributes for `colnames` and `rownames`, respectively, the time series names and time stamps. We can access the elements of the time series using indexes and/or names

```
quotes[2,2:4]

##              AMD  DELL INTEL
## 2006-01-04 32.56 30.76 25.91

quotes[10:20,"INTEL"]



## 2006-01-16 2006-01-17 2006-01-18 2006-01-19 2006-01-20
##    25.5875    25.5200    22.6000    22.4000    21.7600
## 2006-01-23 2006-01-24 2006-01-25 2006-01-26 2006-01-27
##    21.3500    21.2800    21.2100    21.4900    21.6700
## 2006-01-30
##    21.6500
```

but we can use `$` as for `data.frame`

```
quotes$INTEL[10:20]

## 2006-01-16 2006-01-17 2006-01-18 2006-01-19 2006-01-20
##    25.5875    25.5200    22.6000    22.4000    21.7600
## 2006-01-23 2006-01-24 2006-01-25 2006-01-26 2006-01-27
##    21.3500    21.2800    21.2100    21.4900    21.6700
## 2006-01-30
##    21.6500
```

or by dates

```
mydate <- as.Date(sprintf("2006-08-%.2d",20:10))
mydate

##  [1] "2006-08-20" "2006-08-19" "2006-08-18"
##  [4] "2006-08-17" "2006-08-16" "2006-08-15"
##  [7] "2006-08-14" "2006-08-13" "2006-08-12"
## [10] "2006-08-11" "2006-08-10"

quotes[mydate, 5:9]

##               HP  SONY  MOTO NOKIA    EA
## 2006-08-10 33.01 44.31 23.05 19.95 48.48
## 2006-08-11 33.05 44.36 22.69 19.63 49.88
## 2006-08-14 33.29 44.81 23.02 19.85 48.92
## 2006-08-15 33.99 45.43 23.54 20.73 50.82
## 2006-08-16 34.43 45.06 23.81 21.34 51.64
## 2006-08-17 35.15 44.97 23.72 21.29 51.59
## 2006-08-18 35.52 46.09 23.80 21.35 51.29
```

Missing dates are automatically skipped. In the above, we have used `sprintf` with `%.2d` to convert integer numbers like 1, 2 and 3 into a two digits string, i.e. ‘01’, ‘02’ and ‘03’. Of course, because dates are objects, we can do selection on dates like this:

```
initial <- as.Date("2007-05-15")
terminal <- as.Date("2007-05-21")
quotes[ (time(quotes) >= initial) & (time(quotes)<= terminal), 4:9]

##            INTEL    HP  SONY  MOTO NOKIA    EA
## 2007-05-15 22.01 44.75 52.70 17.92 26.31 48.70
## 2007-05-16 22.18 45.21 55.85 18.22 26.66 49.14
## 2007-05-17 22.23 44.87 55.05 18.60 26.57 48.41
## 2007-05-18 22.70 44.58 55.56 18.79 27.04 48.58
## 2007-05-21 22.63 45.22 57.38 18.90 26.97 49.21
```

Package **xts** is more flexible than others in date subsetting. Let use consider again the IBM data.

```
getSymbols("IBM", from="2015-01-01", to = "2016-12-31")

## [1] "IBM"

str(IBM)

## An 'xts' object on 2015-01-02/2016-12-30 containing:
##   Data: num [1:504, 1:6] 180 180 178 175 174 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:6] "IBM.Open" "IBM.High" "IBM.Low" "IBM.Close" ...
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
## List of 2
##  $ src    : chr "yahoo"
##  $ updated: POSIXct[1:1], format: "2018-02-02 15:44:23"
```

We can subset this time series with

```
IBM["2015-01","IBM.Close"]

##            IBM.Close
## 2015-01-02    162.06
## 2015-01-05    159.51
## 2015-01-06    156.07
## 2015-01-07    155.05
## 2015-01-08    158.42
## 2015-01-09    159.11
## 2015-01-12    156.44
## 2015-01-13    156.81
## 2015-01-14    155.80
## 2015-01-15    154.57
## 2015-01-16    157.14
## 2015-01-20    156.95
## 2015-01-21    152.09
## 2015-01-22    155.39
## 2015-01-23    155.87
## 2015-01-26    156.36
## 2015-01-27    153.67
## 2015-01-28    151.55
## 2015-01-29    155.48
## 2015-01-30    153.31
```

to extract only data for January 2015. Or

```
IBM["2016-02-11/2016-03-05","IBM.Close"]

##            IBM.Close
## 2016-02-11    117.85
## 2016-02-12    121.04
## 2016-02-16    122.74
## 2016-02-17    126.10
## 2016-02-18    132.45
## 2016-02-19    133.08
## 2016-02-22    133.77
## 2016-02-23    132.40
## 2016-02-24    132.80
## 2016-02-25    134.50
## 2016-02-26    132.03
## 2016-02-29    131.03
## 2016-03-01    134.37
## 2016-03-02    136.30
## 2016-03-03    137.80
## 2016-03-04    137.80
```

to subset on a interval, or even

```
IBM["/2015-02-11","IBM.Close"]

##            IBM.Close
## 2015-01-02    162.06
## 2015-01-05    159.51
## 2015-01-06    156.07
## 2015-01-07    155.05
## 2015-01-08    158.42
## 2015-01-09    159.11
## 2015-01-12    156.44
## 2015-01-13    156.81
## 2015-01-14    155.80
## 2015-01-15    154.57
## 2015-01-16    157.14
## 2015-01-20    156.95
## 2015-01-21    152.09
## 2015-01-22    155.39
## 2015-01-23    155.87
## 2015-01-26    156.36
## 2015-01-27    153.67
## 2015-01-28    151.55
## 2015-01-29    155.48
## 2015-01-30    153.31
## 2015-02-02    154.66
## 2015-02-03    158.47
## 2015-02-04    156.96
## 2015-02-05    157.91
## 2015-02-06    156.72
## 2015-02-09    155.75
## 2015-02-10    158.56
## 2015-02-11    158.20
```

to select dates up to a given date.

1.15 Miscellanea

1.15.1 From Yuima to LaTeX

The R method `toLatex` has been extended to most of the models in the **yuima** package. For example, consider again this parametric model

```
mod2 <- setModel(drift = "-mu*x", diffusion = "1/(1+x^gamma)")
mod2

##
## Diffusion process
## Number of equations: 1
## Number of Wiener noises: 1
## Parametric model with 2 parameters
```

We can obtain the LaTeX code ready to be inserted in a scientific paper by invoking `toLatex`

```
toLatex(mod2)

## %%% Copy and paste the following output in your LaTeX file
## $$
##  dx
##  =
##  -\mu  \cdot  x dt
##  +
##  1/(1 + x^\gamma )
##  dW1
## $$
## $$
##  x(0)=0
## $$
```

and the above LaTeX code will show, after typesetting, as follows

$$dx = -\mu \cdot x dt + 1/(1 + x^\gamma) dW1$$

$$x(0) = 0$$

For multidimensional models, **yuima** uses the matrix notation

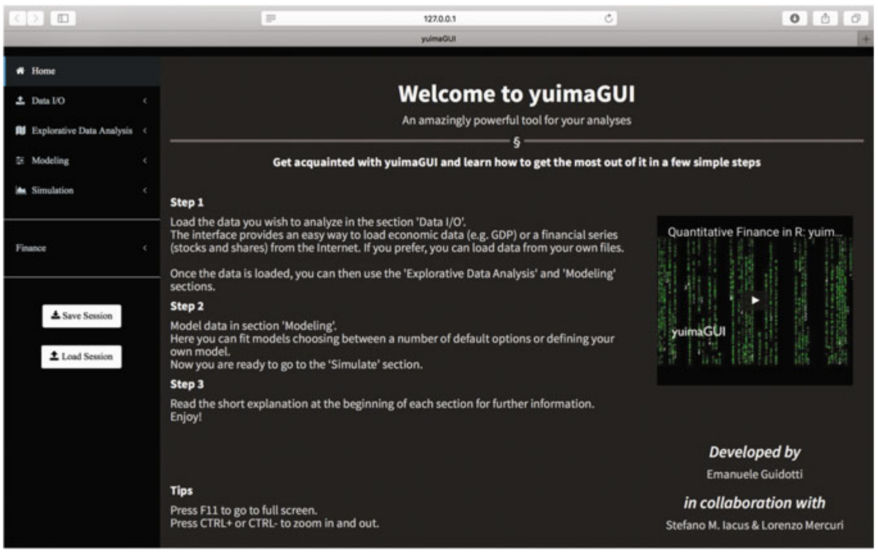
```
sol <- c("x1","x2") # variable for numerical solution
b <- c("-theta*x1","-x1-gamma*x2") # drift vector
s <- matrix(c("1","x1","0","delta","x2","0"),2,3) # diff. mat.
mymod <- setModel(drift = b, diffusion = s, solve.variable = sol)
```

and then `toLatex(mymod)` will produce the following output

$$\begin{pmatrix} dx1 \\ dx2 \end{pmatrix} = \begin{pmatrix} -\theta \cdot x1 \\ -x1 - \gamma \cdot x2 \end{pmatrix} dt + \begin{bmatrix} 1 & 0 & x2 \\ x1 & \delta & 0 \end{bmatrix} \begin{pmatrix} dW1 \\ dW2 \\ dW3 \end{pmatrix}$$

$$\begin{pmatrix} x1(0) = 0 \\ x2(0) = 0 \end{pmatrix}$$

which can be easily adjusted to one's notation. The reader can try with the different **yuima** models presented in this book.



Welcome to yuimaGUI

An amazingly powerful tool for your analyses

§

Get acquainted with yuimaGUI and learn how to get the most out of it in a few simple steps

Step 1

Load the data you wish to analyze in the section 'Data I/O'.
The interface provides an easy way to load economic data (e.g. GDP) or a financial series (stocks and shares) from the Internet. If you prefer, you can load data from your own files.

Once the data is loaded, you can then use the 'Explorative Data Analysis' and 'Modeling' sections.

Step 2

Model data in section 'Modeling'.
Here you can fit models choosing between a number of default options or defining your own model.
Now you are ready to go to the 'Simulate' section.

Step 3

Read the short explanation at the beginning of each section for further information.
Enjoy!

Tips

Press F11 to go to full screen.
Press CTRL+ or CTRL- to zoom in and out.

Developed by

Emanuele Guidotti

in collaboration with

Stefano M. Iacus & Lorenzo Mercuri

Fig. 1.14 How the graphical user interface of **yuima** appears thanks to the **yuimaGUI** package

1.15.2 The Yuima GUI

There exists also a graphical user interface for **yuima** which allows for easy practical analysis of real data using most of the functionalities of the package with interactive graphics. To use this `yuimaGUI`, this is the name of the GUI for **yuima**, and one need to install at first the package **yuimaGUI** with

```
install.packages("yuimaGUI")
```

This package depends on other packages, so all the other packages should be installed first. Once the package is ready for use, the only commands to type in the R console are the following

```
library(yuimaGUI)
yuimaGUI()
```

The `yuimaGUI` is a Web-based interface as shown in Fig. 1.14. More information on how to use the GUI and its functionalities are provided by the GUI itself.

Part II
Models and Inference

Chapter 2
Diffusion Processes

2.1 Model Specification

Let $\{X_t, t \geq 0\}$ be a one-dimensional diffusion process defined by the stochastic differential equation

$$\mathrm{d}X_t = a(t, X_t, \theta)dt + b(t, X_t, \theta)\mathrm{d}W_t, \tag{2.1}$$

with an initial value X_0, where W_t is a standard Brownian motion. We assume that sufficient regularity conditions hold for the drift function $a(\cdot)$ and the diffusion coefficient $b(\cdot)$ as well as for the initial condition X_0 so that a solution of (2.1) exists. The functions $a(\cdot)$ and $b(\cdot)$ may or may not depend on t or a statistical parameter $\theta \in \Theta \subset R^d, d \geq 1$.

In Sect. 1.9.1 we have seen how to specify the simple model

$$\mathrm{d}X_t = -3X_t\mathrm{d}t + \frac{1}{1+X_t^2}\mathrm{d}W_t$$

with some constant initial condition X_0. It is also possible to specify a random initial condition for the process via the argument xinit as follows

```
mod1 <- setModel(drift = "-3*x", diffusion = "1/(1+x^2)",
 xinit="rnorm(1)")
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 : list()
## ..@ measure : list()
## ..@ measure.type : chr(0)
## ..@ parameter :Formal class 'model.parameter' [package
```

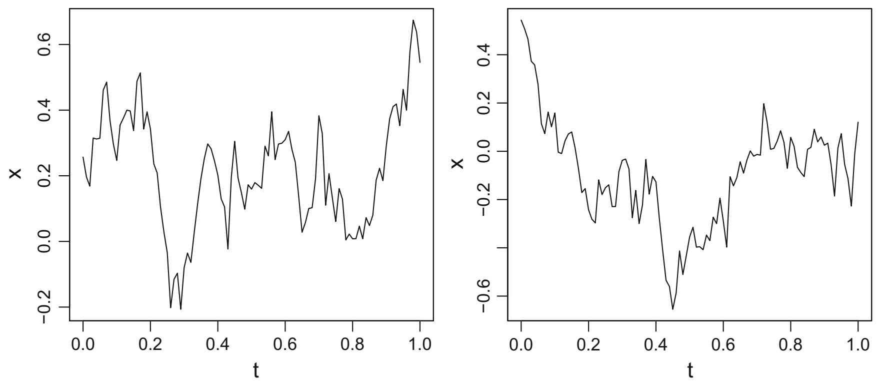


Fig. 2.1 A `yuima.model` with random initial condition

```
## "yuima"] with 7 slots
## .. .. ..@ all : chr(0)
## .. .. ..@ common : chr(0)
## .. .. ..@ diffusion: chr(0)
## .. .. ..@ drift : chr(0)
## .. .. ..@ jump : chr(0)
## .. .. ..@ measure : chr(0)
## .. .. ..@ xinit : 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 : expression((rnorm(1)))
## ..@ J.flag : logi FALSE
```

As we can see, the `xinit` slot of the `yuima.model` is set properly. If we simulate trajectories from this model, the initial condition will be generated as well according to the specified random distribution:

```
set.seed(123)
x1 <- simulate(mod1)
x2 <- simulate(mod1)
par(mfrow=c(1,2))
plot(x1)
plot(x2)
```

where the initial condition is taken from the standard Gaussian distribution (see Fig. 2.1).
The initial condition can also be parametrized as

```
mod2 <- setModel(drift = "-3*x", diffusion = "1/(1+x^2)",
 xinit="rnorm(1, mean=mu)")
mod2
```

```
str(mod2)
```

```
## 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 : list()
## ..@ measure : list()
## ..@ measure.type : chr(0)
## ..@ parameter :Formal class 'model.parameter' [package
## "yuima"] with 7 slots
## .. .. ..@ all : chr "mu"
## .. .. ..@ common : chr(0)
## .. .. ..@ diffusion: chr(0)
## .. .. ..@ drift : chr(0)
## .. .. ..@ jump : chr(0)
## .. .. ..@ measure : chr(0)
## .. .. ..@ xinit : chr "mu"
## ..@ state.variable : chr "x"
## ..@ jump.variable : chr(0)
## ..@ time.variable : chr "t"
## ..@ noise.number : num 1
## ..@ equation.number: int 1
## ..@ dimension : int [1:6] 1 0 0 0 0 0
## ..@ solve.variable : chr "x"
## ..@ xinit : expression((rnorm(1, mean = mu)))
## ..@ J.flag : logi FALSE
```

and in this case, the parameter mu in the initial condition $X_0 \sim N(\mu, 1)$ becomes a parameter in the model. Thus, for example, in order to simulate a trajectory, we need to specify this parameter as well, i.e.

```
x <- simulate(mod2, true.par=list(mu=1))
```

The initial condition can also be overridden at the time of the simulate command either putting a deterministic initial value or replacing a deterministic or random initial condition with another random condition (see Fig. 2.2)

```
mod1 <- setModel(drift = "-3*x", diffusion = "1/(1+x^2)")
set.seed(123)
x1 <- simulate(mod1, xinit=1)
x2 <- simulate(mod1, xinit=expression(rnorm(1)))
x3 <- simulate(mod2, xinit=3)
par(mfrow=c(1,3))
plot(x1, main="mod1, xinit=1")
plot(x2, main="mod1, xinit=expression(rnorm(1))")
plot(x3, main="mod2, xinit=3")
par(mfrow=c(1,1))
```

We present here a list of well-known models which can be easily specified in **yuima**.

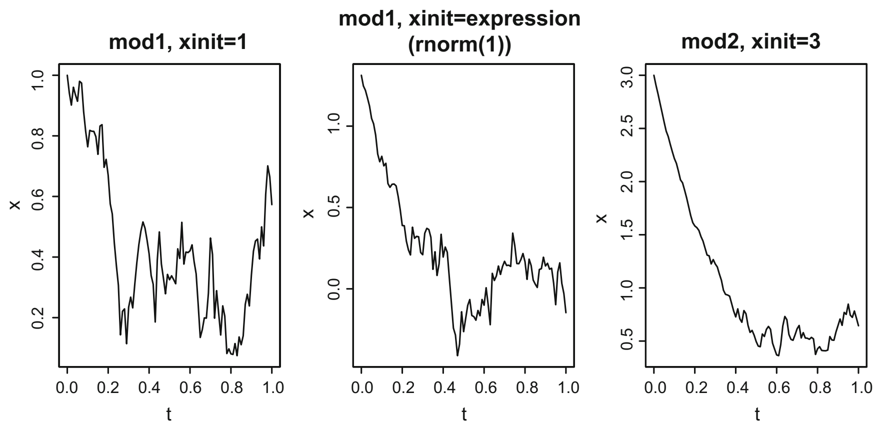


Fig. 2.2 Random versus deterministic initial condition

2.1.1 *Ornstein–Uhlenbeck (OU)*

The simple Ornstein–Uhlenbeck (Uhlenbeck and Ornstein 1930) has the following stochastic differential equation:

$$\mathrm{d}X_t = -\theta X_t \mathrm{d}t + \mathrm{d}W_t, \quad X_0 = x_0,$$

and $\theta > 0$ ensures stationarity. This can be coded as

```
ou <- setModel(drift="-theta*x", diffusion=1)
```

2.1.2 *Geometric Brownian Motion (gBm)*

The geometric Brownian motion (Osborne 1959) has the following stochastic differential equation:

$$\mathrm{d}X_t = \mu X_t \mathrm{d}t + \sigma X_t \mathrm{d}W_t, \quad X_0 = x_0,$$

and $\sigma > 0$. Its solution is the exponential of a linear transform of the Brownian motion and hence always nonnegative. This can be coded as

```
gBm <- setModel(drift="mu*x", diffusion="sigma*x")
```

This process is the building block of the Black and Scholes (1973) and Merton (1973a) theory for option pricing.

2.1.3 Vasicek Model (VAS)

The Vasicek (1977) process is the unique solution to the following stochastic differential equation

$$dX_t = (\theta_1 - \theta_2 X_t)dt + \theta_3 dW_t$$

with $\theta_3 \in \mathbb{R}_+$ and $\theta_1, \theta_2 \in \mathbb{R}$. This is essentially the Ornstein–Uhlenbeck process with the mean reverting property. This process can be coded as follows:

```
vasicek <- setModel(drift="theta1-theta2*x", diffusion="theta3")
```

2.1.4 Constant Elasticity of Variance (CEV)

The constant elasticity of variance (CEV) process introduced in finance in option pricing (see Schroder 1989; Beckers 1980; Cox 1996) is a solution of the stochastic differential equation:

$$dX_t = \mu X_t dt + \sigma X_t^\gamma dW_t.$$

This process is quite useful in modelling a skewed implied volatility. In particular, for $\gamma < 1$, the skewness is negative, and for $\gamma > 1$ the skewness is positive. For $\gamma = 1$, the CEV process is a particular version of the geometric Brownian motion. The model can be coded as follows

```
cev <- setModel(drift="mu*x", diffusion="sigma*x^gamma")
```

2.1.5 Cox–Ingersoll–Ross Process (CIR)

This model was introduced by Feller as a model for population growth (see Feller 1951b, a) and became quite popular in finance after Cox, Ingersoll and Ross proposed it to model short-term interest rates (Cox et al. 1985). The process is a solution to this stochastic differential equation:

$$dX_t = (\theta_1 - \theta_2 X_t)dt + \theta_3 \sqrt{X_t} dW_t,$$

where $\theta_1, \theta_2, \theta_3 \in \mathbb{R}_+$. If $2\theta_1 > \theta_3^2$, the process is strictly positive, otherwise it is nonnegative, which means that it can reach the state 0. This can be coded as follows:

```
cir <- setModel(drift="theta1-theta2*x", diffusion="theta3*sqrt(x)")
```

Table 2.1 Family of CKLS process $\mathrm{d}X_t = (\theta_1 + \theta_2 X_t)\mathrm{d}t + \theta_3 X_t^{\theta_4}\mathrm{d}W_t$ and its embedded elements under different parametric specifications. In all cases, $\theta_3 > 0$

	θ_1	θ_2	θ_4	See
Merton	Any	0	0	Merton (1973b)
Vasicek or Ornstein–Uhlenbeck	Any	Any	0	Vasicek (1977)
CIR or square root process	Any	Any	1/2	Cox et al. (1985)
Dothan	0	0	1	Dothan (1978)
Geometric BM or Black and Scholes	0	Any	1	Black and Scholes (1973)
Brennan and Schwartz	Any	Any	1	Brennan and Schwartz (1980)
CIR VR	0	0	3/2	Cox et al. (1980)
CEV	0	Any	Any	Cox (1996)

2.1.6 *Chan–Karolyi–Longstaff–Sanders Process (CKLS)*

The Chan–Karolyi–Longstaff–Sanders (CKLS) family of models (see Chan et al. 1992) is a class of parametric stochastic differential equations widely used in many financial applications, in particular to model interest rates or asset prices. The CKLS process solves the stochastic differential equation

$$\mathrm{d}X_t = (\theta_1 + \theta_2 X_t)\mathrm{d}t + \theta_3 X_t^{\theta_4}\mathrm{d}W_t\,.$$

The CKLS model does not admit an explicit transition density unless $\theta_1 = 0$ or $\theta_4 = \frac{1}{2}$. It takes values in $(0, +\infty)$ if $\theta_1, \theta_2 > 0$, and $\theta_4 > \frac{1}{2}$. In all cases, θ_3 is assumed to be positive. This model is an extension of several other models as can be seen from Table 2.1. This model can be coded as follows:

```
ckls <- setModel(drift="theta1-theta2*x", diffusion="theta3*x^theta4")
```

2.1.7 *Hyperbolic Diffusion Processes*

The hyperbolic distribution has density

$$p(x;\alpha,\beta,\delta,\mu) = \frac{\kappa}{2\alpha\delta K_1(\delta\kappa)} \exp\left\{-\alpha\sqrt{\delta^2 + (x-\mu)^2} + \beta(x-\mu)\right\}, \quad x \in \mathbb{R}. \tag{2.2}$$

In this parametrization, $\mu \in \mathbb{R}$ is the location parameter, $\delta > 0$ is the scale parameter, β is a real parameter which controls the asymmetry around μ, $\alpha > |\beta| \geq 0$ is called the tail parameter, K_1 is the modified Bessel function of the third kind with index one

$$K_1(t) = \frac{1}{2}\int_0^\infty e^{-\frac{1}{2}t(x+x^{-1})}\mathrm{d}x$$

and $\kappa = \sqrt{\alpha^2 - \beta^2}$ (see also Sect. 4.10 for an extended treatment of the hyperbolic distribution and related processes). The tails of the distribution are exponentially decreasing with rate $\varphi = \beta + \alpha$ for $x \to -\infty$ and $\gamma = \alpha - \beta$ for $x \to +\infty$; thus, playing with the parameters α and β, it is possible to obtain different shapes including asymmetric and heavy-tail distributions. The name of the distribution comes from the fact that the graph of $\ln f(x)$ represents a hyperbola. The hyperbolic diffusion process introduced in Barndorff-Nielsen (1978) and later applied in finance by many authors (e.g., Küchler et al. 1999) has the following stochastic differential equation:

$$\mathrm{d}X_t = \frac{\sigma^2}{2}\left[\beta - \alpha\frac{X_t - \mu}{\sqrt{\delta^2 + (X_t - \mu)^2}}\right]\mathrm{d}t + \sigma\mathrm{d}W_t. \tag{2.3}$$

The name of this process comes from the fact that the invariant density of the process follows the hyperbolic distribution (2.2). Indeed, remind that for any ergodic diffusion of the type $\mathrm{d}X_t = b(X_t)\mathrm{d}t + \sigma(X_t)\mathrm{d}W_t$, the invariant law $\pi(x)$ has always the following form:

$$\pi(x) = \frac{m(x)}{M}$$

where

$$m(x) = \frac{1}{\sigma^2(x)s(x)}$$

is the speed measure, $M = \int m(x)\mathrm{d}x$ and

$$s(x) = \exp\left\{-2\int_{x_0}^{x}\frac{b(y)}{\sigma^2(y)}\mathrm{d}y\right\}$$

is the scale function and x_0 is any point in the state space of the process X_t. Hence, in this case, it is easy to write down these expressions as follows:

$$\begin{aligned}
s(x) &= \exp\left\{\int_0^x \left(\frac{\alpha(y-\mu)}{\sqrt{\delta^2+(y-\mu)^2}} - \beta\right) \mathrm{d}y\right\} \\
&\propto \exp\left\{\alpha\sqrt{\delta^2+(x-\mu)^2} - \beta x\right\} \\
m(x) &= \frac{1}{\sigma^2}\exp\left\{-\int_0^x \left(\frac{\alpha(y-\mu)}{\sqrt{\delta^2+(y-\mu)^2}} - \beta\right) \mathrm{d}y\right\} \\
&\propto \frac{1}{\sigma^2}\exp\left\{-\alpha\sqrt{\delta^2+(x-\mu)^2} + \beta x\right\}
\end{aligned}$$

Given that $m(\pm\infty)$ is finite because of the exponential decreasing tails, we have that $\pi(x) = p(x; \alpha, \beta, \delta, \mu)$ and the normalizing constant is $M = \frac{\kappa}{2\alpha\delta K_1(\delta\kappa)}$. Notice that this distribution is independent of σ. We can input into **yuima** this model quite easily as follows:

```
hyper1 <- setModel( diff="sigma",
 drift="(sigma^2/2)*(beta-alpha*((x-mu)/(sqrt(delta^2+(x-mu)^2))))")
```

In this case, the parameter σ is common to both drift and diffusion coefficients. We can look at the structure of the `parameter` slot of the `yuima.model`:

```
hyper1

##
## Diffusion process
## Number of equations: 1
## Number of Wiener noises: 1
## Parametric model with 5 parameters

str(hyper1@parameter)

## Formal class 'model.parameter' [package "yuima"] with 7 slots
##   ..@ all      : chr [1:5] "sigma" "beta" "alpha" "mu" ...
##   ..@ common   : chr "sigma"
##   ..@ diffusion: chr "sigma"
##   ..@ drift    : chr [1:5] "sigma" "beta" "alpha" "mu" ...
##   ..@ jump     : chr(0)
##   ..@ measure  : chr(0)
##   ..@ xinit    : chr(0)
```

With the idea of generalizing the geometric Brownian motion model to a richer class, Bibby and Sørensen (1997) proposed another type of stochastic differential equation whose solution has a stationary hyperbolic distribution. Let

$$S_t = e^{mt+X_t} \tag{2.4}$$

where

$$X_t = X_0 + \int_0^t v(X_s)\mathrm{d}W_s.$$

By Itô formula, it follows that the stochastic differential equation which represents the dynamics of (2.4) is as follows:

$$\mathrm{d}S_t = S_t\left\{\left[m + \frac{1}{2}v^2(\log S_t - mt)\right]\mathrm{d}t + v(\log S_t - mt)\mathrm{d}W_t\right\}. \tag{2.5}$$

Notice that, when $v(t)$ is constant, (2.5) is of the same type of the stochastic differential equation for the geometric Brownian motion of Sect. 2.1.2. If

$$v(x) = \sigma \exp\left\{\frac{1}{2}\left(\alpha\sqrt{\delta^2 + (x-\mu)^2} - \beta(x-\mu)\right)\right\}$$

the process of the drift-adjusted log-prices $X_t = \log S_t - mt$ satisfies the following stochastic differential equation:

$$\mathrm{d}X_t = \sigma \exp\left\{\frac{1}{2}\left(\alpha\sqrt{\delta^2 + (X_t-\mu)^2} - \beta(X_t-\mu)\right)\right\}\mathrm{d}W_t, \tag{2.6}$$

with the initial condition $X_0 = \log S_0$. It is easy to check that the process X_t has also a hyperbolic invariant distribution. Indeed, in this case $s(x) = 1$ and $m(x) \propto \frac{1}{\sigma^2}\exp\left\{-\alpha\sqrt{\delta^2 + (x-\mu)^2} + \beta x\right\}$. Moreover, the increments over a short time interval have thick tails while over long time intervals the distribution of its increments is almost hyperbolic. This model can be prepared for **yuima** as follows

```
hyper2 <- setModel(drift="0",
 diffusion = "sigma*exp(0.5*alpha*sqrt(delta^2+(x-mu)^2)-
  beta*(x-mu))")
```

and, as before, we can look at the slot `parameter` of the `yuima.model`:

```
hyper2

##
## Diffusion process, driftless
## Number of equations: 1
## Number of Wiener noises: 1
## Parametric model with 5 parameters

str(hyper2@parameter)

## Formal class 'model.parameter' [package "yuima"] with 7 slots
##   ..@ all      : chr [1:5] "sigma" "alpha" "delta" "mu" ...
##   ..@ common   : chr(0)
##   ..@ diffusion: chr [1:5] "sigma" "alpha" "delta" "mu" ...
##   ..@ drift    : chr(0)
##   ..@ jump     : chr(0)
##   ..@ measure  : chr(0)
##   ..@ xinit    : chr(0)
```

Further relations between the models in (2.6) and (2.3) and extensions to the class of the generalized hyperbolic diffusions can be found in Ryder (1999). Generalized hyperbolic processes will be introduced in Sect. 4.10.

2.2 More About Simulation

In Sect. 1.10 we have seen the basic options available for simulation. The basic simulation scheme used for multi- and unidimensional diffusion processes is the Euler–Maruyama scheme, which is based on the discretization of the stochastic differential equation

$$\mathrm{d}X_t = a(t, X_t)\mathrm{d}t + b(t, X_t)\mathrm{d}W_t$$

on a regular grid of times $t_i = i \cdot \Delta$, $i = 0, \ldots, n$, where Δ is a given time lag. For simplicity, we denote by $X_i = X(t_i)$, $i = 0, 1, \ldots, n$. Roughly speaking, the approximation is valid only at the points of the grid t_i and only if Δ is very small (Iacus 2008). From the previous stochastic differential equation, we can write

$$X_{t_{i+1}} - X_{t_i} = a(t_i, X_{t_i})\Delta + b(t_i, X_{t_i})\Delta W_i$$

where $\Delta W_i = W_{t_{i+1}} - W_{t_i} \sim \sqrt{\Delta}N(0, 1)$ is the sequence of independent increments of the Brownian motion. Thus, conditionally on the value X_{t_i}, we have

$$X_{t_{i+1}} = X_{t_i} + a(t_i, X_{t_i})\Delta + b(t_i, X_{t_i})\Delta W_i, \quad i = 1, \ldots, n-1,$$

and then

$$X_{t_{i+1}} \sim N(X_{t_i} + a(t_i, X_{t_i})\Delta, \Delta b^2(t_i, X_{t_i})), \quad i = 1, \ldots, n-1.$$

The method `simulate` automatically generates increments of a Wiener process, but the current implementation allows to input directly some other type of increments via the argument `increment.W`. If this argument is specified, then the Euler–Maruyama scheme uses this time series instead of generating new increments. This flexibility of the **yuima** simulator allows for different tasks, including replication of Monte Carlo experiments, using the same increments in different models, or even specifying increments which are not necessarily Gaussian. The name of the argument `increment.W` means only that those user-input increments replace the standard ΔW_i's in Euler–Maruyama scheme.

```
set.seed(123)
modA <- setModel(drift="-0.3*x", diffusion=1)
modB <- setModel(drift="0.3*x", diffusion=1)

## Set the model in an 'yuima' object with a sampling scheme.
Terminal <- 1
```

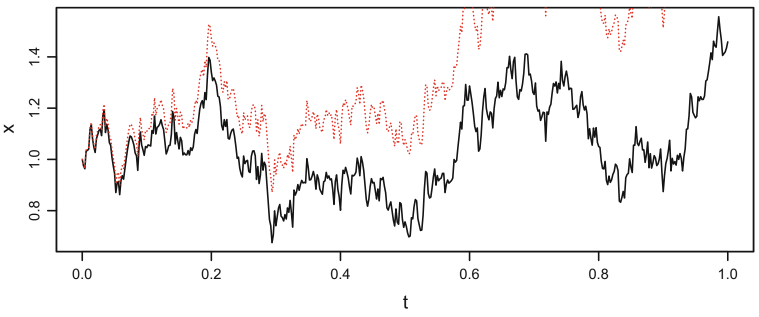


Fig. 2.3 Using the same increments to simulate two different models using argument `increment.W` in `simulate`

```r
n <- 500
mod.sampling <- setSampling(Terminal=Terminal, n=n)
yuima1 <- setYuima(model=modA, sampling=mod.sampling)
yuima2 <- setYuima(model=modB, sampling=mod.sampling)

##use original increment
delta <- Terminal/n
my.dW <- matrix( rnorm(n , 0, sqrt(delta)), nrow=1, ncol=n)

## Solve SDEs using Euler-Maruyama method.
y1 <- simulate(yuima1, xinit=1, increment.W=my.dW)
y2 <- simulate(yuima2, xinit=1, increment.W=my.dW)
```

and now `y1` and `y2` contain two different trajectories corresponding to `modA` and `modB`, respectively, as we see from Fig. 2.3 using the following R code

```r
plot(y1)
lines(get.zoo.data(y2)[[1]], col="red",lty=3)
```

The `simulate` method provides also space-discretized Euler–Maruyama method to solve SDEs. This is at the moment designed for 1 factor SDEs only; i.e., the situation when the driving Brownian motion W is one-dimensional. In this case, the discretization scheme $\{\tau_j\}$ is defined as

$$\tau_0 = 0, \quad \tau_{j+1} = \inf\{t > \tau_j ; |W_t - W_{\tau_j}| = \varepsilon\}$$

for each $j \geq 0$. Internally, `simulate` takes $\varepsilon^2 = T/n = \Delta_n$ which coincides with the mean of the interval $\tau_{j+1} - \tau_j$. This space-discretizing scheme is known to be three times 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 scheme for one-dimensional diffusion processes, as explained in Iacus (2008), will be implemented in the near future.

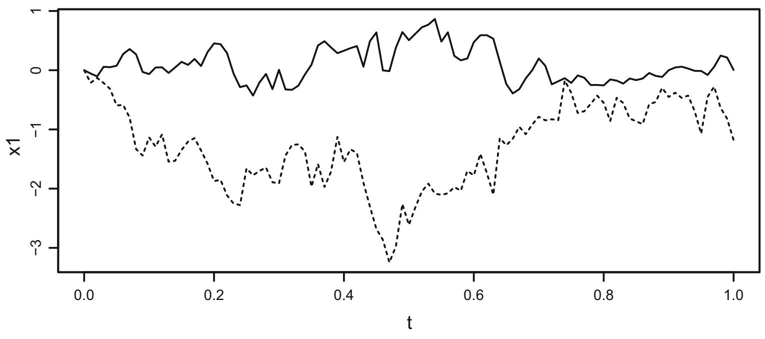


Fig. 2.4 A trajectory of the multidimensional SDE described in `mod3`

2.3 Multidimensional Processes

Next is an example of a system of two stochastic differential equations for the couple $(X_{1,t}, X_{2,t})$ driven by three independent Brownian motions $(W_{1,t}, W_{2,t}, W_{3,t})$

$$\begin{aligned} \mathrm{d}X_{1,t} &= -3X_{1,t}\mathrm{d}t + \mathrm{d}W_{1,t} + X_{2,t}\mathrm{d}W_{3,t} \\ \mathrm{d}X_{2,t} &= -(X_{1,t} + 2X_{2,t})\mathrm{d}t + X_{1,t}\mathrm{d}W_{1,t} + 3\mathrm{d}W_{2,t} \end{aligned}$$

This system has to be organized into matrix form with a vector of drift expressions and a diffusion matrix as follows:

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

For this system, it is now necessary to instruct **yuima** about the state variable on both the left-hand side of the equation and the right-hand side of the equation; i.e., one needs to specify also the `solve.variable` for the left-hand side of the SDE:

```
sol <- c("x1","x2") # variable for numerical solution
a <- c("-3*x1","-x1-2*x2")    # drift vector
b <- matrix(c("1","x1","0","3","x2","0"),2,3)  #  diffusion matrix
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 Fig. 2.4.

```
set.seed(123)
X <- simulate(mod3)
plot(X, plot.type="single",lty=1:2)
```

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

$$\begin{aligned} \mathrm{d}X_{2,t} &= -3X_{1,t}\mathrm{d}t + \mathrm{d}W_{1,t} + X_{2,t}\mathrm{d}W_{3,t} \\ \mathrm{d}X_{1,t} &= -(X_{1,t} + 2X_{2,t})\mathrm{d}t + X_{1,t}\mathrm{d}W_{1,t} + 3\mathrm{d}W_{2,t} \end{aligned}$$

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`.

It is also possible to specify more complex models like the following

$$\begin{cases} \mathrm{d}X_{1,t} = X_{2,t}\left|X_{1,t}\right|^{2/3}\mathrm{d}W_{1,t}, \\ \mathrm{d}X_{2,t} = g(t)\mathrm{d}X_{3,t}, \\ \mathrm{d}X_{3,t} = X_{3,t}(\mu\mathrm{d}t + \sigma(\rho\mathrm{d}W_{1,t} + \sqrt{1-\rho^2}\mathrm{d}W_{2,t})) \end{cases}$$

$$(X_{1,0}, X_{2,0}, X_{3,0}) = (1.0, 0.1, 1.0)$$

with $\mu = 0.1$, $\sigma = 0.2$, $\rho = -0.7$ and $g(t) = 0.4 + (0.1 + 0.2t)e^{-2t}$, for example, where $W = (W_1, W_2)$ is a two-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

$$X_t = X_0 + \int_0^t a(s, X_s)\mathrm{d}s + \int_0^t b(s, X_s)\mathrm{d}W_s$$

with

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

for $x = (x_1, x_2, x_3)$.

```
mu <- 0.1
sig <- 0.2
rho <- -0.7

g <- function(t) {0.4 + (0.1 + 0.2*t)* exp(-2*t)}


f1 <- function(t, x1, x2, x3) {
    ret <- 0
    if(x1 > 0 && x2 > 0) ret <- x2*exp(log(x1)*2/3)
    return(ret)
}

f2 <- function(t, x1, x2, x3) {
     ret <- 0
     if(x3 > 0) ret <- rho*sig*x3
```

```r
     return(ret)
}

f3 <- function(t, x1, x2, x3) {
     ret <- 0
     if(x3 > 0) ret <- sqrt(1-rho^2)*sig*x3
     return(ret)
}

diff.coef.matrix <- matrix(c("f1(t,x1,x2,x3)",
 "f2(t,x1,x2,x3) * g(t)", "f2(t,x1,x2,x3)", "0",
     "f3(t,x1,x2,x3)*g(t)", "f3(t,x1,x2,x3)"),  3, 2)

sabr.mod <- setModel(drift = c("0", "mu*g(t)*x3", "mu*x3"),
diffusion = diff.coef.matrix, state.variable = c("x1", "x2", "x3"),
    solve.variable = c("x1", "x2", "x3"))
str(sabr.mod@parameter)
```

The functions `f1`, `f2` and `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 **yuima** is `mu` as `rho` and `sig` are inside the bodies of the functions `f2` and `f3`. If we want to instruct **yuima** about these parameters, they should appear explicitly as arguments of the functions as shown in the following R code

```r
 f2 <- function(t, x1, x2, x3, rho, sig) {
     ret <- 0
     if(x3 > 0) ret <- rho*sig*x3
     return(ret)
 }

 f3 <- function(t, x1, x2, x3, rho, sig) {
     ret <- 0
     if(x3 > 0) ret <- sqrt(1-rho^2)*sig*x3
     return(ret)
 }

 diff.coef.matrix <- matrix(c("f1(t,x1,x2,x3)",
 "f2(t,x1,x2,x3,rho, sig) * g(t)", "f2(t,x1,x2,x3,rho,sig)",
 "0", "f3(t,x1,x2,x3,rho,sig)*g(t)", "f3(t,x1,x2,x3,rho,sig)"),  3, 2)

 sabr.mod <- setModel(drift = c("0", "mu*g(t)*x3", "mu*x3"),
 diffusion = diff.coef.matrix, state.variable = c("x1", "x2", "x3"),
 solve.variable = c("x1", "x2", "x3"))
str(sabr.mod@parameter)
```

2.3.1 *The Heston Model*

Consider a system of stochastic differential equations

$$
\begin{aligned}
\mathrm{d}X_{1,t} &= \mu X_{1,t}\mathrm{d}t + \sqrt{X_{2,t}}X_{1,t}\mathrm{d}W_{1,t} \\
\mathrm{d}X_{2,t} &= \kappa(\theta - X_{2,t})\mathrm{d}t + \varepsilon\sqrt{X_{2,t}}\mathrm{d}W_{2,t}
\end{aligned}
$$

where the first equation represents the asset price and the second equation represents the dynamics of the volatility process through a CIR model. Conditions on the parameters are that $2\kappa\theta > \varepsilon^2$ to ensure positiveness of the volatility process. The two Brownian motions $(W_{1,t}, W_{2,t})$ are correlated. In order to input this model into **yuima**, we need to rewrite this system by a transform of two independent Brownian motions, say $(B_{1,t}, B_{2,t})$ via the Cholesky decomposition. Indeed, let Y be a multivariate Gaussian random variable with variance–covariance matrix Σ, i.e. $Y \sim N(0, \Sigma)$. By Cholesky decomposition, one can find a matrix A such that $A^T \cdot A = \Sigma$. Then, if Z is a standard multivariate Gaussian random variable, we have that $AZ \sim N(0, \Sigma)$. Now we should extract the variance–covariance matrix Σ and apply the `chol` command to it to obtain the matrix A and transform the original diffusion matrix into a new one as we show in the next steps. First of all, we need to think at this system in matrix form

$$
\begin{pmatrix} \mathrm{d}X_{1,t} \\ \mathrm{d}X_{2,t} \end{pmatrix} = \begin{pmatrix} \mu X_{1,t} \\ \kappa(\theta - X_{2,t}) \end{pmatrix} \mathrm{d}t + \begin{pmatrix} \sqrt{X_{2,t}}X_{1,t} & 0 \\ 0 & \varepsilon\sqrt{X_{2,t}} \end{pmatrix} \begin{pmatrix} \mathrm{d}W_{1,t} \\ \mathrm{d}W_{2,t} \end{pmatrix}
$$

then introduce the matrix Σ to describe the correlation structure, for example

$$
\Sigma = \begin{pmatrix} s_{1,1} & s_{1,2} \\ s_{2,1} & s_{2,2} \end{pmatrix} = \begin{pmatrix} 0.5 & 0.7 \\ 0.7 & 2 \end{pmatrix}
$$

and apply the Cholesky decomposition using `chol`

```
Sigma <- matrix(c(0.5, 0.7, 0.7, 2), 2, 2)
C <- chol(Sigma)
C

##           [,1]      [,2]
## [1,] 0.7071068 0.9899495
## [2,] 0.0000000 1.0099505

crossprod(C)

##      [,1] [,2]
## [1,]  0.5  0.7
## [2,]  0.7  2.0

Sigma

##      [,1] [,2]
## [1,]  0.5  0.7
## [2,]  0.7  2.0
```

so that $C^T \cdot C = \Sigma$. Then, we use two independent Brownian motions $(B_{1,t}, B_{2,t})$ to rewrite the above system

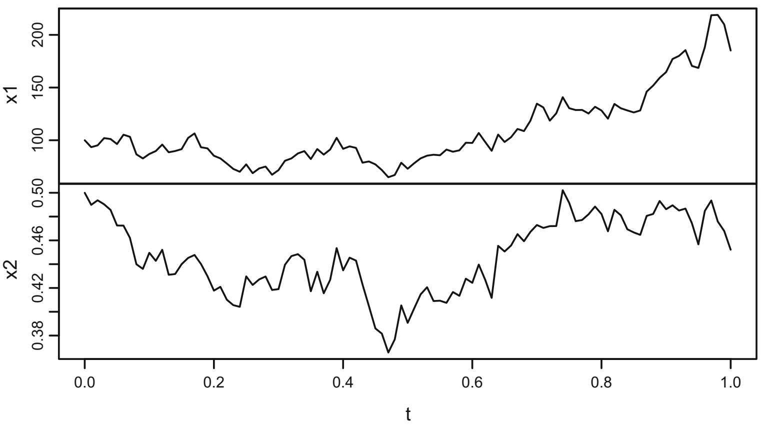


Fig. 2.5 A trajectory of the Heston stochastic volatility model

$$\begin{pmatrix} \mathrm{d}X_{1,t} \\ \mathrm{d}X_{2,t} \end{pmatrix} = \begin{pmatrix} \mu X_{1,t} \\ \kappa(\theta - X_{2,t}) \end{pmatrix} \mathrm{d}t + \begin{pmatrix} \sqrt{X_{2,t}} X_{1,t} & 0 \\ 0 & \varepsilon\sqrt{X_{2,t}} \end{pmatrix} \begin{pmatrix} c_{1,1} & c_{1,2} \\ 0 & c_{2,2} \end{pmatrix} \begin{pmatrix} \mathrm{d}B_{1,t} \\ \mathrm{d}B_{2,t} \end{pmatrix}$$

to get

$$\begin{pmatrix} \mathrm{d}X_{1,t} \\ \mathrm{d}X_{2,t} \end{pmatrix} = \begin{pmatrix} \mu X_{1,t} \\ \kappa(\theta - X_{2,t}) \end{pmatrix} \mathrm{d}t + \begin{pmatrix} c_{1,1}\sqrt{X_{2,t}} X_{1,t} & c_{1,2}\sqrt{X_{2,t}} X_{1,t} \\ 0 & c_{2,2}\varepsilon\sqrt{X_{2,t}} \end{pmatrix} \begin{pmatrix} \mathrm{d}B_{1,t} \\ \mathrm{d}B_{2,t} \end{pmatrix}$$

so we can finally code the Heston model as follows

```
set.seed(123)
drift <- c("mu*x1", "kappa*(theta-x2)")
diffusion <- matrix(c("c11*sqrt(x2)*x1", "0",
  "c12*sqrt(x2)*x1", "c22*epsilon*sqrt(x2)"),2,2)
heston <- setModel(drift=drift, diffusion=diffusion,
 state.var=c("x1","x2"))
X <- simulate(heston, true.par=list(theta=0.5, mu=1.2, kappa=2,
 epsilon=0.2, c11=C[1,1], c12=C[1,2], c22=C[2,2]),
 xinit=c(100,0.5))
```

Figure 2.5 shows a trajectory of the Heston model.

2.4 Parametric Inference

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

2.4.1 Quasi-maximum Likelihood Estimation

Consider a multidimensional diffusion process

$$\mathrm{d}X_t = a(X_t, \theta_2)\mathrm{d}t + b(X_t, \theta_1)\mathrm{d}W_t, \quad X_0 = x_0, \tag{2.7}$$

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 $b : \mathbb{R}^d \times \Theta_1 \to \mathbb{R}^d \otimes \mathbb{R}^r$. The naming of θ_2 and θ_1 is theoretically natural in view of the optimal convergence rates of the estimators for these parameters as we will see in the following. Given sampled data $\mathbf{X}_n = (X_{t_i})_{i=0,\dots,n}$, with $t_i = i\Delta_n$, $\Delta_n \to 0$ as $n \to \infty$, quasi-maximum likelihood estimator (QMLE) makes use of the following approximation of the true log-likelihood for multidimensional diffusions

$$\ell_n(\mathbf{X}_n, \theta) = -\frac{1}{2}\sum_{i=1}^{n}\left\{\log\det(\Sigma_{i-1}(\theta_1)) + \frac{1}{\Delta_n}\Sigma_{i-1}^{-1}(\theta_1)[(\Delta X_i - \Delta_n a_{i-1}(\theta_2))^{\otimes 2}]\right\} \tag{2.8}$$

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, $A[B] = \mathrm{tr}(AB)$. Then the QMLE of θ is an estimator that satisfies

$$\hat{\theta} = \underset{\theta}{\mathrm{argmax}}\, \ell_n(\mathbf{X}_n, \theta)$$

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, as well as the data. 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.

2.4.1.1 QMLE in Practice

As an example, we consider the simple model

$$\mathrm{d}X_t = (2 - \theta_2 X_t)\mathrm{d}t + (1 + X_t^2)^{\theta_1}\mathrm{d}W_t, \quad X_0 = 1 \tag{2.9}$$

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

```
ymodel <- setModel(drift="(2-theta2*x)", diffusion="(1+x^2)^theta1")
n <- 750
ysamp <- setSampling(Terminal = n^(1/3), n = n)
yuima <- setYuima(model = ymodel, sampling = ysamp)
set.seed(123)
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 θ_1 and θ_2 as follows

```
param.init <- list(theta2=0.5,theta1=0.5)
low.par <-  list(theta1=0, theta2=0)
upp.par <-  list(theta1=1, theta2=1)
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

```
summary(mle1)
```

```
## Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = yuima, start = param.init, lower = low.par,
## upper = upp.par)
##
## Coefficients:
## Estimate Std. Error
## theta1 0.1969182 0.008095453
## theta2 0.2998350 0.126410524
##
## -2 log L: -282.8676
```

2.4.1.2 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 2013; Yoshida 2011). On the other hand, one needs $T = n\Delta_n \to \infty$ for the consistency of $\hat{\theta}_2$ since the Fisher information for θ_2 is finite for a finite T, and consistent estimation of θ_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 (2.8) 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 (2012), 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 θ_2 appears, which will be discussed in Sect. 2.4.2.2. Modules for QMLE and Bayes estimators are going to be available for jump-diffusions. See Shimizu and Yoshida (2006) and Ogihara and Yoshida (2011, 2014).

2.4.2 Adaptive Bayes Estimation

Consider again the diffusion process in (2.7) and the quasi-likelihood defined in (2.8). The adaptive Bayes-type estimator (Yoshida 2011) is defined as follows: first we choose an initial arbitrary value $\theta_2^\star \in \Theta_2$ and pretend θ_1 is the unknown parameter for which we construct the Bayesian type estimator $\tilde{\theta}_1$ as follows

$$\tilde{\theta}_1 = \left[\int_{\Theta_1} \exp\{\ell_n(\mathbf{X}_n, (\theta_1, \theta_2^\star))\}\pi_1(\theta_1)d\theta_1\right]^{-1} \int_{\Theta_1} \theta_1 \exp\{\ell_n(\mathbf{X}_n, (\theta_1, \theta_2^\star))\}\pi_1(\theta_1)d\theta_1 \tag{2.10}$$

where π_1 is a prior density on Θ_1. According to the asymptotic theory under high-frequency samplings, any function π_1 can be used if it is positive on Θ_1. For the estimation of θ_2, we use $\tilde{\theta}_1$ to reform the quasi-likelihood function. That is, the Bayes-type estimator for θ_2 is defined by

$$\tilde{\theta}_2 = \left[\int_{\Theta_2} \exp\{\ell_n(\mathbf{X}_n, (\tilde{\theta}_1, \theta_2))\}\pi_2(\theta_2)d\theta_2\right]^{-1} \int_{\Theta_2} \theta_2 \exp\{\ell_n(\mathbf{X}_n, (\tilde{\theta}_1, \theta_2))\}\pi_2(\theta_2)d\theta_2 \tag{2.11}$$

where π_2 is a prior density on Θ_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 **yuima** via the method `adaBayes`. Consider again the model (2.9) 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]

```
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 `adaBayes`, on the same sample data we used for the `qmle` function, as follows:

```
lower <- list(theta1=0,theta2=0)
upper <- list(theta1=1,theta2=1)
bayes1 <- adaBayes(yuima, start=param.init, prior=prior,
 lower=lower,upper=upper, method="nomcmc")
```

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

```
coef(summary(bayes1))

##          Estimate  Std. Error
## theta1 0.1967596 0.008091151
## theta2 0.3029086 0.126341506

coef(summary(mle1))

##          Estimate  Std. Error
## theta1 0.1969182 0.008095453
## theta2 0.2998350 0.126410524
```

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

2.4.2.1 Theoretical Remarks on Adaptive Bayes Estimator

Under the same conditions, the adaptive Bayes estimators $\tilde{\theta}_1$ and $\tilde{\theta}_2$ perform in the same way as $\hat{\theta}_1$ and $\hat{\theta}_2$, respectively. See the remark in Sect. 2.4.1 and also Yoshida (2011) and Uchida and Yoshida (2012) for details.

2.4.2.2 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

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

```
bayes2 <- adaBayes(yuima, start=param.init, prior=prior,
 lower=lower,upper=upper)
```

and we look at the estimates

```
coef(summary(bayes2))

##         Estimate Std. Error
## theta1 0.1961799 0.01000809
## theta2 0.4121887 0.13561660

coef(summary(mle2))

##         Estimate  Std. Error
## theta1 0.1947225 0.009974792
## theta2 0.2193002 0.134937463
```

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.

2.4.2.3 The Effect of Δ on the Estimation

We have seen that, for sufficiently large sample size n, the estimators of the parameters in the diffusion coefficient converge at the speed $\sqrt{n}$, which means that the standard error of the estimates is also proportional to $\frac{1}{\sqrt{n}}$. In the case of drift estimation, consistency is possible only when $T = n\Delta_n$ is large and the standard error of the estimate is proportional to $\frac{1}{\sqrt{T}}$. The next example shows the effect of changing Δ and the number of observations n keeping fixed $n\Delta_n = T$. We start with 100000 observations and $\Delta_n = 0.001$, then we subsample the same trajectory for different values of $\Delta = 0.01, 0.05, 0.1, 0.5$, and we look at the estimates and their standard errors. We proceed as follows:

```
ymodel <- setModel(drift="(2-theta2*x)", diffusion="(1+x^2)^theta1")
n <- 100000
ysamp <- setSampling(delta=0.001, n = n)
mod <- setYuima(model = ymodel, sampling=ysamp)
set.seed(123)
yuima <- simulate(mod, xinit = 1,
true.parameter = list(theta1 = 0.2, theta2 = 0.3))
param.init <- list(theta2=0.5,theta1=0.5)
yuima0.01 <- subsampling(yuima,
 sampling=setSampling(delta=0.01,n=NULL,Terminal=100))
yuima0.1 <- subsampling(yuima,
 sampling=setSampling(delta=0.1,n=NULL,Terminal=100))
yuima1.0 <- subsampling(yuima,
 sampling=setSampling(delta=1,n=NULL,Terminal=100))
```

```
par(mfrow=c(2,2))
plot(yuima,main="delta=0.001, n=100000")
```

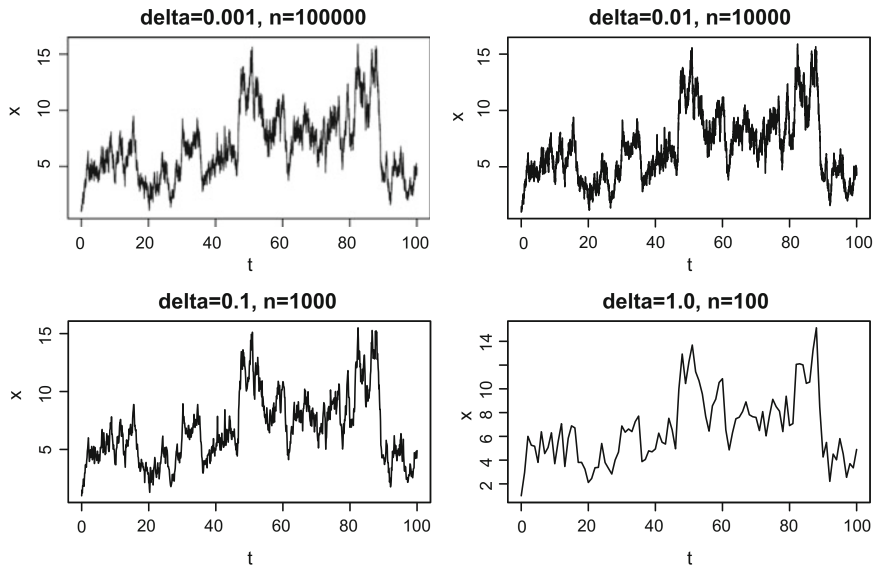


Fig. 2.6 Same trajectory subsampled at different Δ_n and fixed $T = 100$

```
plot(yuima0.01,main="delta=0.01, n=10000")
plot(yuima0.1,main="delta=0.1, n=1000")
plot(yuima1.0,main="delta=1.0, n=100")
```

Figure 2.6 shows the same trajectory subsampled at different Δ_n's.

We now perform quasi-maximum likelihood estimation on the different data and extract the information about the estimated coefficients and their standard errors using the slot `coef` from the output of the `summary` function:

```
low <- list(theta1=0, theta2=0)
up <-  list(theta1=1, theta2=1)
mle0.001 <- qmle(yuima, start = param.init, lower = low, upper = up)
summary(mle0.001)@coef

##          Estimate   Std. Error
## theta1 0.2001445 0.0005939987
## theta2 0.2909200 0.0316264349

mle0.01 <- qmle(yuima0.01, start = param.init, lower = low,
 upper = up)
summary(mle0.01)@coef

##          Estimate  Std. Error
## theta1 0.1984925 0.001883003
## theta2 0.2907650 0.031415656

mle0.1 <- qmle(yuima0.1, start = param.init, lower = low, upper = up)
summary(mle0.1)@coef
```

Table 2.2 Result of QMLE estimates on the same trajectory subsampled at different $\varDelta$'s and given $T = 100$. Standard errors in parentheses. True values: $\theta_1 = 0.2$, $\theta_2 = 0.3$

θ_1	θ_2	$\varDelta_n$	n
0.200 (0.001)	0.291 (0.032)	0.001	100000
0.198 (0.002)	0.291 (0.031)	0.010	10000
0.192 (0.006)	0.290 (0.031)	0.100	1000
0.223 (0.024)	0.293 (0.035)	1.000	100

```
##          Estimate  Std. Error
## theta1 0.1921024 0.006063728
## theta2 0.2896495 0.030602457

mle1.0 <- qmle(yuima1.0, start = param.init, lower = low, upper = up)
summary(mle1.0)@coef

##          Estimate Std. Error
## theta1 0.2226859 0.02449657
## theta2 0.2927327 0.03490789
```

Table 2.2 summarizes the analysis. Although this is an experiment based on a single simulated trajectory and not a Monte Carlo experiment, the results of the analysis clearly match the asymptotic theory for high-frequency data.

2.5 Example of Real Data Estimation for gBm

We now provide a couple of examples of estimation from real data. For this, we use library **quantmod** to import the data using the function `getSymbols` to download the daily quotation from Apple stock, using symbol `AAPL`.

```
library(quantmod)
getSymbols("AAPL",to="2016-12-31")

## [1] "AAPL"

head(AAPL)

##            AAPL.Open AAPL.High AAPL.Low AAPL.Close
## 2007-01-03    13.751    13.798   13.052   11.97143
## 2007-01-04    13.394    13.697   13.358   12.23714
## 2007-01-05    13.668    13.737   13.450   12.15000
## 2007-01-08    13.699    13.790   13.590   12.21000
## 2007-01-09    13.777    14.817   13.570   13.22429
```

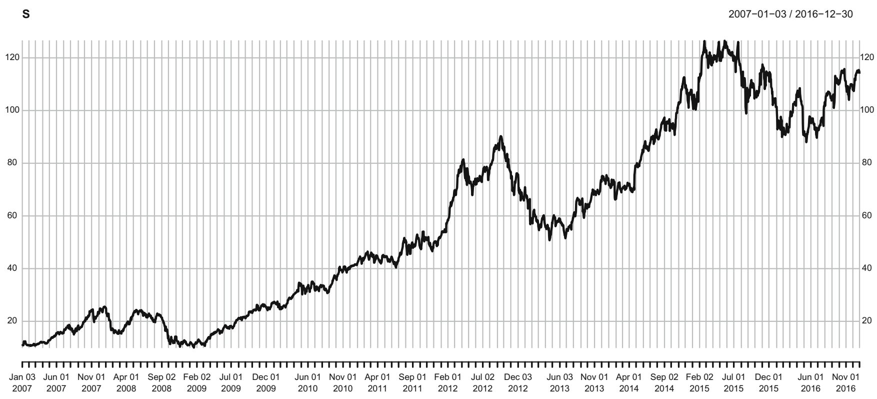


Fig. 2.7 Historical adjusted values of the Apple Inc. stock

```
## 2007-01-10    15.100    15.586   14.892   13.85714
##             AAPL.Volume AAPL.Adjusted
## 2007-01-03   309579900      10.73159
## 2007-01-04   211815100      10.96978
## 2007-01-05   208685400      10.89166
## 2007-01-08   199276700      10.94545
## 2007-01-09   837324600      11.85469
## 2007-01-10   738220000      12.42201

S <- AAPL$AAPL.Adjusted
```

Object S contains the adjusted (for dividends and splits) closing values of the Apple stock series. We can now set up a geometric Brownian motion model and set as data the closing values with rescaled time to delta=1/252 being that we use daily data.

```
Delta <- 1/252
gBm <- setModel(drift="mu*x", diffusion="sigma*x")
mod <- setYuima(model=gBm, data=setData(S, delta=Delta))
```

Looking at the plot in Fig. 2.7 of the stock price, we can imagine that the driving process is indeed a geometric Brownian motion because it looks like an exponential process, although we have no guarantee that the driving motion is a Gaussian noise.

```
set.seed(123)
plot(S)
```

We proceed as if it is a real geometric Brownian motion and then estimate the parameters via quasi-maximum likelihood estimation as follows:

```
fit <- qmle(mod, start=list(mu=1, sigma=1),
  lower=list(mu=0.1, sigma=0.1),
  upper=list(mu=100, sigma=10))
summary(fit)
```

```
## Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = mod, start = list(mu = 1, sigma = 1), lower =
## list(mu = 0.1,
## sigma = 0.1), upper = list(mu = 100, sigma = 10))
##
## Coefficients:
## Estimate Std. Error
## sigma 0.3320259 0.004741881
## mu 0.2909865 0.105058353
##
## -2 log L: 6799.443
```

and compare with the plug-in estimators for this model. Indeed, let

$$X_i = X_{t_i} = \log\left(\frac{S_{t_i}}{S_{t_{i-1}}}\right), \quad i = 1, \dots, n$$

be the approximate log-returns of the process. It is easy to show that the X_i form a sequence of independent and identically distributed Gaussian random variables $X_i \sim N(\alpha\Delta, , \sigma^2\Delta)$, where $\Delta = t_i - t_{i-1}$ and $\alpha = \mu - \frac{1}{2}\sigma^2$. Then, the plug-in estimators for μ and σ can be calculated according to the following code:

```
X <- diff(log(S))
X <- as.numeric(na.omit(diff(log(S))))
alpha <- mean(X)/Delta
sigma <- sqrt(var(X)/Delta)
mu <- alpha +0.5*sigma^2
mu

## [1] 0.2909959

sigma

## [1] 0.3299242

coef(fit)

##     sigma        mu
## 0.3320259 0.2909865
```

which looks close to the QMLE estimates as it should be in this situation.

2.6 Example of Real Data Estimation for CIR

We now look at exchange rate data, which are usually mean reverting processes of CIR type and try to estimate the parameters using quasi-maximum likelihood approach. To this aim, we consider the US/Euro foreign exchange rate available at

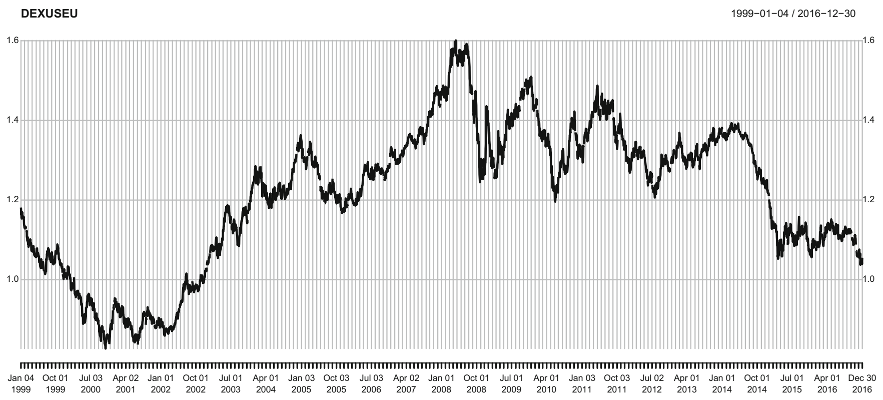


Fig. 2.8 Historical values of US/Euro Foreign Exchange rates

FRED (Federal Reserve Back of St. Luis) which can be obtained using `getSymbols` with argument `src=FRED`. In this case, the symbol is named `DEXUSEU`.

```
getSymbols("DEXUSEU", src="FRED")

## [1] "DEXUSEU"

DEXUSEU <- DEXUSEU["/2016"]
head(DEXUSEU)

##             DEXUSEU
## 1999-01-04  1.1812
## 1999-01-05  1.1760
## 1999-01-06  1.1636
## 1999-01-07  1.1672
## 1999-01-08  1.1554
## 1999-01-11  1.1534

meanCIR <- mean(DEXUSEU, na.rm=TRUE)
meanCIR

## [1] 1.212526
```

Looking at the plot in Fig. 2.8, we see that there is a long-run mean of the process which is `meanCIR` = 1.21.

```
set.seed(123)
plot(DEXUSEU)
```

We can parametrize the model in two ways

$$dX_t = (\theta_1 - \theta_2 X_t)dt + \sigma\sqrt{X_t}dW_t$$

or

$$dX_t = \kappa(\mu - X_t)dt + \sigma\sqrt{X_t}dW_t$$

where $\theta_1 = \kappa \cdot \mu$ and $\theta_2 = \kappa$. From the point of view of numerical estimation, it is better to estimate the model using the first form, because the parameters θ_1 and θ_2 do not multiply together and this makes the numerical optimization more stable. For the point of view of the interpretation, it is better to consider the second form as μ is the long-run mean of the process and κ plays the role of the mean reversion rate. We proceed with both parametrizations

```
cir1 <- setModel(drift="theta1-theta2*x", diffusion="sigma*sqrt(x)")
cir2 <- setModel(drift="kappa*(mu-x)", diffusion="sigma*sqrt(x)")
mod1 <- setYuima(model=cir1, data=setData(na.omit(DEXUSEU),
 delta=Delta))
mod2 <- setYuima(model=cir2, data=setData(na.omit(DEXUSEU),
 delta=Delta))
```

```
fit1 <- qmle(mod1, start=list(theta1=1, theta2=1, sigma=0.5),
  lower=list(theta1=0.1, theta2=0.1, sigma=0.1),
  upper=list(theta1=10, theta2=10, sigma=100),
   method="L-BFGS-B")
summary(fit1)
```

```
## Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = mod1, start = list(theta1 = 1, theta2 = 1,
## sigma = 0.5),
## method = "L-BFGS-B", lower = list(theta1 = 0.1, theta2 =
## 0.1,
## sigma = 0.1), upper = list(theta1 = 10, theta2 = 10,
## sigma = 100))
##
## Coefficients:
## Estimate Std. Error
## sigma 0.1113335 0.001178086
## theta1 0.2367827 0.189280186
## theta2 0.2010643 0.157913224
##
## -2 log L: -31277.3
```

```
fit2 <- qmle(mod2, start=list(kappa=1, mu=meanCIR, sigma=0.5),
  lower=list(kappa=0.1, mu=0.1, sigma=0.1),
  upper=list(kappa=10, mu=10, sigma=100),
   method="L-BFGS-B")
summary(fit2)
```

```
## Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = mod2, start = list(kappa = 1, mu = meanCIR,
## sigma = 0.5),
## method = "L-BFGS-B", lower = list(kappa = 0.1, mu = 0.1,
## sigma = 0.1), upper = list(kappa = 10, mu = 10, sigma =
## 100))
##
## Coefficients:
## Estimate Std. Error
## sigma 0.1108413 0.001165038
## kappa 0.2010811 0.157213044
## mu 1.1777081 0.141737640
##
## -2 log L: -31277.71
```

From the second parametrization, we can calculate θ_1 as $\kappa \cdot \mu$

```
theta1 <- as.numeric( coef(fit2)["kappa"] * coef(fit2)["mu"] )
theta1

## [1] 0.2368149

coef(fit1)["theta1"]

##    theta1
## 0.2367827
```

and from the first parametrization, we can get μ as θ_1/θ_2

```
mu <- as.numeric( coef(fit1)["theta1"] / coef(fit1)["theta2"] )
mu

## [1] 1.177647

coef(fit2)["mu"]

##       mu
## 1.177708
```

2.7 Hypotheses Testing

Consider a d-dimensional ergodic diffusion process X_t satisfying the stochastic differential equation:

$$dX_t = a(\alpha, X_t)dt + b(\beta, X_t)dW_t,$$

where functions a and b are suitably regular and known up to the parameters $\alpha \in \mathbb{R}^p$ and $\beta \in \mathbb{R}^q$ like in Sect. 2.4. Let us denote the parameter vector $\theta = (\alpha, \beta)$, and consider the following hypotheses testing problem for two simple hypotheses

$$H_0 : \theta = \theta_0 \quad \text{versus} \quad H_1 : \theta \neq \theta_0.$$

The problem of testing parametric hypotheses for diffusion processes is still a developing stream of research. In continuous time, Kutoyants (2004) and Dachian and Kutoyants (2008) considered the problem for ergodic diffusion models; Kutoyants (1994) considered the same problem for small diffusion processes. In discrete time, Lee and Wee (2008) dealt with a parametric version of the score-marked empirical process test statistic, while Aït-Sahalia (1996), Giet and Lubrano (2008) and Chen et al. (2008) proposed tests based on the several distances between parametric estimation and nonparametric estimation of the invariant density of ergodic diffusion processes.

Let $\{p(X,\theta), \theta \in \Theta\}$ be a family of probability densities. Denote by E_θ the expected value with respect to P_θ, the true law of the observations, say X. Let $\phi : [0,\infty) \to \mathbb{R}$ be a convex and continuous function. Furthermore, its restriction on $(0,\infty)$ is finite, two times continuously differentiable and such that $\phi(1) = \phi'(1) = 0$ and $\phi''(1) = 1$. Then the ϕ-divergence measure between the two models $p(X,\theta)$ and $p(X,\theta_0)$, $\theta \neq \theta_0$ is defined as

$$D_\phi(\theta,\theta_0) = E_{\theta_0}\phi\left(\frac{p(X,\theta)}{p(X,\theta_0)}\right) \tag{2.12}$$

The ϕ-divergences contain, as special cases, many divergences like the α-divergences (Csiszár 1967; Amari 1985), the Kullback-Leibler and the Hellinger divergences (Beran 1977; Simpson 1989) the Rényi's divergence (Rényi 1961), the power divergences studied in Cressie and Read (1984). Liese and Vajda (1987) provide extensive study of a modified version of α-divergences. Table 2.3 provides a list of well-known divergences that are special cases of the family of the ϕ-divergences. The ϕ-divergence measures have been used for testing hypotheses in parametric models. The reader can consult on this point, for example Morales et al. (1997) and Pardo (2006).

Given a sample of n independent and identically distributed (i.i.d.) observations and some asymptotically efficient estimator $\hat\theta_n$, to test $H_0 : \theta = \theta_0$ against $H_1 : \theta \neq \theta_0$, the ϕ-divergence test statistic is given by $D_\phi(\hat\theta_n,\theta_0)$, which cannot be used directly as is for diffusion processes. As in De Gregorio and Iacus (2013), let us consider the following quantity

$$\mathscr{D}_{\phi,n}(\theta,\theta_0) := \frac{1}{n}\sum_{i=1}^{n}\phi\left(\frac{p_i(\theta)}{p_i(\theta_0)}\right) \tag{2.13}$$

where $p_i(\theta) := \exp(\mathbb{H}_i(\theta))$ and $\mathbb{H}_i(\theta)$ is one of the summands of $-\ell_n(\mathbf{X}_n,\theta)$ in (2.8), i.e.

$$\mathbb{H}_i(\theta) = \frac{1}{2}\left\{\log\det(\Sigma_{i-1}(\theta_1)) + \frac{1}{\Delta_n}\Sigma_{i-1}^{-1}(\theta_1)[(\Delta X_i - \Delta_n a_{i-1}(\theta_2))^{\otimes 2}]\right\}.$$

Table 2.3 Family of ϕ-divergences include many special cases

$\phi(x)$ with $x = p(\theta,\cdot)/p(\theta_0,\cdot)$	Divergence
$x \log x - x + 1$	Kullback–Leibler
$-\log x + -1$	Minimum discrimination information
$(x-1)\log x$	J-divergence
$\frac{1}{2}(x-1)^2$	Pearson, Kagan
$\frac{(x-1)^2}{(x+1)^2}$	Balakrishnan and Sanghvi
$\frac{-x^s+s(x-1)+1}{1-s}, \quad s \neq 1$	Rathie and Kannappan
$\frac{1-x}{2} - \left(\frac{1+x^{-r}}{2}\right)^{-1/r} \quad r > 0$	Harmonic mean (Mathai and Rathie)
$\frac{(1-x)^2}{2(a+(1-a)x)} \quad 0 \le a \le 1$	Rukhin
$\frac{ax\log x-(ax+1-a)\log(ax+1-a)}{a(1-a)} \quad a \neq 0,1$	Lin
$\frac{x^{\lambda+1}-x-\lambda(x-1)}{\lambda(\lambda+1)} \quad \lambda \neq 0,-1$	Cressie and Read
$\lvert 1-x^a\rvert^{1/a} \quad 0 < a < 1$	Matusita
$\lvert 1-x\rvert^a \quad a \ge 1$	χ-divergence of order a (Vajda) and total variation if $a = 1$ (Saks)

The statistic $\mathscr{D}_{\phi,n}(\theta,\theta_0)$ represents the empirical mean of the functions $\phi\left(\frac{p_i(\theta)}{p_i(\theta_0)}\right)$ which measure the discrepancy between two (approximated) parametric models given the sample $\mathbf{X}_n$. The statistic $\mathscr{D}_{\phi,n}(\theta,\theta_0)$ is not an approximation of the ϕ-divergence (2.12) because it does not converge to

$$\int \phi\left(\frac{\mu_\theta(x)}{\mu_{\theta_0}(x)}\right)\mu_{\theta_0}(x)\mathrm{d}x,$$

with μ_{θ_0} the ergodic measure under θ_0, but it proves to be useful in the construction of a new class of asymptotically distribution-free test statistic as follows

$$T_{\phi,n}(\theta,\theta_0) := 2n\mathscr{D}_{\phi,n}(\theta,\theta_0).$$

The above quantity is similar to that used by Morales et al. (1997), where $\mathscr{D}_{\phi,n}(\theta,\theta_0)$ is replaced by the true ϕ-divergence.

Let $\hat{\theta}_n$ be the quasi-maximum likelihood estimator or the Bayes-type estimator defined in Sect. 2.4. It is known (De Gregorio and Iacus 2013) that the family of test statistics

$$T_{\phi,n}(\hat{\theta}_n,\theta_0) \tag{2.14}$$

is asymptotically distribution-free under H_0, i.e.

$$T_{\phi,n}(\hat{\theta}_n,\theta_0) \xrightarrow{d} \chi^2_{p+q}$$

as $n \to \infty$, $n\Delta_n^2 \to 0$, $\Delta_n \to 0$ and $n\Delta_n = T \to \infty$.

Given the level α, such test rejects H_0 if $T_{\phi,n} > c_\alpha$ where c_α is the $1-\alpha$ quantile of the limiting random variable χ^2_{p+q}.

The power function of the test and higher-order expansion of the distribution of the test statistic is also known but not yet implemented in **yuima**; see De Gregorio and Iacus (2013) for further details.

The ϕ-divergence test is available in **yuima** through the function `phi.test`. This function requires as input a `yuima` object with the `model` and `data` slot as well as the definition of the function $\phi(\cdot)$. The estimator $\hat{\theta}_n$ is the quasi-maximum likelihood estimator of Sect. 2.4.1. Consider the simple Gaussian model

$$\mathrm{d}X_t = \theta_1 * (\theta_2 - X_t)\mathrm{d}t + \theta_3 \mathrm{d}W_t$$

and assume that we want two hypotheses for $\theta = (\theta_1, \theta_2, \theta_3)$

$$H_{00} : \theta_{00} = (0.3, 1, 0.25)$$

and

$$H_{01} : \theta_{01} = (0.3, 0.2, 0.1)$$

```
model<- setModel(drift="t1*(t2-x)",diffusion="t3")
```

We simulate the model according to $\theta_0 = H_{00}$

```
T<-300
n<-3000
sampling <- setSampling(Terminal=T, n=n)
yuima<-setYuima(model=model, sampling=sampling)
h00 <- list(t1=0.3, t2=1, t3=0.25)
h01 <- list(t1=0.3, t2=0.2, t3=0.1)
set.seed(123)
X <- simulate(yuima, xinit=1, true.par=h00)
```

and we choose as $\phi(x) = 1 - x + x\log(x)$ which gives a test equivalent to the likelihood ratio test

```
phi1 <- function(x) 1-x+x*log(x)
```

Now we assume that we do not know the true value of θ, and we set up a generalized likelihood ratio test of the form $T_{\phi,n}(\hat{\theta}_n, \theta_0)$, with $\hat{\theta}_n$ the quasi-maximum likelihood estimator. To this aim, we apply the function `phi.test` specifying $H_0 = H_{00}$

```
phi.test(X, H0=h00, phi=phi1, start=h00,
   lower=list(t1=0.1, t2=0.1, t3=0.1),
   upper=list(t1=2,t2=2,t3=2),method="L-BFGS-B")

##
## estimating parameters via QMLE...
## Phi-Divergence test statistic based on phi = 'phi1'
## H0: t1 = 0.300 t2 = 1.000 t3 = 0.250
## versus
## H1: t1 = 0.342 t2 = 1.033 t3 = 0.248
## H1 parameters estimated using QMLE
##
## Test statistic = 1.419, df = 3, p-value = 0.7011919
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In the above H_1 corresponds to the value of the QMLE $\hat{\theta}_n$. As we can see, the above test does not reject $H_0 = H_{00}$ as the QMLE estimator performs quite well and provides estimated values for θ almost equal to H_{00}, with a p value of 0.701. We now test against a false H_0 replacing H_{00} with H_{01}, and we expect the test to reject the null hypothesis.

```
phi.test(X, H0=h01, phi=phi1, start=h00,
  lower=list(t1=0.1, t2=0.1, t3=0.1),
  upper=list(t1=2,t2=2,t3=2),method="L-BFGS-B")

##
## estimating parameters via QMLE...
## Phi-Divergence test statistic based on phi = 'phi1'
## H0: t1 = 0.300 t2 = 0.200 t3 = 0.100
## versus
## H1: t1 = 0.342 t2 = 1.033 t3 = 0.248
## H1 parameters estimated using QMLE
##
## Test statistic = 8.5e+17, df = 3, p-value = 0 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This time the test rejects very sharply the null hypothesis $H_0 = H_{01}$ in favour of $H_1 = \hat{\theta}_n$. Notice that if the function $\phi(\cdot)$ is not specified, the `phi.test` command assumes by default the same `phi1`.

2.8 AIC Model Selection

The Akaike information criterion (AIC) is a measure of the relative quality of a statistical model, for a given set of data (Akaike 1973; Sakamoto et al. 1986). As such, AIC is a tool for model selection, possibly of non-nested models, but it is not a statistical test in the sense of Sect. 2.7. Indeed, the AIC statistic says nothing about the quality of the model in an absolute sense; i.e., if all the candidate models fit poorly AIC will not give any warning of that. AIC deals with the trade-off between the goodness of fit of the model and the complexity of the model. Let $\theta \in \Theta \subset \mathbb{R}^d$, and $\hat{\theta}_n$ a maximum likelihood estimator of θ, and then, the AIC statistic is defined as

$$AIC = -2\ell_n(\hat{\theta}_n) + 2\dim(\Theta).$$

Given a set of candidate models for the data, the preferred model is the one with the minimum AIC value. Hence, AIC not only rewards goodness of fit, but also includes a penalty that is an increasing function of the number of estimated parameters. The penalty discourages overfitting (increasing the number of parameters in the model always improves the goodness of the fit). R has a function called `AIC` which evaluates this statistic for statistical models and in particular for objects which extends the class `mle` like the output of the method `qmle`. Unfortunately, the construction of the true AIC statistic is a delicate problem for diffusion processes and this function should be used with care as the standard `AIC` function in R does not know when the theoretical conditions to apply it hold or not. In particular, the necessary conditions hold for ergodic diffusion processes such that $\Delta_n \to 0$, $n\Delta^2 \to 0$ and $n\Delta = T \to \infty$ as $n \to \infty$. Uchida (2010) contains several results on information criteria, including the AIC statistic. Other criteria like the quasi-Bayesian information criterion are becoming available in `yuima`.

2.8.1 An Example of AIC Model Selection for Exchange Rates Data

Consider again the data in Sect. 2.6, which clearly looks like a CIR model (see Fig. 2.8). We now try to fit both a geometric Brownian motion model

$$\mathrm{d}X_t = \mu X_t \mathrm{d}t + \sigma X_t \mathrm{d}W_t \quad (\texttt{mod})$$

and the CIR model according to the two different parametrizations

$$\mathrm{d}X_t = (\theta_1 - \theta_2 X_t)\mathrm{d}t + \sigma\sqrt{X_t}\mathrm{d}W_t \quad (\texttt{mod1})$$

or

$$\mathrm{d}X_t = \kappa(\mu - X_t)\mathrm{d}t + \sigma\sqrt{X_t}\mathrm{d}W_t \quad (\texttt{mod2})$$

and the CKLS model

$$\mathrm{d}X_t = \kappa(\mu - X_t)\mathrm{d}t + \sigma X_t^\gamma \mathrm{d}W_t \quad (\texttt{mod3})$$

```
library(quantmod)
Delta <- 1/252
getSymbols("DEXUSEU", src="FRED")

## [1] "DEXUSEU"

DEXUSEU <- DEXUSEU["/2016"]
USEU <- setData(na.omit(DEXUSEU),  delta=Delta)
meanCIR <- mean(get.zoo.data(USEU)[[1]])
gBm <- setModel(drift="mu*x", diffusion="sigma*x")
mod <- setYuima(model=gBm, data=USEU)
```

```
cir1 <- setModel(drift="theta1-theta2*x", diffusion="sigma*sqrt(x)")
cir2 <- setModel(drift="kappa*(mu-x)", diffusion="sigma*sqrt(x)")
ckls <- setModel(drift="theta1-theta2*x", diffusion="sigma*x^gamma")
mod1 <- setYuima(model=cir1, data=USEU)
mod2 <- setYuima(model=cir2, data=USEU)
mod3 <- setYuima(model=ckls, data=USEU)
gBm.fit <- qmle(mod, start=list(mu=1, sigma=1),
  lower=list(mu=0.1, sigma=0.1),
  upper=list(mu=100, sigma=10))
cir1.fit <- qmle(mod1, start=list(theta1=1, theta2=1, sigma=0.5),
  lower=list(theta1=0.1, theta2=0.1, sigma=0.1),
  upper=list(theta1=10, theta2=10, sigma=100),
   method="L-BFGS-B")
cir2.fit <- qmle(mod2, start=list(kappa=1, mu=meanCIR, sigma=0.5),
  lower=list(kappa=0.1, mu=0.1, sigma=0.1),
  upper=list(kappa=10, mu=10, sigma=100),
   method="L-BFGS-B")
ckls.fit <- qmle(mod3, start=list(theta1=1, theta2=1, sigma=0.5,
  gamma=0.5),  lower=list(theta1=0.1, theta2=0.1, sigma=0.1,
  gamma=0.1), upper=list(theta1=10, theta2=10, sigma=10,
  gamma=2), method="L-BFGS-B")
```

we now pass the output of the estimated models to the `AIC` function and select the model with the lowest AIC statistic

```
AIC(gBm.fit,cir1.fit,cir2.fit,ckls.fit)

##           df       AIC
## gBm.fit    2 -31064.73
## cir1.fit   3 -31271.30
## cir2.fit   3 -31271.71
## ckls.fit   4 -31300.71
```

which turns to be the “ckls.fit” model, and, as expected, the two CIR models perform almost in the same manner. We now run the same experiment on some simulated data from the geometric Brownian motion

```
set.seed(123)
S <- simulate(gBm, true.par=list(mu=1, sigma=0.25),
  sampling=setSampling(T=1, n=1000), xinit=100)
mod <- setYuima(model=gBm, data=S@data)
mod1 <- setYuima(model=cir1, data=S@data)
mod2 <- setYuima(model=cir2, data=S@data)
mod3 <- setYuima(model=ckls, data=S@data)
 gBm.fit <- qmle(mod, start=list(mu=1, sigma=1),
  lower=list(mu=0.1, sigma=0.1),
  upper=list(mu=100, sigma=10))
cir1.fit <- qmle(mod1, start=list(theta1=1, theta2=1, sigma=0.5),
  lower=list(theta1=0.1, theta2=0.1, sigma=0.1),
  upper=list(theta1=10, theta2=10, sigma=100),
   method="L-BFGS-B")
cir2.fit <- qmle(mod2, start=list(kappa=1, mu=meanCIR, sigma=0.5),
  lower=list(kappa=0.1, mu=0.1, sigma=0.1),
  upper=list(kappa=10, mu=10, sigma=100),
   method="L-BFGS-B")
ckls.fit <- qmle(mod3,
  start=list(theta1=1, theta2=1, sigma=0.5, gamma=0.5),
  lower=list(theta1=0.1, theta2=0.1, sigma=0.1, gamma=0.1),
  upper=list(theta1=10, theta2=10, sigma=10, gamma=2),
   method="L-BFGS-B")
```

```
AIC(gBm.fit,cir1.fit,cir2.fit,ckls.fit)

##           df      AIC
## gBm.fit    2 3449.091
## cir1.fit   3 3538.850
## cir2.fit   3 3540.618
## ckls.fit   4 3474.849
```

The best model turns to be "gBm.fit", and, as before, the two CIR models perform almost at the same manner.

2.9 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 et al. 2004).

To simplify the idea, consider a fully specified regression model

$$Y = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \cdots + \theta_k X_k + \varepsilon,$$

where ε is the Gaussian error term, and perform least squares estimation under L_1 constraints, i.e.

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \left\{ (Y - \theta X)^T (Y - \theta X) + \sum_{i=1}^{k} |\theta_i| \right\}.$$

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

$$\mathrm{d}X_t = a(X_t, \alpha)\mathrm{d}t + b(X_t, \beta)\mathrm{d}W_t$$

$$\alpha = (\alpha_1, ..., \alpha_p)' \in \Theta_p \subset \mathbb{R}^p, \quad p \geq 1$$

$$\beta = (\beta_1, ..., \beta_q)' \in \Theta_q \subset \mathbb{R}^q, \quad q \geq 1$$

with $a : \mathbb{R}^d \times \Theta_p \to \mathbb{R}^d$, $b : \mathbb{R}^d \times \Theta_q \to \mathbb{R}^d \otimes \mathbb{R}^m$ 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 α and β. We denote by $\theta = (\alpha, \beta) \in \Theta_p \times \Theta_q = \Theta$ the parameter vector, with

$\theta_0 = (\alpha_0, \beta_0)$ the unknown true value. Let $\mathbb{H}_n(\mathbf{X}_n, \theta) = -\ell_n(\mathbf{X}_n, \theta)$ from equation (2.8). Denote by $\hat{\theta}_n$ the quasi-maximum likelihood estimator for this model. The adaptive LASSO estimator is defined as the solution to the quadratic problem under L_1 constraints

$$\check{\theta}_n = (\check{\alpha}_n, \check{\beta}_n) = \underset{\theta}{\text{argmin}}\ \mathscr{F}(\theta).$$

with

$$\mathscr{F}(\theta) = (\theta - \hat{\theta}_n)^T \ddot{\mathbb{H}}_n(\mathbf{X}_n, \hat{\theta}_n)(\theta - \hat{\theta}_n) + \sum_{j=1}^{p} \lambda_{n,j}|\alpha_j| + \sum_{k=1}^{q} \gamma_{n,k}|\beta_k|$$

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

$$\lambda_{n,j} = \lambda_0|\hat{\alpha}_{j,n}|^{-\delta_1}, \qquad \gamma_{n,k} = \gamma_0|\hat{\beta}_{k,n}|^{-\delta_2} \tag{2.15}$$

where $\hat{\alpha}_{j,n}$ and $\hat{\beta}_{k,n}$ are the unpenalized QMLE's of α_j and β_k, respectively, $\delta_1, \delta_2 > 0$ and usually taken unitary. Suppose to have this two-dimensional stochastic differential equation

$$\begin{pmatrix} \mathrm{d}X_{1,t} \\ \mathrm{d}X_{2,t} \end{pmatrix} = \begin{pmatrix} 1 - \mu_{11}X_{1,t} + \mu_{12}X_{2,t} \\ 2 + \mu_{21}X_{1,t} - \mu_{22}X_{2,t} \end{pmatrix} \mathrm{d}t + \begin{bmatrix} s_1X_{1,t} & -s_3X_{2,t} \\ s_2X_{1,t} & s_4X_{2,t} \end{bmatrix} \begin{pmatrix} \mathrm{d}W_{1,t} \\ \mathrm{d}W_{2,t} \end{pmatrix}$$

and let us generate data from this model setting $\mu_{12} = \mu_{21} = s_2 = s_3 = 0$.

```r
a <- c("1-mu11*X1+mu12*X2","2+mu21*X1-mu22*X2")
b <- matrix(c("s1*X1","s2*X1", "-s3*X2","s4*X2"),2,2)
mod.est <- setModel(drift=a, diffusion=b,
 solve.var=c("X1","X2"),state.variable=c("X1","X2"))
truep <- list(mu11=.9, mu12=0, mu21=0, mu22=0.7,
 s1=.3, s2=0,s3=0,s4=.2)
low <- list(mu11=1e-5, mu12=1e-5, mu21=1e-5, mu22=1e-5,
 s1=1e-5, s2=1e-5, s3=1e-5,s4=1e-5)
upp <- list(mu11=2, mu12=2, mu21=1, mu22=1,
 s1=1, s2=1, s3=1,s4=1)
set.seed(123)
n <- 1000
X <- simulate(mod.est, T=n^(1/3), n=n, xinit=c(1,1),
 true.parameter=truep)
```

We now run the `lasso` on the simulated data

```r
myest <- lasso(X,  delta=2, start=truep, lower=low, upper=upp,
 method="L-BFGS-B")
myest
```

```
## Adaptive Lasso estimation
##
## Call:
## lasso(yuima = X, start = truep, delta = 2, lower = low,
## upper = upp,
## method = "L-BFGS-B")
##
## QMLE estimates
## Estimate Std. Error
## s1 0.27067445 0.02064474
## s3 0.01924009 0.02634989
## s2 0.03562399 0.07725490
## s4 0.19273459 0.01413467
## mu11 1.05919238 0.33439039
## mu12 0.11540187 0.14993938
## mu21 0.00001000 0.40618976
## mu22 0.76739528 0.21222991
##
## LASSO estimates
## Estimate Std. Error
## s1 0.2717053 0.0138140011
## s3 0.0000100 0.0006085617
## s2 0.0000100 0.0011283299
## s4 0.1915552 0.0097779142
## mu11 0.8074355 0.0616505395
## mu12 0.0000100 0.0036563071
## mu21 0.0000100 0.0003170211
## mu22 0.7612227 0.0431996864
```

and in this simulation example the Lasso method selects correctly the model. We can this result against the AIC method.

```
fit1 <- qmle(X, start=truep, lower=low, upper=upp, method="L-BFGS-B")
```

against the selected model which should be written anew

```
a <- c("1-mu11*X1","2-mu22*X2")
b <- matrix(c("s1*X1","0", "0","s4*X2"),2,2)
mod.est2 <- setModel(drift=a, diffusion=b,
 solve.var=c("X1","X2"),state.variable=c("X1","X2"))
truep <- list(mu11=.9, mu22=0.7, s1=.3,s4=.2)
low <- list(mu11=1e-5,   mu22=1e-5, s1=1e-5,  s4=1e-5)
upp <- list(mu11=2,   mu22=2, s1=1,  s4=1)
Y <- setYuima(model=mod.est2, data=X@data)
fit2 <- qmle(Y, start=truep, lower=low, upper=upp, method="L-BFGS-B")
summary(fit1)

## Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = X, start = truep, method = "L-BFGS-B", lower = low,
##     upper = upp)
##
## Coefficients:
##        Estimate Std. Error
## s1   0.27067445 0.02064474
## s3   0.01924009 0.02634989
## s2   0.03562399 0.07725490
## s4   0.19273459 0.01413467
## mu11 1.05919238 0.33439039
## mu12 0.11540187 0.14993938
## mu21 0.00001000 0.40618976
## mu22 0.76739528 0.21222991
##
## -2 log L: -287.3061

summary(fit2)

## Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = Y, start = truep, method = "L-BFGS-B", lower = low,
##     upper = upp)
##
## Coefficients:
##       Estimate Std. Error
## s1   0.2738216 0.01949123
## s4   0.1936306 0.01379907
## mu11 0.8142334 0.08659001
## mu22 0.7680186 0.06123139
##
## -2 log L: -286.2465

AIC(fit1, fit2)

##      df       AIC
## fit1  8 -271.3061
## fit2  4 -278.2465
```

and in this case LASSO and AIC select the same model with the difference that in the LASSO method there is no need to specify two different models.

2.9.1 An Example of Lasso Model Selection for Interest Rates Data

Let us consider the full CKLS model (Chan et al. 1992)

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

and let us try to estimate the parameter on the US Interest Rates monthly data from June 1964 to December 1989 (see Fig. 2.9). We prepare the data, taking into account that these are monthly data, the model and the constraints for optimization

```
library("Ecdat")
data("Irates")
rates <- Irates[,"r1"]
```

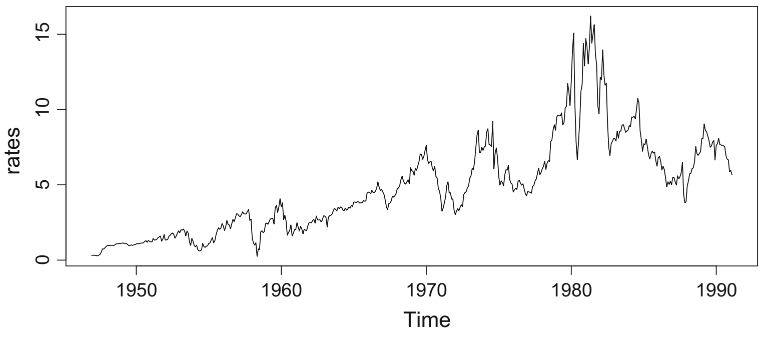


Fig. 2.9 US Interest Rates monthly data from June 1964 to December 1989

```
plot(rates)
X <- window(rates, start=1964.471, end=1989.333)
mod <- setModel(drift="alpha+beta*x", diffusion="sigma*x^gamma")
yuima <- setYuima(data=setData(X,delta=1/12), model=mod)
start <- list(alpha=1, beta =-.1, sigma =.1, gamma =1)
low <- list(alpha=-5, beta =-5, sigma =-5, gamma =-5)
upp <- list(alpha=8, beta =8, sigma =8, gamma =8)
```

Looking at the data, we can see that this time series is not ergodic, so the application of the Lasso method in this case may be questionable but we proceed anyway.

Now we apply the `lasso` function

```
lasso.est <- lasso(yuima, start=start, lower=low, upper=upp,
   method="L-BFGS-B", delta=2)
lasso.est
```

From which we see that, instead of the general model

$$dX_t = (\alpha + \beta X_t)dt + \sigma X_t^{\gamma} dW_t$$

the LASSO method selects the reduced model

$$dX_t = \alpha dt + \sigma X_t^{\gamma} dW_t$$

We can compare with the AIC statistic to see if this conclusion is further supported or not.

```
mod1 <- setModel(drift="alpha", diffusion="sigma*x^gamma")
yuima1 <- setYuima(data=setData(X,delta=1/12), model=mod1)
start1 <- list(alpha=1, sigma =.1, gamma =1)
low1 <- list(alpha=-5, sigma =-5, gamma =-5)
upp1 <- list(alpha=8, sigma =8, gamma =8)
fit <- qmle(yuima, start=start, lower=low, upper=upp,
   method="L-BFGS-B")
fit1 <- qmle(yuima1, start=start1, lower=low1, upper=upp1,
   method="L-BFGS-B")
```

```
summary(fit)
summary(fit1)
AIC(fit, fit1)
```

```
## Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = yuima, start = start, method = "L-BFGS-B",
## lower = low,
## upper = upp)
##
## Coefficients:
## Estimate Std. Error
## sigma 0.1325225 0.0255461
## gamma 1.4432799 0.1027345
## alpha 2.0755483 0.9917822
## beta -0.2629820 0.1958201
##
## -2 log L: 475.7687
## Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = yuima1, start = start1, method = "L-BFGS-B",
## lower = low1,
## upper = upp1)
##
## Coefficients:
## Estimate Std. Error
## sigma 0.1297013 0.02413509
## gamma 1.4555461 0.09944181
## alpha 0.8072980 0.29588245
##
## -2 log L: 477.5659
## df AIC
## fit 4 483.7687
## fit1 3 483.5659
```

and we can see a slight preference for the reduced model.

2.10 Change Point Estimation

Consider a multidimensional stochastic differential equation of the form

$$\mathrm{d}Y_t = a_t \mathrm{d}t + b(X_t, \theta)\mathrm{d}W_t, \quad t \in [0, T],$$

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

When $Y = X$ and a_t is a function of X_t, 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

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

The change point τ^* instant is unknown and is to be estimated, along with θ_0^* and θ_1^*, from the observations sampled from the path of (X, Y). The **yuima** package implements the quasi-maximum likelihood approach as described in the following Iacus and Yoshida (2012). Let $\Delta_i Y = Y_{t_i} - Y_{t_{i-1}}$ and define

$$\Phi_n(t; \theta_0, \theta_1) = \sum_{i=1}^{[nt/T]} G_i(\theta_0) + \sum_{i=[nt/T]+1}^{n} G_i(\theta_1), \tag{2.16}$$

with

$$G_i(\theta) = \log\det S(X_{t_{i-1}}, \theta) + \Delta_n^{-1}(\Delta_i Y)' S(X_{t_{i-1}}, \theta)^{-1}(\Delta_i Y) \tag{2.17}$$

and $S = b^{\otimes 2}$. Suppose that there exists an estimator $\hat{\theta}_k$ for each θ_k, $k = 0, 1$. In case θ_k^* are known, we define $\hat{\theta}_k$ just as $\hat{\theta}_k = \theta_k^*$. The change point estimator of τ^* is

$$\hat{\tau} = \operatorname*{argmin}_{t \in [0,T]} \Phi_n(t; \hat{\theta}_0, \hat{\theta}_1).$$

2.10.1 *Example of Volatility Change Point Estimation for Two-Dimensional SDEs*

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

$$\begin{pmatrix} \mathrm{d}X_{1,t} \\ \mathrm{d}X_{2,t} \end{pmatrix} = \begin{pmatrix} a_1(X_{1,t}) \\ a_2(X_{2,t}) \end{pmatrix} \mathrm{d}t + \begin{pmatrix} \theta_{1.k} \cdot X_{1,t} & 0 \cdot X_{1,t} \\ 0 \cdot X_{2,t} & \theta_{2.k} \cdot X_{2,t} \end{pmatrix} \begin{pmatrix} \mathrm{d}W_{1,t} \\ \mathrm{d}W_{2,t} \end{pmatrix}, \quad t \in [0, T],$$

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 two-dimensional stochastic differential equation:

$$\begin{pmatrix} \mathrm{d}X_{1,t} \\ \mathrm{d}X_{2,t} \end{pmatrix} = \begin{pmatrix} \sin(X_{1,t}) \\ 3 - X_{2,t} \end{pmatrix} \mathrm{d}t + \begin{pmatrix} \theta_{1.k} \cdot X_{1,t} & 0 \cdot X_{1,t} \\ 0 \cdot X_{2,t} & \theta_{2.k} \cdot X_{2,t} \end{pmatrix} \begin{pmatrix} \mathrm{d}W_{1,t} \\ \mathrm{d}W_{2,t} \end{pmatrix}, \quad t \in [0, T],$$

$$X_{1,0} = 3, \quad X_{2,0} = 3,$$

with change point at time $\tau = 4$. We set $T = 10$. First, we describe the model to be simulated

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

##
## Diffusion process
## Number of equations: 2
## Number of Wiener noises: 2
## Parametric model with 2 parameters
```

and then simulate two trajectories. The first trajectory is simulated up to the change point $\tau = 4$ with parameters $\theta_{1.0} = 0.5$ and $\theta_{2.0} = 0.3$ as follows:

```
n <- 1000

set.seed(123)

t0 <- list(theta1.k=0.5, theta2.k=0.3)
T <- 10
tau <- 4
pobs <- tau/T
ysamp1 <- setSampling(n=n*pobs, Initial=0, delta=0.01)
yuima1 <- setYuima(model=ymodel, sampling=ysamp1)
yuima1 <- simulate(yuima1, xinit=c(3, 3), true.parameter=t0)

v11 <- get.zoo.data(yuima1)[[1]]
x1 <- as.numeric(v11[length(v11)]) # terminal value
v21 <- get.zoo.data(yuima1)[[2]]
x2 <- as.numeric(v21[length(v21)]) # terminal value
```

The second trajectory is then generated with parameters $\theta_{1.1} = 0.2$ and $\theta_{2.1} = 0.4$, from τ till T. The initial value of the second trajectory is set equal to the last value of the first trajectory stored in `x1` and `x2` for the two component of the process (see Fig. 2.10)

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

Finally, we collate the two trajectories

```
v12 <- get.zoo.data(yuima2)[[1]]
v22 <- get.zoo.data(yuima2)[[2]]
v1 <- c(v11,v12[-1])
v2 <- c(v21,v22[-1])
new.data <- setData(zoo(cbind(v1,v2)),delta=0.01)
yuima <- setYuima(model=ymodel, data=new.data)
```

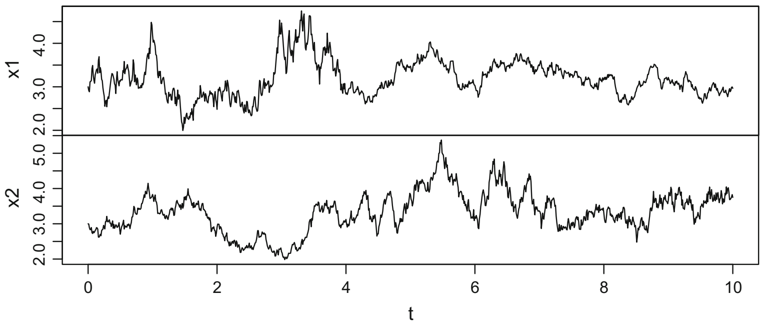


Fig. 2.10 Two-dimensional trajectory with change point around $\tau = 4$

The composed trajectory is visible in Fig. 2.10 and can be plotted simply typing this command:

```
plot(yuima)
```

As said, the change point analysis does not consider the information coming from the drift part of the model and, indeed, **yuima** ignores it 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.

```
noDriftModel <- setModel(drift=c(0,0), diffusion=diff.matrix,
 time.variable="t", state.variable=c("x1", "x2"),
 solve.variable=c("x1", "x2"))
noDriftModel <- setYuima(noDriftModel, data=new.data)
noDriftModel@model@drift

## expression((0), (0))

noDriftModel

##
## Diffusion process, driftless
## Number of equations: 2
## Number of Wiener noises: 2
## Parametric model with 2 parameters
##
## Number of original time series: 2
## length = 1001, time range [0 ; 10]
##
## Number of zoo time series: 2
##    length time.min time.max delta
## v1   1001        0       10  0.01
## v2   1001        0       10  0.01
```

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 $\theta_{2.k}$

```
t.est <- CPoint(yuima,param1=t0,param2=t1)
t.est$tau

## [1] 3.98

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

## [1] 3.98
```

As it can be seen, the above estimates of the change point are the same for the complete model `yuima` and the model without drift, i.e. `noDriftModel`.

2.10.2 *An Example of Two-Stage Estimation*

In practical situations, the initial values of the parameters are not known and it is necessary to provide some preliminary estimators of them. One possible solution is the two-stage change point estimation approach (Iacus and Yoshida 2012). The idea is to take a small subset of observations at the very beginning and the end of the time series to obtain initial guess of the parameters θ, estimate a change point and then refine the estimation of θ using the information about the change point.

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. Similarly for `qmleR`, which uses only observations in the time interval $[t, T]$. In our example, we set `t=1.5` and `t=8.5`, respectively.

```
qmleL(noDriftModel, t=1.5, 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
qmleR(noDriftModel, t=8.5, 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
t0.est <- coef(estL)
t1.est <- coef(estR)
```

and now we proceed with change point estimation

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

## $tau
## [1] 3.98
##
## $param1
## theta1.k theta2.k
```

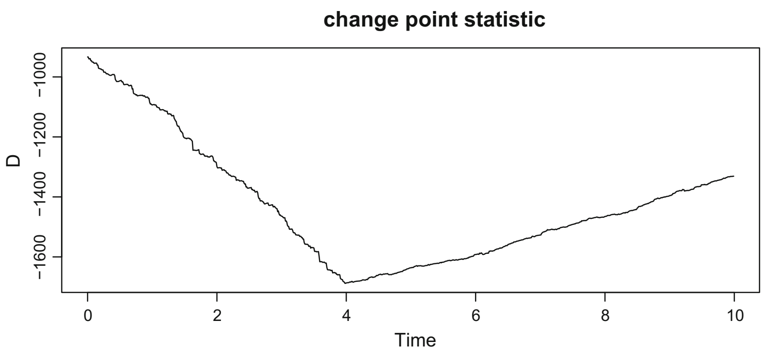


Fig. 2.11 Change point statistic reaches the minimum at change point date

```
## 0.474565 0.287590
##
## $param2
##   theta1.k  theta2.k
## 0.1921615 0.4413915
```

Notice that, even if the estimated parameters are not too accurate because we use small subsets of observations, the change point estimate remains good. Setting the argument `plot=TRUE`, it is possible to see the graph of the change point statistic $\Phi_n(t;\theta_0,\theta_1)$ from (2.16), denoted as D in the plot shown in Fig. 2.11 and obtained typing

```
CPoint(noDriftModel,param1=t0.est,param2=t1.est, plot=TRUE)
```

We can now refine the estimate of θ at the first stage making use of the change point estimate:

```
qmleL(noDriftModel, t=t.est3$tau,
 start=list(theta1.k=0.1, theta2.k=0.1),
 lower=list(theta1.k=0, theta2.k=0),
 upper=list(theta1.k=1, theta2.k=1),
 method="L-BFGS-B") -> estL
qmleR(noDriftModel, t=t.est3$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
t02s.est <- coef(estL)
t12s.est <- coef(estR)
t2s.est3 <- CPoint(noDriftModel,param1=t02s.est,param2=t12s.est)
t2s.est3

## $tau
## [1] 3.98
##
## $param1
##   theta1.k  theta2.k
## 0.4859283 0.2995279
##
## $param2
##  theta1.k  theta2.k
## 0.2036140 0.4087094
```

and these new estimates of the second stage are qualitatively better than the estimates at the first stage. There is no need to further estimate τ.

2.10.3 *Example of Volatility Change Point Estimation in Real Data*

We now apply the change point analysis to real stock data. We consider for our experiment the Apple stock exchange, and we focus on the adjusted values. For simplicity, we assume a geometric Brownian motion model $\mathrm{d}X_t = \mu X_t \mathrm{d}t + \sigma X_t \mathrm{d}W_t$:

```
library(quantmod)
getSymbols("AAPL", to="2016-12-31")

## [1] "AAPL"

S <- AAPL$AAPL.Adjusted
Delta <- 1/252
gBm <- setModel(drift="mu*x", diffusion="sigma*x")
mod <- setYuima(model=gBm, data=setData(S, delta=Delta))
lower <- list(mu=0.1, sigma=0.1)
upper <- list(mu=100, sigma=10)
start <- list(mu=1, sigma=1)
fit <- qmle(mod, start= start, upper=upper, lower=lower)
summary(fit)

## Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = mod, start = start, lower = lower, upper = upper)
##
## Coefficients:
##        Estimate  Std. Error
## sigma 0.3320259 0.004741881
## mu    0.2909865 0.105058353
##
## -2 log L: 6799.443
```

Now we prepare initial estimates for (μ, σ) using the very beginning and the very end of the time series

```
fit1 <- qmleL(mod, t=1, start= list(mu=1,sigma=1))
fit2 <- qmleR(mod, t=6, start= list(mu=1,sigma=1))
fit1

##
```

```
## Call:
## qmle(yuima = <S4 object of class "yuima">, start = list(mu = 1,
##     sigma = 1))
##
## Coefficients:
##     sigma        mu
## 0.3759766 0.9164062

fit2

##
## Call:
## qmle(yuima = <S4 object of class "yuima">, start = list(mu = 1,
##     sigma = 1))
##
## Coefficients:
##     sigma        mu
## 0.2567383 0.1625000
```

The above estimates indeed look different. We now look at the change point estimate using these initial guess

```
cp <- CPoint(mod,param1=coef(fit1),param2=coef(fit2))
cp

## $tau
## [1] 2.384921
##
## $param1
##     sigma        mu
## 0.3759766 0.9164062
##
## $param2
##     sigma        mu
## 0.2567383 0.1625000
```

To check if this change point is meaningful, one can look at Fig. 2.12 which shows the nonconstant volatility log-returns and the value of the change point. The change point time τ has been estimated as 2.385. A quick inspection of the plot shows that the volatility of the stock is higher ($\hat{\sigma} = 0.376$) before τ and lower ($\hat{\sigma} = 0.257$) in the second part of the trajectory (Fig. 2.12). The change-point statistics reaches its minimum at τ (see Fig 2.13)

```
X <- diff(log(get.zoo.data(mod)[[1]]))
plot(X)
abline(v=cp$tau, lty=3,lwd=2,col="red")
```

2.11 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 between the two

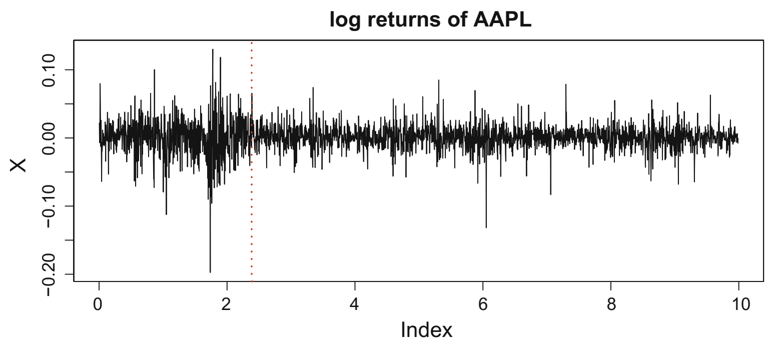


Fig. 2.12 Log-returns of the Apple stock indeed show non constant volatility. The dotted line represents the time of the change point

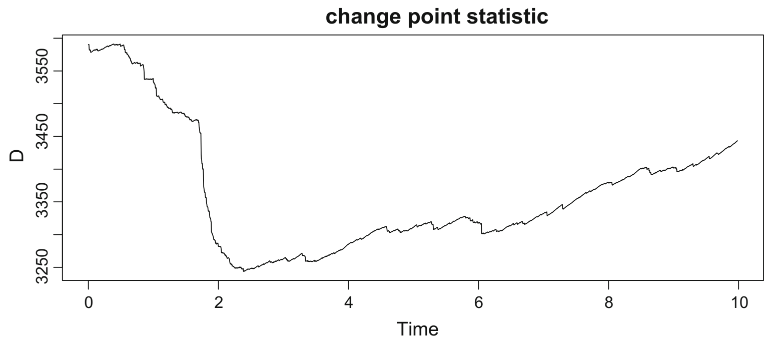


Fig. 2.13 Change point statistic reaches the minimum at change point date for the AAPL stock

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

$$\begin{aligned} \mathrm{d}X_{l,t} &= \mu_{l,t}\mathrm{d}t + \sigma_{l,t}\mathrm{d}W_{l,t}, \quad t \in [0, T] \\ X_{l,0} &= x_{l,0} \end{aligned}$$

for $l = 1, 2$. Here, W_l denote standard Wiener processes with a progressively measurable correlation process $\mathrm{d}\langle W_1, W_2\rangle_t = \rho_t \mathrm{d}t$, $\mu_{l,t}$ 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, however, 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 $T^{l,i}$ 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. The two sequences of stopping times $T^{1,i}$ and $T^{2,j}$ 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:

$$\theta = \langle X_1, X_2\rangle_T = \int_0^T \sigma_{1,t}\sigma_{2,t}\rho_t \mathrm{d}t. \tag{2.18}$$

The target variable θ is random in general, and it can be estimated with the nonsynchronous covariance estimator (Hayashi–Yoshida estimator)

$$U_n = \sum_{i,j:T^{1,i}\le T, T^{2,j}\le T} \Delta X_1(I^{1,i})\Delta X_2(J^{2,j})1_{\{I^{1,i}\bigcap J^{2,j}\neq\emptyset\}}. \tag{2.19}$$

That is, the product of any pair of increments $\Delta X_1(I^{1,i}) = (X_{1,T^{1,i}} - X_{1,T^{1,i-1}})$ and $\Delta X_2(J^{2,j}) = (X_{2,T^{2,j}} - X_{2,T^{2,j-1}})$ will make a contribution to the sum only when the respective observation intervals $I^{1,i} = (T^{1,i-1}, T^{1,i}]$ and $J^{2,j} = (T^{2,j-1}, T^{2,j}]$ 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, 2008a, 2006, 2008b) for details.

2.11.1 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:

$$\begin{aligned} \mathrm{d}X_{1,t} &= \sigma_{1,t}\mathrm{d}B_{1,t}, \\ \mathrm{d}X_{2,t} &= \sigma_{2,t}\mathrm{d}B_{2,t}. \end{aligned} \tag{2.20}$$

Here, $B_{1,t}$ and $B_{2,t}$ denote two standard Wiener processes; however, they are correlated as

$$B_{1,t} = W_{1,t}, \tag{2.21}$$

$$B_{2,t} = \int_0^t \rho_s \mathrm{d}W_{1,s} + \int_0^t \sqrt{1-\rho_s^2}\mathrm{d}W_{2,s}, \tag{2.22}$$

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

$$\sigma_{1,t} = \sqrt{1+t},$$
$$\sigma_{2,t} = \sqrt{1+t^2},$$
$$\rho_t = \frac{1}{\sqrt{2}}.$$

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.

```
# diffusion coefficient for process 1
diff.coef.1 <- function(t,x1=0, x2=0) sqrt(1+t)
# diffusion coefficient for process 2
diff.coef.2 <- function(t,x1=0, x2=0) sqrt(1+t^2)
# correlation
cor.rho <- function(t,x1=0, x2=0) sqrt(1/2)
# coefficient matrix for diffusion term
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)
# Model SDE using yuima.model
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:

$$\theta = \langle X_1, X_2 \rangle_T = \int_0^T \sigma_{1,t}\sigma_{2,t}\rho_t \mathrm{d}t. \tag{2.23}$$

Later, we will compare estimated values with the true value of θ given by

```
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 the independent Poisson sampling. That is, each configuration of the sampling times $T^{l,i}$ is realized as the Poisson random measure with intensity np_l, and the two random measures are independent each other as well as the stochastic processes. Under this scheme, the data become asynchronous. It is known that

$$n^{1/2}(U_n - \theta) \to N(0, c), \tag{2.24}$$

as $n \to \infty$, where

$$c = \left(\frac{2}{p_1} + \frac{2}{p_2}\right) \int_0^T \left(\sigma_{1,t}\sigma_{2,t}\right)^2 \mathrm{d}t + \left(\frac{2}{p_1} + \frac{2}{p_2} - \frac{2}{p_1+p_2}\right) \int_0^T \left(\sigma_{1,t}\sigma_{2,t}\rho_t\right)^2 \mathrm{d}t. \tag{2.25}$$

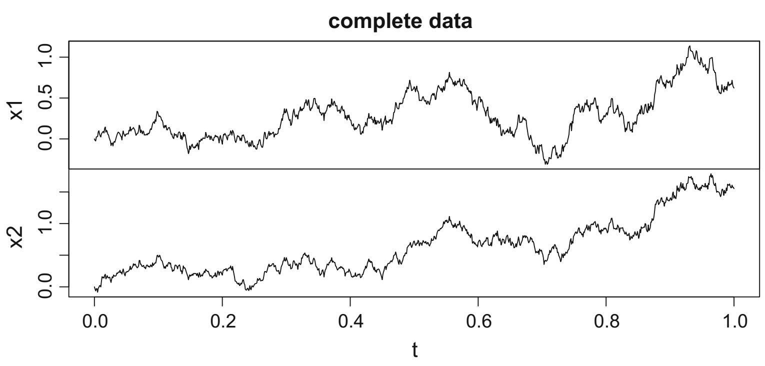


Fig. 2.14 Complete simulated data

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

## theta=1.000
```

so in our case $\theta = 1$.

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

`cce` takes the sample and returns an estimate of the quadratic covariation. For example, for the complete data in Fig. 2.14, we obtain the following estimates:

```
cce(X)

## $covmat
##          Series 1 Series 2
## Series 1 1.490955 1.085304
## Series 2 1.085304 1.473602
##
## $cormat
##           Series 1  Series 2
## Series 1 1.0000000 0.7321991
## Series 2 0.7321991 1.0000000
```

```
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 ith component of X, we specify an

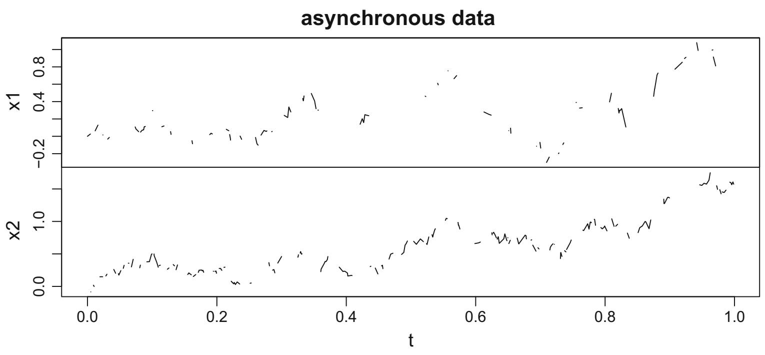


Fig. 2.15 Asynchronous data generated from the simulated ones using Poisson random subsampling

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

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

Now, we use the `subsampling` function to subsample the original data `X` into new asynchronous data `Y` (see Fig. 2.15)

```
Y <- subsampling(X, sampling=newsamp)
```

```
plot(Y,main="asynchronous data")
```

We calculate the covariance estimator on the asynchronous data `Y`

```
cce(Y)$covmat    # asynch data

##           Series 1 Series 2
## Series 1 1.396354 1.083400
## Series 2 1.083400 1.265823

cce(X)$covmat    # full data

##           Series 1 Series 2
## Series 1 1.490955 1.085304
## Series 2 1.085304 1.473602
```

and we obtain an unbiased estimate of the covariance.

2.11.2 Asynchronous Estimation for Nonlinear Systems

Consider now the two-dimensional system with nonlinear feedback

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

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 generate data from it

```
b1 <- function(x,y) y
b2 <- function(x,y) -x
s1 <- function(t,x,y) sqrt(abs(x)*(1+t))
s2 <- function(t,x,y) sqrt(abs(y))
cor.rho <- function(t,x,y) 1/(1+x^2)
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)
cor.mod <- setModel(drift = c("b1","b2"), diffusion = diff.mat,
solve.variable = c("x", "y"),state.var=c("x","y"))

## Generate a path of the process
set.seed(111)
Terminal <- 1
n <- 10000
yuima.samp <- setSampling(Terminal = Terminal, n = n)
yuima <- setYuima(model = cor.mod, sampling = yuima.samp)
yuima <- simulate(yuima, xinit=c(2,3))
```

We apply the same Poisson random sampling so that the object Y will contain asynchronous data (see Fig. 2.16)

```
p1 <- 0.2
p2 <- 0.3
newsamp <- setSampling(random=list(rdist=c(
 function(x) rexp(x, rate=p1*n/Terminal),
 function(x) rexp(x, rate=p2*n/Terminal))) )
Y <- subsampling(yuima, sampling = newsamp)
```

We can plot again the asynchronous data as in Fig. 2.16.

```
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`

```
cce(yuima)$covmat # full data

##           Series 1 Series 2
## Series 1 2.709112 0.780349
## Series 2 0.780349 3.470497


cce(Y)$covmat        # asynch data
```

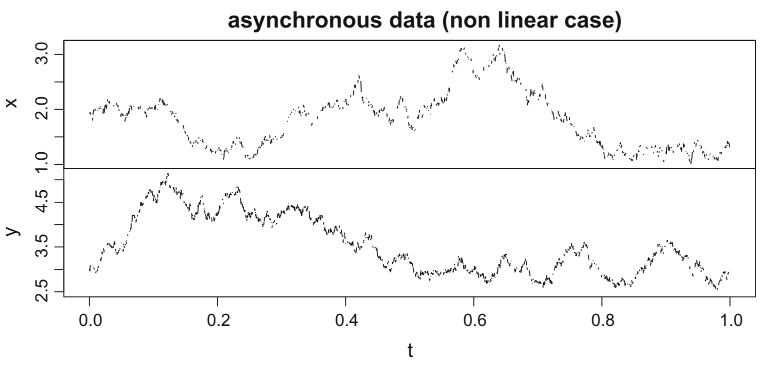


Fig. 2.16 Asynchronous data for the nonlinear system

```
##            Series 1  Series 2
## Series 1 2.7132456 0.7600877
## Series 2 0.7600877 3.3807434
```

Notice that the argument `psd` of `cce` gives a positive semi-definite version of estimated matrices by the HY-estimator.

2.11.3 Other Covariance Estimators

The `cce` command also evaluates other type of covariance estimators proposed in the literature. The default estimator is the Hayashi and Yoshida (2005) estimator, but the argument `method` accepts several options listed below:

- `method="HY"`: default, the Hayashi and Yoshida (2005) estimator;
- `method="PHY"`: the pre-averaged Hayashi–Yoshida estimator proposed in Christensen et al. (2010);
- `method="MRC"`: the modulated realized covariance based on refresh time sampling proposed in Christensen et al. (2010);
- `method="TSCV"`: the previous tick two-scale realized covariance based on refresh time sampling proposed in Zhang (2011);
- `method="GME"`: the generalized multiscale estimator proposed in Bibinger (2011);
- `method="RK"`: the multivariate realized kernel based on refresh time sampling proposed in Barndorff-Nielsen et al. (2011);
- `method="QMLE"`: the nonparametric quasi-maximum likelihood estimator proposed in Ait-Sahalia et al. (2010);
- `method="SIML"`: the separating information maximum likelihood estimator proposed in Kunitomo and Sato (2008) with the basis of refresh time sampling;

- `method="THY"`: the truncated Hayashi–Yoshida estimator proposed in Mancini and Gobbi (2012);
- `method="PHTY"`: the pre-averaged truncated Hayashi–Yoshida estimator, which is a thresholding version of the pre-averaged Hayashi–Yoshida estimator (Christensen et al. 2010; Koike 2014);
- `method="SRC"`: the calendar time subsampled realized covariance.
- `method="SBPC"`: the calendar time subsampled realized bipower covariation.

For details on the different use of the `cce` function, we refer the reader to the manual page of the command or use `?cce`.

2.12 Lead–Lag Estimation

Market participants usually agree that certain pairs of assets (X_1, X_2) share a "lead–lag effect", in the sense that the lagger (or follower) price process Y tends to partially reproduce the oscillations of the leader (or driver) price process X, with some temporal delay, or vice versa. This property is usually referred to as the "lead–lag effect". The lead–lag effect may have some importance in practice, when assessing the quality of risk management indicators, for instance, or, more generally, when considering statistical arbitrage strategies. Also, note that it can be measured at various temporal scales (daily, hourly or even at the level of seconds, for flow products traded on electronic markets). The lead–lag effect is a concept of common practice that has some history in financial econometrics. In time series analysis for instance, this notion can be linked to the concept of Granger causality, and we refer to Comte and Renaut (1996) for a general approach. From a phenomenological perspective, the lead–lag effect is supported by empirical evidence reported in Chiao et al. (2004), de Jong and Nijman (1997) and Kang et al. (2006), together with Robert and Rosenbaum (2011) and the references therein.

The **yuima** package implements the lead–lag estimator recently proposed Hoffmann et al. (2013) which is based on the asynchronous covariance estimator of Sect. 2.11. Let $\theta \in (-\delta, \delta)$ be the time lag between the two diffusion processes X_1 and X_2. Roughly speaking, the idea is to construct a contrast function $U_n(\theta)$ which evaluates the Hayashi–Yoshida estimator in formula (2.19) for the times series $X_{1,t}$ and $X_{2,t+\theta}$ and then to maximize it as a function of θ; i.e., using the same notation of Sect. 2.11, the contrast function is given by

$$\begin{aligned} U_n(\theta) = & 1_{\{\theta \geq 0\}} \sum_{i,j:T^{1,i}\leq T, T^{2,j}\leq T} \Delta X_1(I^{1,i}) \Delta X_2(J^{2,j}) 1_{\{I^{1,i} \bigcap J^{2,j}_{-\theta} \neq \emptyset\}} \\ & + 1_{\{\theta < 0\}} \sum_{i,j:T^{1,i}\leq T, T^{2,j}\leq T} \Delta X_1(I^{1,i}) \Delta X_2(J^{2,j}) 1_{\{I^{1,i}_{\theta} \bigcap J^{2,j} \neq \emptyset\}} \end{aligned}$$

where $J^{2,j}_{-\theta} = (T^{2,j-1} - \theta, T^{2,j} - \theta]$ and $I^{1,i}_{\theta} = (T^{1,i-1} + \theta, T^{1,i} + \theta]$. The lead–lag estimator $\hat{\theta}_n$ of θ is defined as

$$\hat{\theta}_n = \underset{-\delta<\theta<+\delta}{\operatorname{argmax}} |U_n(\theta)|.$$

Let us consider the following three-dimensional diffusion process

$$\begin{pmatrix} \mathrm{d}X_{1,t} \\ \mathrm{d}X_{2,t} \\ \mathrm{d}X_{3,t} \end{pmatrix} = \begin{pmatrix} 1 - X_{1,t} \\ 2 \cdot (10 - X_{2,t}) \\ 3 \cdot (4 - X_{3,t}) \end{pmatrix} \mathrm{d}t + \begin{bmatrix} \sqrt{X_{1,t}} & 0 & 0 \\ 3/5 \cdot \sqrt{X_{2,t}} & 4/5 \cdot \sqrt{X_{2,t}} & 0 \\ 1/3 \cdot \sqrt{X_{3,t}} & 2/3 \cdot \sqrt{X_{3,t}} & 2/3 \cdot \sqrt{X_{3,t}} \end{bmatrix} \begin{pmatrix} \mathrm{d}W_{1,t} \\ \mathrm{d}W_{2,t} \\ \mathrm{d}W_{3,t} \end{pmatrix}$$

We generate data from this model and then artificially change the time of the second and third processes, respectively, by $\theta_2 = 0.05$ and $\theta_3 = 0.12$.

```r
diff.coef.matrix <- matrix(c("sqrt(x1)", "3/5*sqrt(x2)",
 "1/3*sqrt(x3)", "", "4/5*sqrt(x2)","2/3*sqrt(x3)","","",
  "2/3*sqrt(x3)"), 3, 3)
drift <- c("1-x1","2*(10-x2)","3*(4-x3)")
cor.mod <- setModel(drift = drift, diffusion = diff.coef.matrix,
  solve.variable = c("x1", "x2","x3"))

set.seed(111)
Terminal <- 1
yuima.samp <- setSampling(Terminal = Terminal, n = 1200)
yuima <- setYuima(model = cor.mod, sampling = yuima.samp)
yuima <- simulate(yuima, xinit=c(1,7,5))

# intentionally displace the second time series

data1 <- get.zoo.data(yuima)[[1]]
data2 <- get.zoo.data(yuima)[[2]]
time2 <- time( data2 )
theta2 <- 0.05   # the lag of x2 behind x1
stime2 <- time2 + theta2
time(data2) <- stime2
data3 <- get.zoo.data(yuima)[[3]]
time3 <- time( data3 )
theta3 <- 0.12   # the lag of x3 behind x1
stime3 <- time3 + theta3
time(data3) <- stime3
syuima <- setYuima(data=setData(merge(data1, data2, data3)))
yuima

##
## Diffusion process
## Number of equations: 3
## Number of Wiener noises: 3
##
## Number of original time series: 3
```

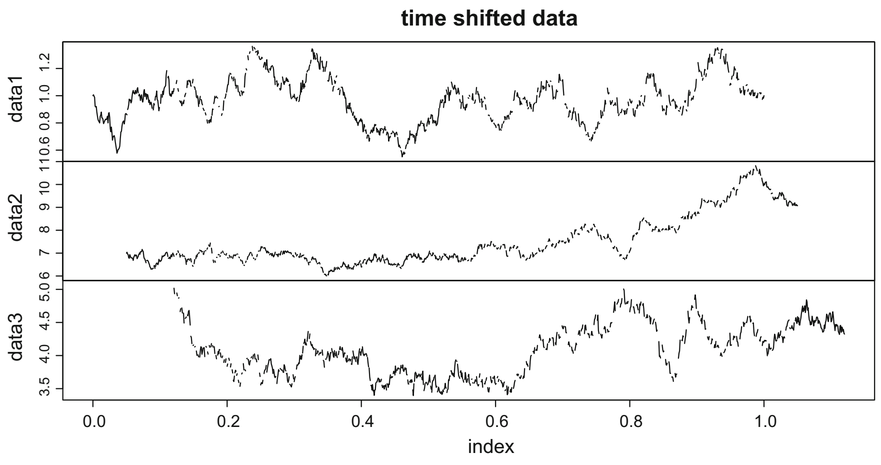


Fig. 2.17 Simulated data shifted in time contained in the object `syuima`

```
## length = 1201, time range [0 ; 1]
##
## Number of zoo time series: 3
##          length time.min time.max        delta
## Series 1   1201        0        1 0.0008333333
## Series 2   1201        0        1 0.0008333333
## Series 3   1201        0        1 0.0008333333

syuima

##
##
## Number of original time series: 3
## length = 1842, time range [0 ; 1.12]
##
## Number of zoo time series: 3
##       length time.min time.max        delta note
## data1   1842        0     1.12 0.0008333333    *
## data2   1842        0     1.12 0.0008333333    *
## data3   1842        0     1.12 0.0008333333    *
## ================
## * : maximal mesh
```

Fig. 2.17 shows clearly the effect of these shifting in time

```
plot(syuima,main="time shifted data")
```

We now apply the lead–lag estimator `llag` to the original data `yuima` and to the shifted data `syuima`

```
llag(yuima)

##          Series 1 Series 2 Series 3
```

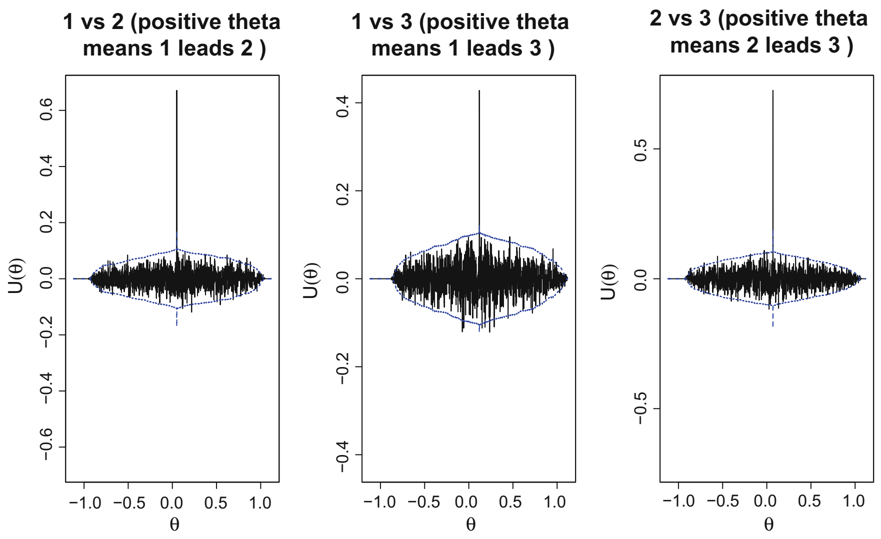


Fig. 2.18 Confidence interval lines around the $U(\theta)$ contrast function. This is the output of the `plot` method for the `llag` object. The function $U(\theta)$ crosses the lines at the point of the lag estimate

```
## Series 1         0         0         0
## Series 2         0         0         0
## Series 3         0         0         0

llag(syuima)

##             data1       data2     data3
## data1  0.00000000  0.04985885 0.1197828
## data2 -0.04985885  0.00000000 0.0699240
## data3 -0.11978284 -0.06992400 0.0000000
```

The `llag` function returns also the p value associated to each lag estimate and the pointwise confidence intervals by specifying the argument `ci=TRUE`. The `plot` method for `llag` produces a graphical representation of the same information as shown in Fig. 2.18 obtained by typing `plot(llag(syuima,ci=TRUE))`.

From the above result, we can see that the original data present no lag, while for the shifted data we obtain the expected result. We can do one step further; i.e., to avoid the suspect that the initial delay is the cause of this precise estimation, we cut the series in a window of time on which all processes take values, for example for $t \in (0.5, 1)$

```
data2 <- get.zoo.data(yuima)[[2]]
time2 <- time( data2 )
theta2 <- 0.05   # the lag of x2 behind x1
stime2 <- time2 + theta2
time(data2) <- stime2
data3 <- get.zoo.data(yuima)[[3]]
```

```
time3 <- time( data3 )
theta3 <- 0.12   # the lag of x3 behind x1
stime3 <- time3 + theta3
time(data3) <- stime3
data1 <- data1[which(time(data1)>0.5 & time(data1)<1)]
data2 <- data2[which(time(data2)>0.5 & time(data2)<1)]
data3 <- data3[which(time(data3)>0.5 & time(data3)<1)]
syuima2 <- setYuima(data=setData(merge(data1, data2, data3)))
syuima2

##
##
## Number of original time series: 3
## length = 882, time range [0.500833333333333 ; 0.999166666666667]
##
## Number of zoo time series: 3
##       length time.min time.max        delta note
## data1    882    0.501    0.999 0.0008333333    *
## data2    882    0.501    0.999 0.0008333333    *
## data3    882    0.501    0.999 0.0008333333    *
## ================
## * : maximal mesh

llag(syuima2)

##             data1       data2      data3
## data1  0.00000000  0.04972033 0.11978080
## data2 -0.04972033  0.00000000 0.06949546
## data3 -0.11978080 -0.06949546 0.00000000
```

and, as we can see, this is not an issue for the estimator. Furthermore, the lead–lag estimator of Hoffmann et al. (2013) also works for asynchronous data. To this aim, we perform Poisson random sampling on the data to obtain asynchronous series and we re-estimate the lead–lag parameters

```
p1 <- 0.2
p2 <- 0.3
p3 <- 0.4
n <- 1000
newsamp <- setSampling(
random=list(rdist=c( function(x) rexp(x, rate=p1*n/Terminal),
function(x) rexp(x, rate=p2*n/Terminal),
function(x) rexp(x, rate=p3*n/Terminal))) )
psample <- subsampling(syuima, sampling = newsamp)
psample

##
##
## Number of original time series: 3
## length = 1842, time range [0 ; 1.12]
##
## Number of zoo time series: 3
##       length time.min time.max      delta note
## data1    250        0    1.117 0.03091730    *
## data2    370        0    1.118 0.01631701    *
## data3    419        0    1.119 0.01712454    *
## ================
## * : maximal mesh

llag(psample)

##              data1       data2      data3
## data1  0.00000000  0.05165984 0.12103949
## data2 -0.05165984  0.00000000 0.07004067
## data3 -0.12103949 -0.07004067 0.00000000
```

and still the estimator works as expected.

2.12.1 Application of the Lead–Lag Estimator to Real Data

We now consider the daily closing values for the year 2013 of six IT companies: Apple, IBM, Amazon, EBay, Facebook and Microsoft. We run a lead–lag analysis just to verify if there is any leader among these assets.

```
library(quantmod)
getSymbols("AAPL", from="2013-01-01", to="2013-12-31")
getSymbols("IBM", from="2013-01-01", to="2013-12-31")
getSymbols("AMZN", from="2013-01-01", to="2013-12-31")
getSymbols("EBAY", from="2013-01-01", to="2013-12-31")
getSymbols("FB", from="2013-01-01", to="2013-12-31")
getSymbols("MSFT", from="2013-01-01", to="2013-12-31")
data1 <- AAPL$AAPL.Close
data2 <- IBM$IBM.Close
data3 <- AMZN$AMZN.Close
data4 <- EBAY$EBAY.Close
data5 <- FB$FB.Close
data6 <- MSFT$MSFT.Close
market.data <- merge(data1, data2, data3, data4,data5,data6)
colnames(market.data) <- c("AAPL", "IBM", "AMZN", "EBAY",
 "FB", "MSFT")
mkt <- setYuima(data=setData(market.data, delta=1/252))
```

```
mkt

##
##
## Number of original time series: 6
## length = 251, time range [2013-01-02 ; 2013-12-30]
##
## Number of zoo time series: 6
##      length time.min time.max       delta
## AAPL    251        0    0.992 0.003968254
## IBM     251        0    0.992 0.003968254
## AMZN    251        0    0.992 0.003968254
## EBAY    251        0    0.992 0.003968254
## FB      251        0    0.992 0.003968254
## MSFT    251        0    0.992 0.003968254
```

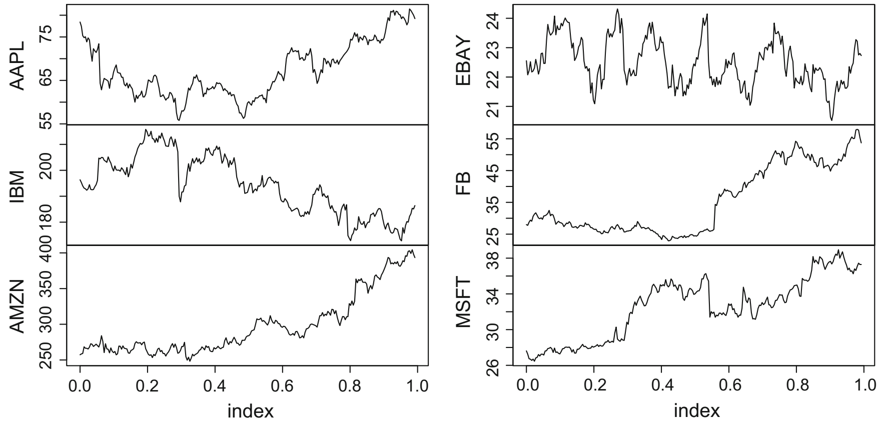


Fig. 2.19 Closing values of six time series in 2013. Which one is the leader?

```
round(cce(mkt)$cormat,2) # correlation matrix

##        AAPL  IBM  AMZN EBAY   FB MSFT
## AAPL   1.00 0.11 -0.01 0.05 0.13 0.06
## IBM    0.11 1.00  0.16 0.33 0.03 0.17
## AMZN  -0.01 0.16  1.00 0.36 0.29 0.25
## EBAY   0.05 0.33  0.36 1.00 0.20 0.23
## FB     0.13 0.03  0.29 0.20 1.00 0.10
## MSFT   0.06 0.17  0.25 0.23 0.10 1.00
```

Looking at the data (see Fig. 2.19) and at the correlation matrix, we see that there is some non-negligible link between the time series, as expected given that they belong to the same sector.

```
plot(mkt)
```

We now look at the lead–lag estimator

```
round(llag(mkt),4)

##          AAPL     IBM    AMZN    EBAY      FB    MSFT
## AAPL   0.0000  0.2332  0.2727  0.2213  0.4980  0.4822
## IBM   -0.2332  0.0000  0.5217 -0.0040  0.6047  0.2253
## AMZN  -0.2727 -0.5217  0.0000  0.0040 -0.0040  0.0040
## EBAY  -0.2213  0.0040 -0.0040  0.0000 -0.0040  0.0040
## FB    -0.4980 -0.6047  0.0040  0.0040  0.0000 -0.2648
## MSFT  -0.4822 -0.2253 -0.0040 -0.0040  0.2648  0.0000
```

and we see a clear leader, which is Apple, as probably expected. Figure 2.20 presents both the correlation matrix and the lead–lag matrix in graphical form and has been obtained using the following R commands:

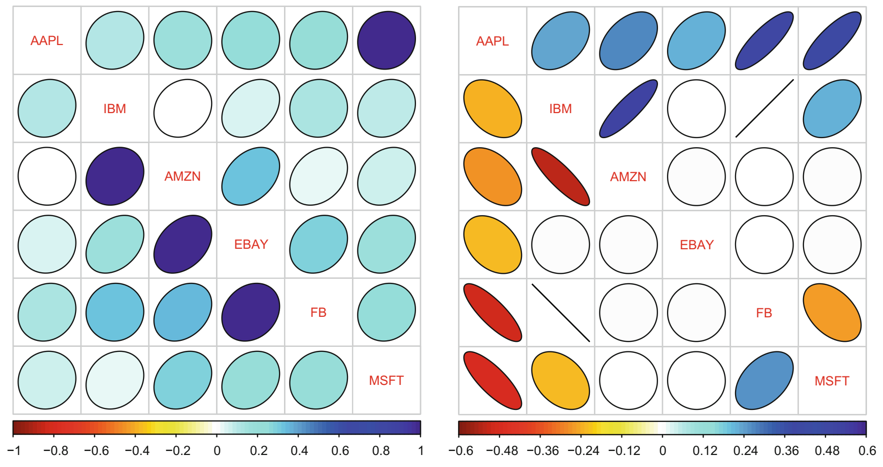


Fig. 2.20 Correlation matrix (left) and lead–lag estimation matrix (right)

```
require(corrplot)
cols <- colorRampPalette(c("#7F0000", "red", "#FF7F00",
  "yellow", "white", "cyan",
  "#007FFF", "blue", "#00007F"))
corrplot(cce(mkt)$cormat,method="ellipse",
 cl.pos = "b", tl.pos = "d", tl.srt = 60,
 col=cols(100), outline=TRUE)
corrplot(llag(mkt),method="ellipse",is.corr=FALSE,
 cl.pos = "b", tl.pos = "d", tl.srt = 60,
 col=cols(100), outline=TRUE)
```

2.13 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 (Iacus 2008). 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 computational 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

$$X_t^{(\varepsilon)} = x_0 + \int_0^t a(X_s^{(\varepsilon)},\varepsilon)\mathrm{d}s + \int_0^t b(X_s^{(\varepsilon)},\varepsilon)\mathrm{d}W_s, \qquad t \in [0,T] \tag{2.26}$$

for $\varepsilon \in (0, 1]$, where $W_t = (W_{1,t}, \dots, W_{r,t})$ is an r-dimensional Wiener process. A functional of interest is expressed in the following abstract form:

$$F^{(\varepsilon)} = \sum_{\alpha=0}^{r} \int_0^T f_\alpha(X_t^{(\varepsilon)}, \varepsilon) \mathrm{d}W_t^\alpha + F(X_T^{(\varepsilon)}, \varepsilon), \qquad W_t^0 = t. \tag{2.27}$$

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

$$\mathrm{d}X_t^{(\varepsilon)} = \mu X_t^{(\varepsilon)} \mathrm{d}t + \varepsilon X_t^{(\varepsilon)} \mathrm{d}W_t, \tag{2.28}$$

the price of the option under zero interest rate is of the form

$$\mathbb{E}\left[\max\left(\frac{1}{T}\int_0^T X_t^{(\varepsilon)}\mathrm{d}t - K, 0\right)\right].$$

Thus, the functional of interest is

$$F^{(\varepsilon)} = \frac{1}{T}\int_0^T X_t^{(\varepsilon)}\mathrm{d}t, \qquad r = 1$$

with

$$f_0(x, \varepsilon) = \frac{x}{T}, \quad f_1(x, \varepsilon) = 0, \quad F(x, \varepsilon) = 0$$

in (2.27). Similarly, for $F(x, \varepsilon) = x$, the functional becomes $F^{(\varepsilon)} = X_T^{(\varepsilon)}$ and the price of the European call option is $\mathbb{E}[\max(X_T^{(\varepsilon)} - K, 0)]$. 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 (2.26)–(2.27), we will assume that the stochastic system is deterministic in the limit as $\varepsilon \downarrow 0$, that is,

$$b(\cdot, 0) = 0 \quad \text{and} \quad f_\alpha(\cdot, 0) = 0 \quad (\alpha = 1, ..., r).$$

Since $X_t^{(0)}$ is the deterministic solution to the ordinary differential equation

$$\frac{\mathrm{d}X_t^{(0)}}{\mathrm{d}t} = a(X_t^{(0)}, 0), \quad X_0^{(0)} = x_0,$$

the functional $F^{(0)}$ becomes a constant:

$$F^{(0)} = \int_0^T f_0(X_t^{(0)}, 0)\mathrm{d}t + F(X_T^{(0)}, 0). \tag{2.29}$$

Under standard regularity of a, b, f_α and F, it is possible to show $F^{(\varepsilon)}$ has a version that is smooth in $\varepsilon \in [0, 1)$ almost surely, and hence,

$$\tilde{F}^{(\varepsilon)} := \varepsilon^{-1}(F^{(\varepsilon)} - F^{(0)})$$

admits a stochastic expansion

$$\tilde{F}^{(\varepsilon)} \sim \tilde{F}^{[0]} + \varepsilon \tilde{F}^{[1]} + \varepsilon^2 \tilde{F}^{[2]} + \cdots \tag{2.30}$$

as $\varepsilon \downarrow 0$. This stochastic expansion makes sense in the Sobolev spaces of the Malliavin calculus. Then the so-called Watanabe's theory (Watanabe 1987) validates the asymptotic expansion of the (generalized) expectation

$$\mathbb{E}[g(\tilde{F}^{(\varepsilon)})] \sim d_0(g) + \varepsilon d_1(g) + \varepsilon^2 d_2(g) + \cdots \tag{2.31}$$

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 $\tilde{F}^{(\varepsilon)}$.[1] 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 $v = \mathrm{Cov}[\tilde{F}^{(0)}]$. Polynomials p_i are given by the conditional expectation of multiple Wiener integrals. The expansion (2.31) 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 formulas. The **yuima** package provides functions to construct the functional $F^{(\varepsilon)}$ and perform automatic asymptotic expansion based on the Malliavin calculus starting from a `yuima` object. This asymptotic expansion approach to option pricing was proposed in the early 1990s (Yoshida 1992a; Takahashi 1999; Kunitomo and Takahashi 2001), and a huge number of related papers are available today.

Though the method can be applied to the nonlinear system (2.26)–(2.27), just as an example, we shall consider the Asian call option of the geometric Brownian motion of equation (2.28) with $\mu = 1$ and $x_0 = 1$, and

$$g(x) = \max\left(F^{(0)} - K + \varepsilon x, 0\right) \tag{2.32}$$

Set the model (2.26) and the functional (2.27) as follows:

```
model <- setModel(drift = "x", diffusion = matrix( "x*e", 1,1))
T <- 1
xinit <- 150
K <- 100
```

[1]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 $\tilde{F}^{(0)}$.

```
f <- list( expression(x/T), expression(0))
F <- 0
e <- 0.5
yuima <- setYuima(model = model,
  sampling = setSampling(Terminal=T, n=1000))
yuima <- setFunctional( yuima, f=f,F=F, xinit=xinit,e=e)
```

This time the `setFunctional` command fills the appropriate slots inside the `yuima` object

```
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 150
##   ..@ e    : num 0.5
```

Then the limit $F^{(0)}$ of $F^{(\varepsilon)}$ is easily obtained by calling the function `F0` on the `yuima` object:

```
F0 <- F0(yuima)
F0

## [1] 257.6134
```

Set the function g according to (2.32):

```
rho <- expression(0)
epsilon <- e  # noise level
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 $\mathbb{E}[\max(F^{(\varepsilon)} - K, 0)]$ by applying the function `asymptotic_term`:

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

Then the sums

```
asy1 <- asymp$d0 + e * asymp$d1
# 1st order asymp. exp. of asian call price
asy1

## [1] 156.608

asy2 <- asymp$d0 + e * asymp$d1 +  e^2* asymp$d2
# 2nd order asymp. exp. of asian call price
asy2
```

```
##           [,1]
## [1,] 157.6082
```

give the first- and second-order asymptotic expansions, respectively.

We remark that the expansion of $\mathbb{E}[g(\tilde{F}^{(\varepsilon)})G^{(\varepsilon)}]$ is also possible by the same method for a functional $G^{(\varepsilon)}$ having a stochastic expansion like (2.30). Thus, the 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).

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

## [1] 157.7712
```

and the relative difference between the two approximations is –0.1%.

2.13.1 *Asymptotic Expansion for General Stochastic Processes*

Of course, **yuima** approach is more general in that the above Lévy approximation only holds when the process X_t is a geometric Brownian motion. We now give an example when the underlying process X_t is the following CIR model of Sect. 2.1.5:

$$\mathrm{d}X_t = 0.9X_t\mathrm{d}t + \varepsilon\sqrt{X_t}\mathrm{d}W_t, \quad X_0 = 1$$

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

```
a <- 0.9
e <- 0.4
Terminal <- 3

xinit <- 1
K <- 10

drift <- "a * x"
diffusion <- "e * sqrt(x)"

model <- setModel(drift=drift,diffusion=diffusion)

n <- 1000*Terminal
yuima <- setYuima(model = model,
  sampling = setSampling(Terminal=Terminal,n=n))
```

```
f <- list(c(expression(0)),c(expression(0)))
F <- expression(x)

yuima.ae <- setFunctional(yuima,f=f,F=F,xinit=xinit,e=e)
rho <- expression(0)
F1 <- F0(yuima.ae)


get_ge <- function(x,epsilon,K,F0){
        tmp <- (F0 - K) + (epsilon * x[1])
        tmp[(epsilon * x[1]) > (K - F0)] <- 0
        return( - tmp )
}


g <- function(x){
        return(get_ge(x,e,K,F1))
}

time1 <- proc.time()
asymp <- asymptotic_term(yuima.ae,block=100,rho,g)

## [1] "compute X.t0"

time2 <- proc.time()
```

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

```
ae.value0 <- asymp$d0
ae.value0

## [1] 0.7219652

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

## [1] 0.5787545

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

## [1] 0.5617722

ae.time <- time2 - time1
ae.time

##    user  system elapsed
##   3.637   0.033   3.694
```

As it can be seen, the contribution of the terms corresponding to 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 computational time. The relative difference among the two estimates is 0.1%.

Chapter 3
Compound Poisson Processes

The *compound Poisson* process is defined as $M_t = m_0 + \sum_{i=1}^{N_t} Y_{\tau_i}$, where N_t is a Poisson process and Y_{τ_i} are the jumps at random times τ_i. As will be explained in Chap. 4, the compound Poisson process plays an important role in the construction of the Lévy process. The `yuima model` has some slots to describe the jump structure in a stochastic differential equation with jumps and in particular when those jumps are of compound Poisson type. Nevertheless, given the peculiarities of both the compound Poisson process and stochastic differential equations with jumps, **yuima** has an extension of the basic `yuima-model` class and a constructor method explicitly designed for the compound Poisson process.

To define a compound Poisson process in **yuima**, one needs to use the `setPoisson` function. Assume that N_t is a Poisson process with intensity λ, i.e. $N_t \sim \text{Poi}(\lambda t)$:

$$P(N_t = k) = e^{-\lambda t}\frac{(\lambda t)^k}{k!}, \quad k = 0, 1, 2, \dots$$

with $\mathbb{E}(N_t) = \lambda t$. To define such process in **yuima**, we proceed as follows

```
mod1 <- setPoisson(intensity="lambda", df=list("dconst(z,1)"))
mod1

##
## Compound Poisson process
## Number of equations: 1
## Parametric model with 1 parameters
```

where the argument `intensity` specifies the intensity function, in this case the constant λ, and `df` indicates the distribution of jumps. We have chosen `dconst`, i.e. constant jumps of size `1` in this case. We can then simulate the process by just specifying the sampling structure and the value of `lambda` as follows:

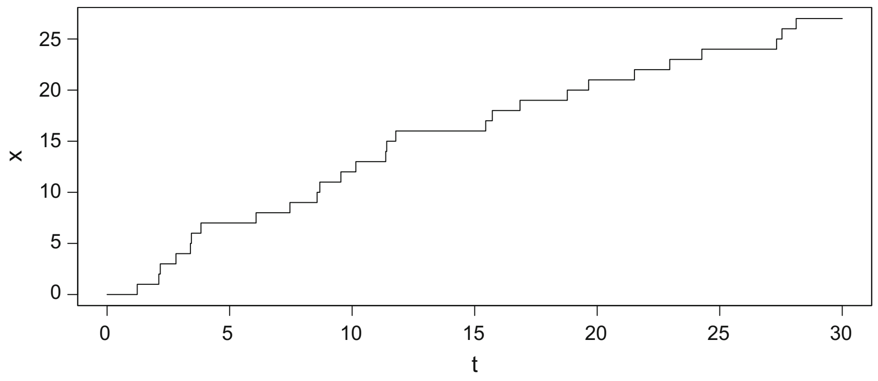


Fig. 3.1 A trajectory of homogeneous Poisson process with parameter $\lambda = 1$ as defined in `mod1`

```
Terminal <- 30
samp <- setSampling(T=Terminal,n=3000)
set.seed(123)
poisson1 <- simulate(mod1, true.par=list(lambda=1),sampling=samp)
poisson1

##
## Compound Poisson process
## Number of equations: 1
## Parametric model with 1 parameters
##
## Number of original time series: 1
## length = 29, time range [0 ; 30.1613615503751]
##
## Number of zoo time series: 1
##    length time.min time.max delta
## x    3001        0       30  0.01

plot(poisson1)
```

As usual, as **yuima** assumes the high-frequency set-up, the process is returned on a regular grid unless other sampling schemes are specified. Figure 3.1 shows a trajectory of the process.

The `setPoisson` function has two additional arguments: `scale`, which is a constant used to rescale the jumps and defaults to `1`, and `dimension` which is used to describe multidimensional compound Poisson processes. In the case of model `mod1`, the following to lines of code describe essentially the same model of homogeneous Poisson process with jumps of size 5

```
setPoisson(intensity="lambda", df=list("dconst(z,1)"), scale=5)
setPoisson(intensity="lambda", df=list("dconst(z,5)"))
```

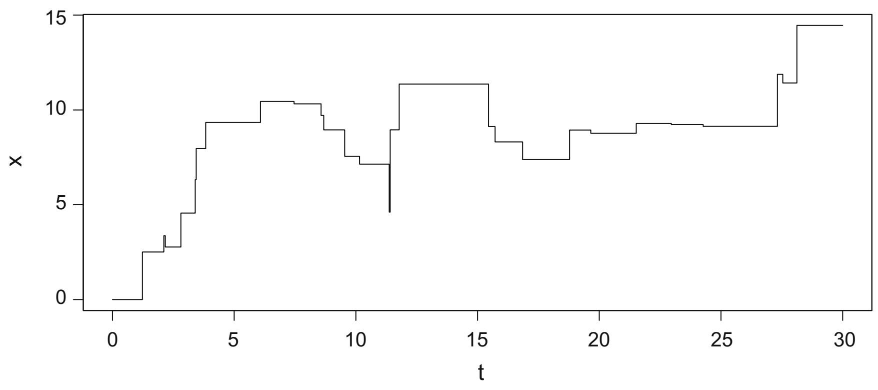


Fig. 3.2 A trajectory of homogeneous Poisson process with Gaussian jumps as defined in `mod2`

Any distribution can be specified for the jump distribution. For example, if we want to specify a compound Poisson process with Gaussian jumps, we can input the following code (see Fig. 3.2 for the simulated path):

```
mod2 <- setPoisson(intensity="lambda", df=list("dnorm(z,mu,sigma)"))
set.seed(123)
poisson2 <- simulate(mod2, sampling=samp,
 true.par=list(lambda=1,mu=0, sigma=2))
poisson2

##
## Compound Poisson process
## Number of equations: 1
## Parametric model with 3 parameters
##
## Number of original time series: 1
## length = 29, time range [0 ; 30.1613615503751]
##
## Number of zoo time series: 1
##    length time.min time.max delta
## x   3001        0       30  0.01

plot(poisson2)
```

Any other distribution for which a density function and a random number generator exist in R can be specified as well. For example, the next code specifies normal inverse Gaussian jumps using the `dNIG` density function and the corresponding random number generator `rNIG` existing in the **yuima** package:

```
mod3 <- setPoisson(intensity="lambda",
 df=list("dNIG(z,alpha,beta,gamma,mu)"))
poisson3 <- simulate(mod3, sampling=samp,
 true.par=list(lambda=10,alpha=2,beta=0.3,gamma=1,mu=0))
poisson3
```

```
##
## Compound Poisson process
## Number of equations: 1
## Parametric model with 5 parameters
##
## Number of original time series: 1
## length = 316, time range [0 ; 30.0762710060585]
##
## Number of zoo time series: 1
##   length time.min time.max delta
## x   3001        0       30  0.01
```

Moreover, one can use random generators and distributions defined in other packages. For example, the package **fBasics** defines its own version of the normal inverse Gaussian distribution called `dnig`/`rnig`. The next code describes the same model in the above (although the `dnig` and `dNIG` are different in the interface and scope, check the relative documentation from the two packages)

```
require(fBasics)
mod4 <- setPoisson(intensity="lambda",
 df=list("dnig(z,alpha,beta,gamma)"))
poisson4 <- simulate(mod4,  sampling=samp,
 true.par=list(lambda=10,alpha=2,beta=0.3,gamma=1))
poisson4

##
## Compound Poisson process
## Number of equations: 1
## Parametric model with 4 parameters
##
## Number of original time series: 1
## length = 298, time range [0 ; 30.0430063761144]
##
## Number of zoo time series: 1
##   length time.min time.max delta
## x   3001        0       30  0.01
```

3.1 Inhomogeneous Compound Poisson Process

It is also possible to specify a nonconstant intensity to describe time-inhomogeneous compound Poisson processes. In this case, the intensity function can be time dependent, i.e. $\lambda = \lambda(t)$, the process N_t is characterized by its *intensity function* $\Lambda(t) = \int_0^t \lambda(s)\mathrm{d}s$, and its distribution has the following form

$$P(N_t = k) = e^{-\Lambda(t)} \frac{\Lambda(t)^k}{k!}, \quad k = 0, 1, 2, \ldots$$

i.e. $N_t \sim \text{Poi}(\Lambda(t))$ with $\mathbb{E}(N_t) = \Lambda(t)$. For time-inhomogeneous models, **yuima** uses the thinning methods; see, e.g., Lewis and Shedler (1979) and Ogata (1981).

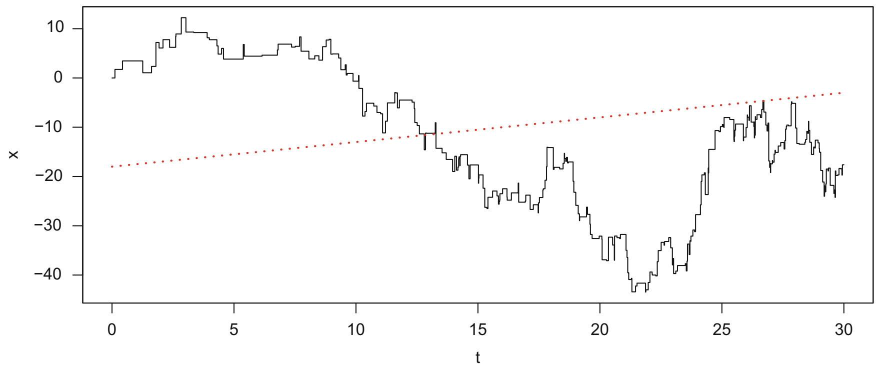


Fig. 3.3 A trajectory of time-inhomogeneous Poisson process with linear intensity $\lambda(t) = \alpha + \beta t$ and Gaussian jumps as defined in mod5. The dotted line is the intensity $\lambda(t)$, shifted down to -20 for graphical representation purposes

3.1.1 *Linear Intensity Function*

The following code is an example of compound Poisson process with linear intensity $\lambda(t) = \alpha + \beta t$ and Gaussian jumps (see Fig. 3.3 as well):

```
mod5 <- setPoisson(intensity="alpha+beta*t",
 df=list("dnorm(z,mu,sigma)"))
set.seed(123)
poisson5 <- simulate(mod5,  sampling=samp,
 true.par=list(alpha=2,beta=.5,mu=0, sigma=2))
plot(poisson5)
f <- function(t,alpha,beta) alpha + beta*t
curve(f(x,alpha=2,beta=0.5)-20,0,30,add=TRUE,col="red",lty=3,lwd=2)
```

3.1.2 *The Weibull Model*

The Weibull model has an intensity of the form $\lambda(t) = \theta * t^{\theta-1}, 0 < \alpha < \theta < \beta < \infty$. The model can be specified as follows (see Fig. 3.4 as well):

```
mod6 <- setPoisson(intensity="theta*t^(theta-1)",
 df=list("dnorm(z,mu,sigma)"))
set.seed(123)
poisson6 <- simulate(mod6,  sampling=samp,
 true.par=list(theta=1.5,mu=0, sigma=2))
plot(poisson6)
f <- function(t,theta) theta*t^(theta-1)
curve(f(x,theta=1.5),0,30,add=TRUE,col="red",lty=3,lwd=2)
```

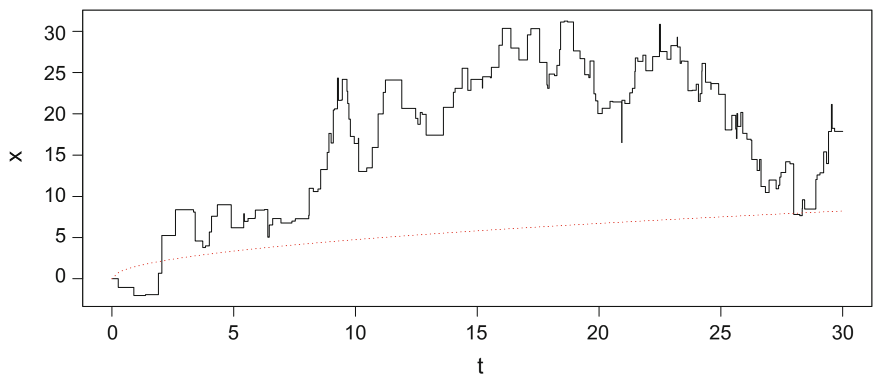


Fig. 3.4 A trajectory of time-inhomogeneous Poisson process with Weibull intensity $\lambda(t) = \theta t^{\theta-1}$ and Gaussian jumps as defined in `mod6`. The dotted line is the intensity $\lambda(t)$

3.1.3 The Exponentially Decaying Intensity Model

Next is an example of exponentially decreasing intensity $\lambda(t) = \beta \exp(-\lambda t)$ and exponential jumps with parameter γ (see Fig. 3.5 as well):

```
mod7 <- setPoisson(intensity="beta*exp(-lambda*t)",
 df=list("dexp(z,gamma)"))
set.seed(123)
poisson7 <- simulate(mod7, sampling=samp,
 true.par=list(lambda=.2,beta=10,gamma=1))
plot(poisson7)
f <- function(t,beta,lambda) beta*exp(-lambda*t)
curve(f(x,beta=10,lambda=0.2),0,30,add=TRUE,col="red",lty=3,lwd=2)
```

3.1.4 Modulated and Periodical Intensity Model

This is a physical model studied in Kutoyants (1998) and has a periodic intensity of the form $\lambda(t) = \frac{a}{2}(1 + \cos(\omega t + \varphi)) + \lambda$, where a is called the *amplitude* (the maximal value), ω is the *frequency*, and φ is the *phase* of the harmonic signal observed in background of a homogenous Poisson noise of intensity $\lambda > 0$ (the so-called *dark current*). We simply modify this model by adding Gaussian jumps (see Fig. 3.6 as well):

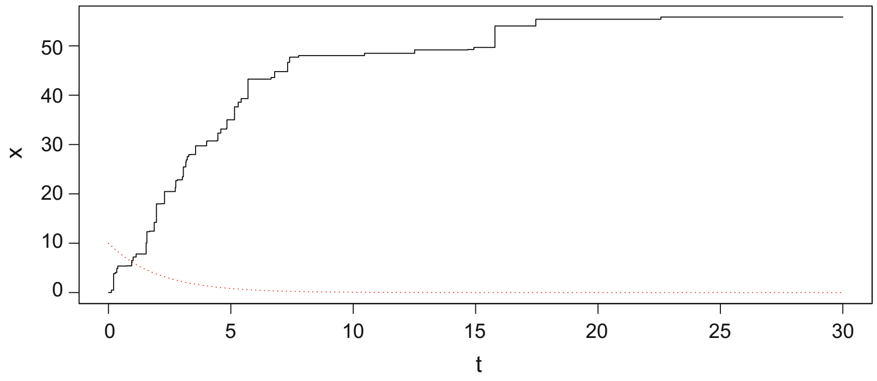


Fig. 3.5 A trajectory of time-inhomogeneous Poisson process with exponentially decaying intensity $\lambda(t) = \beta \exp(-\lambda t)$ and exponential jumps as defined in `mod7`. The dotted line is the intensity $\lambda(t)$

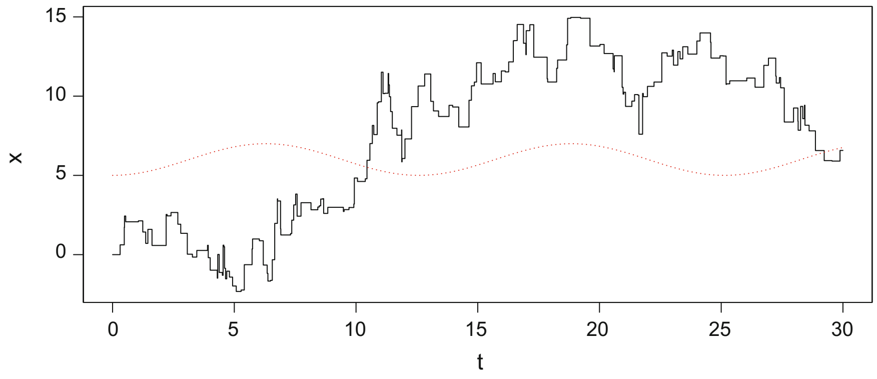


Fig. 3.6 A trajectory of time-inhomogeneous Poisson process with modulated and periodic intensity $\lambda(t) = \frac{a}{2}(1 + \cos(\omega t + \varphi)) + \lambda$ and Gaussian jumps as defined in `mod8`. The dotted line is the intensity $\lambda(t)$

```
mod8 <- setPoisson(intensity="0.5*a*(1+cos(omega*t+phi))+lambda",
 df=list("dnorm(z,mu,sigma)"))
set.seed(123)
poisson8 <- simulate(mod8, sampling=samp,
 true.par=list(a=2,omega=0.5,phi=3.14,lambda=5,mu=0,sigma=1))
plot(poisson8)
f <- function(t,a,omega,phi,lambda) 0.5*a*(1+cos(omega*t+phi))+lambda
curve(f(x,a=2,omega=0.5,phi=3.14,lambda=5),0,30,add=TRUE,
 col="red",lty=3,lwd=2)
```

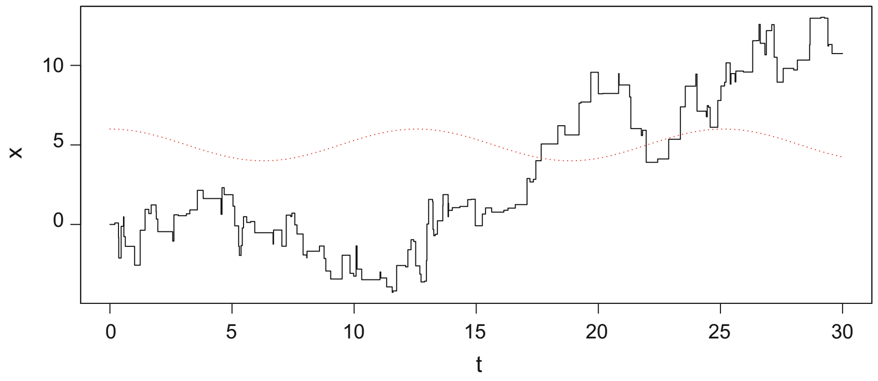


Fig. 3.7 A trajectory of time-inhomogeneous Poisson process with modulated and periodic intensity $\lambda(t) = a\cos(\theta t) + \lambda$ and Gaussian jumps as defined in `mod9`. The dotted line is the intensity $\lambda(t)$

3.1.5 Frequency Modulation Model

A model similar to the previous one, in which properties are also described in Kutoyants (1998), is the one that mixes exponential behaviour and periodicity in the intensity $\lambda(t) = a\cos(\theta t) + \lambda$, with $0 < a < \lambda$. We input it in **yuima** in this way (see Fig. 3.7 as well):

```
mod9 <- setPoisson(intensity="a*cos(theta*t)+lambda",
 df=list("dnorm(z,mu,sigma)"))
set.seed(123)
poisson9 <- simulate(mod9, sampling=samp,
true.par=list(a=1,theta=0.5,lambda=5,mu=0,sigma=1))
plot(poisson9)
f <- function(t,a,theta,lambda) a*cos(theta*t)+lambda
curve(f(x,a=1,theta=0.5,lambda=5),0,30,add=TRUE,col="red",lty=3,lwd=2)
```

3.2 Multidimensional Compound Poisson Processes

In the case of k-dimensional compound processes, it is necessary to specify the k-dimensional distribution of the random jumps Y_{τ_i} through one of the existing possibilities in R. Unfortunately, as there is no way to deduce the dimensionality of the distribution from its specification when one constructs the model it is necessary to specify this parameter in the argument `dimension` of `setPoisson`.

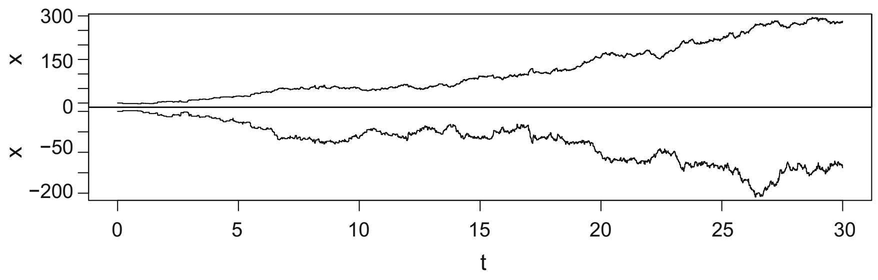


Fig. 3.8 A trajectory of time-inhomogeneous two-dimensional compound Poisson process with multivariate Gaussian jumps as defined in `mod10`

3.2.1 Multivariate Gaussian Jumps

Next example specifies a two-dimensional inhomogeneous compound Poisson process with multivariate Gaussian jumps (see Fig. 3.8 as well):

```
mod10 <- setPoisson(intensity="lambda*t",
 df=list("dmvnorm(z,c(0.15,-0.1),matrix(c(2,-1.9,-1.9,4.3),2,2))"),
 dimension=2)
set.seed(123)
poisson10 <- simulate(mod10, true.par=list(lambda=5), sampling=samp)
poisson10

##
## Compound Poisson process
## Number of equations: 2
## Parametric model with 1 parameters
##
## Number of original time series: 2
## length = 2305, time range [0 ; 30.0201047532617]
##
## Number of zoo time series: 2
##     length time.min time.max delta
## x.1   3001        0       30  0.01
## x.2   3001        0       30  0.01

plot(poisson10)
```

The following is an example with a three-dimensional multivariate compound Poisson process (see Fig. 3.9 as well):

```
mod11 <- setPoisson(intensity="lambda*t",
 df=list("dmvnorm(z,c(0.01,-0.01,.05),
 matrix(c(1,.5,0,.5,1,0,0,0,1),3,3))"),
 dimension=3)
set.seed(123)
poisson11 <- simulate(mod11, true.par=list(lambda=5),
```

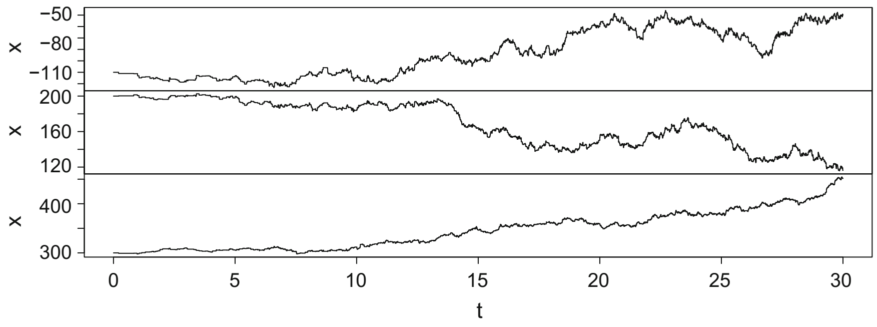


Fig. 3.9 A trajectory of time-inhomogeneous three-dimensional compound Poisson process with multivariate Gaussian jumps as defined in `mod11`

```
sampling=samp,xinit=c(-100,200,300))
plot(poisson11)
```

3.2.2 *User-Specified Jump Distribution*

Parametric versions of the compound Poisson processes can be constructed. In this case, the parameters must be set explicitly and specified during the simulation step. Next example describes a two-dimensional compound Poisson process with two-dimensional normal inverse Gaussian jumps. To this aim, we construct our functions to describe the density and the random number generator. For simulation purposes, **yuima** does not use the information coming from the distribution but only the random number generator. For this reason, we can create a density object `d2DNIG` which is not a true density which is needed only in model specification to allow **yuima** to resume the random number generator `r2DNIG` from its name `d2DNIG`. This approach will be replaced in future release of the package by the concept of *law*.

```
r2DNIG <- function(n,alpha){
 alpha <- 2
 beta <- c(0,0)
 delta0 <- 0.55
 mu <- c(0,0)
 Lambda <- matrix(c(1,0,0,1),2,2)
 t(rNIG(n,alpha=alpha,beta=beta,delta=delta0,mu=mu,Lambda=Lambda))
}
# the next fake density plays no role in simulation
# but it is needed for model specification
d2DNIG <- function(n,alpha){
 rep(0,2)
}
```

and we proceed with the construction of the model

```
mod12 <- setPoisson(intensity="lambda", df=list("d2DNIG(z,)"),
 dim=2)
set.seed(123)
poisson12 <- simulate(mod12, true.par= list(lambda=1),
 sampling=samp)
poisson12

##
## Compound Poisson process
## Number of equations: 2
## Parametric model with 1 parameters
##
## Number of original time series: 2
## length = 29, time range [0 ; 30.1613615503751]
##
## Number of zoo time series: 2
##     length time.min time.max delta
## x.1   3001        0       30  0.01
## x.2   3001        0       30  0.01
```

One can describe a model where each component is independent and with different distribution for the jumps like in the following example of a three-dimensional model where the first component is Gaussian, the second is exponential, and the third is normal inverse gamma:

```
rMydis <- function(n,a=1){
 cbind(rnorm(n), rexp(n), rNIG(n,1,1,1,1))
}
dMydis <- function(n,a=1){
 rep(0,3)
}
mod13 <- setPoisson(intensity="lambda*t",
 df=list("dMydis(z,1)"), dimension=3)
set.seed(123)
poisson13 <- simulate(mod13, true.par=list(lambda=5),
sampling=samp)
poisson13

##
## Compound Poisson process
## Number of equations: 3
## Parametric model with 1 parameters
##
## Number of original time series: 3
## length = 2305, time range [0 ; 30.0201047532617]
##
## Number of zoo time series: 3
##     length time.min time.max delta
## x.1   3001        0       30  0.01
## x.2   3001        0       30  0.01
## x.3   3001        0       30  0.01
```

Note that the above argument `a=1` is ignored but it is needed only to define formally the density function and the random number generator.

3.3 Estimation

Estimation of compound Poisson processes in **yuima** is possible only in the one-dimensional case at present. To this aim, the usual `qmle` can be used. The likelihood function exists in explicit form. Unfortunately, the asymptotic theory is model specific, and the estimators of the different coefficients in the intensity function have non-standard rates (see Kutoyants 1998). In general, both high frequency and growing T must be assumed to have good convergence properties. The large T requirement is due to the fact that in the Poisson model the intensity function can only be estimated as T diverges. The high-frequency requirement is needed in order to observe enough events from the Poisson process.

3.3.1 Compound Poisson Process with Gaussian Jumps

Here is an example of compound Poisson process with nonconstant intensity $\lambda(t) = \alpha + \lambda t$ and Gaussian jumps $N(\mu, \sigma^2)$. The aim is to estimate all the unknown parameters in the model $\theta = (\alpha, \lambda, \mu, \sigma)$ where the true value is $\theta_0 = (1, \frac{1}{2}, 0, 2)$:

```
mod14 <- setPoisson(intensity="alpha+lambda*t",
 df=list("dnorm(z,mu,sigma)"))
set.seed(123)
poisson14 <- simulate(mod14, sampling=samp,
true.par=list(alpha=1,lambda=.5,mu=0, sigma=2))
poisson14

##
## Compound Poisson process
## Number of equations: 1
## Parametric model with 4 parameters
##
## Number of original time series: 1
## length = 264, time range [0 ; 30.1784105685482]
##
## Number of zoo time series: 1
##    length time.min time.max delta
## x    3001        0       30  0.01

fit14 <- qmle(poisson14, start=list(alpha=2,lambda=1,mu=0,sigma=1),
 lower=list(alpha=0.1, lambda=0.1,mu=-1,sigma=0.1),
 upper=list(alpha=10,lambda=10,mu=3,sigma=4),
 method="L-BFGS-B")
coef(fit14)
```

```
##       alpha      lambda          mu       sigma
##  0.97964830  0.47913522 -0.00102812  2.05406584
```

The summary of the `qmle` in the case of jump process returns also additional information on the observed jumps which are common to the case of diffusion processes with jumps that will be considered in Chap. 4:

```
summary(fit14)

## Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = poisson14, start = list(alpha = 2, lambda = 1, mu = 0,
##     sigma = 1), method = "L-BFGS-B", lower = list(alpha = 0.1,
##     lambda = 0.1, mu = -1, sigma = 0.1), upper = list(alpha = 10,
##     lambda = 10, mu = 3, sigma = 4))
##
## Coefficients:
##           Estimate Std. Error
## alpha   0.97964830 0.55599883
## lambda  0.47913522 0.04789083
## mu     -0.00102812 0.13122945
## sigma   2.05406584 0.09279316
##
## -2 log L: 434.4788
##
##
## Number of estimated jumps: 245
##
## Average inter-arrival times: 0.122131
##
## Average jump size: -0.001028
##
## Standard Dev. of jump size: 2.058270
##
## Jump Threshold: 0.000000
##
## Summary statistics for jump times:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.13   14.55   20.37   19.59   25.97   29.93
##
## Summary statistics for jump size:
##      Min.   1st Qu.    Median      Mean   3rd Qu.
## -5.619549 -1.386609  0.063052 -0.001028  1.278984
##      Max.
##  5.383428
```

Notice that the summary statistics for jump size suggest that the jumps come from the Gaussian distribution with zero mean and standard deviation 2.

3.3.2 *NIG Compound Poisson Process*

Let us consider an example of homogenous compound Poisson process with normal inverse Gaussian jumps.

```
mod15 <- setPoisson(intensity="lambda",
 df=list("dNIG(z,alpha,beta,gamma,mu)"))
set.seed(123)
poisson15 <- simulate(mod15,sampling=samp,
 true.par=list(lambda=10,alpha=2,beta=0.3,gamma=1,mu=0))
poisson15
fit15 <- qmle(poisson15,
 start=list(beta=5,lambda=2,gamma=0.5,alpha=1,mu=0),
 lower=list(alpha=1,beta=0.1,lambda=0.1,gamma=0.1,mu=-1),
 upper=list(alpha=5,beta=0.99,lambda=20,gamma=2,mu=2),
 method="L-BFGS-B")
summary(fit15)
```

```
## Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = poisson15, start = list(beta = 5, lambda = 2,
## gamma = 0.5,
## alpha = 1, mu = 0), method = "L-BFGS-B", lower = list(alpha
## = 1,
## beta = 0.1, lambda = 0.1, gamma = 0.1, mu = -1), upper =
## list(alpha = 5,
## beta = 0.99, lambda = 20, gamma = 2, mu = 2))
##
## Coefficients:
## Estimate Std. Error
## lambda 9.93334954 0.5754235
## alpha 1.76434876 0.5016348
## beta 0.41462936 0.2435998
## gamma 0.88237689 0.1981752
## mu -0.04894914 0.1064062
##
## -2 log L: -131.3317
##
##
## Number of estimated jumps: 298
##
## Average inter-arrival times: 0.099899
##
## Average jump size: 0.164390
##
## Standard Dev. of jump size: 0.731894
##
## Jump Threshold: 0.000000
##
## Summary statistics for jump times:
## Min. 1st Qu.  Median Mean 3rd Qu.  Max.
## 0.120 7.415 15.190 14.884 22.510 29.790
```

```
##
## Summary statistics for jump size:
## Min. 1st Qu.  Median Mean 3rd Qu.  Max.
## -1.8322 -0.2782 0.0915 0.1644 0.5329 3.3452
```

3.3.3 *Exponential Jump Compound Poisson Process*

In this example, we assume that the jump distribution and the intensity function of the process have one common parameter λ:

```
mod16 <- setPoisson(intensity="beta*exp(-lambda*t)",
 df=list("dexp(z,lambda)"))
set.seed(123)
poisson16 <- simulate(mod16, true.par=list(lambda=.2,beta=10),
 sampling=samp)
poisson16

##
## Compound Poisson process
## Number of equations: 1
## Parametric model with 2 parameters
##
## Number of original time series: 1
## length = 56, time range [0 ; 22.576481101042]
##
## Number of zoo time series: 1
##   length time.min time.max delta
## x   3001        0       30   0.01

fit16 <- qmle(poisson16,
 start=list(beta=.5,lambda=2),
 lower=list(beta=0.1,lambda=0.1),
 upper=list(beta=20,lambda=10),
 method="L-BFGS-B")
summary(fit16)

## Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = poisson16, start = list(beta = 0.5, lambda = 2),
##     method = "L-BFGS-B", lower = list(beta = 0.1, lambda = 0.1),
##     upper = list(beta = 20, lambda = 10))
##
## Coefficients:
##          Estimate Std. Error
## beta   10.0271900 1.70795286
## lambda  0.1922273 0.01936245
##
## -2 log L: 241.873
##
```

```
##
## Number of estimated jumps: 52
##
## Average inter-arrival times: 0.440196
##
## Average jump size: 5.368982
##
## Standard Dev. of jump size: 5.344620
##
## Jump Threshold: 0.000000
##
## Summary statistics for jump times:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.120   1.823   3.320   4.941   5.935  22.570
##
## Summary statistics for jump size:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##  0.02908  1.53416  3.54106  5.36898  7.78484 21.76591
```

3.3.4 *The Weibull Compound Poisson Process*

We consider again the Weibull compound Poisson process with Gaussian jumps:

```
mod17 <- setPoisson(intensity="lambda*t^(lambda-1)",
 df=list("dnorm(z,mu,sigma)"))
set.seed(123)
poisson17 <- simulate(mod17, true.par=list(lambda=2,mu=0, sigma=2),
 sampling=samp)
poisson17
fit17 <- qmle(poisson17,
 start=list(lambda=5,mu=0,sigma=1),
 lower=list(lambda=0.1,mu=-1,sigma=0.1),
 upper=list(lambda=10,mu=3,sigma=4),
 method="L-BFGS-B")
summary(fit17)
```

```
## Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = poisson17, start = list(lambda = 5, mu = 0,
## sigma = 1),
## method = "L-BFGS-B", lower = list(lambda = 0.1, mu = -1,
## sigma = 0.1), upper = list(lambda = 10, mu = 3, sigma = 4))
##
## Coefficients:
## Estimate Std. Error
## lambda 1.94548958 0.01063067
## mu -0.04916873 0.07964453
## sigma 2.18696388 0.05631872
```

```
##
## -2 log L: -264.4562
##
##
## Number of estimated jumps: 754
##
## Average inter-arrival times: 0.039110
##
## Average jump size: -0.049143
##
## Standard Dev. of jump size: 2.188374
##
## Jump Threshold: 0.000000
##
## Summary statistics for jump times:
## Min. 1st Qu.  Median Mean 3rd Qu.  Max.
## 0.54 13.90 20.39 19.37 25.54 29.99
##
## Summary statistics for jump size:
## Min.   1st Qu.  Median Mean 3rd Qu.  Max.
## -7.29017 -1.47597 -0.04692 -0.04914 1.40901 7.35593
```

also in this case the maximum likelihood approach produces good estimates.

3.3.5 Modulated and Periodical Intensity Model

Finally, we consider the estimation problem for the modulated and periodical intensity model with Gaussian jumps

```
mod18 <- setPoisson(intensity="0.5*a*(1+cos(omega*t+phi))+lambda",
 df=list("dnorm(z,mu,sigma)"))
set.seed(123)
poisson18 <- simulate(mod18, sampling=samp,
 true.par=list(a=2,omega=0.5,phi=3.14,lambda=5,mu=0,sigma=1))
 fit18 <- qmle(poisson18,
 start=list(a=1, omega=0.2, phi=1, lambda=2, mu=1, sigma=2),
 lower=list(a=0.1, omega=0.1, phi=0.1, lambda=0.1, mu=-2, sigma=0.1),
 upper=list(a=5, omega=1, phi=5, lambda=10, mu=2, sigma=3),
 method="L-BFGS-B")
summary(fit18)
```

```
## Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = poisson18, start = list(a = 1, omega = 0.2,
## phi = 1,
## lambda = 2, mu = 1, sigma = 2), method = "L-BFGS-B", lower
## = list(a = 0.1,
## omega = 0.1, phi = 0.1, lambda = 0.1, mu = -2, sigma =
## 0.1),
## upper = list(a = 5, omega = 1, phi = 5, lambda = 10, mu =
## 2,
## sigma = 3))
##
## Coefficients:
## Estimate Std. Error
## a 1.88325585 1.28883394
## omega 0.42829021 0.13037074
## phi 2.99248228 1.27490712
## lambda 4.97892402 0.82826617
## mu 0.03712969 0.07342504
## sigma 0.97685658 0.05191948
##
## -2 log L: 217.4368
##
##
## Number of estimated jumps: 177
##
## Average inter-arrival times: 0.168011
##
## Average jump size: 0.037138
##
## Standard Dev. of jump size: 0.979623
##
## Jump Threshold: 0.000000
##
## Summary statistics for jump times:
## Min. 1st Qu.  Median Mean 3rd Qu.  Max.
## 0.31 6.78 12.82 14.18 21.44 29.88
##
## Summary statistics for jump size:
## Min.  1st Qu.  Median Mean 3rd Qu.  Max.
## -2.46590 -0.58948 0.03455 0.03714 0.70352 2.57146
```

even in this 6-parameters model the estimation procedure seems to work properly.

Chapter 4
Stochastic Differential Equations Driven by Lévy Processes

4.1 Lévy Processes

Definition 4.1 An $\mathbb{R}^{\mathrm{d}}$-valued stochastic process $X = (X_t)_{t\in\mathbb{R}_+}$ defined on a probability space $(\Omega, \mathscr{F}, P)$ is called a **Lévy process** if it satisfies the following conditions:

(i) $X_0 = 0$ a.s.

(ii) (independent increments) $X_{t_i} - X_{t_{i-1}}$ $(i = 1, ..., n)$ are independent for any $(t_i)_{i=0,\dots,n}$ $(0 \le t_0 \le \cdots \le t_n)$ and $n \in \mathbb{N}$.

(iii) (stationary increments) $\mathscr{L}\{X_t - X_s\} = \mathscr{L}\{X_{t-s}\}$ for $s, t \in \mathbb{R}_+$, $s \le t$.[1]

(iv) (stochastic continuity) For any $\varepsilon > 0$ and $t \in \mathbb{R}_+$, $\lim_{s\to t} P[|X_s - X_t| > \varepsilon] = 0$.

In Definition 4.1, Condition (iv) can be replaced by

(iv') For any $\varepsilon > 0$, $\lim_{t\downarrow 0} P[|X_t| > \varepsilon] = 0$.

It is known that the Lévy process X in the sense of Definition 4.1 admits a càdlàg version.[2] That is,

(v) (càdlàg) There exists an event Ω_0 such that $P[\Omega_0] = 1$ and each path $\mathbb{R}_+ \ni t \mapsto X_t(\omega)$ is càdlàg for $\omega \in \Omega_0$.

Hereafter, we will assume the property (v) for Lévy processes unless otherwise specified. Two stochastic processes $X = (X_t)_{\mathbb{R}_+}$ and $Y = (Y)_{\mathbb{R}_+}$ are identified if they are indistinguishable, i.e. $P[X_t = Y_t\ (\forall t \in \mathbb{R}_+)] = 1$. For a complete treatment on Lévy processes see Itô (2013) and Sato (1999).

[1]The distribution of a random variable Z is denoted by $\mathscr{L}\{Z\}$.

[2]The term càdlàg stands for "right-continuous and admitting left-hand limits".

S. M. Iacus and N. Yoshida, *Simulation and Inference for Stochastic Processes with YUIMA*, Use R!, https://doi.org/10.1007/978-3-319-55569-0_4

4.1.1 Infinitely Divisible Distributions

A probability distribution μ on $\mathbb{R}^{\mathsf{d}}$ is called to be **infinitely divisible** if for every $n \in \mathbb{N}$, there exists a probability distribution μ_n on $\mathbb{R}^{\mathsf{d}}$ such that $\mu = \mu_n^{*n}$ (n-convolution of μ_n). Infinitely divisible laws are specified by the following representation.

Theorem 4.1 (Lévy-Khintchine formula) *For any infinitely divisible law μ on $\mathbb{R}^{\mathsf{d}}$, the characteristic function $\hat{\mu}$ of μ admits the representation*

$$\hat{\mu}(u) = \exp(\psi_c(u)) \qquad (u \in \mathbb{R}^{\mathsf{d}}) \tag{4.1}$$

with

$$\psi_c(u) = i\, b \cdot u - \frac{1}{2} C[u^{\otimes 2}] + \int_{\mathbb{R}^{\mathsf{d}}} \big(e^{iu\cdot x} - 1 - iu \cdot x 1_{\{|x|\le 1\}}\big)\nu(dx), \tag{4.2}$$

where

(i) $b \in \mathbb{R}^{\mathsf{d}}$,
(ii) *C is a $\mathsf{d} \times \mathsf{d}$-nonnegative symmetric matrix, and*
(iii) *ν is a measure on $\mathbb{R}^{\mathsf{d}}$ such that $\nu(\{0\}) = 0$ and $\int_{\mathbb{R}^{\mathsf{d}}} (|x|^2 \wedge 1)\nu(dx) < \infty$.*

The representation (4.1) and (4.2) is unique. Moreover, for a triplet (b, C, ν) satisfying (i)–(iii), there exists an infinitely divisible distribution μ whose characteristic function $\hat{\mu}$ is given by (4.1) and (4.2).

When $\int_{\{|x|\le 1\}} |x|\nu(dx) < \infty$, (4.2) can be written as

$$\psi_c(u) = i\, b_- \cdot u - \frac{1}{2} C[u^{\otimes 2}] + \int_{\mathbb{R}^{\mathsf{d}}} \big(e^{iu\cdot x} - 1\big)\nu(dx) \tag{4.3}$$

with correspondence

$$b_- = b - \int_{\{|x|\le 1\}} x\ \nu(dx).$$

When $\int_{\{|x|>1\}} |x|\nu(dx) < \infty$, (4.2) is rewritten as

$$\psi_c(u) = i\, b_+ \cdot u - \frac{1}{2} C[u^{\otimes 2}] + \int_{\mathbb{R}^{\mathsf{d}}} \big(e^{iu\cdot x} - 1 - iu \cdot x\big)\nu(dx)$$

with correspondence

$$b_+ = b + \int_{\{|x|>1\}} x\nu(dx).$$

4.1.2 *Infinite Divisible Distributions, Lévy Processes, Lévy-Itô Decomposition*

Obviously, for any Lévy process X, the law $\mathscr{L}\{X_t\}$ is infinitely divisible for $t \in \mathbb{R}_+$. Thanks to stationarity, independency of increments and stochastic continuity, the law of X on the space $\mathbb{D}(\mathbb{R}^{\mathsf{d}})$, the space of all càdlàg functions $f : \mathbb{R}_+ \to \mathbb{R}^{\mathsf{d}}$, is determined by the infinitely divisible law $\mu = \mathscr{L}\{X_1\}$. In particular, $\widehat{\mathscr{L}\{X_t\}}(u) = \exp\left(t\psi_c(u)\right)$ with ψ_c in Theorem 4.1.

Conversely, Kolmogorov's extension theorem ensures that for any infinitely divisible distribution μ on $\mathbb{R}^{\mathsf{d}}$, there exists a unique Lévy process $X = (X_t)_{t\in\mathbb{R}^{\mathsf{d}}}$ such that $\mathscr{L}\{X_1\} = \mu$ (See e.g., p.35, Theorem 7.10, Sato 1999, for details). Thus, there is a one-to-one correspondence between Lévy processes X and infinitely divisible distributions $\mu = \mathscr{L}\{X_1\}$. The triplet (b, C, ν) of μ characterizes a Lévy process. The measure ν is called a **Lévy measure**.

The Lévy-Itô decomposition gives a description of paths of Lévy processes. Given a probability space $(\Omega, \mathscr{F}, P)$, a collection of measure $\mu = (\mu_\omega)_{\omega\in\Omega}$ is called a **Poisson random measure** on $\mathsf{E} = \mathbb{R}_+ \times \mathbb{R}^{\mathsf{d}}$ if it satisfies the following conditions.

(1) For each $\omega \in \Omega$, $\mu_\omega(\cdot)$ is a measure on E.
(2) For each $B \in \mathbb{B}(\mathsf{E})$, $\mu_\cdot(B)$ is measurable.
(3) For any disjoint measurable sets $B_1, ..., B_n$, the family $\{\mu(B_1), ..., \mu(B_n)\}$ is independent.
(4) For any $B \in \mathbb{B}(\mathsf{E})$ with $\bar{\mu}(B) = E[\mu(B)] < \infty$, $\mu(B)$ has the Poisson distribution with parameter $\bar{\mu}(B)$.

Here $\mathbb{B}(\mathsf{E})$ denotes the Borel σ-field of E. The measure $\bar{\mu}$ is called the intensity measure of μ.

Theorem 4.2 (Lévy-Itô decomposition) *Any* d*-dimensional Lévy process X has a representation*

$$X_t = bt + C^{1/2}W_t + \int_0^t \int_{|x|\le 1} x(\mu - \bar{\mu})(ds, dx) + \int_0^t \int_{|x|>1} x\mu(ds, dx),$$

where $b \in \mathbb{R}$, $C \in \mathbb{R}_+$, $W = (W_t)_{t\in\mathbb{R}_+}$ is a standard Wiener process, and μ is an independent Poisson random measure on $\mathbb{R}_+ \times \mathbb{R}^{\mathsf{d}}$ with an intensity measure $\bar{\mu}(ds, dx) = ds\nu(dx)$ for a measure ν such that $\nu(\{0\}) = 0$ and $\int_{\mathbb{R}^{\mathsf{d}}} |x|^2\nu(dx) < \infty$.

The integral with respect to $\mu - \bar{\mu}$ is a stochastic integral that should be read in L^2-sense.

In the following sections, we shall observe several processes that have more or less explicit representations. We will assume that $\mathsf{d} = 1$ unless otherwise stated.

4.2 Wiener Process

A d-dimensional standard Wiener process $W = (W_t)_{t\in\mathbb{R}_+}$ is a Lévy process with characteristics $(0, I_{\mathsf{d}}, 0)$, where I_{d} is the d-dimensional identity matrix. Obviously the law $\mathscr{L}\{W_t\}$ of W_t is infinitely divisible.

4.3 Compound Poisson Process

Consider a family of i.i.d. random variables $\{\xi_j\}_{j\in\mathbb{N}}$ taking values in $\mathbb{R}^{\mathsf{d}}$. Let $N = (N_t)_{t\in\mathbb{R}_+}$ be a Poisson process with intensity parameter λ, independent of $\{\xi_j\}_{j\in\mathbb{N}}$. Define a stochastic process $X = (X_t)_{t\in\mathbb{R}_+}$ by

$$X_t = \sum_{j=1}^{N_t} \xi_j .$$

By convention, the summation reads 0 if there is no summand: $X_t = 0$ when $N_t = 0$. With the occurrence times $(T_j)_{j\in\mathbb{N}}$, $0 < T_1 < T_2 < \cdots$, we can write

$$X_t = \sum_{j=1}^{\infty} \xi_j 1_{[T_j,\infty)}(t).$$

Then X is a compound Poisson process.

It is easy to show that X is a Lévy process. Indeed, by using the conditional independency of $\{\xi_j\}_{j\in\mathbb{N}}$ given N and the property of independent increments of N, we have

$$\begin{aligned}
E\Big[\exp\Big(i\sum_{j=1}^{n} u_j\cdot(X_{t_j}-X_{t_{j-1}})\Big)\Big] &= E\Big[\prod_{j=1}^{n} E\Big[\exp\Big(iu_j\cdot\sum_{j=N_{t_{j-1}}+1}^{N_{t_j}}\xi_j\Big)\Big|N\Big]\Big]\\
= E\Big[\prod_{j=1}^{n}\varphi_{\xi_1}(u_j)^{N_{t_{j-1}}-N_{t_j}}\Big] &= \prod_{j=1}^{n} E\Big[\varphi_{\xi_1}(u_j)^{N_{t_{j-1}}-N_{t_j}}\Big]\\
= \prod_{j=1}^{n} E\Big[\exp\Big(iu_j\cdot(X_{t_j}-X_{t_{j-1}})\Big)\Big]
\end{aligned}$$

for t_j $(0 \le t_0 < t_1 < \cdots < t_n)$ and $\{u_j\}_{j=1,\dots,n} \subset \mathbb{R}^{\mathsf{d}}$. The the Kac theorem ensures that X has independent increments. The stationarity of increments and the stochastic continuity are easily observed.

As a particular case of the above computation, we also obtain

$$\begin{aligned} E\big[\exp\big(iu\cdot X_t\big)\big] = E\big[\varphi_{\xi_1}(u)^{N_t}\big] &= \exp\big[\lambda t(\varphi_{\xi_1}(u)-1)\big] \\ &= \exp\left[t\int_{\mathbb{R}^d}(e^{iu\cdot x}-1)\lambda P^{\xi_1}(dx)\right] \end{aligned}$$

for the distribution P^{ξ_1} of ξ_1. Therefore, X is a Lévy process with characteristics $b_- = 0$, $C = 0$ and $\nu(dx) = \lambda P^{\xi_1}(dx)$.

The function `setPoisson` can be used to build a `yuima` model of a compound Poisson type. The jump distribution is specified by the argument `df`. Simulation is run by applying `simulate` to `yuima` model so constructed.

```
set.seed(123)
mu <- 0
sigma <- 1
lambda <- 10
samp <- setSampling(Terminal=10, n=1000)
mod10b <- setPoisson(intensity="lambda", df=list("dnorm(z,mu,sigma)"))
y10b <- simulate(mod10b,sampling=samp,
  true.par=list(lambda=lambda,mu=0.1, sigma=2))
y10b

##
## Compound Poisson process
## Number of equations: 1
## Parametric model with 3 parameters
##
## Number of original time series: 1
## length = 113, time range [0 ; 10.2515652111658]
##
## Number of zoo time series: 1
##    length time.min time.max delta
## x   1001        0       10  0.01
```

See Chap. 3 for more details on compound Poisson processes and their simulation and inference within **yuima**.

4.4 Gamma Process and Its Variants

4.4.1 Gamma Process

For positive numbers δ and γ, the **Gamma distribution** $\Gamma(\delta,\gamma)$ with density

$$p_\Gamma(x) = \frac{\gamma^\delta}{\Gamma(\delta)}x^{\delta-1}e^{-\gamma x}1_{(0,\infty)}(x), \qquad x\in\mathbb{R}, \tag{4.4}$$

has the characteristic function

$$\varphi_\Gamma(u) = \left(1 - iu/\gamma\right)^{-\delta}.$$

$\Gamma(\delta, \gamma)$ is infinitely divisible, in particular,

$$\Gamma(\delta_1, \gamma) * \Gamma(\delta_2, \gamma) = \Gamma(\delta_1 + \delta_2, \gamma).$$

There exists a Lévy process X such that

$$\varphi_{X_t} = \left(1 - iu/\gamma\right)^{-\delta t},$$

and $\mathscr{L}\{X_t\} = \Gamma(\delta t, \gamma)$. By the formula[3] $\log\left(1 - iu/\gamma\right) = \int_0^\infty (1 - e^{iux}) \frac{e^{-\gamma x}}{x} dx$, we obtain

$$\varphi_{X_t}(u) = \exp\left(t \int_0^\infty (e^{iux} - 1) \frac{\delta e^{-\gamma x}}{x} dx\right)$$

Therefore, $b_- = 0$, $C = 0$ and $d\nu/dx(x) = \delta x^{-1} e^{-\gamma x} 1_{\{x>0\}}$. The Lévy process X with these characteristics is called a **gamma process** $\Gamma\mathrm{P}(\delta, \gamma)$ for parameters δ, $\gamma > 0$, (see, e.g., Applebaum 2004)

4.4.2 *Variance Gamma Process*

A process $X^0 = (X_t^0)_{t \in \mathbb{R}_+}$ is called a **variance gamma process** $\mathrm{VGP}^0(\delta, \gamma_-, \gamma_+)$ if it has a decomposition

$$X_t^0 = X_t^+ - X_t^-,$$

where $X^+ = (X_t^+)_{t \in \mathbb{R}_+}$ and $X^- = (X_t^-)_{t \in \mathbb{R}_+}$ are independent gamma processes with Lévy measures $d\nu_{X^+}/dx(x) = \delta x^{-1} e^{-\gamma_- x} 1_{\{x>0\}}$ and $d\nu_{X^-}/dx(x) = \delta x^{-1} e^{-\gamma_+ x}$ $1_{\{x>0\}}$, respectively. Here δ and $\gamma_\pm$ are positive constants.

By definition, the characteristic function $\varphi_{X_t^0}$ is given by

$$\varphi_{X_t^0}(u) = \varphi_{X_t^+}(u)\varphi_{X_t^-}(-u) = \left[1 - \left(\frac{1}{\gamma_+} - \frac{1}{\gamma_-}\right)iu + \frac{1}{\gamma_+\gamma_-}u^2\right]^{-\delta t}.$$

[3] $\log\left(1 - iu/\gamma\right) = \int_0^1 \frac{-iu}{\gamma}(1 - ius/\gamma)^{-1} ds = \int_0^1 \int_0^\infty (-iu) e^{iusx}\, e^{-\gamma x} dx ds = \int_0^\infty \int_0^1 (-iu) e^{iusx} ds\, e^{-\gamma x} dx = \int_0^\infty (1 - e^{iux}) \frac{e^{-\gamma x}}{x} dx.$

The process X^0 is a Lévy process with characteristics $b_- = 0$, $C = 0$ and

$$\frac{d\nu}{dx}(x) = \delta |x|^{-1} e^{-\gamma_-|x|} 1_{\{x<0\}} + \delta x^{-1} e^{-\gamma_+ x} 1_{\{x>0\}}.$$

The parameterization

$$(\lambda, \alpha, \beta) = \left(\delta, \frac{\gamma_- + \gamma_+}{2}, \frac{\gamma_- - \gamma_+}{2}\right),$$

that is,

$$(\delta, \gamma_-, \gamma_+) = \left(\lambda, \alpha + \beta, \alpha - \beta\right)$$

is often used. Then,

$$\varphi_{X_t^0}(u) = \left[\frac{\alpha^2 - (\beta + iu)^2}{\alpha^2 - \beta^2}\right]^{-\lambda t}$$

For a constant $\mu \in \mathbb{R}$, the process $X_t = \mu t + X_t^0$ is called a normal gamma process NGP$(\lambda, \alpha, \beta, \mu)$ or a variance gamma process VGP$(\lambda, \alpha, \beta, \mu)$ in the present **yuima** package. The characteristics are $b_- = \mu$, $C = 0$ and ν. In particular, $\varphi_{X_t}(u) = e^{i\mu t u}\varphi_{X_t^0}(u)$. As it will be seen in Sect. 4.8.4, the density of X_t is

$$p_{X_t}(x) = \frac{1}{\sqrt{\pi}\Gamma(\lambda t)}\left(\alpha^2 - \beta^2\right)^{\lambda t}\left(\frac{|x - \mu t|}{2\alpha}\right)^{\lambda t - \frac{1}{2}} K_{\lambda t - \frac{1}{2}}\left(\alpha|x - \mu t|\right)\exp(\beta(x - \mu t)).$$

Further reading on the variance gamma process can be found in Madan and Seneta (1990a), Madan et al. (1998) and Seneta (2007).

4.4.3 *Bilateral Gamma Process*

The **yuima** package is equipped with the **bilateral gamma process** B$\Gamma(\delta_+, \gamma_+, \delta_-, \gamma_-)$ that is a Lévy process with characteristics $b_- = 0$, $C = 0$ and

$$\frac{d\nu}{dx}(x) = \delta_- |x|^{-1} e^{-\gamma_-|x|} 1_{\{x<0\}} + \delta_+ x^{-1} e^{-\gamma_+ x} 1_{\{x>0\}}.$$

Therefore, for the bilateral gamma process X_t,

$$\varphi_{X_t}(u) = \left(1 + iu/\gamma_-\right)^{-\delta_- t}\left(1 - iu/\gamma_+\right)^{-\delta_+ t}.$$

The reader can refer to Küchler and Tappe (2008a, b) for more details on the bilateral gamma distribution and process.

4.4.4 *Simulation of Gamma Processes*

The correspondences between the parameters of the bilateral gamma process and the arguments of the functions `rbgamma` and `dbgamma` of **yuima** are as follows:

$$\text{delta.plus} = \delta_+, \quad \text{gamma.plus} = \gamma_+, \quad \text{delta.minus} = \delta_-, \quad \text{gamma.minus} = \gamma_-$$

Let us generate paths of the bilateral gamma process with $\delta_+ = 1.4$, $\gamma_+ = 0.3$, $\delta_- = 2$, and $\gamma_- = 0.6$.[4]

```
BGmodel <- setModel(drift="0", xinit="0", jump.coeff="1",
 measure.type="code", measure=list(df="rbgamma(z, delta.plus=1.4,
 gamma.plus=0.3, delta.minus=2,
 gamma.minus=0.6)"))
n <- 1000
samp <- setSampling(Terminal=1, n=n)
BGyuima <- setYuima(model=BGmodel, sampling=samp)
set.seed(127)
for (i in 1:5) {
 result <- simulate(BGyuima)
 plot(result,xlim=c(0,1),ylim=c(-6,6),
  main="Paths of bilateral gamma process",col=i,par(new=T))
}
```

The simulated paths can be seen in Fig. 4.1. When $\delta_- = \delta_+$, the bilateral gamma process is a variance gamma process.[5]

```
VGmodel <- setModel(drift="0", xinit="0", jump.coeff="1",
 measure.type="code", measure=list(df="rbgamma(z, delta.minus=2,
 gamma.minus=0.6, delta.plus=2, gamma.plus=0.3)"))
VGyuima <- setYuima(model=VGmodel, sampling=samp)
set.seed(127)
for (i in 1:5) {
 result <- simulate(VGyuima)
 plot(result,xlim=c(0,1),ylim=c(-4,12),
  main="Paths of variance gamma process",col=i,par(new=T))
}
```

Again, the simulated trajectories can be seen in Fig. 4.2. In the case where $\delta_- = 0$, the bilateral gamma process becomes a gamma process. We use the simulator `rgamma` instead of `rbgamma` (Fig. 4.3).

[4]We owe YUIMA's random number generators of Lévy processes to Hiroki Masuda.

[5]The function `rvgamma` is also available to generate variance gamma processes.

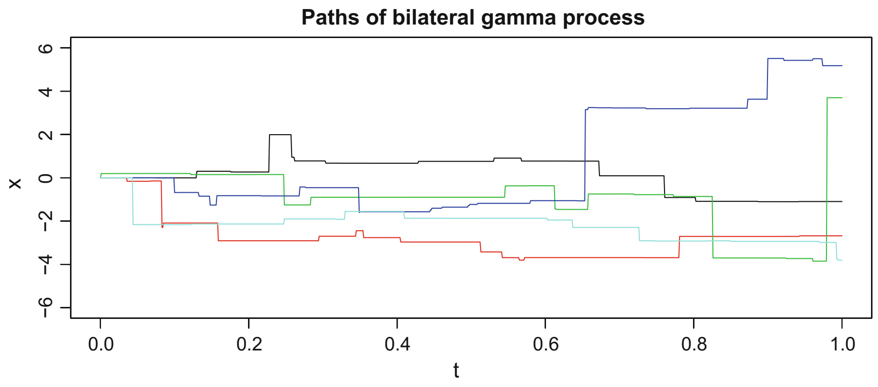


Fig. 4.1 Simulated trajectories of a bilateral gamma process

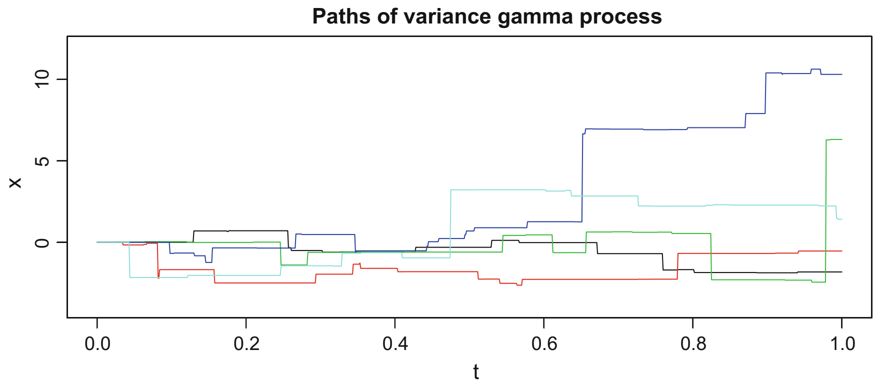


Fig. 4.2 Simulated trajectories of a variance gamma process

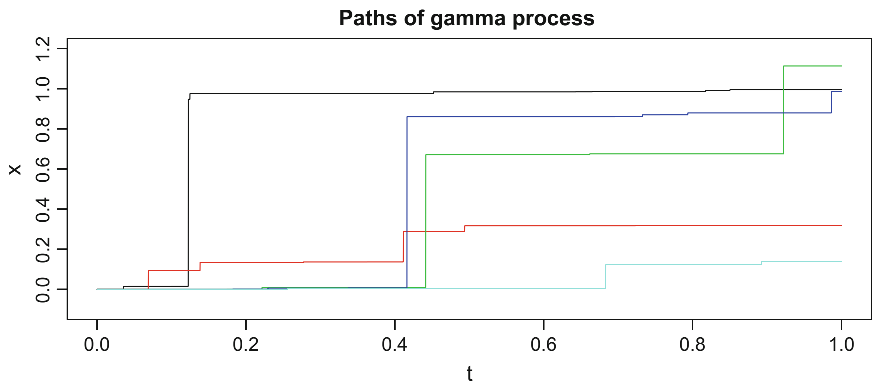


Fig. 4.3 Simulated trajectories of a gamma process

```
Gmodel <- setModel(drift="0", xinit="0", jump.coeff="1",
 measure.type="code", measure=list(df="rgamma(z,
 shape=0.7, scale=1)"))
n <- 10000
samp <- setSampling(Terminal=1, n=n)
Gyuima <- setYuima(model=Gmodel, sampling=samp)
set.seed(129)
for (i in 1:5){
result <- simulate(Gyuima)
 plot(result,xlim=c(0,1),ylim=c(-0.1,1.2),
  main="Paths of gamma process",col=i,par(new=T))
}
```

We shall compare the histogram of X_1 obtained by `simulate` with the density function of Gamma(0.7, 1) (Fig. 4.4).

```
n <- 5
sampling <- setSampling(Terminal=1, n=n)
Gmodel <- setModel(drift="0", xinit="0", jump.coeff="1",
measure.type="code", measure=list(df="rgamma(z,
shape=0.7, scale=1)"))
Gyuima <- setYuima(model=Gmodel, sampling=samp)
simdata <- NULL
set.seed(127)
for (i in 1:3000){
result <- simulate(Gyuima)
 x1 <- result@data@original.data[n+1,1]
 simdata <- c(simdata,as.numeric(x1))
}
hist(simdata, xlim=c(0,2), ylim=c(0,3), breaks=100, freq=FALSE,
 main=expression(paste("Distribution of ", X[1],
 " and Density of Gamma(0.7,1)")))
curve(dgamma(x,0.7,1),add=TRUE,col="red")
```

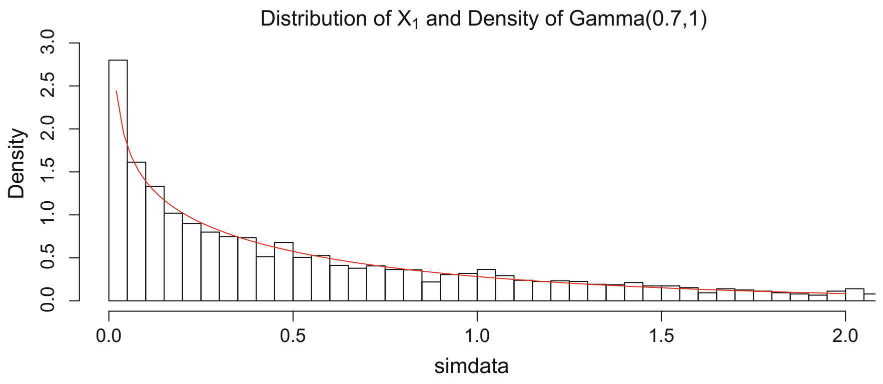


Fig. 4.4 Distribution of simulated gamma process X_1 versus theoretical gamma distribution

4.5 Generalized Tempered Stable Process, Tempered α Stable Process, CGMY Process, Positive Tempered Stable Process

A generalization of the bilateral gamma process is the **generalized tempered stable process** $\mathrm{GTSP}(\alpha_+, \delta_+, \gamma_+, \alpha_-, \delta_-, \gamma_-, b)$ that has the Lévy triplet $b \in \mathbb{R}$, $C = 0$ and ν such that

$$\frac{d\nu}{dx}(x) = \delta_- |x|^{-1-\alpha_-} e^{\gamma_- x} 1_{\{x<0\}} + \delta_+ x^{-1-\alpha_+} e^{-\gamma_+ x} 1_{\{x>0\}},$$

where $\delta_\pm \geq 0$, $\gamma_\pm > 0$ and $\alpha_\pm < 2$ (see, e.g., Cont and Tankov 2004; Schoutens 2003; Kyprianou et al. 2005; Koponen 1995; Levendorskii and Boyarchenko 2002, related with this section).

A special case is the **tempered α stable process** $\mathrm{TSP}(\alpha, \delta_+, \gamma_+, \delta_-, \gamma_-, b)$ with components $C = 0$ and

$$\frac{d\nu}{dx}(x) = \delta_- |x|^{-1-\alpha} e^{\gamma_- x} 1_{\{x<0\}} + \delta_+ x^{-1-\alpha} e^{-\gamma_+ x} 1_{\{x>0\}},$$

where $\delta_\pm \geq 0$, $\gamma_\pm > 0$ and $\alpha < 2$.

The **CGMY process** (Carr et al. 2002) is a tempered α stable process for constants $\delta_\pm = c$, $\gamma_- = g$, $\gamma_+ = m$, $\alpha = y$. In particular, the CGMY process for $y = 0$ is a variance gamma process. The **positive tempered stable process** $\mathrm{PTSP}(\alpha_+, \delta_+, \gamma_+)$ is $\mathrm{GTSP}(\alpha_+, \delta_+, \gamma_+, 0, 0, 0, 0) = \mathrm{TSP}(\alpha_+, \delta_+, \gamma_+, 0, 0, 0)$.

The **yuima** package provides a random number generator of X_1 of the positive tempered stable process X with parameters $\alpha_+ \in (0, 1)$, $\delta_+ > 0$ and $\gamma_+ > 0$. More information on the simulation of tempered stable random variables can be found in Barndorff-Nielsen and Shephard (2001b) and Kawai and Masuda (2011).

4.6 Inverse Gaussian Process

The **inverse Gaussian distribution** $\mathrm{IG}(\delta, \gamma) = \mathrm{GIG}(-1/2, \delta, \gamma)$ is a distribution with the probability density

$$\begin{aligned} p_{\mathrm{IG}}(x; \delta, \gamma) &= \frac{\delta}{\sqrt{2\pi}} e^{\delta\gamma} x^{-\frac{3}{2}} \exp\left[-\frac{1}{2}\left(\frac{\delta^2}{x} + \gamma^2 x \right) \right] \\ &= \left(\frac{\delta^2}{2\pi} \right)^{\frac{1}{2}} x^{-\frac{3}{2}} \exp\left[-\frac{\delta^2 (x - \delta\gamma^{-1})^2}{2(\delta\gamma^{-1})^2 x} \right] \end{aligned} \tag{4.5}$$

for $x > 0$, $\delta > 0$ and $\gamma > 0$. The case $\gamma = 0$ results in (4.13) with $\lambda = -1/2$. The fact the function $p_{\mathrm{IG}}(\cdot; \delta, \gamma)$ defines a probability distribution follows

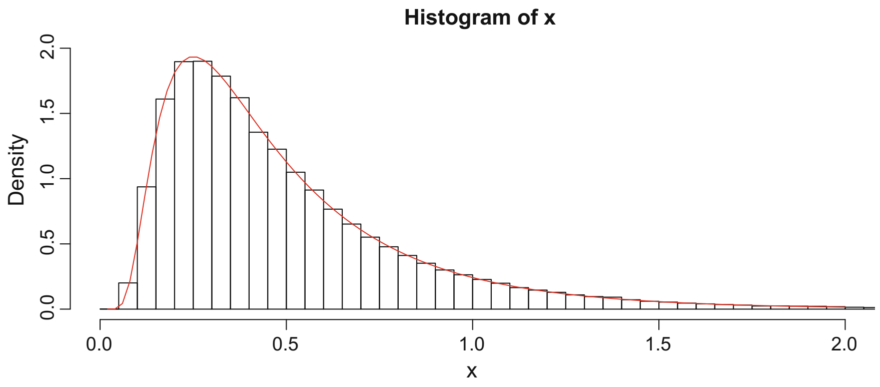


Fig. 4.5 Theoretical versus simulated Inverse Gaussian distribution

more generally from the expression of its Fourier transform[6] $\varphi_{\mathrm{IG}}(u;\delta,\gamma) = \int_0^\infty e^{iux} p_{\mathrm{IG}}(x;\delta,\gamma)dx$:

$$\varphi_{\mathrm{IG}}(u;\delta,\gamma) = \exp\left[\gamma\delta\left(1-\sqrt{1-2iu/\gamma^2}\right)\right] \quad, u \in \mathbb{R}. \tag{4.6}$$

Thus, the class of inverse Gaussian distributions satisfies the following reproducing property:

$$\mathrm{IG}(\delta_1,\gamma) * \mathrm{IG}(\delta_2,\gamma) = \mathrm{IG}(\delta_1+\delta_2,\gamma)$$

The infinite divisibility of $\mathrm{IG}(\delta,\gamma)$ is clear, and the corresponding Lévy process is called an **inverse Gaussian process** $\mathrm{IG}(\delta,\gamma)$ with parameters $\delta > 0$ and $\gamma > 0$. $\mathrm{IG}(\delta,\gamma)$ has mean $\delta\gamma^{-1}$ and variance $\delta\gamma^{-3}$. The reader can also refer to Barndorff-Nielsen (1997) and Chhikara and Folks (1989).

The function `rIG` with arguments `n`, `delta` and `gamma` generates n independent random numbers having the distribution $\mathrm{IG}(\delta,\gamma)$ (Michael et al. 1976) (Fig. 4.5).

[6]For $a > 0$ and $t \geq 0$, let $f(t) = \int_0^\infty x^{-3/2}\exp(-a/x - tx)dx = \int_0^\infty y^{-1/2}\exp(-ay - t/y)dy$. In particular, $f(0) = \sqrt{\pi/a}$. We see $f'(t) = -\sqrt{a/t}\, f(t)$ by changing variables by $x = a(yt)^{-1}$ for $f'(t) = -\int_0^\infty x^{-1/2}\exp(-a/x - tx)\, dx$. Solve the differential equation to show $\int_0^\infty x^{-3/2}\exp(-a/x - tx)dx = \sqrt{\pi/a}\, \exp(-2\sqrt{at})$. Then, we obtain (4.6) by substituting $\delta^2/2$ into a and $\gamma^2/2 - iu$ into t with analytic continuation.

```
delta <- 1
gamma <- 2
set.seed(127)
x <- rIG(100000,delta,gamma)
hist(x,xlim=c(0,2),ylim=c(0,2),breaks=100,freq=FALSE)
curve(dIG(x,delta,gamma),add=TRUE,col="red",
 from=min(x), to=max(x), n=500)
mean(x)

## [1] 0.5012324

var(x)

## [1] 0.1263361
```

We can build a `yuima` model of the inverse Gaussian process by `setModel`. We select `code` for the argument `measure.type` and `rIG(z,delta,gamma)` for the measure. Let us generate five paths as follows (Fig. 4.6).

```
IGmodel <- setModel(drift=0, xinit=0, jump.coeff=1,
 measure.type="code", measure=list(df="rIG(z, delta=1, gamma=2)"))
n <- 1000
samp <- setSampling(Terminal=1, n=n)
IGyuima <- setYuima(model=IGmodel, sampling=samp)
set.seed(127)
for (i in 1:5){
 result <- simulate(IGyuima,xinit=0)
 plot(result, xlim=c(0,1), ylim=c(0,1),
  main="Paths of IG process (delta=1, gamma=2)",par(new=T),col=i)
}
```

Next, let us compare the empirical distribution $\mathscr{L}\{X_1\}$ of simulated paths and IG(δ, γ). We take $n = 5$ to reduce time of simulation since the theoretical distribution is independent of a choice of n (Fig. 4.7).

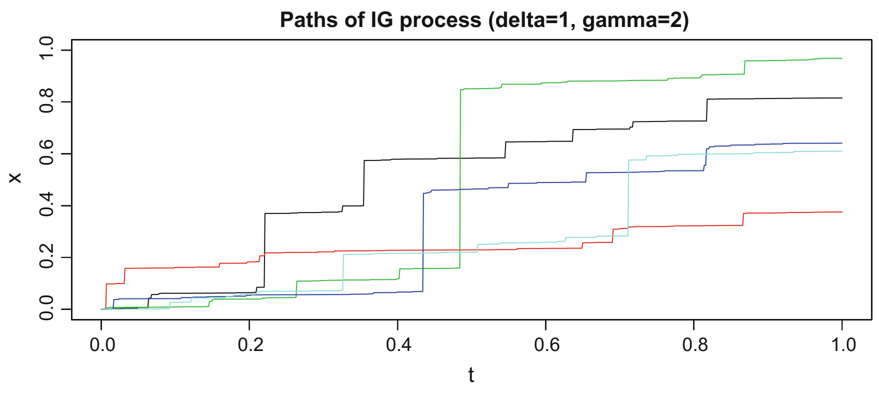


Fig. 4.6 Simulated paths of IG process

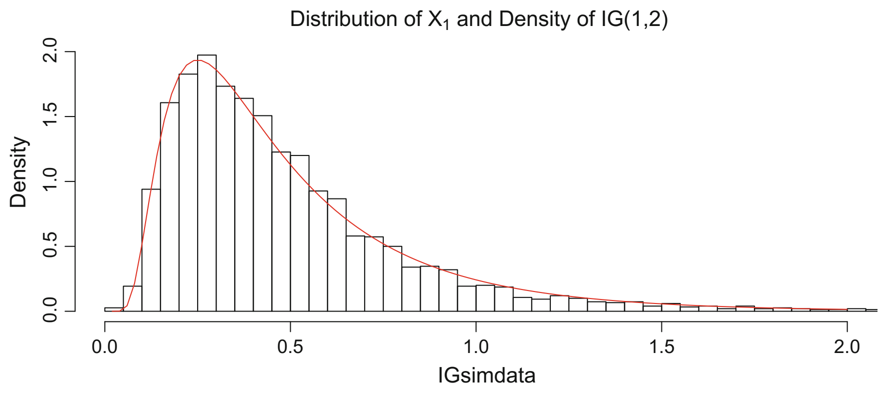


Fig. 4.7 Theoretical versus simulated IG(1,2) process distribution

```
n <- 5
samp <- setSampling(Terminal=1, n=n)
IGyuima <- setYuima(model=IGmodel, sampling=samp)
IGsimdata <- NULL
for (i in 1:3000){
 result <- simulate(IGyuima)
 x1 <- result@data@original.data[n+1,1]
 IGsimdata <- c(IGsimdata,as.numeric(x1))
}
hist(IGsimdata,xlim=c(0,2), ylim=c(0,2), breaks=100, freq=FALSE,
 main=expression(paste("Distribution of ",X[1],
 " and Density of IG(1,2)")))
curve(dIG(x,delta,gamma),add=TRUE,col="red",
 from = 0.001, to = 5, n=500)
```

It is know that the hitting time $\tau_\delta = \inf\{t > 0;\ \gamma t + W_t = \delta\}$ for a standard Wiener process W_t with $W_0 = 0$ has the inverse Gaussian distribution IG(δ, γ). The reproducing property looks natural by strong Markovian property of the Wiener process.

The characteristic function φ_{IG} admits a representation

$$\varphi_{\mathrm{IG}}(u;\delta,\gamma) = \exp\left[\int_0^\infty (e^{iux}-1)\frac{1}{\sqrt{2\pi}}\delta x^{-3/2}e^{-\frac{1}{2}\gamma^2 x}dx\right].$$

Indeed,

$$\sqrt{1-2iu/\gamma^2}-1 = \int_0^1 2^{-1}(-2iu/\gamma^2)/\sqrt{1-2ius/\gamma^2}ds$$
$$= \int_0^1 2^{-1}(-2iu/\gamma^2)\int_0^\infty \Gamma\left(\frac{1}{2}\right)^{-1}(\gamma^2/2)^{1/2}x^{-1/2}\exp\left[iusx - 2^{-1}\gamma^2 x\right]dxds$$
$$= \int_0^\infty (1-e^{iux})\gamma^{-1}(2\pi)^{-1/2}x^{-3/2}e^{-2^{-1}\gamma^2 x}dx.$$

Therefore, the characteristics are given by $b_- = 0$, $C = 0$ and

$$\frac{d\nu}{dx}(x) = 1_{\{x>0\}} \frac{1}{\sqrt{2\pi}} \delta x^{-3/2} e^{-\frac{1}{2}\gamma^2 x}.$$

In the case $\gamma = 0$, the distribution IG$(\delta, 0)$ is called a **Lévy distribution**. It has a density

$$p_{\text{Levy}}(x; \delta) = \frac{\delta}{\sqrt{2\pi}} x^{-\frac{3}{2}} \exp\left(- \frac{\delta^2}{x} \right),$$

and the corresponding Lévy measure is

$$\frac{d\nu}{dx}(x) = 1_{\{x>0\}} \frac{1}{\sqrt{2\pi}} \delta x^{-3/2}.$$

The process X is an increasing stable process for $\alpha = 1/2$ and $b_- = 0$ described in details in the next section.

4.7 Increasing Stable Process

For $\alpha \in (0, 1)$, the Lévy process with characteristics $b_- \geq 0$, $C = 0$ and

$$\frac{d\nu}{dx}(x) = 1_{\{x>0\}} \frac{\alpha c}{\Gamma(1-\alpha)} x^{-1-\alpha}$$

is called an **increasing stable process** $\text{S}_+(\alpha, b_-)$ (See, e.g., Sato 1999; Çınlar 2011). An $\text{S}_+(\alpha, b_-)$ process X_t has the characteristic function

$$\varphi_{X_t}(u) = \exp\left[t \left\{ - c_\alpha |u|^\alpha \left(1 - i \tan \frac{\pi\alpha}{2} \text{sign}(u)\right) + i b_- u \right\} \right]$$

where[7]

$$c_\alpha = c \cos \frac{\pi\alpha}{2}.$$

The positive tempered stable process PTSP(α, δ, γ) given in Sect. 4.5 is the Lévy process having characteristics $b_- = 0$, $C = 0$ and

[7]For $\alpha \in (0, 1)$, $u \in \mathbb{R}$ and $\lambda > 0$, one has $\int_0^\infty \left(e^{(-\lambda+iu)x} - 1\right) x^{-1-\alpha} dx = \int_0^\infty (-\lambda + iu) \int_0^x e^{(-\lambda+iu)s} ds x^{-1-\alpha} dx = \alpha^{-1}(-\lambda + iu) \int_0^\infty e^{(-\lambda+iu)s} s^{-\alpha} ds = -(\lambda - iu)^\alpha \alpha^{-1} \Gamma(1-\alpha)$. Let $\lambda \downarrow 0$ with $\lim_{\lambda\downarrow 0}(\lambda - iu)^\alpha = |u|^\alpha \exp(-i\frac{\pi\alpha}{2}\text{sign}(u))$.

$$\frac{d\nu}{dx}(x) = \delta x^{-1-\alpha} e^{-\gamma x} 1_{\{x>0\}},$$

where $\delta \geq 0$, $\gamma > 0$ and $\alpha \in (0, 1)$.

The function `rpts` generates random numbers of the positive tempered stable distribution. Let us generate random numbers of X_1 for a positive tempered stable process X in two ways X_1 and $X_{1/2} + (X_1 - X_{1/2}) = X_{1/2} + X'_{1/2}$ for an independent copy $X'_{1/2}$ of $X_{1/2}$.

```
rep <- 3000000
set.seed(129)
X1 <- rpts(rep,0.5,0.2,1)
hist(X1,xlim=c(0,3),ylim=c(0,3),breaks=100,
 main=expression(X[1]),probability=TRUE)
X05 <- rpts(rep,0.5,0.1,1)
X05.prime <- rpts(rep,0.5,0.1,1)
Xsum <- X05+X05.prime
summary(X1)

##      Min.   1st Qu.    Median      Mean   3rd Qu.
##  0.008864  0.111001  0.212039  0.354374  0.427638
##      Max.
## 10.587877

summary(Xsum)

##      Min.   1st Qu.    Median      Mean   3rd Qu.
##  0.008556  0.110882  0.211898  0.354134  0.426951
##      Max.
## 12.397137

ks.test(X1,Xsum)

##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  X1 and Xsum
## D = 0.00069933, p-value = 0.4555
## alternative hypothesis: two-sided
```

The above code shows the summary statistics for the simulated data and the results of the Kolmogorov–Smirnov test which does not reject the hypothesis of equality in distribution of the simulated data from X_1 and $X_{1/2} + X'_{1/2}$. Figure 4.8 represents the histogram for the simulated data from X_1. The random number generator uses acceptance/rejection method with the acceptance rate $= \exp(\delta\Gamma(-\alpha)\gamma^\alpha)$. A warning appears if the acceptance rate is too small. This will be not a restriction in simulation of stochastic differential equations because δ is proportional to the discretization step size.

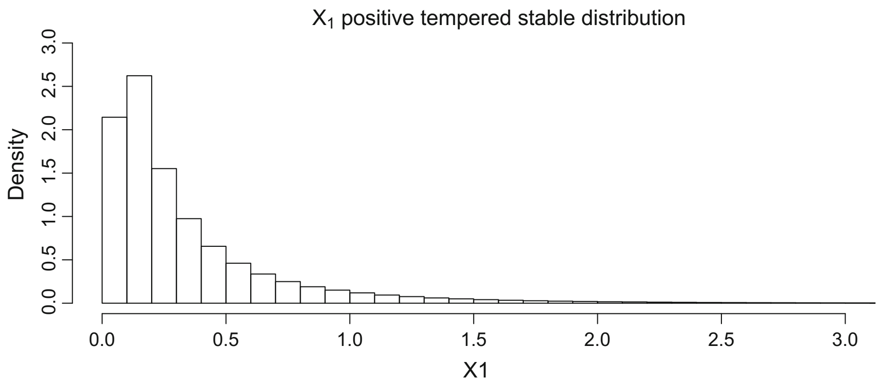


Fig. 4.8 Histogram of simulated data from a positive tempered stable distribution

4.8 Subordination

4.8.1 Definition

Given a probability space $(\Omega, \mathscr{F}, P)$, let $Y = (Y_s)_{t\in\mathbb{S}}$ be a d-dimensional stochastic process with a parameter s in a set $\mathbb{S}$. For example, $\mathbb{S} = \mathbb{R}_+$, $\mathbb{N} = \{1, 2, ...\}$, $\mathbb{Z}_+ = \{0, 1, ...\}$ and so on. For a process $S = (S_t)_{t\in\mathbb{T}}$ defined on Ω, taking values in $\mathbb{S}$ and having a parameter space $\mathbb{T}$, we can make a new process $X = (X_t)_{t\in\mathbb{T}}$ by

$$X_t(\omega) = Y_{S_t(\omega)}(\omega) \qquad (\omega \in \Omega,\ t \in \mathbb{T}).$$

This procedure to obtain Y_S is called **subordination** and S is called a **subordinator**. The process X made by subordinating Y to a nondecreasing process in an order of S is denoted by Y_S. Usually "subordinator" means a nondecreasing process.

4.8.2 Compound Poisson Process by Subordination

The compound Poisson process discussed in Sect. 4.3 is an example of subordination. That is, for an i.i.d. sequence $\xi = (\xi_j)_{j\in\mathbb{N}}$ and a Poisson process $S = (S_t)_{t\in\mathbb{Z}_+}$ independent of ξ, the compound Poisson process is given by $Y_S = (Y_S)_{t\in\mathbb{Z}_+}$ for $Y = (Y_s)_{s\in\mathbb{Z}_+}$, with $Y_0 = 0$ and $Y_s = \sum_{j=1}^{s} \xi_j$ for $s \geq 1$.

4.8.3 Subordination of a Wiener Process with Drift

For a d-dimensional standard Wiener process $W = (W_t)_{t\in\mathbb{R}_+}$ and $\beta \in \mathbb{R}^{\mathsf{d}}$, we consider subordination of the process $(\beta t + W_t)_{t\in\mathbb{R}_+}$ to a one-dimensional nondecreasing Lévy process S with characteristics $b_- = b_-^S \geq 0$ in the sense of (4.3), $C = 0$ and $\nu = \nu^S$ charging only on $(0, \infty)$ and $\int_0^\infty (x \wedge 1)\nu^S(dx) < \infty$.

In a similar way as in Sect. 4.3, it is easy to see that the process $X_t = S_t\beta + W_{S_t}$ is a Lévy process. In a special case where S is a compound Poisson process with bounded jumps, with the aid of analytic continuation (in the second equality below), we have

$$
\begin{aligned}
&E\big[\exp\{iu\cdot(S_t\beta + W_{S_t})\}\big]\\
&= E\big[\exp\{iu\cdot(\beta + 2^{-1}iu)S_t\}\big]\\
&= \exp\left[t\left\{b_-^S iu\cdot(\beta+2^{-1}iu) + \int_0^\infty \Big(\exp\big(iu\cdot(\beta+2^{-1}iu)s\big) - 1\Big)\nu^S(ds)\right\}\right]\\
&= \exp\left[t\left\{b_-^S iu\cdot(\beta+2^{-1}iu) + \int_0^\infty \left(\int_{\mathbb{R}^{\mathsf{d}}} (e^{iu\cdot x} - 1)\phi(x; s\beta, sI_{\mathsf{d}})dx\right)\nu^S(ds)\right\}\right]\\
&= \exp\left[t\left\{iu\cdot\left(b_-^S\beta + \int_0^\infty \nu^S(ds)\int_{|x|\leq 1} x\phi(x; s\beta, sI_{\mathsf{d}})dx\right) - 2^{-1}b_-^S|u|^2\right.\right.\\
&\quad\left.\left. + \int_0^\infty\int_{\mathbb{R}^{\mathsf{d}}} (e^{iu\cdot x} - 1 - iu\cdot x 1_{\{|x|\leq 1\}})\phi(x; \beta s, s)dx\, \nu^S(ds)\right\}\right]\\
&= \exp\left[t\left\{iu\cdot\left(b_-^S\beta + \int_0^\infty \nu^S(ds)\int_{|x|\leq 1} x\phi(x; s\beta, sI_{\mathsf{d}})dx\right) - 2^{-1}b_-^S|u|^2\right.\right.\\
&\quad\left.\left. + \int_{\mathbb{R}^{\mathsf{d}}} (e^{iu\cdot x} - 1 - iu\cdot x 1_{\{|x|\leq 1\}})\left(\int_0^\infty \phi(x; s\beta, sI_{\mathsf{d}})\nu^S(ds)\right)dx\right\}\right]
\end{aligned}
$$

for $u \in \mathbb{R}^{\mathsf{d}}$.[8] Thus, the resulting characteristics (b^X, C^X, ν^X) for (b, C, ν) in (4.2) are given by

$$
\begin{cases}
b^X = b_-^S\beta + \int_0^\infty \nu^S(ds)\int_{|x|\leq 1} x\phi(x; s\beta, sI_{\mathsf{d}})dx,\\
C^X = b_-^S I_{\mathsf{d}},\\
\nu^X(dx) = \int_0^\infty \phi(x; s\beta, sI_{\mathsf{d}})\nu^S(ds)\, dx,
\end{cases}
\tag{4.7}
$$

where $\phi(\cdot; \mu, \Sigma)$ denotes the normal density function with mean vector μ and covariance matrix Σ.

As a matter of fact, the representation (4.7) of characteristics is valid for general subordinator S. Note that

[8]The above computation (second equation) works by analytic continuation with respect to iu in the present case, but it is incorrect in general, where Fubini's theorem cannot be applied.

$$\left|\int_{|x|\le 1} x\phi(x; s\beta, sI_{\mathsf{d}})dx\right| = O(s)$$

and

$$\int_{|x|\le 1} |x|^2\phi(x; s\beta, sI_{\mathsf{d}})dx = O(s)$$

as $s \downarrow 0$ are necessary conditions for the validation of the above result, and that

$$\int_{|x|\le 1} |x|\phi(x; s\beta, sI_{\mathsf{d}})dx \ge c\sqrt{s}$$

as $s \downarrow 0$ for some positive constant c for the validity of the representation (4.2). The Lévy measure has a representation

$$\nu^X(B) = \int_0^\infty E[1_B(Y_s)]\nu^S(ds) \qquad (B \in \mathbb{B}[\mathbb{R}^{\mathsf{d}}])$$

with subordinated process $Y_s = \beta s + W_s$.

For a d-dimensional standard Gaussian variable ζ, $\mathscr{L}\{S_t\beta + W_{S_t}\} = \mathscr{L}\{S_t\beta + \sqrt{S_t}\zeta\}$. Therefore, the density of $X_t = S_t\beta + W_{S_t}$ is

$$p_{X_t}(x) = \int_0^\infty \phi(x; s\beta, sI_{\mathsf{d}})p_{S_t}(s)ds \tag{4.8}$$

for the density function p_{S_t} of S_t.

4.8.4 Variance Gamma Process with Drift

Let $\lambda, \alpha \in (0, \infty)$, $\beta \in \mathbb{R}$ $(\alpha > |\beta|)$ and $\mu \in \mathbb{R}$. For a gamma process $S = (S_t)_{t\in\mathbb{R}_+}$ such that

$$S_t \sim \Gamma(\lambda t, (\alpha^2 - \beta^2)/2),$$

let

$$X_t = \mu t + \beta S_t + W_{S_t}.$$

Then, $X = (X_t)_{t\in\mathbb{R}_+}$ is a Lévy process with

$$\varphi_{X_t}(u) = e^{i\mu t u}\left[\frac{\alpha^2 - (\beta + iu)^2}{\alpha^2 - \beta^2}\right]^{-\lambda t} \tag{4.9}$$

for $u \in \mathbb{R}$. Let us call the process X a **variance gamma process** VGP$(\lambda, \alpha, \beta, \mu)$, and the distribution of X_1 the **variance gamma distribution** VG$(\lambda, \alpha, \beta, \mu)$. From (4.8) and (4.4), we obtain the density of X_t

$$p_{X_t}(x) = \frac{1}{\sqrt{\pi}\Gamma(\lambda t)} \left(\alpha^2 - \beta^2\right)^{\lambda t} \left(\frac{|x - \mu t|}{2\alpha}\right)^{\lambda t - \frac{1}{2}} K_{\lambda t - \frac{1}{2}}\left(\alpha |x - \mu t|\right) \exp(\beta(x - \mu t)).$$

Remark 4.1 The **yuima** package provides the random number generator `rvgamma` and the density function `dvgamma`. Very old releases of the **yuima** package used to name the above functions as "normal gamma"; however the term "normal gamma distribution" is often used for the two-dimensional distribution that has the following density

$$p(x, t) = 1_{\mathbb{R}\times(0,\infty)}(x, t) \frac{\beta^\alpha}{\Gamma(\alpha)} \sqrt{\frac{\lambda}{2\pi}} t^{\alpha - \frac{1}{2}} e^{-\beta t} \exp\left(- \lambda t (x - \mu)^2/2\right),$$

which is not considered here.

We now generate a few paths of the variance gamma process as follows (the results are shows in Fig. 4.9).

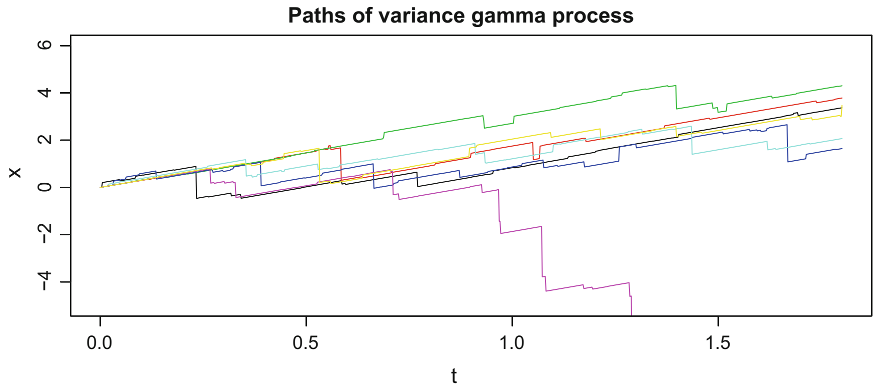


Fig. 4.9 Examples of variance gamma process as Wiener subordinator

```
lambda <- 2
alpha <- 1.5
beta <- -0.7
mu <- 3
xinit <- 0
gamma <- sqrt(alpha^2-beta^2)
n <- 1000
T <- 1.8
VGPmodel <- setModel(drift=0, jump.coeff=1, measure.type="code",
 measure=list(df="rvgamma(z,lambda,alpha,beta,mu)"))
samp <- setSampling(Terminal=T, n=n)
VGPyuima <- setYuima(model=VGPmodel, sampling=samp)
# simulation
set.seed(127)
for (i in 1:7) {
 result <- simulate(VGPyuima, xinit=xinit,
 true.par=list(lambda=lambda,alpha=alpha,beta=beta,mu=mu))
plot(result,xlim=c(0,T),ylim=c(-5,6),col=i,
main="Paths of variance gamma process",par(new=T))
}
```

Next, X_t is simulated and its histogram is compared with the density function of VG$(\lambda t, \alpha, \beta, \mu t)$ (see Fig. 4.10)

```
n <- 5
samp <- setSampling(Terminal=T, n=n)
VGPyuima <- setYuima(model=VGPmodel, sampling=samp)
VGPsimdata <- NULL
for (i in 1:5000){
 result <- simulate(VGPyuima, xinit=xinit,
  true.par=list(lambda=lambda,alpha=alpha,beta=beta,mu=mu))
  x1 <- result@data@original.data[n+1,1]
  VGPsimdata <- c(VGPsimdata,as.numeric(x1[1]))
}
hist(VGPsimdata,xlim=c(-7,10),ylim=c(0,0.22),breaks=100,freq=FALSE,
 main=expression(paste("Distribution of ",X[1.8],
 " and Density of VG")))
curve(dvgamma(x,lambda*T,alpha,beta,mu*T),add=TRUE,col="red")
```

4.8.5 Normal Inverse Gaussian Process

Let $S = (S_t)_{t\in\mathbb{R}_+}$ be an inverse Gaussian process with $\mathscr{L}\{S_1\} = \mathrm{IG}(\delta, \sqrt{\alpha^2-\beta^2})$, where $\delta, \alpha \in (0,\infty)$, $\beta \in \mathbb{R}$ satisfying $\alpha \geq |\beta|$; $\gamma = \sqrt{\alpha^2-\beta^2}$ in the notation in Sect. 4.6. Define $X = (X_t)_{t\in\mathbb{R}_+}$ by

$$X_t = \mu t + \beta S_t + W_{S_t},$$

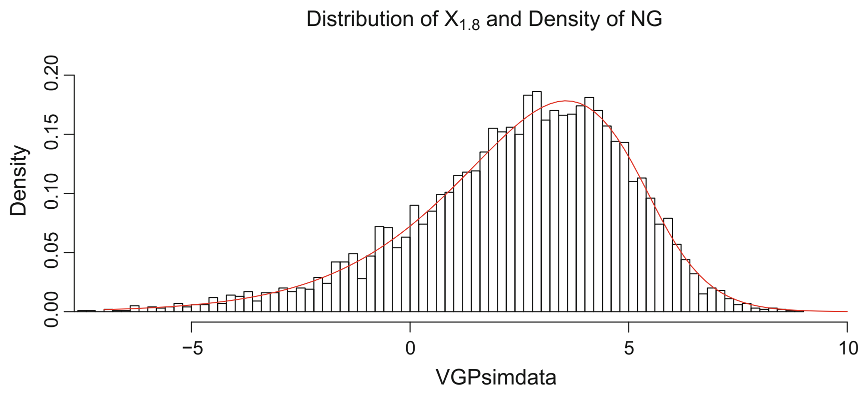


Fig. 4.10 Theoretical and empirical variance gamma distribution

where $\mu \in \mathbb{R}$ and $W = (W_t)_{t\in\mathbb{R}_+}$ is a one-dimensional Wiener process independent of S. Since $\beta S + W_S$ is a subordination of a one-dimensional drifting Wiener process $Y_s = \beta s + W_s$ by the inverse Gaussian process S, the process $\beta S + W_S$ and hence X is a Lévy process. The process X is called a **normal inverse Gaussian process** NIGP$(\alpha, \beta, \delta, \mu)$. This Lévy process X is characterized by the distribution

$$\mathscr{L}\{X_1\} = \mathscr{L}\{\mu + \beta S_1 + W_{S_1}\}. \tag{4.10}$$

and the distribution (4.10) is called a **normal inverse Gaussian distribution** NIG$(\alpha, \beta, \delta, \mu)$. By

$$E[\exp(iu(\beta S_1 + W_{S_1})] = E[\exp(iu(\beta + 2^{-1}iu)S_1)]$$

and (4.6), we obtain the characteristic function of NIG$(\alpha, \beta, \delta, \mu)$:

$$\varphi_{\mathrm{NIG}}(u) = \exp\left[i\mu u + \delta\left(\sqrt{\alpha^2 - \beta^2} - \sqrt{\alpha^2 - (\beta + iu)^2}\right)\right]$$

for $u \in \mathbb{R}$. A simple reproductive property of the NIG family is

$$\mathrm{NIG}(\alpha, \beta, \delta_1, \mu_1) * \mathrm{NIG}(\alpha, \beta, \delta_2, \mu_2) = \mathrm{NIG}(\alpha, \beta, \delta_1 + \delta_2, \mu_1 + \mu_2).$$

For the NIGP$(\alpha, \beta, \delta, \mu)$ X, we have $X_t \sim \mathrm{NIG}(\alpha, \beta, \delta t, \mu t)$ with characteristic function

$$\varphi_{X_t}(u) = \exp\left[it\mu u + t\delta\left(\sqrt{\alpha^2 - \beta^2} - \sqrt{\alpha^2 - (\beta + iu)^2}\right)\right].$$

Simple calculus gives the characteristics (b, C, ν) of the NIGP$(\alpha, \beta, \delta, \mu)$ as follows. The Lévy density

$$\frac{d\nu}{dx}(x) = \frac{\alpha\delta}{\pi|x|}e^{\beta x}K_1(\alpha|x|)$$

is obtained by (4.7) and (4.16) with (4.17) after transforming $s = |x|\alpha^{-1}t$. K_1 is the Bessel function of third kind whose definition is reminded in Sect. 4.13 for the benefit of the reader. The second characteristic $C_{\mathrm{NIG}} = 0$ and the first one become

$$\begin{aligned} b &= \int_0^\infty \frac{\delta}{\sqrt{2\pi}} s^{-3/2} e^{-\frac{1}{2}\gamma^2 s} ds \left\{ \int_0^1 x \frac{1}{\sqrt{2\pi s}} \exp\left[-\frac{(x-\beta s)^2}{2s}\right] dx \right. \\ &\quad \left. - \int_0^1 x \frac{1}{\sqrt{2\pi s}} \exp\left[-\frac{(x+\beta s)^2}{2s}\right] dx \right\} \\ &= \frac{2\alpha\delta}{\pi} \int_0^1 \sinh(\beta x) K_1(\alpha x) dx. \end{aligned}$$

Since $K_1(x) \sim x^{-1}$ when $x \downarrow 0$, $d\nu(x)/dx \sim$ constant $\cdot\, x^{-2}$ near $x = 0$. The Blumenthal-Getoor index of NIGP is 1.

By (4.5), (4.8) taking the shift μt into account, (4.16) and (4.17) and the transform $s = \alpha^{-1}\sqrt{(\delta t)^2 + (x-\mu t)^2}\, y$, we obtain the density function of X_t as

$$\begin{aligned} p_{X_t}(x) &= p_{\mathrm{NIG}}(x; \alpha, \beta, \delta t, \mu t) \\ &= a(-1/2, \alpha, \beta, \delta t)\left((\delta t)^2 + (x-\mu t)^2\right)^{-1/2} \times \\ &\quad K_1\left(\alpha\sqrt{(\delta t)^2 + (x-\mu t)^2}\right) \exp\left(\beta(x-\mu t)\right) \end{aligned}$$

where

$$a(-1/2, \alpha, \beta, \delta t) = \frac{\alpha\delta t}{\pi} \exp\left(\delta t\sqrt{\alpha^2 - \beta^2}\right).$$

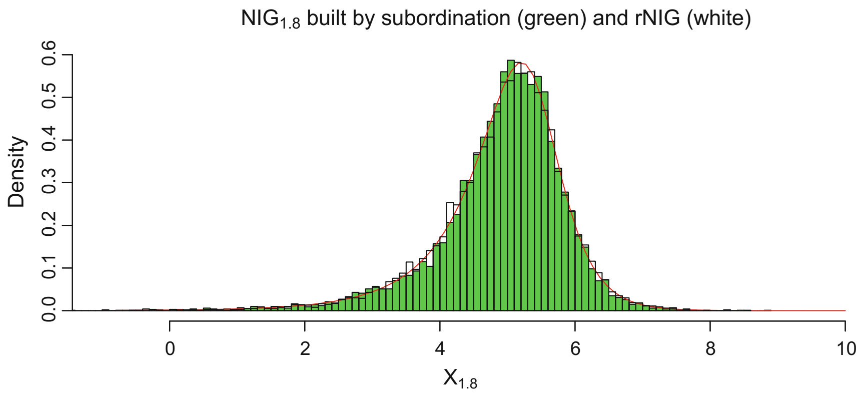


Fig. 4.11 Simulated and theoretical data from the NIG distribution

Let us now build X_t by subordination and compare its histogram with the empirical distribution of X_t generated by `rNIG(`$\alpha, \beta, \delta t, \mu t$`)` and the density function `dNIG(`$\alpha, \beta, \delta t, \mu t$`)`. The results are shown in Fig. 4.11.

```
delta <- 0.5
alpha <- 1.5
beta <- -0.7
mu <- 3
gamma <- sqrt(alpha^2-beta^2)
n <- 10000
T <- 1.8
set.seed(127)
normal.rn <- rnorm(n,0,1)
iv.rn <- rIG(n,delta*T,gamma)
z <- mu*T+beta*iv.rn+sqrt(iv.rn)*normal.rn
title <- expression(paste(NIGP[1.8],
 " built by subordination (green) and rNIG (white)"))
nig.rn <- rNIG(n,alpha,beta,delta*T,mu*T)
hist(z,xlim=c(-1,10),ylim=c(0,0.61),breaks=100, freq=FALSE,
 col="green", main=title, xlab=expression(X[1.8]) )
curve(dNIG(x,alpha,beta,delta*T,mu*T),add=TRUE,col="red")
par(new=T)
hist(nig.rn,xlim=c(-1,10),ylim=c(0,0.61),breaks=100,
 freq=FALSE, main="", xlab="")
```

It is also possible to generate trajectories of X_t as follows (results for the next code are given in Fig. 4.12).

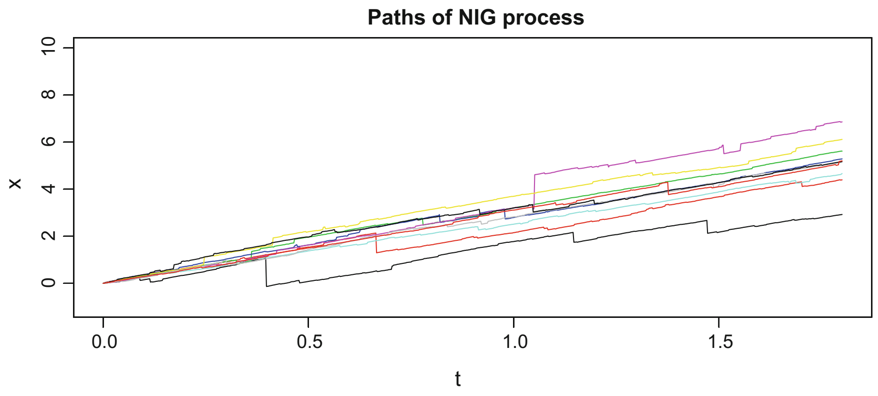


Fig. 4.12 Simulated and theoretical data from the NIG distribution

```
delta1 <- 0.5
alpha <- 1.5
beta <- -0.7
mu <- 3
xinit <- 0
gamma <- sqrt(alpha^2-beta^2)
n <- 1000
T <- 1.8
NIG2model <- setModel(drift=0, jump.coeff=1, measure.type="code",
 measure=list(df="rNIG(z,alpha,beta,delta1,mu)"))
samp <- setSampling(Terminal=T, n=n)
NIG2yuima <- setYuima(model=NIG2model, sampling=samp)
set.seed(127)
for (i in 1:10) {
 result <- simulate(NIG2yuima, xinit=xinit,
  true.par=list(delta1=delta1, alpha=alpha, beta=beta,
  mu=mu, gamma=gamma))
plot(result,xlim=c(0,T),ylim=c(-1,10),col=i,
 main="Paths of NIG process",par(new=T))
}
```

Function `simulate` also allows to generate the empirical distribution of X_t for any n as shown in Fig. 4.13.

```
n <- 5
samp <- setSampling(Terminal=T, n=n)
NIG2yuima <- setYuima(model=NIG2model, sampling=samp)
NIG2data <- NULL
for (i in 1:3000){
 result <- simulate(NIG2yuima, xinit=xinit,
  true.par=list(delta1=delta1, alpha=alpha, beta=beta,
  mu=mu, gamma=gamma))
 x1 <- result@data@original.data[n+1,1]
 NIG2data <- c(NIG2data,as.numeric(x1[1]))
}
hist(NIG2data,xlim=c(2,8),ylim=c(0,0.8),breaks=100, freq=FALSE,
 main=expression(paste("Distribution of ",X[1.8],
 " and Density of NIG")))
curve(dNIG(x,alpha,beta,delta*T,mu*T),add=TRUE,col="red")
```

4.8.6 *Normal Tempered Stable Process*

For a positive tempered stable process S in PTSP(α, δ, γ) with $\alpha \in (0, 1)$, $\delta > 0$ and $\gamma > 0$, the subordination

$$X_t = \mu t + \beta S_t + W_{S_t}$$

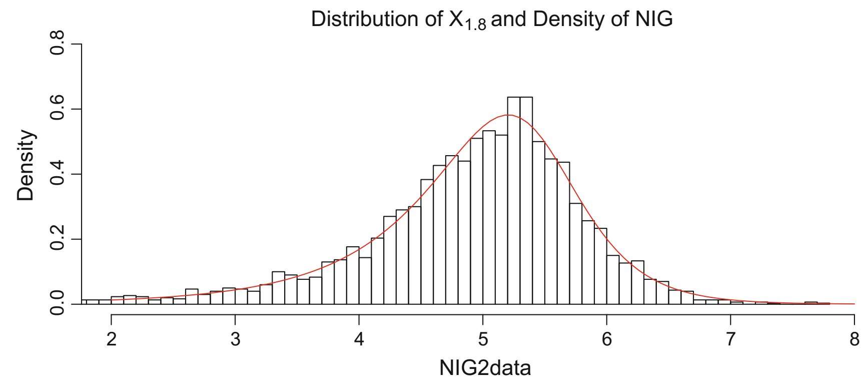


Fig. 4.13 Simulated and theoretical data from the NIG process at time 1.8

defines the one-dimensional **normal tempered stable process** NTSP$(\alpha, \delta, \gamma, \beta, \mu)$. We call the distribution of X_1 the **normal tempered stable distribution**, and we denote it by NTS$(\alpha, \delta, \gamma, \beta, \mu)$. The **yuima** package provides a random number generator obeying the distribution of X_t by the function `rnts(x,alpha,a,b, beta,mu,Lambda)`. The parameter `Lambda` is for multi-dimensional extension but `Lambda=1` in the one-dimensional case. We now simulate the normal tempered stable process directly and by convolution as in the previous examples. Figure 4.14 shows the results and the Kolmogorov–Smirnov test confirm the equality in distribution among the data.

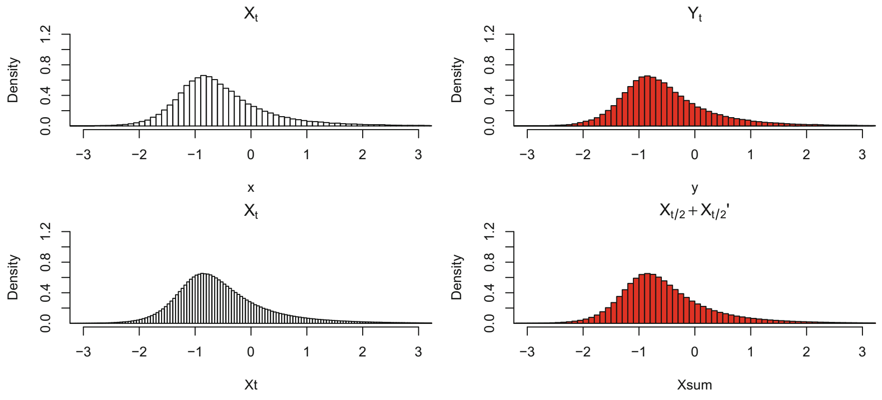


Fig. 4.14 Normal tempered stable process with direct simulation and convolution

```
nrep <- 100000
alpha <- 0.5
delta <- 0.2
gamma <- 1
beta <- 1
mu <- -0.7
Lambda <- matrix(1,1,1)
t <- 1.5
par(mfrow=c(2,2))
set.seed(127)
x <- rnts(nrep,alpha,delta*t,gamma,beta,mu*t,Lambda)
s <- rpts(nrep,alpha,delta*t,gamma)
w <- rnorm(nrep,0,1)
y <- rep(mu*t,nrep) + beta*s + sqrt(s)*w
hist(x,xlim=c(-3,3),ylim=c(0,1.2),breaks=200,
 main=expression(X[t]),probability=TRUE)
hist(y,xlim=c(-3,3),ylim=c(0,1.2),breaks=200,
 main=expression(Y[t]),probability=TRUE,col="red")
## experiment by convolution
nrep <- 3000000
Xt <- rnts(nrep,alpha,delta*t,gamma,beta,mu*t,Lambda)
X05 <- rnts(nrep,alpha,delta*t/2,gamma,beta,mu*t/2,Lambda)
X05.prime <- rnts(nrep,alpha,delta*t/2,gamma,beta,mu*t/2,Lambda)
Xsum <- X05+X05.prime
hist(Xt,xlim=c(-3,3),ylim=c(0,1.2),breaks=300,
 main=expression(X[t]),probability=TRUE)
hist(Xsum,xlim=c(-3,3),ylim=c(0,1.2),breaks=300,
 main=expression(paste(X[t/2]+X[t/2],"'")),
 probability=TRUE,col="red")
ks.test(Xt,Xsum)

##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  Xt and Xsum
## D = 0.00096067, p-value = 0.1255
## alternative hypothesis: two-sided
```

4.9 Stable Process

For a Lévy process $X = (X_t)_{t\in\mathbb{R}_+}$, suppose that each $\mathscr{L}\{X_t\}$ is a stable distribution, that is, for any $a > 0$, there exist $c > 0$ and $b \in \mathbb{R}$ such that

$$\varphi_{X_t}(u)^a = \varphi_{X_t}(cu)e^{ibu} \quad (u \in \mathbb{R}).$$

The, it is known that $c = a^{1/\alpha}$ for some $\alpha \in (0, 2]$, and that the characteristic function of X_t has one of the following representations:

(i) for $\alpha \in (0, 1) \cup (1, 2)$,

$$\varphi_{X_t}(u) = \exp\left[i\gamma tu - t\sigma^\alpha |u|^\alpha \left(1 - i\beta \operatorname{sign}(u) \tan \frac{\pi\alpha}{2}\right)\right],$$

(ii) for $\alpha = 1$,

$$\varphi_{X_t}(u) = \exp\left[i\gamma tu - t\sigma |u| \left(1 + i\beta \frac{2}{\pi} \operatorname{sign}(u) \log |u|\right)\right],$$

(iii) for $\alpha = 2$,

$$\varphi_{X_t}(u) = \exp\left[i\gamma tu - \frac{1}{2} t\sigma^2 u^2\right]$$

for $u \in \mathbb{R}$, where $\sigma \in (0, \infty)$, $\beta \in [-1, 1]$ and $\gamma \in \mathbb{R}$ are constants.

The process X is called an **α-stable Lévy process** or **stable process** $\mathrm{SP}(\alpha, \beta, \sigma, \gamma)$. The increasing stable process in Sect. 4.7 is an α-stable Lévy process. The **Cauchy process** is the 1-stable Lévy process with parameters $\sigma \in (0, \infty)$, $\beta = 0$ and $\gamma \in \mathbb{R}$.

Any nontrivial α-stable Lévy process X with $\alpha \in (0, 2)$ has a Lévy measure of the form

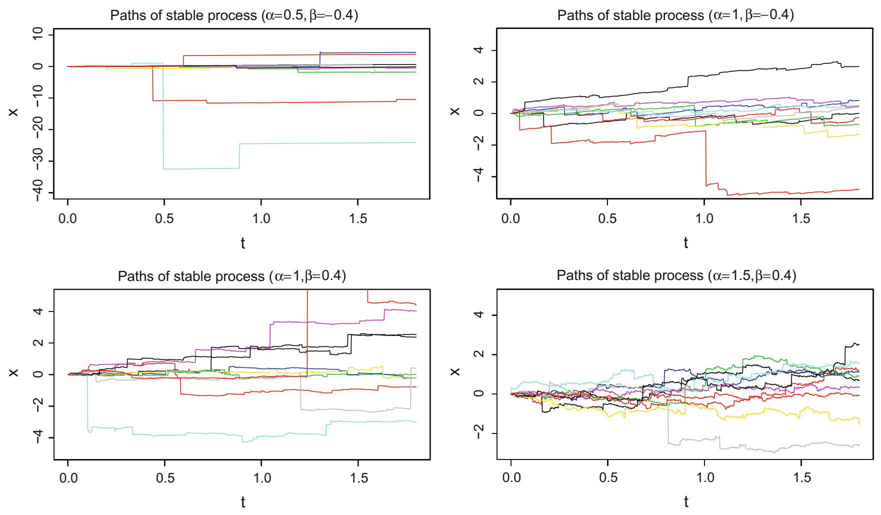


Fig. 4.15 The α-stable process with different parametrizations for α and β

$$\frac{d\nu}{dx}(x) = \delta_-|x|^{-1-\alpha}1_{(-\infty,0)}(x) + \delta_+ x^{-1-\alpha}1_{(0,\infty)}(x)$$

for some constants $\delta_\pm \geq 0$ ($\delta_- + \delta_+ > 0$). The **yuima** package provides random number generators for α-stable Lévy processes. The next example simulates some paths from the α-stable process for different values of α and β. The results are in Fig. 4.15.

```
alpha <- 0.5
beta <- -0.4
sigma <- 0.7
gamma <- 0.5
n <- 1000
T <- 1.8
ASmodel <- setModel(drift=0, jump.coeff=1, measure.type="code",
 measure=list(df="rstable(z,alpha,beta,sigma,gamma)"))
samp <- setSampling(Terminal=T, n=n)
ASyuima <- setYuima(model=ASmodel, sampling=samp)
set.seed(129)
for (i in 1:10) {
 result <- simulate(ASyuima, true.par=list(alpha=alpha,
  beta=beta,sigma=sigma,gamma=gamma))
 plot(result,xlim=c(0,T),ylim=c(-40,10),col=i,
  main=expression(paste("Paths of stable process (",
  alpha==0.5,",",beta==-0.4,")")),par(new=T))
 }

#param2
alpha <- 1
beta <- -0.4
sigma <- 0.7
gamma <- 0.5
AS2model <- setModel(drift=0, jump.coeff=1, measure.type="code",
 measure=list(df="rstable(z,alpha,beta,sigma,gamma)"))
AS2yuima <- setYuima(model=AS2model, sampling=samp)
for (i in 1:10) {
 result <- simulate(AS2yuima, true.par=list(alpha=alpha,
  beta=beta,sigma=sigma,gamma=gamma))
 plot(result,xlim=c(0,T),ylim=c(-5,5),col=i,
 main=expression(paste("Paths of stable process (",
 alpha==1,",",beta==-0.4,")")),par(new=T))
}

#param3
alpha <- 1
beta <- 0.4
sigma <- 0.7
gamma <- 0.5
AS3model <- setModel(drift=0, jump.coeff=1, measure.type="code",
 measure=list(df="rstable(z,alpha,beta,sigma,gamma)"))
AS3yuima <- setYuima(model=AS3model, sampling=samp)
for (i in 1:10) {
 result <- simulate(AS3yuima, true.par=list(alpha=alpha,
  beta=beta,sigma=sigma,gamma=gamma))
plot(result,xlim=c(0,T),ylim=c(-5,5),col=i,
 main=expression(paste("Paths of stable process (",
 alpha==1,",",beta==0.4,")")),par(new=T))
}

#param4
alpha <- 1.5
beta <- 0.4
sigma <- 0.7
gamma <- 0.5
AS4model <- setModel(drift=0, jump.coeff=1, measure.type="code",
 measure=list(df="rstable(z,alpha,beta,sigma,gamma)"))
AS4yuima <- setYuima(model=AS4model, sampling=samp)
for (i in 1:10) {
 result <- simulate(AS4yuima, true.par=list(alpha=alpha,
  beta=beta, sigma=sigma,gamma=gamma))
 plot(result,xlim=c(0,T),ylim=c(-3,5),col=i,
  main=expression(paste("Paths of stable process (",
  alpha==1.5,",",beta==0.4,")")),par(new=T))
}
```

4.10 Generalized Hyperbolic Processes

4.10.1 Generalized Inverse Gaussian Distribution

The **generalized inverse Gaussian distribution** GIG$(\lambda, \delta, \gamma)$ is a probability measure on $(0, \infty)$ with the density function

$$p_{\mathrm{GIG}}(x; \lambda, \delta, \gamma) = \frac{(\gamma/\delta)^{\lambda}}{2K_{\lambda}(\gamma\delta)} x^{\lambda-1} \exp\left[-\frac{1}{2}\left(\frac{\delta^2}{x} + \gamma^2 x\right)\right] \quad (x > 0) \quad (4.11)$$

The characteristic function of GIG$(\lambda, \delta, \gamma)$ is

$$\varphi_{\mathrm{GIG}}(u; \lambda, \delta, \gamma) = \left(1 - 2iu/\gamma^2\right)^{-\frac{\lambda}{2}} \frac{K_{\lambda}\left(\gamma\delta\sqrt{1 - 2iu/\gamma^2}\right)}{K_{\lambda}(\gamma\delta)} \quad (u \in \mathbb{R}) \quad (4.12)$$

The inverse Gaussian distribution IG(δ, γ) in Sect. 4.6 is a special case of GIG $(\lambda, \delta, \gamma)$ for $\lambda = -1/2$.

The function (4.11) is not generally integrable. Integrability of the right-hand side constrains the range of parameters as follows.

(i) $\lambda > 0$, $\delta \geq 0$, $\gamma > 0$. In particular, GIG$(\lambda, 0, \gamma) = \Gamma(\lambda, \gamma^2/2)$, the gamma distribution with density function

$$\begin{aligned}p_{\mathrm{GIG}}(x;\lambda,0,\gamma) &= p_{\Gamma}(x;\lambda,\gamma^2/2)\\ &= \frac{1}{\Gamma(\lambda)}\left(\frac{\gamma^2}{2}\right)^{\lambda} x^{\lambda-1}\exp\left(-\frac{\gamma^2}{2}x\right),\quad x>0,\end{aligned}$$

interpreted by (4.18) as the limit when $\delta \downarrow 0$.

(ii) $\lambda = 0, \delta > 0, \gamma > 0$.

(iii) $\lambda < 0, \delta > 0, \gamma \geq 0$. In particular, $\mathrm{GIG}(\lambda,\delta,0) = \mathrm{I}\Gamma(|\lambda|,\delta^2/2)$, the inverse-gamma distribution with density function

$$p_{\mathrm{GIG}}(x;\lambda,\delta,0) = p_{\mathrm{I}\Gamma}(x;|\lambda|,\delta^2/2) = \frac{1}{\Gamma(|\lambda|)}\left(\frac{\delta^2}{2}\right)^{|\lambda|} x^{-|\lambda|-1}e^{-\frac{\delta^2}{2x}},\quad x>0, \tag{4.13}$$

interpreted by (4.17) and (4.18) as the limit when $\gamma \downarrow 0$.

A useful property of the class of GIG distributions is

$$X \sim \mathrm{GIG}(\lambda,\delta,\gamma) \Rightarrow X^{-1} \sim \mathrm{GIG}(-\lambda,\gamma,\delta).$$

In particular,

$$X \sim \Gamma(\lambda,c) \Rightarrow X^{-1} \sim \mathrm{I}\Gamma(\lambda,c)$$

for $\lambda, c > 0$. There is a scaling invariance of the class of GIG distributions

$$X \sim \mathrm{GIG}(\lambda,\delta,\gamma) \Rightarrow cX \sim \mathrm{GIG}(\lambda,\sqrt{c}\delta,\gamma/\sqrt{c})$$

for $c > 0$. More information on the analytical properties of the GIG distribution can be found in Masuda (2002).

4.10.2 Generalized Inverse Gaussian Process and Generalized Hyperbolic Process

It is known that GIG distributions are infinitely divisible (Barndorff-Nielsen and Halgreen 1977). Therefore, there is a Lévy process S for which $S_1 \sim \mathrm{GIG}(\lambda,\delta,\gamma)$. We call such a Lévy process a **generalized inverse Gaussian process** $\mathrm{GIGP}(\lambda,\delta,\gamma)$. From $\operatorname{supp} \mathscr{L}\{S_1\} \subset \mathbb{R}_+$, it is easy to show any increments of S is distributed on $\mathbb{R}_+$. Therefore, S is an increasing Lévy process, i.e. subordinator.

Suppose that $\alpha > 0$, $\beta \in \mathbb{R}$ with $\alpha > |\beta|$, $\delta > 0$ and $\mu \in \mathbb{R}$. Let $S = (S_t)_{t\in\mathbb{R}_+}$ be a $\mathrm{GIGP}(\lambda,\delta,\gamma)$ for $\gamma = \sqrt{\alpha^2-\beta^2}$. For a one-dimensional Wiener process $W = (W_t)_{t\in\mathbb{R}_+}$ independent of S, let

$$X_t = \mu t + \beta S_t + W_{S_t}.$$

By definition, $X = (X_t)_{t\in\mathbb{R}_+}$ is a Lévy process and we see[9]

$$\begin{aligned}\varphi_{X_1}(u) &= \varphi_{\mathrm{GH}}(u;\lambda,\alpha,\beta,\delta,\mu)\\ &= e^{i\mu u}\frac{\left(\alpha^2-\beta^2\right)^{\lambda/2}}{\left(\alpha^2-(\beta+iu)^2\right)^{\lambda/2}}\frac{K_\lambda\left(\delta\sqrt{\alpha^2-(\beta+iu)^2}\right)}{K_\lambda\left(\delta\sqrt{\alpha^2-\beta^2}\right)}\end{aligned} \tag{4.14}$$

for $u \in \mathbb{R}$. The Lévy process X is called a **generalized hyperbolic process** or **generalized hyperbolic Lévy motion** and denoted by $\mathrm{GHP}(\lambda,\alpha,\beta,\delta,\mu)$. For this process, the reader can also refer to Eberlein and Keller (1995), Eberlein (2001) and Eberlein and von Hammerstein (2004).

4.10.3 GH Distributions

For a $\mathrm{GHP}(\lambda,\alpha,\beta,\delta,\mu)$ X, the distribution $\mathscr{L}\{X_1\}$ is called a **generalized hyperbolic distribution** $\mathrm{GH}(\lambda,\alpha,\beta,\delta,\mu)$ (Barndorff-Nielsen and Halgreen 1977). Due to (4.11) and (4.8), the density function of $\mathrm{GH}(\lambda,\alpha,\beta,\delta,\mu)$ is expressed as

$$\begin{aligned}p_{\mathrm{GH}}(x;\lambda,\alpha,\beta,\delta,\mu) &= a(\lambda,\alpha,\beta,\delta)\left(\delta^2+(x-\mu)^2\right)^{(\lambda-\frac{1}{2})/2}\\ &\quad\times K_{\lambda-\frac{1}{2}}\left(\alpha\sqrt{\delta^2+(x-\mu)^2}\right)\exp\left(\beta(x-\mu)\right)\end{aligned}$$

with parameters $\lambda \in \mathbb{R}$, $\alpha > 0$, β satisfying $0 \le |\beta| < \alpha$, $\delta > 0$ and $\mu \in \mathbb{R}$. The coefficient $a(\lambda,\alpha,\beta,\delta)$ is given by

$$a(\lambda,\alpha,\beta,\delta) = \frac{(\alpha^2-\beta^2)^{\lambda/2}}{\sqrt{2\pi}\alpha^{\lambda-\frac{1}{2}}\delta^\lambda K_\lambda(\delta\sqrt{\alpha^2-\beta^2})}.$$

The moment generating function of $\mathrm{GH}(\lambda,\alpha,\beta,\delta,\mu)$ is

$$\begin{aligned}\mathscr{M}_{\mathrm{GH}}(t;\lambda,\alpha,\beta,\delta,\mu) &= e^{\mu t}\frac{a(\lambda,\alpha,\beta,\delta,\mu)}{a(\lambda,\alpha,\beta+t,\delta,\mu)}\\ &= e^{\mu t}\frac{\left(\alpha^2-\beta^2\right)^{\lambda/2}}{\left(\alpha^2-(\beta+t)^2\right)^{\lambda/2}}\frac{K_\lambda\left(\delta\sqrt{\alpha^2-(\beta+t)^2}\right)}{K_\lambda\left(\delta\sqrt{\alpha^2-\beta^2}\right)}\end{aligned}$$

for $t \in \mathbb{R}$ such that $|\beta+t| < \alpha$, and the characteristic function is $\varphi_{\mathrm{GH}}(u;\lambda,\alpha,\beta,\delta,\mu)$ of (4.14).

[9]Replace u by $u(\beta + 2^{-1}iu)$ in (4.12) and shift by μ.

4.10.4 Subclasses of the GH Distributions

(1) Normal inverse Gaussian distribution NIG$(\alpha, \beta, \delta, \mu)$. $\lambda = -1/2$: the characteristic function is

$$\varphi_{\mathrm{NIG}}(u; \alpha, \beta, \delta, \mu) = \exp\left[iu\mu + \delta\left(\sqrt{\alpha^2 - \beta^2} - \sqrt{\alpha^2 - (\beta + iu)^2}\right)\right].$$

The density function is

$$\begin{aligned} p_{\mathrm{NIG}}(x; \alpha, \beta, \delta, \mu) &= p_{\mathrm{GH}}(x; -1/2, \alpha, \beta, \delta, \mu) \\ &= a(-1/2, \alpha, \beta, \delta)\left(\delta^2 + (x-\mu)^2\right)^{-1/2} K_1\left(\alpha\sqrt{\delta^2 + (x-\mu)^2}\right) \exp\left(\beta(x-\mu)\right) \end{aligned}$$

where

$$a(-1/2, \alpha, \beta, \delta) = \frac{\alpha\delta}{\pi} \exp\left(\delta\sqrt{\alpha^2 - \beta^2}\right).$$

A special case is the Cauchy distribution C(μ, δ) appears when $\alpha = \beta = 0$ with

$$\varphi_{\mathrm{NIG}}(u; 0, 0, \delta, \mu) = \exp\left(i\mu u - \delta|u|\right).$$

(2) Hyperbolic distribution H$(\alpha, \beta, \delta, \mu)$. $\lambda = 1$:

$$\begin{aligned} p_{\mathrm{H}}(x; \alpha, \beta, \delta, \mu) &= p_{\mathrm{GH}}(x; 1, \alpha, \beta, \delta, \mu) \\ &= \frac{(\alpha^2 - \beta^2)^{1/2}}{2\alpha\delta K_1(\delta\sqrt{\alpha^2 - \beta^2})} \exp\left[-\alpha\sqrt{\delta^2 + (x-\mu)^2} + \beta(x-\mu)\right]. \end{aligned}$$

The characteristic function of H$(\alpha, \beta, \delta, \mu)$ is

$$\begin{aligned} \varphi_{\mathrm{H}}(u; \alpha, \beta, \delta, \mu) &= \varphi_{\mathrm{GH}}(u; 1, \alpha, \beta, \delta, \mu) \\ &= e^{i\mu u} \frac{\left(\alpha^2 - \beta^2\right)^{1/2}}{\left(\alpha^2 - (\beta + iu)^2\right)^{1/2}} \frac{K_1\left(\delta\sqrt{\alpha^2 - (\beta + iu)^2}\right)}{K_1\left(\delta\sqrt{\alpha^2 - \beta^2}\right)}. \end{aligned}$$

(3) Variance gamma distribution VG$(\lambda, \alpha, \beta, \mu)$. As $\delta \downarrow 0$, GH$(\lambda, \alpha, \beta, \delta, \mu)$ converges to VG$(\lambda, \alpha, \beta, \mu)$. Indeed, we see

$$\lim_{\delta \downarrow 0} \varphi_{\mathrm{GH}(\lambda,\alpha,\beta,\delta,\mu)}(u) = \varphi_{\mathrm{VG}(\lambda,\alpha,\beta,\mu)}(u) \quad (u \in \mathbb{R})$$

from (4.14), (4.18) and (4.9) for $t = 1$. In this sense, VG$(\lambda, \alpha, \beta, \mu) =$ GH$(\lambda, \alpha, \beta, 0, \mu)$.

(4) Skew Student's t distribution SkewT$(\nu, \delta, \beta, \mu)$. When $\lambda = -\nu/2$ and $\alpha \to |\beta| \neq 0$, GH$(\lambda, \alpha, \beta, \delta, \mu)$ converges to a distribution with density function

$$p_{\mathrm{SkewT}}(x; \nu, \delta, \beta, \mu) = \frac{\delta^\nu |\beta|^{(\nu+1)/2} K_{(\nu+1)/2}\big(|\beta|\sqrt{\delta^2 + (x-\mu)^2}\big) e^{\beta(x-\mu)}}{2^{(\nu-1)/2}\sqrt{\pi}\Gamma(\nu/2)\sqrt{\delta^2 + (x-\mu)^2}^{(\nu+1)/2}}, \quad x \in \mathbb{R}.$$

Moreover, as $\beta \to 0$, $p_{SkewT}(x; \nu, \delta, \beta, \mu)$ converges to

$$p_{\mathrm{SkewT}}(x; \nu, \delta, 0, \mu) = \frac{\Gamma((\nu+1)/2)}{\sqrt{\pi}\delta\Gamma(\nu/2)}\left[1 + \left(\frac{x-\mu}{\delta}\right)^2\right]^{-(\nu+1)/2}, \quad x \in \mathbb{R}$$

In particular, when $\delta = \nu^{1/2}$ and $\mu = 0$, we have SkewT$(x; \nu, \nu^{1/2}, 0, 0) = t(\nu)$, the t distribution with ν degree of freedom. Instead, letting $\delta = \nu^{1/2}\sigma \to \infty$ for $\sigma > 0$ for $p_{\mathrm{SkewT}}(x; \nu, \delta, 0, \mu)$, we have the normal distribution N(μ, σ^2).

4.11 Stochastic Differential Equation Driven by Lévy Processes and Their Simulation

4.11.1 Semimartingale

Given a measurable space $(\Omega, \mathscr{F})$, an increasing family $\mathbb{F} = (\mathscr{F}_t)_{t\in\mathbb{R}_+}$ of sub σ-fields $\mathscr{F}_t$ of $\mathscr{F}$ is called a **filtration**. Usually, the right continuity $\mathscr{F}_t = \cap_{u>t}\mathscr{F}_u$ is assumed. Then, $\mathscr{B} = (\Omega, \mathscr{F}, \mathbb{F}, P)$ is a stochastic basis. We will fix a stochastic basis $\mathscr{B}$. A random time $T : \Omega \to [0, \infty]$ is called a **stopping time** if $\{T \leq t\} \in \mathscr{F}_t$ for all $t \in \mathbb{R}_+$. For a stopping time T, the σ-field $\mathscr{F}_T$ is, intuitively, the whole information up to time T. More rigorously, $\mathscr{F}_T$ is defined by $\mathscr{F}_T = \{A \in \mathscr{F};\ A \cap \{T \leq t\} \in \mathscr{F}_t$ for all $t \in \mathbb{R}_+\}$.

We say a stochastic process $X = (X_t)_{t\in\mathbb{R}_+}$ is **adapted** if X_t is $\mathscr{F}_t$-measurable for all $t \in \mathbb{R}_+$. An adapted process $X = (X_t)_{t\in\mathbb{R}_+}$ is called a **martingale** if each X_t is integrable and $E[X_t|\mathscr{F}_s] = X_s$ a.s. for every $s < t$. We call X a uniformly integrable martingale if the family $\{X_t\}_{t\in\mathbb{R}_+}$ of random variables is uniformly integrable. For a uniformly integrable martingale X, the limit $X_\infty = \lim_{t\to\infty} X_t$ (a.s. and in L^1) is well defined and the optional sampling theorems holds:

$$E[X_T|\mathscr{F}_S] = X_{T\wedge S} \quad a.s.$$

for any stopping times S and T.

A process $X = (X_t)_{t\in\mathbb{R}_+}$ is called a **local martingale** if there exists an increasing sequence of stopping times $T_n \to \infty$ a.s. such that each stopped process X^{T_n} is a uniformly integrable martingale, where X^{T_n} is defined by $X_t^{T_n} = X_{t\wedge T_n}$. For local martingales, we assume that their paths are almost surely càdlàg , i.e., right continuous and admitting left-hand limits. This is because one can always obtain such a modification by completion of the probability space and augmentation of the filtration.

The localization of stochastic processes is universally applied to various classes of stochastic processes. For example, $X = (X_t)_{t\in\mathbb{R}+}$ is a locally **square-integrable** martingale if each X^{T_n} is a square-integrable martingale for some localizing sequence (T_n) as above. Here a martingale $M = (M_t)_{t\in R_+}$ is called a square-integrable martingale if $\sup_{t\in R+} \mathbb{E}[M_t^2] < \infty$.

We say a process $X = (X_t)_{t\in\mathbb{R}_+}$ has **bounded variation** if the variation of the function $[0, N] \ni t \to X_t$ is a.s. finite for all $N > 0$. A process X is called a **semimartingale** if X has a decomposition $X = X_0 + M + A$ by M and A as follows. M is a local martingale with $M_0 = 0$. The process A is an adapted process with a.s. càdlàg paths of finite variation. This decomposition of X is not unique. It is know that M can be a locally square-integrable martingale with $M_0 = 0$ in the decomposition $X = X_0 + M + A$ for any semimartingale X.

Let us consider a so-called **bounded simple predictable process** $H = (H_t)_{t\in\mathbb{H}}$, that has a expression

$$H_t = H_0 + \sum_{j=0}^{J-1} H_{(j)} 1_{(T_j, T_{j+1}]},$$

where H_0 is a bounded $\mathscr{F}_0$-measurable random variable, $H_{(j)}$ is a bounded $\mathscr{F}_{T_j}$-measurable random variable, $0 = T_0 \le T_1 \le \cdots T_J$ are $\mathbb{R}_+$-valued stopping times. For a square-integrable martingale $X = (X_t)_{t\in\mathbb{R}_+}$, a **stochastic integral** of H with respect to X is naturally defined by

$$J_X(H)_t = \sum_{j=0}^{J-1} H_{(j)}(X_t^{T_{j+1}} - X_t^{T_j}) \quad (t \in \mathbb{R}_+)$$

Then, it turns out that the process $J_X(H)$ is a square-integrable martingale. For simplicity, let us assume that T_j are deterministic. It is not difficult to show $J_X(H)_t$ is $\mathscr{F}_t$-measurable, and $J_X(H)$ is a martingale, i.e. $E[J_X(H)_t|\mathscr{F}_s] = J_X(H)_s$ a.s. for $s < t$; for that, we may assume $s, t \in \{T_j\}$, if necessary. Moreover,

$$\begin{aligned} E[J_X(H)_t^2] &= \sum_{j=0}^{J-1} E\big[H_{(j)}^2(X_t^{T_{j+1}} - X_t^{T_j})^2\big] \\ &\le \sup_t |H_t|^2 \sum_{j=0}^{J-1} E\big[(X_t^{T_{j+1}} - X_t^{T_j})^2\big] \\ &\le \sup_{s\le t} |H_s|^2 E\big[X_t^2\big]. \end{aligned}$$

Here, the conditional expectation $E[\cdot|\mathscr{F}_u]$ and the martingale property are repeatedly used. The last inequality implies a kind of continuity of the mapping J_X, and really we can extend J_X to left continuous adapted processes H with right-hand limits by taking advantage of this continuity. Consequently, if remembering the decomposition of a semimartingale X, we can define a stochastic integral J_X for such processes H. So defined $J_X(H)$ becomes a semimartingale. The stochastic integral $J_X(H)$ is denoted by $\int_0^\cdot H_t dX_t$.

4.11.2 Stochastic Differential Equations

Suppose that we have semimartingales Z^α, $\alpha = 1, ..., \mathbf{r}$. Then, it is possible to produce various functionals of $Z = (Z^\alpha)_{\alpha=1,...,\mathsf{r}}$ by the stochastic equations based on Z. Consider a stochastic integral equation

$$X_t = \eta + \sum_\alpha \int_0^t c_\alpha(s, X_{s-}) dZ_s^\alpha \tag{4.15}$$

where $c_\alpha(t, x)$ are (possibly vector valued) functions of (t, x) and η is $\mathscr{F}_0$-measurable. The integrals on the right-hand side of (4.15) are stochastic integrals. Equivalently to the integral form (4.15) of the equation, we also use the stochastic differential equation

$$dX_t = \sum_\alpha c_\alpha(t, X_{t-}) dZ_t^\alpha, \quad X_0 = \eta.$$

If there is a constant K such that

$$|c_\alpha(t, x_2) - c_\alpha(t, x_1)| \le K|x_2 - x_1|$$

for each $t \in \mathbb{R}_+$ and if the functions $t \mapsto c_\alpha(t, x)$ are càglàd (left continuous with right-hand limits) for each x, then a unique solution to (4.15) extists; see Protter (1990).

The **yuima** package specifies the driving process Z differently according to the Lévy measure type as described in what follows.

4.11.3 Compound Poisson Driving Processes

Lévy models can be specified in two different ways in **yuima**. The first one is in terms of the compound Poisson structure, and the second one presented in the next section allows for direct specification of the density of the Lévy process. The compound Poisson specification is very close to what we have seen for pure, compound Poisson process in Chap. 3 but in this case, the Poisson structure represents only the jump part of the yuima model. Consider the following stochastic differential equation with jumps

$$dX_t = -\theta X_t dt + \sigma dW_t + \left(\gamma + X_{t-}/\sqrt{1 + X_{t-}^2}\right) dJ_t, \quad X_0 = 0$$

on $[0, T]$, where J_t is a compound Poisson process with spot intensity λ and jump sizes distributed as the normal distribution $N(2, 0.1)$. We can specify this model in **yuima** using the arguments `measure` and `measure.type` set to `CP`. The next code illustrates briefly how to proceed

```
modJump <- setModel(drift = c("-theta*x"), diffusion = "sigma",
 jump.coeff=c("gamma+x/sqrt(1+x^2)"),
 measure = list(intensity="lambda",df=list("dnorm(z, -3, 1)")),
 measure.type="CP", solve.variable="x")
modJump

##
## Diffusion process with Levy jumps
## Number of equations: 1
## Number of Wiener noises: 1
## Number of Levy noises: 1
## Parametric model with 4 parameters

samp <- setSampling(n=10000,Terminal=10)
set.seed(125)
X <- simulate(modJump, xinit=2, sampling=samp,
 true.par= list(theta=2, sigma=0.5,gamma=0.3,lambda=0.5))
plot(X)
```

Figure 4.16 shows a simulated path of the above jump-diffusion process with compound Poisson Gaussian jumps.

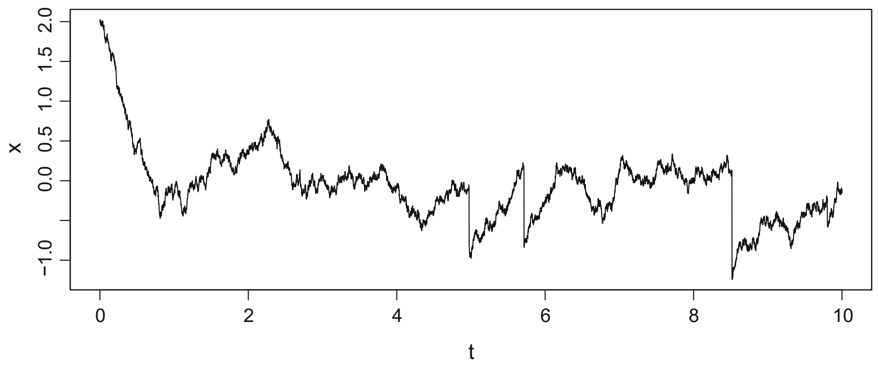


Fig. 4.16 A simulated path of a diffusion process with compound Poisson jumps

4.11.4 Driving Processes of `code` Type

Suppose we want to generate a sample path of the stochastic differential equation

$$\begin{aligned} dX_t &= a\, X_t\, dt + c\, dZ_t, \quad t \in [0, T], \\ X_0 &= x_0, \end{aligned}$$

where $Z = (Z_t)_{t\in[0,T]}$ is an inverse Gaussian Lévy motion with $Z_t \sim \mathrm{IG}(\delta t, \gamma) = \mathrm{GIG}(-1/2, \delta t, \gamma)$. For example, let us take $x_0 = 2$, $a = 0.05$, $c = -1$, $T = 10$, $\delta = 0.55$, $\gamma = 2$. The next code shows the usage of the argument `mesure.type` which is set to `"code"`, meaning that a random number generator will be specified in the `measure` argument. The random number generator is `rIG`. The simulated path of this process is given in Fig. 4.17.

```
x0 <- 2
a <- 0.1
c <- -1
model.ig <- setModel(drift="a*x", xinit=x0, jump.coeff=c,
 measure.type="code", measure=list(df="rIG(z, delta0, gamma)"))
model.ig

## Levy process
## Number of equations: 1
## Number of Levy noises: 1
## Parametric model with 3 parameters

sampling.ig <- setSampling(Terminal=10, n=10000)
yuima.ig <- setYuima(model=model.ig, sampling=sampling.ig)
set.seed(128)
result.ig <- simulate(yuima.ig,true.par=list(delta0=0.55,gamma=2))
plot(result.ig)
```

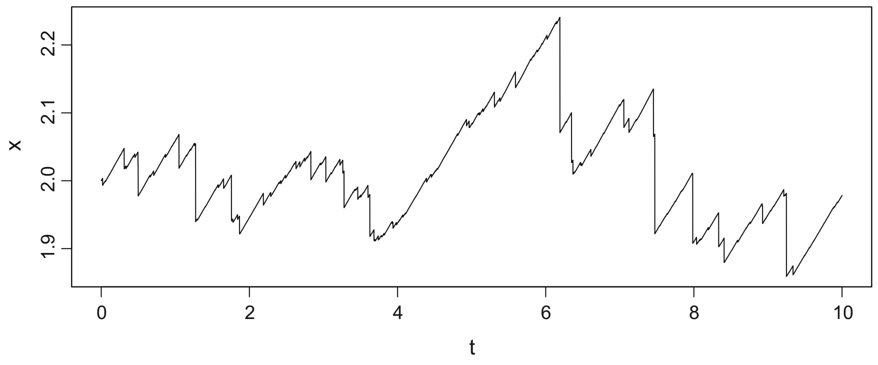


Fig. 4.17 A simulated path of an IG($\delta = 0.55, \gamma = 2$) process

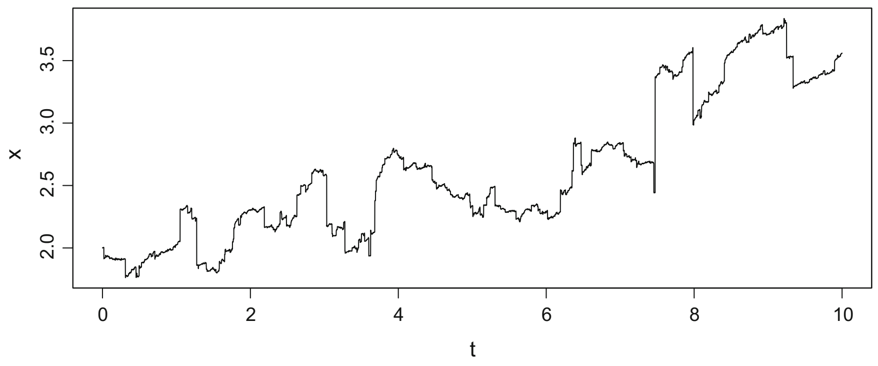


Fig. 4.18 A simulated path of an NIG($\alpha = 2, \beta = 0, \delta = 0.55, \mu = 0$) process

The Lévy measure is specified by the parameters corresponding to the distribution of Z_1. Here we used `delta0` for the parameter δ, otherwise `delta` conflicts with the internal variable `delta` describing the mesh size of the sampling.

Next is a stochastic differential equation where the driving process is replaced by the normal inverse Gaussian Lévy motion Z_t with $Z_1 \sim \text{NIG}(\alpha, \beta, \delta, \mu)$. We apply the normal inverse Gaussian random number generator `rNIG`. The simulated path is shown in Fig. 4.18.

```
x0 <- 2
a <- 0.1
c <- -1
model.nig <- setModel(drift="a*x", xinit=x0, jump.coeff=c,
 measure.type="code",measure=list(df="rNIG(z, alpha,
 beta, delta0, mu)"))
sampling.nig <- setSampling(Terminal=10, n=10000)
yuima.nig <- setYuima(model=model.nig, sampling=sampling.ig)
set.seed(128)
result.nig <- simulate(yuima.nig,true.par=list(alpha=2, beta=0,
 delta0=0.55, mu=0))
plot(result.nig)
```

We should note the relation $\alpha = \sqrt{\gamma^2 + \beta^2}$ between parameters (α, β) of NIG and γ of IG.

The multivariate normal inverse Gaussian random number generator is also available in the present one-dimensional case to obtain the same result as above (see Fig. 4.19).

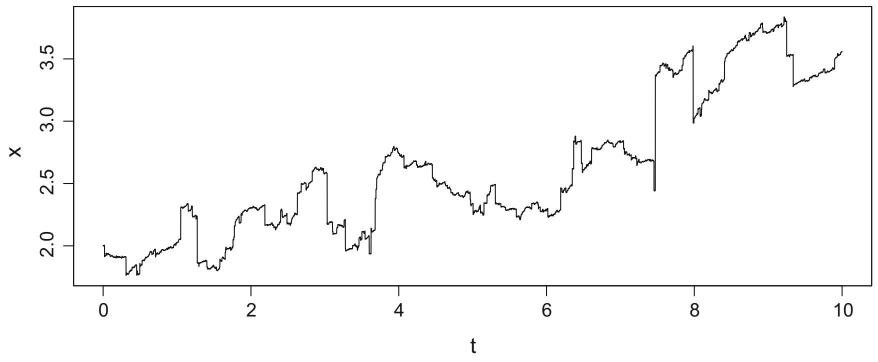


Fig. 4.19 A simulated path of a multidimensional NIG($\alpha = 2, \beta = 0, \delta = 0.55, \mu = 0, \Lambda = [1]$) process with dimension 1

```
x0 <- 2
a <- 0.1
c <- -1
Lambda <- matrix(1,1,1)
model.nig <- setModel(drift="a*x", xinit=x0, jump.coeff=c,
 measure.type="code",measure=list(df="rNIG(z, alpha,
 beta, delta0, mu, Lambda)"))
sampling.nig <- setSampling(Terminal=10, n=10000)
yuima.nig <- setYuima(model=model.nig, sampling=sampling.ig)
set.seed(128)
result.nig <- simulate(yuima.nig,true.par=list(alpha=2,
 beta=0, delta0=0.55, mu=0, Lambda=Lambda))
plot(result.nig)
```

Next, let us consider the following two-dimensional stochastic differential equation for $X_t = (X_{1,t}, X_{2,t})$ driven by a multivariate Lévy process

$$\begin{aligned} dX_t &= a(t, X_t)dt + b(t, X_t)dW_t + c(t, X_{t-})dZ_t, \quad t \in [0, T] \\ X_0 &= x_0, \end{aligned}$$

where a and b are mappings from $[0, T] \times \mathbb{R}^2$ to $\mathbb{R}^2$, and the driving noises are a two-dimensional Wiener process $W_t = (W_{1,t}, W_{2,t})$ and a two-dimensional Lévy process $Z_t = (Z_{1,t}, Z_{2,t})$.

For illustration, we set $x_0 = (2, 3)$, $T = 1$,

$$a(t, x_1, x_2) = \begin{pmatrix} x_1 \cos(2\pi t) - x_2 \sin(2\pi t) \\ x_1 \sin(2\pi t) + x_2 \cos(2\pi t) \end{pmatrix}$$

$$b(t, x_1, x_2) = \begin{pmatrix} t\, x_2 & 0 \\ 1 & x_1 \end{pmatrix}$$

and

$$c(t, x_1, x_2) = \begin{pmatrix} \cos(2\pi t) & \sin(2\pi t) \\ (5-t)x_1 & 1 \end{pmatrix}$$

for $t \in [0, T]$ and $x_1, x_2 \in \mathbb{R}$. For Z_t, we choose a two-dimensional NIG distribution with parameters

$$\alpha = 2, \quad \beta = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \delta = 0.55, \quad \mu = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \Lambda = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

we set up the model as follows

```
x0 <- c(2,3)
a1 <- function(t,x1,x2){ x1*cos(2*pi*t)-x2*sin(2*pi*t) }
a2 <- function(t,x1,x2){ x1*sin(2*pi*t)+x2*cos(2*pi*t) }
a <- c("a1(t,x1,x2)","a2(t,x1,x2)")
b <- matrix(c("t*x2","1","0","x1"),2,2)
c <- matrix(c("cos(2*pi*t)", "(5-t)*x1","sin(2*pi*t)",1),2,2)
alpha <- 2
beta <- c(0,0)
delta0 <- 0.55
mu <- c(0,0)
Lambda <- matrix(c(1,0,0,1),2,2)
model.mnig <- setModel(drift=a, xinit=x0, diffusion=b,
  jump.coeff=c, measure.type="code",
  measure=list(df="rNIG(z, alpha, beta, delta0, mu, Lambda)"),
  state.variable=c("x1","x2"),solve.variable=c("x1","x2") )
model.mnig

##
## Diffusion process with Levy jumps
## Number of equations: 2
## Number of Wiener noises: 2
## Number of Levy noises: 1
## Parametric model with 7 parameters

sampling.mnig <- setSampling(Terminal=1, n=10000)
yuima.mnig <- setYuima(model=model.mnig, sampling=sampling.mnig)
set.seed(128)
result.mnig <- simulate(yuima.mnig,true.par=list(alpha=alpha,
 beta=beta, delta0=delta0, mu=mu, Lambda=Lambda))
plot(result.mnig)
```

and the simulated two-dimensional path is given in Fig. 4.20.

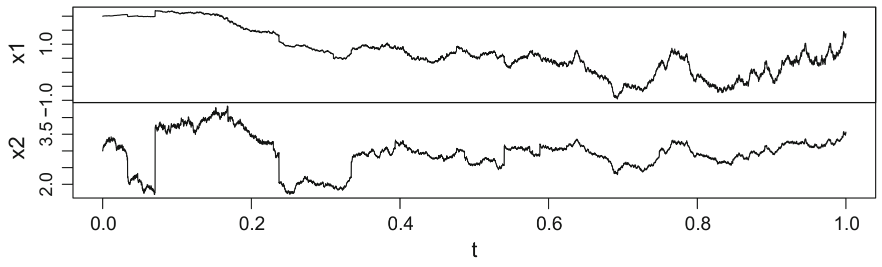


Fig. 4.20 A simulated path of a multidimensional NIG process with dimension 2

4.12 Estimation

Estimation for general Lévy process is in continuous development in **yuima** package; here we present two options available. The the case of diffusion process with compound Poisson jumps and the simple case of the estimation of exponential Lévy processes.

4.12.1 Estimation of Jump-Diffusion Processes

The **yuima** package is providing the function `qmle` for the quasi-maximum likelihood estimation of jump-diffusion processes with randomly amplified compound Poisson jumps. For an illustrative example, we will consider a jump-diffusion process satisfying the stochastic differential equation with jumps

$$dX_t = -\theta X_t dt + \sigma dW_t + \left(\gamma + X_{t-}/\sqrt{1 + X_{t-}^2}\right) dJ_t, \quad X_0 = 0$$

on $[0, T]$, where J_t is a compound Poisson process with spot intensity λ and jump sizes distributed as the normal distribution $N(2, 0.1^2)$. We will estimate $(\sigma, \theta, \lambda, \gamma)$ from the simulated data. The quasi-likelihood inference is based on thresholding the increments of the observed path to separate the continuous part of the increments and the Poissonian jumps. Ergodicity and high-frequency observations are required for consistency and asymptotic normality of the estimators. We refer the reader to Shimizu and Yoshida (2006) and Ogihara and Yoshida (2011) for parametric estimation of jump-diffusion processes. The next code shows the practical implementation using a threshold h_n^p with $p < 1/2$ for the mesh size h_n between two consecutive observation times. In our example, we choose $p = 0.4$; therefore, the threshold is set to $(T/N)^{0.4}$. The argument which specifies the thresholding in `qmle` is `threshold`. One condition on the jumps is, of course, that there should be few jumps around zero to avoid loss of information in the estimation of the Poisson intensity. The simulated path is shown in Fig. 4.21.

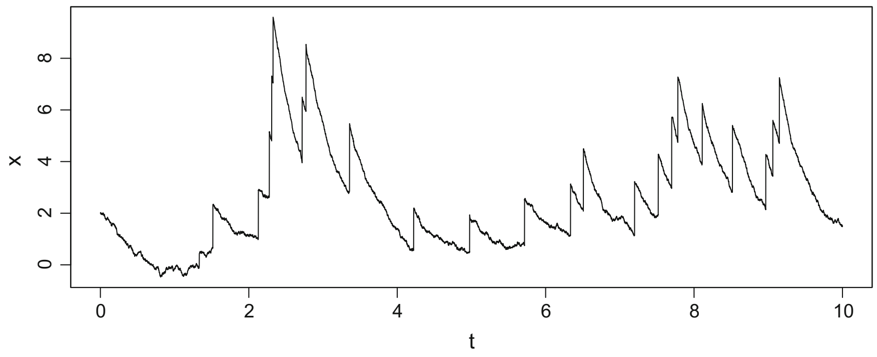


Fig. 4.21 A simulated path of a diffusion process with compound Poisson jumps

```
mod5 <- setModel(drift = c("-theta*x"), diffusion = "sigma",
jump.coeff=c("gamma+x/sqrt(1+x^2)"),
measure = list(intensity="lambda",df=list("dnorm(z, 2, 0.1)")),
measure.type="CP", solve.variable="x")
theta <- 2
sigma <- 0.5
gamma <- 0.3
lambda <- 2.5
T <- 10
N <- 10000
delta <- T/N
h <- T/N
true <- list(theta=theta, sigma=sigma,gamma=gamma,lambda=lambda)
set.seed(125)
X <- simulate(mod5, true.p=true,xinit=2,
sampling=setSampling(n=N,Terminal=T))
plot(X)
r <- h^0.4
est.qmle <- qmle(yuima=X, start=true,
 lower=list(theta=1,sigma=0,gamma=0.1,lambda=0.1),
 upper=list(theta=3,sigma=2,gamma=0.8,lambda=20), method="L-BFGS-B",
 threshold=r)
unlist(true)
summary(est.qmle)
```

```
##   theta  sigma  gamma lambda
##     2.0    0.5    0.3    2.5
## Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = X, start = true, method = "L-BFGS-B", lower =
## list(theta = 1,
## sigma = 0, gamma = 0.1, lambda = 0.1), upper = list(theta =
## 3,
## sigma = 2, gamma = 0.8, lambda = 20), threshold = r)
```

```
##
## Coefficients:
## Estimate Std. Error
## sigma 0.5012518 0.003548335
## theta 2.0530481 0.049587527
## lambda 2.5000000 0.499999919
## gamma 0.3094323 0.008028919
##
## -2 log L: -53574.4
##
##
## Number of estimated jumps: 25
##
## Average inter-arrival times: 0.325708
##
## Average jump size: 2.002308
##
## Standard Dev. of jump size: 0.796882
##
## Jump Threshold: 0.063096
##
## Summary statistics for jump times:
## Min. 1st Qu.  Median Mean 3rd Qu.  Max.
## 1.332 2.718 6.335 5.612 8.110 9.149
##
## Summary statistics for jump size:
## Min. 1st Qu.  Median Mean 3rd Qu.  Max.
## -0.072 1.795 2.335 2.002 2.544 2.742
```

Next code shows the effect of setting the wrong threshold. If it is too large, then the number of Poissonian events will be underestimate and vice versa. For example, looking at the correct threshold, the above estimation result show an average jump size of about 2. So we could have mistakenly chosen the threshold in this way. On the other side, we can take a threshold lower than $h^{0.4} = 0.063$.

```
est.qmle1 <- qmle(yuima=X, start=true,
 lower=list(theta=1,sigma=0,gamma=0.1,lambda=0.1),
 upper=list(theta=3,sigma=2,gamma=0.8,lambda=20), method="L-BFGS-B",
 threshold=2) # too large
coef(est.qmle1)

##     sigma     theta    lambda     gamma
## 1.4538082 1.9782071 1.6000002 0.2954198

est.qmle2 <- qmle(yuima=X, start=true,
 lower=list(theta=1,sigma=0,gamma=0.1,lambda=0.1),
 upper=list(theta=3,sigma=2,gamma=10,lambda=1000), method="L-BFGS-B",
 threshold=0.03) ## too low
coef(est.qmle2)

##      sigma      theta     lambda      gamma
##  0.4274578  1.4596203 77.8515913  0.1114447
```

4.12.2 Estimation of Exponential Lévy Processes

Let Z_t be a Lévy process, then the process S_t defined as

$$S_t = S_0 e^{Z_t}, \quad t > 0,$$

is called an **exponential Lévy** process. This process is often used in finance to model asset prices under the assumption of independent log-returns. Indeed, if we take the log-returns of the process S_t

$$\log\left(\frac{S_{t+\Delta t}}{S_t}\right) = Z_{t+\Delta t} - Z_t = \Delta Z_t$$

these are distributed as the increments of the Lévy process Z_t. Being Z_t a process with independent increments, true likelihood estimation can be applied for this i.i.d. sequence of random variables. Notice that geometric Brownian motion

$$dS_t = \mu S_t dt + \sigma S_t dB_t$$

is a special case of this model as its solutions is given in the form

$$S_t = S_0 e^{\left(\mu - \frac{1}{2}\sigma^2\right)t + \sigma B_t}.$$

The **yuima** package cannot fit yet directly a pure jump Lévy model, but this temporary limitation of **yuima** can be turned around by using a compound Poisson structure with known constant intensity and i.i.d. jumps based on the cumulative sum of the Lévy increments. Indeed, let $Y_{t_j} = \Delta Z_{t_j}$, where $t_j = j \cdot T/N$, with $Y_0 = Z_0$. Assume that the hypothetical arrival times of the Poisson process coincide exactly with the instants t_i where data have been collected and define the process

$$\begin{aligned} X_{t_i} &= \sum_{j=0}^{N_{t_i}} Y_{t_j} = \sum_{j=0}^{i} \Delta Z_{t_j} \\ &= (Z_{t_i} - Z_{t_{i-1}}) + \cdots + (Z_{t_1} - Z_0) = Z_{t_i} \end{aligned}$$

so that X_{t_i} is distributed as Z_{t_i} for each time t_i, as we assume further X_t being a piecewise constant process. This can be seen as a degenerate compound Poisson process where N_{t_i} coincides with the number of observations at time i, with jumps deterministically observed at time t_i. Then, we can use the `setPoisson` function to specify this model in **yuima** and estimate it via exact `qmle` as we explain now. In the following code, we first try to fit on real data with geometric Brownian motion, then a compound Poisson model with Gaussian jumps and, finally, a compound Poisson model with NIG jumps, i.e. an NIG exponential Lévy model in this setup. We collected the data using `getSymbols` from the **quantmod** package.

```
require(quantmod)
getSymbols("ENI.MI",to="2016-12-31")

## [1] "ENI.MI"

S <- ENI.MI$ENI.MI.Adjusted
Z <- na.omit(diff(log(S)))
Dt <- 1/252
# geometric Brownian motion estimation
model1 <- setModel(drift="mu*x", diff="sigma*x")
gBm <- setYuima(model=model1, data=setData(S,delta=Dt))
gBm.fit <- qmle(gBm, start=list(mu=0,sigma=1),method="BFGS")
gBm.cf <- coef(gBm.fit)
zMin <- min(Z)
zMax <- max(Z)
# Gaussian-Levy estimation
model3 <- setPoisson( df="dnorm(z,mu,sigma)")
Norm <- setYuima(model=model3, data=setData(cumsum(Z),delta=Dt))
Norm.fit <- qmle(Norm,start=list(mu=1, sigma=1),
 lower=list(mu=1e-7,sigma=0.01),method="L-BFGS-B")
Norm.cf <- coef(Norm.fit)
# NIG-Levy estimation
model2 <- setPoisson( df="dNIG(z,alpha,beta,delta1,mu)")
NIG <- setYuima(model=model2, data=setData(cumsum(Z),delta=Dt))
NIG.fit <- qmle(NIG,start=list(alpha=10, beta=1, delta1=1,mu=1),
 lower=list(alpha=1,beta=-2, delta1=0.001,mu=0.0001),
  method="L-BFGS-B")
NIG.cf <- coef(NIG.fit)
myfgBm <- function(u)
 dnorm(u, mean=gBm.cf["mu"], sd=gBm.cf["sigma"])
myfNorm <- function(u)
 dnorm(u, mean=Norm.cf["mu"],sd=Norm.cf["sigma"])
myfNIG <- function(u)
 dNIG(u, alpha=NIG.cf["alpha"],beta=NIG.cf["beta"],
 delta=NIG.cf["delta1"], mu=NIG.cf["mu"])
plot(density(Z,na.rm=TRUE),main="Gaussian versus NIG")
curve(myfgBm, zMin, zMax, add=TRUE, lty=2)
curve(myfNorm, zMin, zMax, col="red", add=TRUE, lty=4)
curve(myfNIG, zMin, zMax, col="blue", add=TRUE,lty=3)
```

Figure 4.22 compares the empirical density of the data Z_t with the densities of an estimated geometric Brownian motion (dashed line), a Gaussian distribution where the mean and the variance coincide with the sample means and variance of Z_t and the NIG density where the parameters have been estimated using the degenerate compound Poisson process X_t above. It can be clearly seen that the data are not Gaussian (and especially not coming from a geometric Brownian motion), but more likely to be of NIG type in this particular dataset. The Akaike information criterion evaluated with the `AIC` function also confirms this empirical evidence.

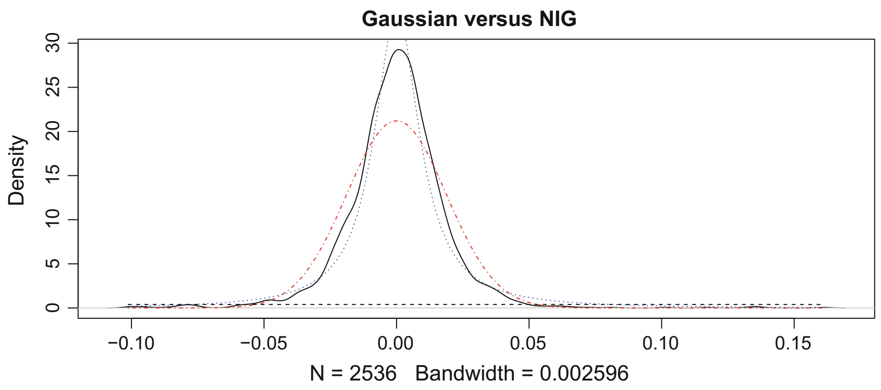


Fig. 4.22 Empirical density (continuous line) against a geometric Brownian motion fit (dashed line), a Gaussian fit (horizontal dashed and dotted line) and the estimated NIG Lévy model density (dotted line)

```
AIC(gBm.fit)

## [1] 2e+10

AIC(Norm.fit)

## [1] -12673.47

AIC(NIG.fit)

## [1] -12950.19
```

4.13 Bessel Function of the Third Kind

Denote by K_ν the modified Bessel function (Bessel function of the third kind) with index ν defined by the integral representation

$$K_\nu(x) = \frac{1}{2}\int_0^\infty y^{\nu-1} \exp\left[-\frac{1}{2}x\left(y+\frac{1}{y}\right)\right] dy. \tag{4.16}$$

Extending domain of ν to a complex region, one has another integral representation

$$K_\nu(z) = \frac{\sqrt{\pi}z^\nu}{2^\nu \Gamma\left(\nu+\frac{1}{2}\right)} \int_1^\infty e^{-zt}(t^2-1)^{\nu-\frac{1}{2}} dt$$

when $\mathrm{Re}(\nu) > -\frac{1}{2}$ and $|\arg z| < \pi/2$.

The function K_ν has the following properties.

$$K_{-\nu}(z) = K_\nu(z) \tag{4.17}$$

As $z \to 0$

$$K_\nu(z) \sim \frac{1}{2}\Gamma(\nu)(z/2)^{-\nu} \tag{4.18}$$

when $\mathrm{Re}(\nu) > 0$, and

$$K_0(z) \sim -\log z.$$

$$K_{\frac{1}{2}}(z) = \sqrt{\frac{\pi}{2z}}e^{-z} \tag{4.19}$$

With

$$a_k(\nu) = \frac{(4\nu^2-1)(4\nu^2-3^2)\cdots(4\nu^2-(2k-1))^2}{8^k k!},$$

$$K_\nu(z) \sim \sqrt{\frac{\pi}{2z}}e^{-z}\sum_{k=0}^{\infty}\frac{a_k(\nu)}{z^k}$$

as $z \to \infty$ under $|\arg z| \le 3\pi/2 - \varepsilon$.

We refer the reader to Abramowitz and Stegun (1964) for more details on Bessel functions.

Chapter 5
Stochastic Differential Equations Driven by the Fractional Brownian Motion

5.1 Model Specification

The `yuima` allows for the description of stochastic differential equations driven by fractional Brownian motion of the following type

$$\mathrm{d}X_t = a(X_t)\mathrm{d}t + b(X_t)\mathrm{d}W_t^H$$

where $W^H = \left(W_t^H, t \geq 0\right)$ is a normalized fractional Brownian motion (fBM), i.e., the zero-mean Gaussian processes with covariance function

$$\mathbb{E}(W_s^H W_t^H) = \frac{1}{2}\left(|s|^{2H} + |t|^{2H} - |t-s|^{2H}\right)$$

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 (Kolmogorov 1940; Mandelbrot and Ness 1968). In order to specify a stochastic differential e-quation 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

$$\mathrm{d}Y_t = 3Y_t\mathrm{d}t + \mathrm{d}W_t^H$$

we can proceed as follows

```
mod4A <- setModel(drift="3*y", diffusion=1, hurst=0.3, solve.var="y")
mod4A

##
## Diffusion process with Hurst index:0.30
## Number of equations: 1
## Number of Wiener noises: 1
```

```
mod4B <- setModel(drift="3*y", diffusion=1, hurst=0.7, solve.var="y")
mod4B

##
## Diffusion process with Hurst index:0.70
## Number of equations: 1
## Number of Wiener noises: 1

set.seed(123)
X1 <- simulate(mod4A,sampling=setSampling(n=1000))
X2 <- simulate(mod4B,sampling=setSampling(n=1000))
par(mfrow=c(2,1))
par(mar=c(2,3,1,1))
plot(X1,main="H=0.3")
plot(X2,main="H=0.7")
```

and the two trajectories can be seen in Fig. 5.1. In this case, the appropriate slot is now filled

```
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
## ..@ jump.coeff : list()
## ..@ measure : list()
## ..@ measure.type : chr(0)
## ..@ parameter :Formal class ‘model.parameter’ [package
## "yuima"] with 7 slots
## .. .. ..@ all : chr(0)
## .. .. ..@ common : chr(0)
## .. .. ..@ diffusion: chr(0)
## .. .. ..@ drift : chr(0)
## .. .. ..@ jump : chr(0)
## .. .. ..@ measure : chr(0)
## .. .. ..@ xinit : 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 "y"
## ..@ xinit : expression((0))
## ..@ J.flag : logi FALSE
```

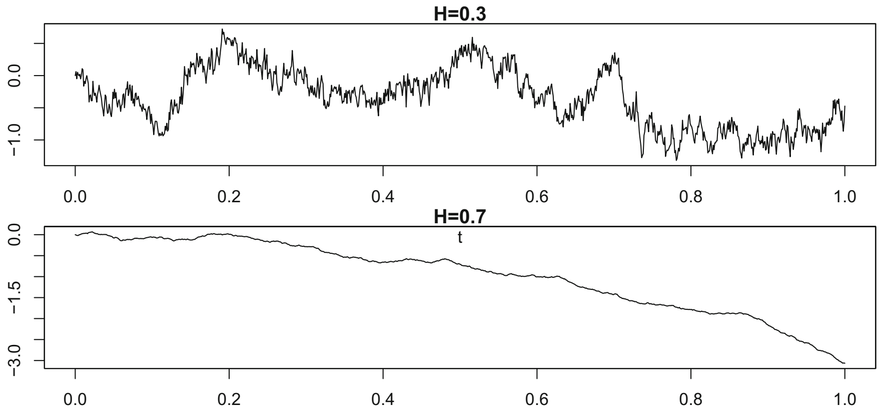


Fig. 5.1 Trajectories of the fractional Ornstein–Uhlenbeck process for different values of the Hurst parameter

5.2 Simulation of the Fractional Gaussian Noise

We briefly recall the two standard methods for the simulation of a fractional Gaussian noise (fGn) which `yuima` implements. It is important to compare different methods in order to understand their computational complexity, memory use and computation time. Both methods presented here are exact methods and the focus will be on the capability to simulate the fractional Gaussian noise for different mesh grids: random, deterministic, Poisson, tick-time (more generally for `yuima.sampling` object). Let $0 = t_0 < t_1 < \cdots < t_{n+1} = T$ be the mesh grid, not necessarily deterministic or regular. For $i = 0 \dots n$, let us define

$$X_i = W^H(t_{i+1}) - W^H(t_i)$$

such that the sequence $X^n = (X_0, X_1, \dots, X_n)$ is the fGn sample to be simulated. Let us, finally, note $\gamma(\cdot)$ the covariance function of the (zero-mean) process, that is, for $i, j \in \{0 \dots n\}^2$,

$$\gamma(i, j) = \mathbb{E}\left(X_i X_j\right) \tag{5.1}$$

and the covariance $(n+1) \times (n+1)$ matrix

$$\Gamma_n = (\gamma(i, j))_{i,j\in\{0\dots n\}^2} \; . \tag{5.2}$$

For the fGn, covariance function has a closed form, namely $\gamma(i, j)$ is equal to

$$\gamma(i, j) = \frac{1}{2}\left(|t_{i+1} - t_j|^{2H} - |t_{i+1} - t_{j+1}|^{2H} - |t_i - t_j|^{2H} + |t_i - t_{j+1}|^{2H}\right) . \tag{5.3}$$

5.2.1 Cholesky Method

This algorithm relies on the Cholesky decomposition of the covariance matrix Γ_n, namely,

$$\Gamma_n = L_n L_n^T$$

where L_n is a lower triangular matrix. It can be proven that such a decomposition exists when Γ_n is a symmetric positive definite matrix.

Then $X^n = \zeta^n L_n^*$, where $\zeta^n = (\zeta_0, \zeta_1, \dots, \zeta_n)$ is a Gaussian $(n+1)$-vector of standard independent component, is a fGn sample associated to the mesh grid because

$$\operatorname{cov}\left(\zeta^n L_n^*\right) = L_n \operatorname{cov}\left(\zeta^n\right) L_n^* = L_n L_n^* = \Gamma_n \, .$$

This method has been implemented in the `CholeskyfGn` function and uses the Cholesky decomposition R base function `chol`. The method is demanding in term of storage and number of operations and it is of order n^3.

5.2.2 Wood and Chan Method

This method proposed by and Wood and Chan (1994) can only be applied to stationary sequences. In this case, for the fGn,

$$\gamma(i, j) = \gamma(|i-j|) \, .$$

The algorithm relies on the embedding of the covariance matrix Γ_n into a circulant covariance matrix C_n of size $(2(n+1)-2) \times (2(n+1)-2)$, namely

$$C_n = \left(\begin{array}{cccc|ccc}
\gamma(0) & \gamma(1) & \dots & \gamma(n) & \gamma(n-1) & \dots & \gamma(1) \\
\gamma(1) & \gamma(0) & \dots & \gamma(n-1) & \gamma(n) & \dots & \gamma(2) \\
\vdots & \vdots & \ddots & \dots & \vdots & \dots & \vdots \\
\gamma(n) & \gamma(n-1) & \dots & \gamma(0) & \gamma(1) & \dots & \gamma(n-1) \\
\hline
\gamma(n-1) & \gamma(n) & \dots & \gamma(1) & \gamma(0) & \dots & \gamma(n-2) \\
\vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\
\gamma(1) & \gamma(2) & \dots & \gamma(n-1) & \gamma(n-2) & \dots & \gamma(0)
\end{array}\right) .$$

Since C_n is a circulant matrix, it has the eigenvalue decomposition

$$C_n = \frac{1}{2n} F_n \Delta_n F_n^*$$

where Λ_n is the diagonal matrix of terms $(\lambda_{1,1}, \ldots, \lambda_{2n,2n})$ which are the Fast Fourier Transform of the first row components of C_n

$$\lambda_{i,i} = \sum_{j=1}^{2n} (C_n)_{1,j} \exp\left(-2\pi\imath \frac{ij}{2n}\right) \quad i = 1 \ldots 2n$$

and F_n is a unitary matrix defined by

$$(F_n)_{i,j} = \frac{1}{\sqrt{2n}} \exp\left(-2\pi\imath \frac{ij}{2n}\right) \quad i, j = 1 \ldots 2n$$

with F_n^* its the conjugate transpose. Here $\imath = \sqrt{-1}$ and we extended $(C_n)_{i,j}$ for all integers j periodically with period $2n$. It has been shown (Crouse and Baraniuk 1999) that, in the fGn case, the matrix C_n is positive semi-definite; therefore the diagonal terms of Λ_n are nonnegative and

$$\tilde{\lambda}_n = \left(\sqrt{\frac{\lambda_{i,i}}{2n}}\right)_{i=1\ldots2n} .$$

Let $Z_n = \zeta_n + \imath\xi_n$ be a complex Gaussian vector where ζ_n and ξ_n are $2n$-vector of standard independent component and $\mathbb{E}[\zeta_n \xi_n^*] = 0$. And compute

$$Y^n = \Re\left(F_n \tilde{\lambda}_n Z_n\right) .$$

The first (n+1) component of Y^n, noted X^n is a fGn sample associated to the mesh grid (with the covariance).

This method has been implemented in the `WoodChanfGn` function and uses the Fast Fourier transform R base function `fft`. It is a fast and exact $n \log(n)$ operations method for regular discretization.

5.3 Simulation of Fractional Stochastic Differential Equations

In `yuima`, the user can choose between the above two simulation schemes specifying the argument `methodfGn` in the `simulate` method. The default simulation scheme is Wood and Chan and it is chosen by setting `methodfGn="WoodChan"`, the other simply by setting `methodfGn` to `Cholesky`.

Let $H > 1/2$. It had been shown that the one-dimensional stochastic differential equation

$$Y_t = x_0 + \int_0^t S(Y_s)ds + \int_0^t \sigma(Y_s)dW_s \quad 0 \le t \le T$$

admits a unique solution whose paths are Hölder continuous of order $\alpha > 1 - H$ a.s., when $\sigma \in \mathscr{C}_b^2$ and S satisfy a global Lipschitz condition, and where the integral is a pathwise Riemann–Stieltjes integral. Let us fix $\tilde{Y}_0^n$, the approximation scheme for the process Y is given by

$$\begin{cases} \tilde{Y}_0^n &= x_0 \quad \text{and for } k \in \{1 \dots n\}, \\ \tilde{Y}_{i+1}^n &= \tilde{Y}_i^n + \left(\theta S(\tilde{Y}_{i+1}^n) + (1-\theta) S(\tilde{Y}_i^n)\right)(t_{i+1} - t_i) \\ & \quad + \left(\mu\sigma(\tilde{Y}_{i+1}^n) + (1-\mu)\sigma(\tilde{Y}_i^n)\right)(W(t_{i+1}) - W(t_i)) \; . \end{cases}$$

which is an Euler scheme with linear-predictor method.

When $\sigma \in \mathscr{C}_b^2$ and $S \in \mathscr{C}_b^3$ the above Euler scheme converges (Neuenkirch and Nourdin 2007) to the true solution and

$$n^{2H-1}||\tilde{Y}^n - Y||_\infty \longrightarrow \frac{1}{2} \sup_{t\in[0,T]} \left| \int_0^t \sigma'(Y_s) D_s Y_t ds \right| \quad \text{almost surely as} \quad n \to +\infty,$$

where $D_s Y_t$ is the Malliavin derivative at time s of Y_t with respect to the fractional Brownian motion. Let us remark that for $H = 1/2$, the Euler explicit scheme converges to the Itô-SDE. Extensions to the multidimensional case are possible (Mishura and Shevchenko 2008) but not yet implemented in `yuima`.

5.4 Parametric Inference for the fOU

Statistical inference for general stochastic differential equations driven by fractional Brownian motion is not available due to its complexity. However, some results are available for the fractional Ornstein–Uhlenbeck process (fOU) solution of

$$Y_t = y_0 - \lambda \int_0^t Y_s ds + \sigma W_t^H, \quad t \geq 0, \tag{5.4}$$

where unknown parameter $\vartheta = (\lambda, \sigma, H)$ belongs to an open subset Θ of $(0, \Lambda) \times [\underline{\sigma}, \overline{\sigma}] \times (0, 1)$, $0 < \Lambda < +\infty$, $0 < \underline{\sigma} < \overline{\sigma} < +\infty$ and $H \in (0, 1)$. The fOU process is neither Markovian nor a semimartingale for $H \neq \frac{1}{2}$ but remains Gaussian and ergodic for $\lambda > 0$ (see Cheridito et al. 2003). For $H > \frac{1}{2}$, it even presents the long-range dependence property that makes it useful for different applications in biology and physics (with the Fractional Langevin Equation), ethernet traffic (Bregni and Erangoli 2005; Willinger et al. 1995) or finance (Xiao et al. 2011).

Estimation for the fOU from discrete observations is a relatively new field. Very recent works considered methods to estimate the drift λ by contrast procedure (Bertin et al. 2011; Hu et al. 2011; Ludena 2004; Neuenkirch and Tindel 2011) or the drift λ and the diffusion coefficient σ with discretization procedure of integral transform

(Xiao et al. 2011). In these papers, the Hurst exponent is supposed to be known and only consistency is obtained. On the other hand, methods to estimate the Hurst exponent H and the diffusion coefficient are presented in Berzin and Leon (2008) with classical order 2 variations convolution filters. The `yuima` implements the estimation procedure of Brouste and Iacus (2013) when all the parameters $\vartheta = (\lambda, \sigma, H)$ are unknown.

5.4.1 *Estimation of the Hurst Exponent and the Diffusion Coefficient via Quadratic Generalized Variations*

The key point here is that the Hurst exponent H and the diffusion coefficient σ can be estimated without prior knowledge on λ. Let us denote by X_i, $i = 0, 1, \ldots, n$, the discrete observations from the process (5.4) on a regular grid. Let $\mathbf{a} = (a_0, \ldots, a_K)$ be a discrete filter of length $K+1$, $K \in \mathbb{N}$, of order $L \geq 1$, $K \geq L$, i.e.

$$\sum_{k=0}^{K} a_k k^{\ell} = 0 \quad \text{for} \quad 0 \leq \ell \leq L-1 \quad \text{and} \quad \sum_{k=0}^{K} a_k k^{L} \neq 0. \tag{5.5}$$

normalized by $\sum_{k=0}^{K} (-1)^{1-k} a_k = 1$. In the following, we will also consider dilated filter $\mathbf{a}^2$ associated to $\mathbf{a}$ defined by

$$a_k^2 = \begin{cases} a_{k'} & \text{if } k = 2k' \\ 0 & \text{otherwise.} \end{cases} \quad \text{for} \quad 0 \leq k \leq 2K\,.$$

Since $\sum\limits_{k=0}^{2K} a_k^2 k^r = 2^r \sum\limits_{k=0}^{K} k^r a_k$, filter $\mathbf{a}^2$ as the same order as $\mathbf{a}$. We denote by

$$V_{N,\mathbf{a}} = \sum_{i=0}^{N-K} \left(\sum_{k=0}^{K} a_k X_{i+k} \right)^2$$

the generalized quadratic variations associated to the filter $\mathbf{a}$ (see for instance Istas and Lang 1997). Finally,

$$\widehat{H}_N = \frac{1}{2} \log_2 \frac{V_{N,\mathbf{a}^2}}{V_{N,\mathbf{a}}} \tag{5.6}$$

and

$$\widehat{\sigma}_N = \left(-2 \frac{V_{N,\mathbf{a}}}{\sum_{k,\ell} a_k a_\ell |k-\ell|^{2\widehat{H}_N} \Delta_N^{2\widehat{H}_N}} \right)^{\frac{1}{2}}. \tag{5.7}$$

If **a** is a filter of order $L \geq 2$, then, both estimators $\widehat{H}_N$ and $\widehat{\sigma}_N$ are strongly consistent and asymptotically Gaussian which exists in closed form (see Brouste and Iacus 2013). Classical filters of order $L \geq 1$ are defined by

$$a_k = c_{L,k} = \frac{(-1)^{1-k}}{2^K}\binom{K}{k} = \frac{(-1)^{1-k}}{2^K}\frac{K!}{k!(K-k)!} \quad \text{for} \quad 0 \leq k \leq K. \tag{5.8}$$

Daubechies filters of even order can also be considered (Daubechies 1992), for instance the order 2 Daubechies' filter:

$$\frac{1}{\sqrt{2}}(.4829629131445341, -.8365163037378077, .2241438680420134, .1294095225512603). \tag{5.9}$$

and this is the default filter used by the function `qgv` by `yuima`. The function `qgv`[1] needs as input a `yuima` object, with the `data` slot and accepts as arguments: `filter.type` (by default `"Daubechies"` but can be also `"Classical"`), the `order`, and eventually the sequence **a** which describes the filter. Let us consider the model

$$X_t = 1 - \lambda \int_0^t X_t dt + \sigma d W_t^H$$

and let us simulate a path with $(H = 0.7, \lambda = 2, \sigma = 1)$ in order to estimate the parameters

```
set.seed(123)
samp <- setSampling(Terminal=100, n=10000)
mod <- setModel(drift="-lambda*x", diffusion="sigma", hurst=NA)
ou <- setYuima(model=mod, sampling=samp)
fou <- simulate(ou, xinit=1,
  true.param=list(lambda=2, sigma=1), hurst=0.7)
fou

##
## Diffusion process
## Number of equations: 1
## Number of Wiener noises: 1
## Parametric model with 2 parameters
##
## Number of original time series: 1
## length = 10001, time range [0 ; 100]
##
## Number of zoo time series: 1
##            length time.min time.max delta
## Series 1   10001        0      100   0.01
```

[1] For 'quadratic generalized variation'.

Notice that in the definition of the model, we need to specify `hurst=NA` as, for real data and the purpose of estimation, we do not know the true value of H. Subsequently, in `simulate`, as the true value of H is unknown to the `yuima` model, we need to specify its value through the argument `hurst`. Now, we can apply `qgv` to the simulated data to get the following results

```
qgv(fou)

##
## Fractional OU estimation
##                  hurst     (sigma)
## Estimate   0.70203080 1.0151571
## Std. Error 0.01057269 0.0699011
```

5.4.2 *Estimation of the Drift Parameter*

It is well known that (Hu and Nualart 2010)

$$\mu_2 = \lim_{t \longrightarrow \infty} \text{var}(Y_t) = \lim_{t \longrightarrow \infty} \frac{1}{t} \int_0^t Y_t^2 dt = \frac{\sigma^2 \Gamma(2H+1)}{2\lambda^{2H}}$$

The above limiting result suggests the following moment-type estimator for λ:

$$\widehat{\lambda}_N = \left(\frac{2\,\widehat{\mu}_{2,N}}{\widehat{\sigma}_N^2 \Gamma\left(2\widehat{H}_N + 1\right)} \right)^{-\frac{1}{2\widehat{H}_N}} \tag{5.10}$$

where

$$\widehat{\mu}_{2,N} = \frac{1}{N} \sum_{n=1}^{N} X_n^2$$

is the empirical moment of order 2. The estimator $\widehat{\lambda}_N$ is consistent but asymptotically Gaussian only for $H \in \left(\frac{1}{2}, \frac{3}{4}\right)$, as shown in Brouste and Iacus (2013). This moment-type estimator is implemented in the function `mmfrac` which calls `qgv` if also H and the diffusion coefficient are unknown.

```
mmfrac(fou)

##
## Fractional OU estimation
##                  hurst     sigma     lambda
## Estimate   0.70203080 1.0151571   2.21593
## Std. Error 0.01057269 0.0699011   0.25919
```

5.5 An Example on Climate Change Data

The package `yuima` provides the data set `MWK151` which contains the measurements of the ring width of pine trees collected by Graybill and Shiyatov (1992). In particular, `MWK151` is a small subset of the general dataset, concerning only one site and the date spans from -608 to 1957.

```
data(MWK151)
str(MWK151)
```

```
## 'zoo' series from -608 to 1957
## Data: num [1:2566] 0.9 0.96 0.94 0.85 0.69 0.74 0.64 0.45
## 0.58 0.66 ...
## Index: num [1:2566] -608 -607 -606 -605 -604 -603 -602 -601
## -600 -599 ...
```

Looking at the data in Fig. 5.2, one can guess the fractional nature of these data and the persistence of the correlation through the plot of the autocorrelation function using `acf`

```
par(mfrow=c(1,2))
plot(MWK151, main="Methuselah Walk ring widths", xlab="year")
plot(acf(MWK151))
```

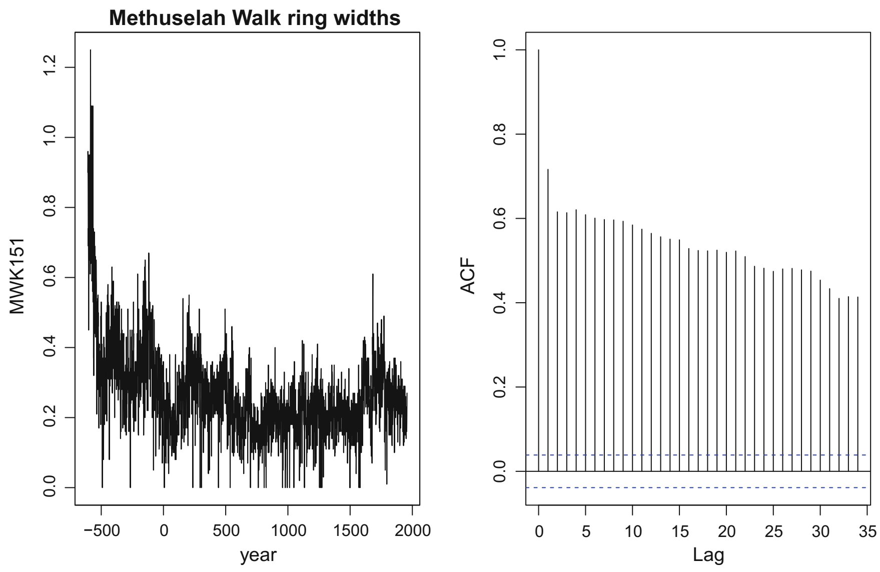


Fig. 5.2 Methuselah Walk ring widths, -608, 1957. The autocorrelation function presents strong temporal correlation

We now estimate a fOU for these data although the process exhibits a mean-reverting behaviour, thus the estimate of λ would not be quite correct in this situation.

```
mod <- setModel(drift="-lambda *x", diffusion="sigma", hurst=NA)
mwk  <- setYuima(model=mod, data=setData(MWK151))
mwk

##
## Diffusion process
## Number of equations: 1
## Number of Wiener noises: 1
## Parametric model with 2 parameters
##
## Number of original time series: 1
## length = 2566, time range [-608 ; 1957]
##
## Number of zoo time series: 1
##           length time.min time.max delta
## Series 1    2566     -608     1957     1

mmfrac(mwk)

##
## Fractional OU estimation
##                  hurst    (sigma)      lambda
## Estimate   0.24027157 0.09808681 0.001823339
## Std. Error 0.02208363 0.01202379 0.007963485
```

and we obtain $H = 0.24$. This results in line with the literature. As guessed, the estimate of λ is statistically not significant, meaning that probably the drift contains some mean-reverting property which we cannot extract from the simple estimation procedure for λ explained in the above.

Chapter 6
CARMA Models

Doob (1944) introduced continuous autoregressive moving average models, also known as CARMA, as continuous versions of the most famous ARMA models. The main assumption of both ARMA and CARMA were the Gaussian innovations of the process. Recently, Brockwell (2001) has extended the class of CARMA to Lévy process with finite second-order moments to allow for asymmetric and heavy-tailed increments frequently noted in real-time series. Examples of uses of these processes include Barndorff-Nielsen and Shephard (2001a), Todorov and Tauchen (2006), Todorov (2011) and Brockwell and Marquardt (2005).

6.1 Lévy-Driven CARMA Models

This section reviews the basic knowledge about Lévy-driven CARMA(p,q) models as described in Brockwell (2001). Let $p,\ q$ nonnegative integers such that $p > q \geq 0$. Then, the CARMA(p,q) process is defined as:

$$a(D)Y_t = b(D)DL_t \tag{6.1}$$

where a and b are polynomials

$$a\left(z\right) = z^p + a_1 z^{p-1} + \cdots + a_p$$

$$b\left(z\right) = b_0 + b_1 z^1 + \cdots + b_{p-1} z^{p-1}$$

with $a_1, \ldots, a_p$ and $b_0, \ldots, b_{p-1}$ are coefficients such that $b_q \neq 0$ and $b_j = 0$ $\forall j > q$ and D is the differentiation operator with respect to t.

The CARMA(p,q) model has the following convenient state-space representation

$$Y_t = \mathbf{b}^\intercal X_t \tag{6.2}$$

S. M. Iacus and N. Yoshida, *Simulation and Inference for Stochastic Processes with YUIMA*, Use R!, https://doi.org/10.1007/978-3-319-55569-0_6

where X_t is p-dimensional process solution to

$$\mathrm{d}X_t = AX_t\mathrm{d}t + \mathbf{e}\mathrm{d}L_t \tag{6.3}$$

and the $p \times p$ matrix A is given by

$$A = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ -a_p & -a_{p-1} & -a_{p-2} & \dots & -a_1 \end{bmatrix}$$

e and **b** are $p \times 1$ vectors defined as

$$\mathbf{e} = [0, \dots, 0, 1]^\intercal$$

$$\mathbf{b} = \left[b_0, \dots, b_{p-1}\right]^\intercal .$$

Given the X_s, the solution of (6.3) has this form:

$$X_t = e^{A(t-s)}X_s + \int_s^t e^{A(t-u)}\mathbf{e}\mathrm{d}L_u, \ \ \forall t > s, \tag{6.4}$$

where e^A is the matrix exponential

$$e^A = \sum_{h=0}^{+\infty} \frac{1}{h!} A^k.$$

If the real part of the eigenvalues $\lambda_1, \dots, \lambda_p$ of A is negative, then X_t in (6.3) has a covariance stationary solution (Brockwell 2001)

$$X_t = \int_{-\infty}^t e^{A(t-u)}\mathbf{e}\mathrm{d}L_u \overset{d}{=} \int_0^{+\infty} e^{Au}\mathbf{e}\mathrm{d}L_u \tag{6.5}$$

with

$$E\left[X_t\right] = \frac{\mu}{a_p}\mathbf{e}$$

$$Cov\left[X_{t+h}; X_t\right] = \sigma^2 e^{Ah} \int_0^{+\infty} e^{Au}\mathbf{e}\mathbf{e}^\intercal e^{A^\intercal u}\mathrm{d}u \ \ for \ \ h \geq 0.$$

where $\mu = E\left[L_1\right]$ and $\sigma^2 = Var\left[L_1\right]$.

6.2 CARMA Model Specification

In **yuima** package, the CARMA(p,q) model is specified by means of `setCarma` that creates an object of class `yuima.carma`. The `yuima.carma-class` extends the `yuima.model-class` and `simulate` method works out of the box as well as the `qmle` method (Iacus and Mercuri 2015).

6.2.1 The `yuima.carma-class`

The `yuima.carma` class stores the model in its generalized linear state-space representation. Let

$$\begin{aligned} Y_t &= c_0 + \sigma\,(\mathbf{b}^\top X_t) \\ \mathrm{d}X_t &= AX_t\mathrm{d}t + \mathbf{e}\,(\gamma_0 + \gamma^\top X_t)\,\mathrm{d}Z_t \end{aligned} \tag{6.6}$$

where $c_0 \in \mathbb{R}$ and $\sigma \in (0, +\infty)$ are location and scale parameters, respectively. Let $\mathbf{b} \in \mathbb{R}^p$ be the vector of moving average parameters $b_0, b_1, \ldots, b_q$ and A the $p \times p$ matrix containing the autoregressive parameters $a_1, \ldots, a_p$

$$A = \begin{bmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \\ -a_p & -a_{p-1} & -a_{p-2} & \ldots & -a_1 \end{bmatrix}.$$

The $\gamma_0 \in \mathbb{R}$ and the vector $\gamma := \left[\gamma_1, \ldots \gamma_p\right]$ are called linear parameters (Brockwell et al. 2006). The `yuima.carma` class extends the `yuima.model` by adding a new slot `info` of class `carma.info-class`. The `info` object is built for the user by `setCarma` function and contains the following slots:

- `p` the order of the autoregressive coefficients.
- `q` the order of the moving average coefficients.
- `loc.par` a label denoting the location coefficient. The default value `loc.par= NULL` means $c_0 = 0$.
- `scale.par` the label of scale coefficient. The default value `scale.par=NULL` implies that `sigma=1`.
- `ar.par` the label of the autoregressive coefficients. The default Value is `ar.par= "a"`.
- `ma.par` the label of the moving average coefficients. The default Value is `ma.par="b"`.
- `Carma.var` the label of the observed process. Defaults to `"v"`.

- `Latent.var` the label of the unobserved process. Defaults to `"x"`.
- `lin.par` the label of the linear coefficients. If `lin.par=NULL`, the `setCarma` builds the CARMA(p,q) model of Brockwell (2001).
- `XinExpr` is a logical variable. The default value `XinExpr=FALSE` implies that the starting condition for `Latent.var` is zero. If `XinExpr=TRUE`, each component of `Latent.var` has a parameter as a initial value.
- `...` Arguments to be passed to `setCarma`, such as the slots of `yuima.model-class`. They play a fundamental role when the underlying noise is a pure jump Lévy process. In particular, the following two arguments are necessary
  - `measure` Lévy measure of jump variables.
  - `measure.type` type specification for Levy measure.

Assume that we want to build a CARMA($p = 3$, $q = 1$) model. The representation (6.6) takes this form

$$\begin{aligned} Y_t &= b_0 X_{0,t} + b_1 X_{1,t} \\ \mathrm{d}X_{0,t} &= X_{1,t}\mathrm{d}t \\ \mathrm{d}X_{1,t} &= X_{2,t}\mathrm{d}t \\ \mathrm{d}X_{2,t} &= \left[-a_3 X_{0,t} - a_2 X_{1,t} - a_1 X_{0,t}\right]\mathrm{d}t + dZ_t \end{aligned} \tag{6.7}$$

here Z_t is a Wiener process. This model is created in **yuima** as follows

```
carma.mod<-setCarma(p=3,q=1,loc.par="c0",Carma.var="y",Latent.var="X")
carma.mod
```

```
##
## Carma process p=3, q=1
## Number of equations: 4
## Number of Wiener noises: 1
## Parametric model with 6 parameters
```

The internal structure of the object `carma.mod`

```
str(carma.mod)
```

```
## Formal class 'yuima.carma' [package "yuima"] with 17 slots
## ..@ info :Formal class 'carma.info' [package "yuima"] with
## 10 slots
## .. .. ..@ p : num 3
## .. .. ..@ q : num 1
## .. .. ..@ loc.par : chr "c0"
## .. .. ..@ scale.par : chr(0)
## .. .. ..@ ar.par : chr "a"
## .. .. ..@ ma.par : chr "b"
## .. .. ..@ lin.par : chr(0)
## .. .. ..@ Carma.var : chr "y"
## .. .. ..@ Latent.var: chr "X"
## .. .. ..@ XinExpr : logi FALSE
## ..@ drift : expression((b0 * X1 + b1 * X2), (X1), (X2)) ...
## ..@ diffusion :List of 4
## .. ..$ : expression((0))
## .. ..$ : expression((0))
## .. ..$ : expression((0))
## .. ..$ : expression((1))
## ..@ hurst : num 0.5
## ..@ jump.coeff : list()
## ..@ measure : list()
## ..@ measure.type : chr(0)
## ..@ parameter :Formal class 'model.parameter' [package
## "yuima"] with 7 slots
## .. .. ..@ all : chr [1:6] "b0" "b1" "a3" "a2" ...
## .. .. ..@ common : chr(0)
## .. .. ..@ diffusion: chr(0)
## .. .. ..@ drift : chr [1:5] "b0" "b1" "a3" "a2" ...
## .. .. ..@ jump : chr(0)
## .. .. ..@ measure : chr(0)
## .. .. ..@ xinit : chr "c0"
## ..@ state.variable : chr [1:4] "y" "X0" "X1" "X2"
## ..@ jump.variable : chr(0)
## ..@ time.variable : chr "t"
## ..@ noise.number : int 1
## ..@ equation.number: int 4
## ..@ dimension : int [1:6] 6 0 0 5 0 0
## ..@ solve.variable : chr [1:4] "y" "X0" "X1" "X2"
## ..@ xinit : expression((c0), (0), (0)) ...
## ..@ J.flag : logi FALSE
```

the slots `measure` and `measure.type` are empty in this case because the driving process is the standard Brownian motion. The slots `drift` and `diffusion` contain the **yuima** matrix-wise representation of model (6.7):

$$
\mathrm{d}\begin{bmatrix} Y_t \\ X_{0,t} \\ X_{1,t} \\ X_{2,t} \end{bmatrix} = \begin{bmatrix} b_0X_1 + b_1X_2 \\ X_{1,t} \\ X_{2,t} \\ -a_3X_{0,t} - a_2X_{1,t} - a_1X_{2,t} \end{bmatrix} \mathrm{d}t + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \mathrm{d}Z_t \tag{6.8}
$$

The observable process Y_t is represented as a stochastic differential equation. The slot `xinit` contains the location parameter c_0 and the starting condition for Y_t

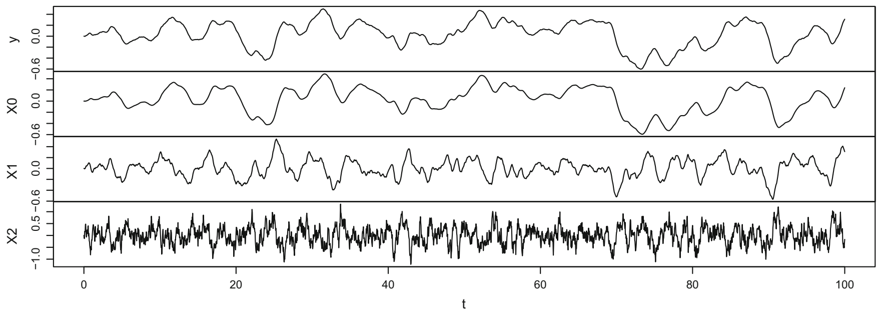


Fig. 6.1 Simulation of a CARMA(3,1) process driven by a standard Brownian motion

$$Y_0 = c_0 + b_0 X_0 + b_1 X_1$$

Let us choose $\mathbf{a} := [a_1, a_2, a_3]$ such that the eigenvalues of the matrix A are real and negative. For example, specifying $a_1 = 4$, $a_2 = 4.75$ and $a_3 = 1.5$ gives $\lambda_1 = -0.5$, $\lambda_2 = -1.5$ and $\lambda_3 = -2$. Now, we simulate a path of the process (6.7).

```r
par.carma<-list(a1=4,a2=4.75,a3=1.5,b0=1,b1=0.23,c0=0)
samp<-setSampling(Terminal=100, n=3000)
set.seed(123)
carma <-simulate(carma.mod,
    true.parameter=par.carma, sampling=samp)
```

The simulated sample path can be drawn using the `plot` function. The upper part of the plot contains a trajectory of the CARMA process Y_t while the remaining plots show the trajectories of each component of the state vector X_t (see Fig. 6.1).

```r
plot(carma)
```

where the initial condition is a vector of zeros.

6.3 CARMA(p,q) Model Estimation

We assume that the condition for canonical state representation (i.e. distinct eigenvalues for A matrix whose real part is negative) is satisfied. The estimation of the CARMA processes from real data involves a three steps procedure as the component X of the model is unobservable. The steps are as follows:

1. estimation of the CARMA parameters $\mathbf{a} = [a_1, \ldots, a_p]$ and $\mathbf{b} = [b_0, \ldots, b_q, b_{q+1} = 0, \ldots, b_{p-1} = 0]$ via quasi-maximum likelihood estimation (see Schlemm and Stelzer 2012, for univariate and multivariate cases). Least square estimation is also possible (see Brockwell et al. 2011, for more details) but not implemented

in package **yuima**. Since the state space representation in system (6.3) is based on the unobservable process X_t, a Kalman Filter procedure (see Tómasson 2013, for a CARMA model driven by a Brownian motion) is applied;
2. once the CARMA parameters have been estimated, the increments of the underlying Lévy process are extracted using the same ideas as in Brockwell et al. (2011, 2007); Brockwell and Schlemm (2013). To this aim the method `CarmaNoise` should be used;
3. finally, with the increments at hand, the parameters of the Lévy measure can be estimated. In **yuima**, the maximum likelihood approach is used.

Although the interface is the same, the `qmle` method accepts additional arguments, like the estimated Lévy increments obtained via the function `CarmaNoise`. The main new argument in the method `qmle` are the character-string variable `Est.Incr` and the logical variable `aggregation`. The variable `Est.Incr` manages the output of the `qmle` function. The variable `Est.Incr` assumes the following three values:

- `IncPar` (Default) returns the increments and the estimated parameters of the Lévy process.
- `Inc` returns just the increments of the Lévy process.
- `Par` returns only the estimated parameters of the Lévy measure.

The logical variable `aggregation` is related to the methodology for the estimation of the Lévy parameter. Indeed if the variable is `TRUE`, the increments are aggregated in order to obtain the increments on unitary time intervals. The function `CarmaNoise` can called directly by the user.

```
CarmaNoise(yuima, param, data=NULL)
```

where the arguments mean:

- `yuima` the CARMA model;
- `param` the `list` of parameters for the CARMA model;
- `data` unitary spaced observation or, if `NULL`, the data in the `yuima` object.

We now fit the model of Sect. 6.2 on the simulated data

```
 fit <- qmle(carma, start=par.carma)


##
## Starting qmle for carma ...

 fit


##
## Call:
## qmle(yuima = carma, start = par.carma)
##
```

```
## Coefficients:
##           b0           b1           a3           a2
## 4.570696e+02 5.786199e+00 2.217946e+03 9.152726e+02
##           a1           c0
## 1.917346e+03 3.536935e-02
```

Since the driven noise is a standard Brownian motion, the estimated parameters are only the AR and MA parameters.

6.4 Examples of Lévy-driven CARMA(*p*,*q*) Models

Given the Lévy process capability of **yuima** and the possibility to filter the Lévy increments, we can now show how to model, simulate and estimate some types of CARMA(p,q) processes driven by different types of Lévy noise.

6.4.1 Compound Poisson CARMA(2,1) Process

We simulate a trajectory from a CARMA(2,1) driven by a compound Poisson process with normally distributed jumps, and then we use this trajectory for the estimation procedure. It is worth to notice that since all the considered models can be seen as mixture of normals, the maximum likelihood estimation could be efficiently performed through an EM algorithm (Dempster et al. 1977). This algorithm was used by Hinde (1982) for the compound Poisson case, Loregian et al. (2012) for the variance gamma case and Dimitris (2002) for the NIG case. Let us consider a CARMA(2,1) model driven by a compound Poisson with Gaussian jumps and constant intensity $\lambda = 1$

```
modCP<-setCarma(p=2,q=1,Carma.var="y",
 measure=list(intensity="Lamb",df=list("dnorm(z, mu, sig)")),
 measure.type="CP")
true.parmCP <-list(a1=1.39631,a2=0.05029,b0=1,b1=2,
                   Lamb=1,mu=0,sig=1)
```

Let us generate sample paths

```
samp.L<-setSampling(Terminal=200,n=4000)
set.seed(123)
simCP<-simulate(modCP,true.parameter=true.parmCP,sampling=samp.L)
```

Figure 6.2 shows the trajectory of the process

```
plot(simCP,main="CP CARMA(2,1) model")
```

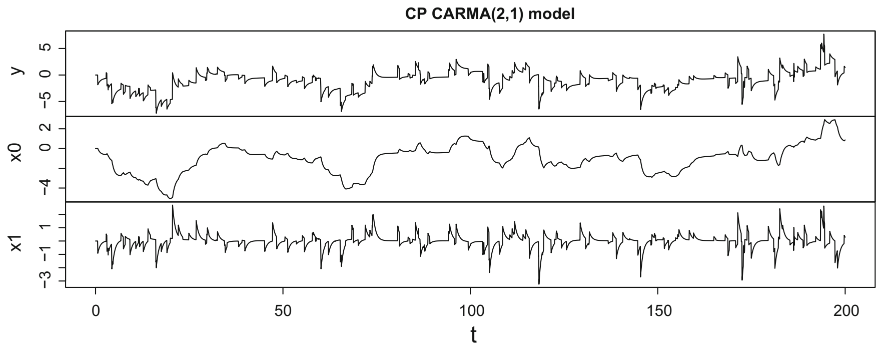


Fig. 6.2 Simulation of a CARMA(2,1) process driven by a compound Poisson process

and now we apply the estimation procedure described in Sect. 6.3

```
carmaoptCP <- qmle(simCP, start=true.parmCP, Est.Incr="Incr")


##
## Starting qmle for carma ...
##
##  Stationarity condition is satisfied...
##  Starting Estimation Increments ...

summary(carmaoptCP)


## Two Stage Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = simCP, start = true.parmCP, Est.Incr = "Incr")
##
## Coefficients:
##         Estimate Std. Error
## b1    1.82692151 0.02127052
## b0    0.78355697 0.25351014
## a2    0.07855805 0.04679159
## a1    1.37994579 0.22821157
## Lamb 1.00000000 0.00000000
## mu    0.00000000 0.00000000
## sig   1.00000000 0.00000000
##
## -2 log L: 4006.415
##
##
## Number of increments: 3999
##
## Average of increments: -0.004649
```

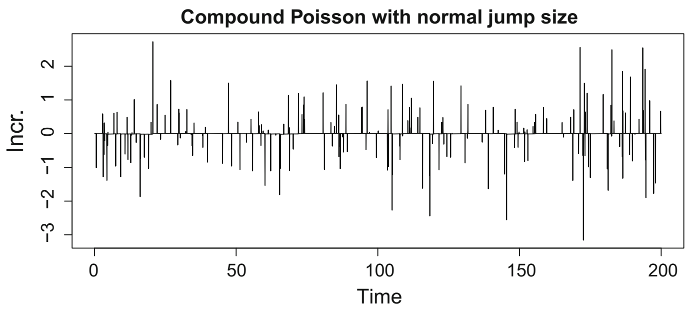


Fig. 6.3 The estimated increments from the CARMA(2,1) model with compound Poisson driving noise

```
##
## Standard Dev. of increments: 0.218469
##
## Summary statistics for increments:
##         Min.    1st Qu.     Median       Mean    3rd Qu.
## -3.1584305 -0.0051499 -0.0015587 -0.0046495  0.0006458
##         Max.
##  2.7267152
##
##
## Carma(2,1) model: Stationarity conditions are satisfied.
```

We can now plot the estimated increments (see, Fig. 6.3) extracted from the data specifying argument `Est.Incr="Incr"`.

```
plot(carmaoptCP,ylab="Incr.",type="l",
 main="Compound Poisson with normal jump size")
```

6.4.2 *Variance Gamma CARMA(2,1) Process*

Now let us consider a variance gamma Lévy process (Madan and Seneta 1990b)

```
modVG<-setCarma(p=2,q=1,Carma.var="y",
     measure=list("rvgamma(z,lambda,alpha,beta,mu)"),
     measure.type="code")
true.parmVG <-list(a1=1.39631, a2=0.05029, b0=1, b1=2,
                   lambda=1, alpha=1, beta=0, mu=0)
```

and let us simulate a sample path from this process (see, Fig. 6.4)

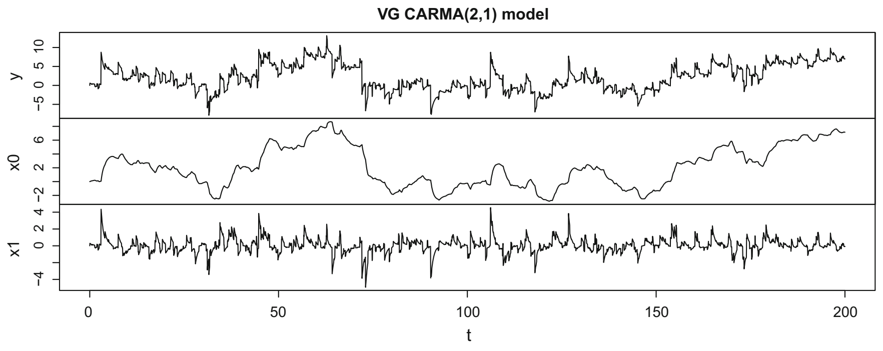


Fig. 6.4 The Variance Gamma CARMA(2,1) process

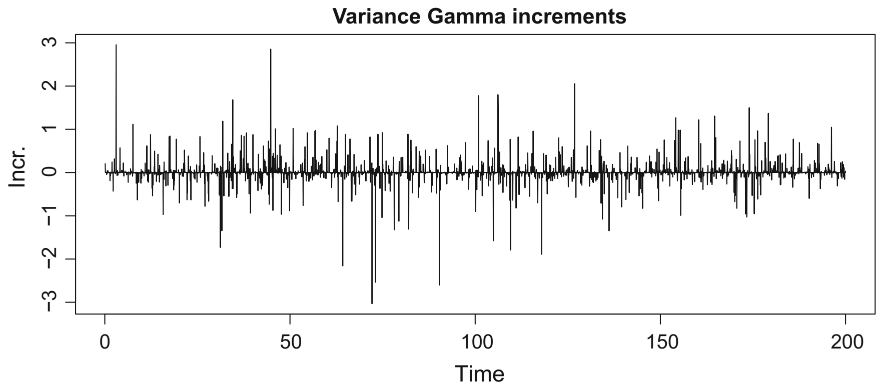


Fig. 6.5 The estimated increments for the variance gamma CARMA(2,1) process

```r
set.seed(100)
simVG<-simulate(modVG, true.parameter=true.parmVG,
 sampling=samp.L)
plot(simVG,main="VG CARMA(2,1) model")
```

We know estimate the parameters via `qmle` and plot the increments as shown in Fig. 6.5

```r
carmaoptVG <- qmle(simVG, start=true.parmVG, Est.Incr="Incr")
summary(carmaoptVG)
plot(carmaoptVG,xlab="Time",
 main="Variance Gamma increments",ylab="Incr.",type="l")
```

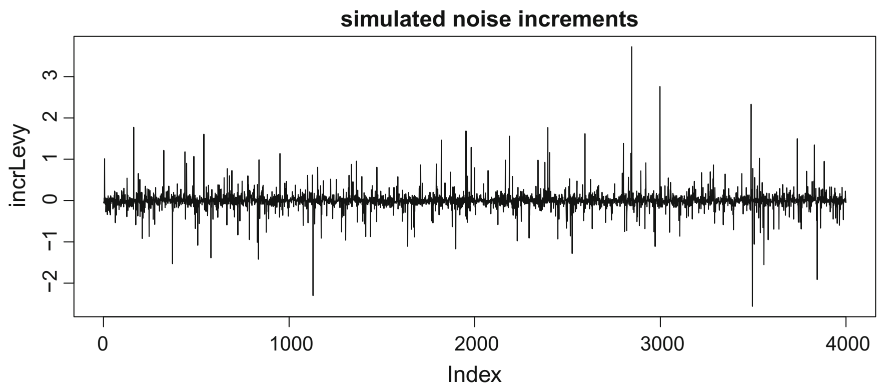


Fig. 6.6 The simulated NIG Lévy increments

6.4.3 Normal Inverse Gaussian CARMA(2,1) Process

We conclude the examples with NIG Lévy noise (Barndorff-Nielsen 1977).

```
modNIG<-setCarma(p=2,q=1,Carma.var="y",
   measure=list("rNIG(z,alpha,beta,delta1,mu)"),
   measure.type="code")
IncMod<-setModel(drift="0",diffusion="0",jump.coeff="1",
  measure=list("rNIG(z,1,0,1,0)"),measure.type="code")
set.seed(100)
simLev<-simulate(IncMod,sampling=samp.L)
incrLevy<-diff(as.numeric(get.zoo.data(simLev)[[1]]))
```

```
plot(incrLevy,main="simulated noise increments",type="l")
```

The simulated Lévy increments (see Fig. 6.6) are necessary for building the sample path of the CARMA(2,1) model driven by a Normal Inverse Gaussian Process. In yuima package, we simulate a trajectory using the code listed below:

```
true.parmNIG <-list(a1=1.39631,a2=0.05029,b0=1,b1=2,
                    alpha=1,beta=0,delta1=1,mu=0)
simNIG<-simulate(modNIG,true.parameter=true.parmNIG,sampling=samp.L)
```

Figure 6.7 shows the trajectory of the simulated process

```
plot(simNIG,main="NIG CARMA(2,1) model")
```

We now move to estimation

```
carmaoptNIG <- qmle(simNIG, start=true.parmNIG, Est.Incr="Incr")
```

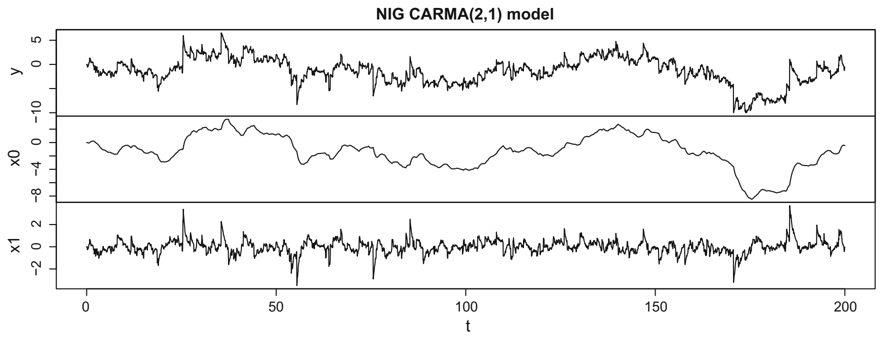


Fig. 6.7 The NIG CARMA(2,1) process

```
##
## Starting qmle for carma ...
##
##  Stationarity condition is satisfied...
##  Starting Estimation Increments ...

summary(carmaoptNIG)


## Two Stage Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = simNIG, start = true.parmNIG, Est.Incr = "Incr")
##
## Coefficients:
##           Estimate Std. Error
## b1      1.96714121 0.02334417
## b0      1.45518670 0.57884115
## a2      0.08622383 0.05543702
## a1      1.63602425 0.41296660
## alpha   1.00000000 0.00000000
## beta    0.00000000 0.00000000
## delta1  1.00000000 0.00000000
## mu      0.00000000 0.00000000
##
## -2 log L: 4610.317
##
##
## Number of increments: 3999
##
## Average of increments: -0.004172
##
## Standard Dev. of increments: 0.218784
##
## Summary statistics for increments:
##       Min.    1st Qu.    Median      Mean    3rd Qu.
```

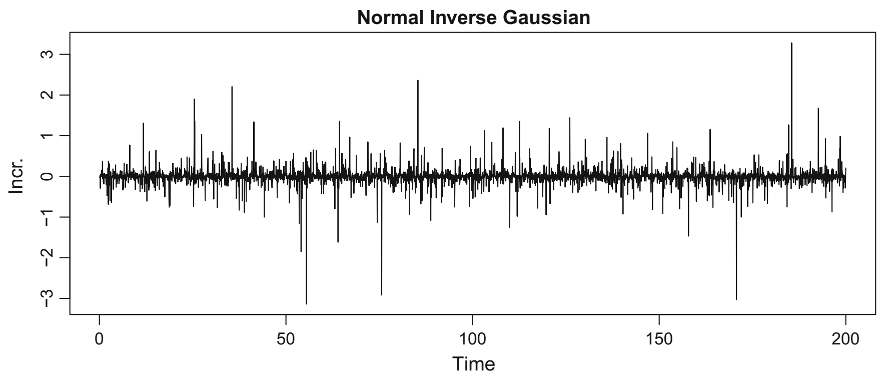


Fig. 6.8 The estimated increments for the NIG CARMA(2,1) process

```
## -3.138450 -0.050587 -0.001904 -0.004172  0.041460
##      Max.
##  3.283058
##
##
## Carma(2,1) model: Stationarity conditions are satisfied.
```

Figure 6.8 shows the estimated increments for the process.

```
plot(carmaoptNIG,main="Normal Inverse Gaussian",ylab="Incr.",type="l")
```

Now, we show how to estimate the parameters of the underlying Normal Inverse Gaussian Lévy Process using the package `GeneralizedHyperbolic` just to test the accuracy of the `qmle` function.

As a first step, we extract the Lévy innovations from the `yuima.carma.qmle` object

```
NIG.Inc<-as.numeric(coredata(carmaoptNIG@Incr.Lev))
NIG.freq<-frequency(carmaoptNIG@Incr.Lev)
```

then, we aggregate the innovations to work with increments on time intervals of length one.

```
t.idx <- seq(from=1, to=length(NIG.Inc), by=NIG.freq)
Unitary.NIG.Inc<-diff(cumsum(NIG.Inc)[t.idx])
```

The function `nigFit`, from **Generalized Hyperbolic**, fits the NIG distribution to generic i.i.d. data `Unitary.NIG.Inc` by exact maximum likelihood method

```
library(GeneralizedHyperbolic)

## Loading required package: DistributionUtils
## Loading required package: RUnit
```

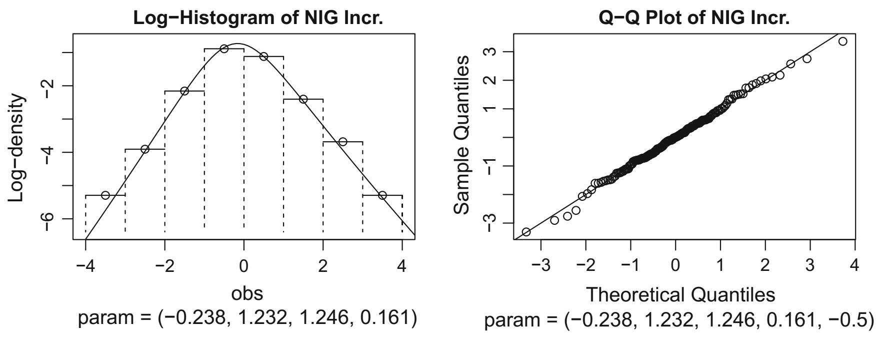


Fig. 6.9 The quality of fitting for the estimated NIG process

```
FitInc.NIG.Lev<-nigFit(Unitary.NIG.Inc)
summary(FitInc.NIG.Lev, hessian = TRUE, hessianMethod = "tsHessian")


##
## Data:       Unitary.NIG.Inc
## Hessian:  tsHessian
##                  mu      delta      alpha        beta
## mu    -236.98726 -17.66162  14.85199 -199.00307
## delta  -17.66162 -55.22145  41.34202  -26.00819
## alpha   14.85199  41.34202 -38.04039   26.13861
## beta  -199.00307 -26.00819  26.13861 -201.81750
## Parameter estimates:
##        mu         delta       alpha       beta
##    -0.2385      1.2323      1.2464      0.1615
##   ( 0.1672)   ( 0.3189)   ( 0.4015)   ( 0.1877)
## Likelihood:          -277.4848
## Method:              Nelder-Mead
## Convergence code:    0
## Iterations:          203
```

The `summary` of `qmle` and `nigFit` produce quite similar results. Figure 6.9 shows the theoretical and empirical log-densities (left side) and the corresponding QQ-plot (right side).

```
par(mfrow = c(1, 2))
plot(FitInc.NIG.Lev, which = 2:3,
       plotTitles = paste(c("Histogram of NIG ",
        "Log-Histogram of NIG ",
        "Q-Q Plot of NIG "), "Incr.", sep = ""))
```

6.5 Application to the VIX Index

In this example, adapted from Iacus and Mercuri (2014), we apply the CARMA model to the VIX CBOE Volatility Index, which is a measure of the implied volatility of

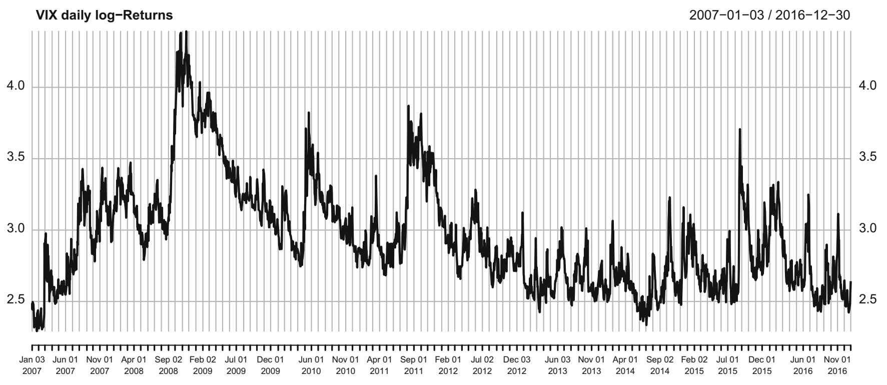


Fig. 6.10 The log(VIX) data

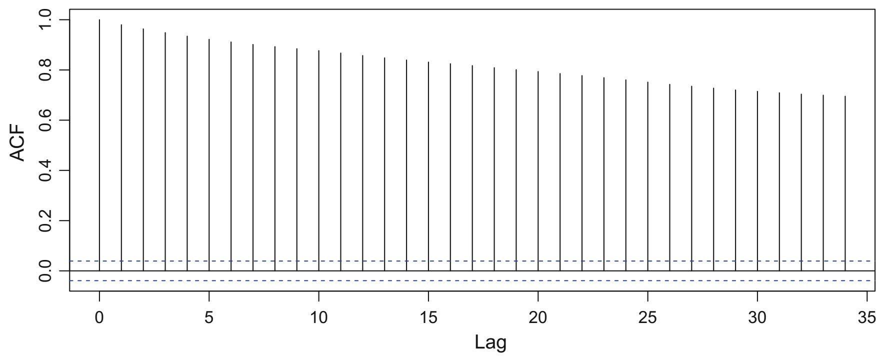


Fig. 6.11 The autocorrelation function for the log(VIX) data

S&P500 index options. VIX data can be accessed using `getSymbols` from the **quantmod** package. We take the log values of the VIX for the index (Figs. 6.10).

```
library(quantmod)
getSymbols("^VIX",  to="2016-12-31")

## [1] "VIX"

X <- VIX$VIX.Close
VIX.returns <- log(X)
plot(VIX.returns, main="VIX daily log-Returns")
```

We can now plot the autocorrelation function (see Fig. 6.11)

```
acf(VIX.returns)
```

Using the extended autocorrelation function `eacf` from package **TSA** (Chan and Ripley 2012), we can see that the most parsimonious model is an ARMA(2,1). For this reason, we now try to fit a CARMA(2,1) using Gaussian noise and two other types of Lévy noises: the VG and the NIG noise.

```
library(TSA)
eacf(VIX.returns,ar.max = 3, ma.max = 4)

## AR/MA
##   0 1 2 3 4
## 0 x x x x x
## 1 x o o o x
## 2 x o o o o
## 3 x o o o o

Delta <- 1/252
VIX.Data<-setData(VIX.returns,delta=Delta)
Normal.model<-setCarma(p=2, q=1,loc.par="mu")
Normal.CARMA<-setYuima(data=VIX.Data, model=Normal.model)
Normal.start <- list(a1=36,a2=56,b0=21,b1=1,mu=0)
Normal.est <- qmle(yuima=Normal.CARMA, start=Normal.start,
 Est.Incr="Incr")

##
## Starting qmle for carma ...
##
## Stationarity condition is satisfied...
##  Starting Estimation Increments ...

summary(Normal.est )


## Two Stage Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = Normal.CARMA, start = Normal.start, Est.Incr = "Incr")
##
## Coefficients:
##     Estimate  Std. Error
## b1   1.229888  0.02141029
## b0 38.038621 14.32246330
## a2 52.585978 33.17085654
## a1 57.771192 16.42969646
## mu  2.876486  0.21132590
##
## -2 log L: -5984.714
##
##
## Number of increments: 2516
##
## Average of increments: 0.000226
##
## Standard Dev. of increments: 0.060366
##
## Summary statistics for increments:
```

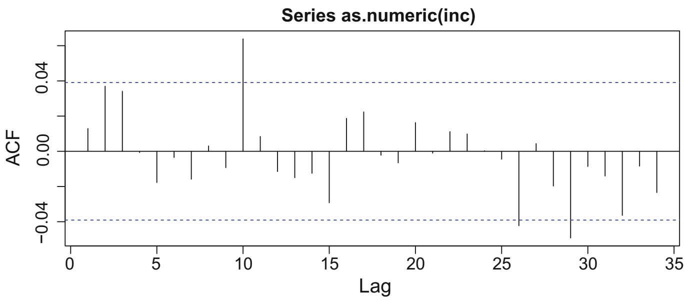


Fig. 6.12 The autocorrelation for the estimated Lévy increments

```
##        Min.    1st Qu.     Median       Mean    3rd Qu.
## -0.3522842 -0.0347673 -0.0065596  0.0002259  0.0284968
##        Max.
##  0.4056629
##
##
## Carma(2,1) model: Stationarity conditions are satisfied.
```

We can now extract the increments and test if they are Gaussian

```
inc <-Normal.est@Incr.Lev
shapiro.test(as.numeric(inc))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  as.numeric(inc)
## W = 0.93811, p-value < 2.2e-16
```

The Shapiro–Wilk test rejects the null hypotheses of normality for these data. We can also check if the CARMA model was able to remove residual autocorrelation

```
plot(acf(as.numeric(inc)))
```

Figure 6.12 suggests an autocorrelation effect at discrete lag 10, so we can apply the tests

```
Box.test(x=as.numeric(inc), lag = 10, type ="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  as.numeric(inc)
## X-squared = 18.87, df = 10, p-value = 0.04195
```

```
Box.test(x=as.numeric(inc), lag = 10, type ="Box-Pierce")
```

```
##
##  Box-Pierce test
##
## data:  as.numeric(inc)
## X-squared = 18.797, df = 10, p-value = 0.04292
```

and they both fail al 1% level of significance. We now proceed with the specification and estimation of the two alternative CARMA models

```
VG.model <- setCarma(p=2, q=1,loc.par="mu",
 measure=list("rvgamma(z,lambda,alpha,beta,mu0)"),
 measure.type="code")
NIG.model <- setCarma(p=2, q=1,loc.par="mu",
 measure=list(df=list("rNIG(z, alpha, beta, delta1, mu0)")),
 measure.type="code")

VG.CARMA<-setYuima(data=VIX.Data, model=VG.model)
NIG.CARMA<-setYuima(data=VIX.Data, model=NIG.model)

VG.start <- list(a1=36,a2=56,b0=21,b1=1,mu=0,
 lambda=1,alpha=1,beta=0,mu0=0)
NIG.start <- list(a1=36,a2=56,b0=21,b1=1,mu=0,
 alpha=2,beta=1,delta1=1,mu0=0)

fit.VG <- qmle(yuima=VG.CARMA,start=VG.start,
 Est.Incr="IncrPar",aggregation=FALSE)

##
## Starting qmle for carma ...
##
##  Stationarity condition is satisfied...
##  Starting Estimation Increments ...
##
## Starting Estimation parameter Noise ...

fit.NIG <- qmle(yuima=NIG.CARMA,start=NIG.start,
  Est.Incr="IncrPar",aggregation=FALSE)


##
## Starting qmle for carma ...
##
##  Stationarity condition is satisfied...
##  Starting Estimation Increments ...
##
## Starting Estimation parameter Noise ...

cf.VG <- coef(fit.VG )
cf.NIG <- coef(fit.NIG )

summary(fit.VG)
```

```
## Two Stage Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = VG.CARMA, start = VG.start, Est.Incr = "IncrPar",
##     aggregation = FALSE)
##
## Coefficients:
##           Estimate  Std. Error
## b0       38.038621 14.32246330
## b1        1.229888  0.02141029
## a2       52.585978 33.17085654
## a1       57.771192 16.42969646
## mu        2.876486  0.21132590
## lambda 340.342772 28.16689722
## alpha   30.227849  1.67107610
## beta     7.129974  0.93903097
## mu0     -5.566823  0.55743874
##
## -2 log L: -5984.714
##
##
## Number of increments: 2516
##
## Average of increments: 0.000226
##
## Standard Dev. of increments: 0.060366
##
##
## -2 log L of increments: -7430.572460
##
## Summary statistics for increments:
##       Min.    1st Qu.     Median       Mean    3rd Qu.
## -0.3522842 -0.0347673 -0.0065596  0.0002259  0.0284968
##       Max.
##  0.4056629
##
##
## Carma(2,1) model: Stationarity conditions are satisfied.

summary(fit.NIG)

## Two Stage Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = NIG.CARMA, start = NIG.start, Est.Incr = "IncrPar",
##     aggregation = FALSE)
##
## Coefficients:
##          Estimate  Std. Error
## b0      38.038621 14.32246330
## b1       1.229888  0.02141029
## a2      52.585978 33.17085654
## a1      57.771192 16.42969646
## mu       2.876486  0.21132590
## alpha   18.288242  1.51942625
## beta     6.735734  0.93264784
## delta1 13.567081  0.75121361
## mu0    -5.317212  0.54450503
##
## -2 log L: -5984.714
##
##
## Number of increments: 2516
##
## Average of increments: 0.000226
##
## Standard Dev. of increments: 0.060366
##
##
## -2 log L of increments: -7439.914384
##
## Summary statistics for increments:
##        Min.    1st Qu.     Median       Mean    3rd Qu.
## -0.3522842 -0.0347673 -0.0065596  0.0002259  0.0284968
##        Max.
##  0.4056629
##
##
## Carma(2,1) model: Stationarity conditions are satisfied.
```

We now compare the empirical density of the Lévy increments against the theoretical Gaussian, variance gamma and NIG densities. To put in evidence the discrepancy of the data from the Gaussian density, we plot each density d as $log(1+d)$. We add 1 because most density estimate will return 0. This scaling has not effect on estimation and it is done with the only purpose of plotting the data. Notice that in the next code the parameters μ and λ of the variance gamma distribution and δ and μ of the NIG distribution have been rescaled by the time mesh as the `qmle` return the estimation of the Lévy process for time $t=1$.

```
d.N <- function(u) log( 1+dnorm(u, mean=mean(inc), sd=sd(inc)) )
d.VG <- function(u) {
 log(1+dvgamma(u, lambda=cf.VG["lambda"]*Delta,
  alpha=cf.VG["alpha"], beta=cf.VG["beta"], mu=cf.VG["mu0"]*Delta))
}
d.NIG <- function(u) {
 log(1+dNIG(u,alpha=cf.NIG["alpha"], beta=cf.NIG["beta"],
   delta=cf.NIG["delta1"]*Delta, mu=cf.NIG["mu0"]*Delta))
}
d.Emp <- density(inc)
plot(d.Emp$x, log(1+d.Emp$y),type="l",
 main="Rescaled log-densities")
curve(d.N, min(d.Emp$x), max(d.Emp$x), col="blue",add=TRUE, lty=3)
curve(d.VG, min(d.Emp$x), max(d.Emp$x), col="red",add=TRUE,lty=4)
curve(d.NIG, min(d.Emp$x), max(d.Emp$x), col="green",add=TRUE,lty=2)
```

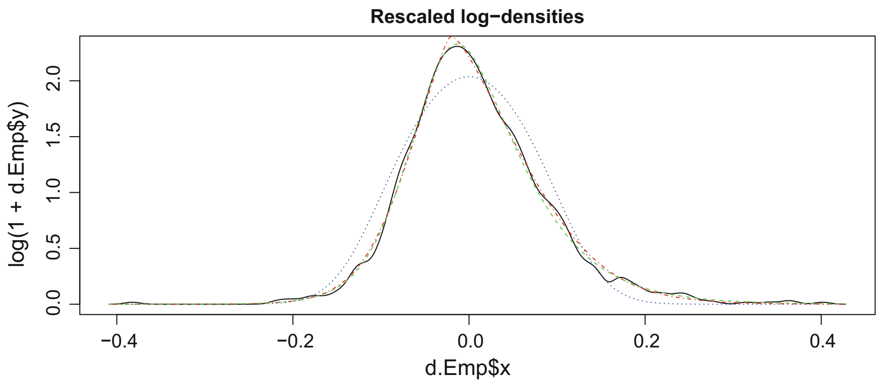


Fig. 6.13 The estimated densities of the Lévy increments. Each density d is plotted as $log(1 + d)$. The continuous line is the empirical density of the data, the dotted one is the Gaussian distribution, the dashed line represents the NIG fitted density and the last one the VG fitted density

Figure 6.13 shows that both NIG and VG fit the data better than the Gaussian distribution, with a slightly preference for the NIG distribution.

Chapter 7
COGARCH Models

In financial literature, stochastic volatility models have been considered to take into account the stylized facts often observed in the market. In general, for these models there is the requirement of two sources of randomness that drive respectively return and volatility processes. In GARCH and COGARCH processes, the idea is to focus on a single source of noise which affects both the return and the volatility processes. The COGARCH(1, 1) is indeed the stochastic volatility model introduced by Klüppelberg et al. (2004) as the continuous counterpart of the GARCH(1, 1), i.e., the following discrete-time series model:

$$\begin{cases} Y_i = \epsilon_i \sigma_i \\ \sigma_i^2 = \beta + \lambda_1 Y_{i-1}^2 + \delta_1 \sigma_{i-1}^2 \qquad i \in \mathbb{N}_0 \end{cases}$$

where β, λ_1, δ_1 are strictly positive, and both ϵ_0 and σ_0 are independent from future values of ϵ and possibly random (Bollerslev 1986). The model developed by Klüppelberg et al. (2004) is built on the idea to replace the discrete-time noise process $\{\epsilon_i, i \in \mathbb{N}_0\}$ with the increments ΔL_t of a Lévy process $\{L_t, t \geq 0\}$. This procedure occurs through several steps. Intuitively, a COGARCH(1, 1) is the limit of an explicit representation of a GARCH(1, 1). In fact, very roughly speaking, first consider the explicit representation of the volatility process for the GARCH(1, 1):

$$\sigma_i^2 = \beta \sum_{k=0}^{i-1} \prod_{j=k+1}^{i-1} (\delta + \lambda \epsilon_j^2) + \sigma_0^2 \prod_{j=0}^{i-1} (\delta + \lambda \epsilon_j^2).$$

The above equation can be extended to continuous time representation as follows:

$$\sigma_i^2 = \beta \int_{k=0}^{i-1} \exp\left[\sum_{j=\lfloor u \rfloor + 1}^{i-1} \log(\delta + \lambda \epsilon_j^2) \right] \mathrm{d}u + \sigma_0^2 \exp\left[\sum_{j=0}^{i-1} \log(\delta + \lambda \epsilon_j^2) \right]$$

S. M. Iacus and N. Yoshida, *Simulation and Inference for Stochastic Processes with YUIMA*, Use R!, https://doi.org/10.1007/978-3-319-55569-0_7

where $\lfloor u \rfloor$ denotes the integer part of u. Now, the idea is to replace the sequence $\{\epsilon_i, i \in \mathbb{N}_0\}$ with the increments of a Lévy process. To this aim, one needs to introduce the following auxiliary process $\{X_t, t \geq 0\}$:

$$X_t = \eta t - \sum_{0<s\leq t} \log(1 + \phi \Delta L_s^2)$$

for which $\mathbb{E}be^{-cX_t} = e^{t\Psi(c)}$, where $\Psi(c)$ is the Laplace exponent

$$\Psi(c) = -\eta c + \int_R \left[(1 + \phi x^2)^c - 1\right] \nu_L(\mathrm{d}x)$$

with $\nu_L(\cdot)$ the Lévy measure of process L. Then, for every $u \geq 0$ and $t > u$, the variance process can be rewritten as

$$\sigma_t^2 = \beta \mathrm{e}^{-(X_t - X_u)} \int_u^t e^{-(X_u - X_s)} \mathrm{d}s + e^{-(X_t - X_u)} \sigma_u^2.$$

It can be proved (Klüppelberg et al. 2004) that such a process, $\{\sigma_t^2, t \geq 0\}$, satisfies the following stochastic differential equation

$$\mathrm{d}\sigma_t^2 = \beta dt + \sigma_{t-}^2 e^{X_{t-}} \mathrm{d}(e^{-X_t})$$

which is solved as

$$\sigma_t^2 = \beta t + \log \delta \int_0^t \sigma_s^2 \mathrm{d}s + \frac{\lambda}{\delta} \sum_{0<s<t} \sigma_s^2 (\Delta L_s)^2 + \sigma_0^2. \tag{7.1}$$

By setting $\eta = -\log \delta$ and $\phi = \frac{\lambda}{\delta}$, Eq. (7.1) can be finally rewritten as

$$\mathrm{d}\sigma_t^2 = (\beta - \eta \sigma_{t-}^2)\mathrm{d}t + \phi \sigma_{t-}^2 \mathrm{d}[L, L]_t^d$$

where $[L, L]_t^d$ is the discrete part of the quadratic variation of the Lévy process and is defined as:

$$[L, L]_t^d := \sum_{0\leq s\leq t} (\Delta L_s)^2\,. \tag{7.2}$$

As result of these steps, the COGARCH(1, 1) model is defined as the solution $G = (G_t)_{t\geq 0}$ to the following stochastic differential equation

$$\begin{cases} \mathrm{d}G_t = \sigma_{t-} \mathrm{d}L_t \\ \mathrm{d}\sigma_t^2 = (\beta - \eta \sigma_{t-}^2)\mathrm{d}t + \phi \sigma_{t-}^2 \mathrm{d}[L, L]_t^d \end{cases} \tag{7.3}$$

where $\beta > 0$, $\eta \geq 0$, $\phi \geq 0$, $G_0 = 0$, and σ_0^2 are independent of the Lévy process L.

7.1 General Order (p,q) Model

The formulation of the COGARCH(1, 1) in formula (7.3) is not particularly suited for the extension to the general (p, q) case. A much convenient formal definition of the COGARCH(p, q) is the one proposed by Brockwell et al. (2006) that includes as a special case a version of (7.3) as we will see later. As mentioned in the above, the idea is to capture market's features using only one driving noise that controls both the dynamics of a return process $\{G_t\}$ and a volatility process $\{V_t\}$. In analogy with the discrete-time GARCH(p, q) model, where the variance process is a "self-exciting" predictable ARMA$(q, p-1)$ model driven by the squared of the innovations in the returns dynamics, Brockwell et al. (2006) constructed the COGARCH(p, q) model using the state-space representation of a CARMA$(q, p-1)$ model (see Chap. 6) for the variance process. The result is the following definition: the process $\{G_t, t \geq 0\}$ is a COGARCH(p, q) model if it satisfies the following system of stochastic differential equations:

$$\begin{cases} \mathrm{d}G_t &= \sqrt{V_t}\mathrm{d}L_t \\ V_t &= a_0 + a_1 Y_{1,t-} + \cdots + a_p Y_{p,t-} \\ \mathrm{d}Y_{1,t} &= Y_{2,t-}\mathrm{d}t \\ \quad\vdots &= \vdots \\ \mathrm{d}Y_{q-1,t} &= Y_{q,t-}\mathrm{d}t \\ \mathrm{d}Y_{q,t} &= \left(-b_q Y_{1,t-} - \cdots - b_q Y_{1,t-}\right)\mathrm{d}t + V_t \mathrm{d}\left[L, L\right]_t^d \end{cases}$$

where $\{L_t, t \geq 0\}$ is a pure jump finite variation Lévy process. As in the above, the quantity $\{[L, L]_t^d, t \geq 0\}$ is the discrete part of the quadratic variation of process L. From a practical point of view, one can think of the underlying Lévy process as a pure jump process for which its variation is composed only by the quadratic part. To be more precise and in order to input the above model into a **yuima** object, we make of the following notation:

$$\begin{cases} \mathrm{d}G_t = \sqrt{V_t}\mathrm{d}L_t \\ V_t = a_0 + \mathbf{a}^\intercal Y_{t-} \\ \mathrm{d}Y_t = BY_{t-}\mathrm{d}t + \mathbf{e}\left(a_0 + \mathbf{a}^\intercal Y_{t-}\right)\mathrm{d}\left[L, L\right]_t^d \end{cases} \tag{7.4}$$

where q and p are integers such that $q \geq p \geq 1$. The state-space process Y_t is a vector with q components:

$$Y_t = \left[Y_{1,t}, \ldots, Y_{q,t}\right]^\intercal .$$

The vector $\mathbf{a} \in \mathbb{R}^q$ is defined as:

$$\mathbf{a} = \left[a_1, \ldots, a_p, a_{p+1}, \ldots, a_q\right]^\intercal$$

with $a_{p+1} = \cdots = a_q = 0$. The companion $q \times q$ matrix B is

$$B = \begin{bmatrix} 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \\ -b_q & -b_{q-1} & \dots & -b_1 \end{bmatrix}.$$

The vector $\mathbf{e} \in \mathbb{R}^q$ contains zero entries except the last component that is equal to one, and $[L, L]_t^d$ is as in formula (7.2).

7.1.1 How to Input a COGARCH(p,q) Model in yuima

To specify a COGARCH(p, q) model in **yuima**, we need to use the constructor function `setCogarch` which will build a new object of class `yuima.cogarch`. This class extends the `yuima.model-class`, and it contains specific information for the COGARCH(p, q) process. The use of the function is similar to `setCarma` so we do not give full details here, but the reader can refer to the manual page of the function. In the following lines, we build two models: a COGARCH(1, 1) and a COGARCH(2, 2) models both driven by a compound Poisson process with standard Gaussian jumps and constant intensity $\lambda = 1$.

```
# COGARCH(1,1) driven by CP
Cog11 <- setCogarch(p = 1, q=1,  measure = list(intensity="1",
  df="dnorm(z, 0, 1)"), measure.type = "CP", XinExpr = TRUE)
Cog11

##
## Cogarch process p=1, q=1 with Levy jumps
## Number of equations: 3
## Number of Levy noises: 1
## Number of quadratic variation: 1
## Parametric model with 4 parameters

# COGARCH(2,2) driven by CP
Cog22 <- setCogarch(p=2, q=2, measure = list(intensity="1",
  df="dnorm(z, 0, 1)"), measure.type = "CP", XinExpr = TRUE)
Cog22

##
## Cogarch process p=2, q=2 with Levy jumps
## Number of equations: 4
## Number of Levy noises: 1
## Number of quadratic variation: 1
## Parametric model with 7 parameters
```

The arguments `measure` and `measure.type` specify the Lévy measure for the underlying noise L_t. Choosing `XinExpr = TRUE`, the user must specify the starting condition of the COGARCH(p, q) model in the simulation and estimation steps for this model. The new object is of class `yuima.cogarch` which is an extension of the `yuima` class with the additional slot `info` as we can see below

```
class(Cog11)

## [1] "yuima.cogarch"
## attr(,"package")
## [1] "yuima"

slotNames(Cog11)

##  [1] "info"             "drift"
##  [3] "diffusion"        "hurst"
##  [5] "jump.coeff"       "measure"
##  [7] "measure.type"     "parameter"
##  [9] "state.variable"   "jump.variable"
## [11] "time.variable"    "noise.number"
## [13] "equation.number"  "dimension"
## [15] "solve.variable"   "xinit"
## [17] "J.flag"

str(Cog11@info,2)

## Formal class 'cogarch.info' [package "yuima"] with 11 slots
##   ..@ p           : num 1
##   ..@ q           : num 1
##   ..@ ar.par      : chr "b"
##   ..@ ma.par      : chr "a"
##   ..@ loc.par     : chr "a0"
##   ..@ Cogarch.var : chr "g"
##   ..@ V.var       : chr "v"
##   ..@ Latent.var  : chr "y"
##   ..@ XinExpr     : logi TRUE
##   ..@ measure     :List of 2
##   ..@ measure.type: chr "CP"
```

7.1.2 Stationarity Conditions

The process G has a stationary solution when the following assumption on the decomposition of the matrix B holds true:

$$B = SDS^{-1}$$

with

$$S = \begin{bmatrix} 1 & \dots & 1 \\ \lambda_1 & \dots & \lambda_q \\ \vdots & & \vdots \\ \lambda_1^{q-1} & \dots & \lambda_q^{q-1} \end{bmatrix}, \; D = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_q \end{bmatrix} \tag{7.5}$$

where $\lambda_1, \lambda_2, \dots, \lambda_q$ are the eigenvalues of matrix A and are ordered as follows:

$$\Re\{\lambda_1\} \geq \Re\{\lambda_2\} \geq \dots \geq \Re\{\lambda_q\}.$$

Applying the theory of stochastic recurrence equations (Brandt 1986; Kesten 1973), Brockwell et al. (2006) provide a sufficient condition for the strict stationarity of the COGARCH(p, q) model under the assumptions that the eigenvalues $\lambda_1, \dots, \lambda_q$ are distinct and the underlying process L has a nontrivial Lévy measure $\nu_L(\cdot)$. If this is the case, the process Y_t converges in distribution to the random variable Y_∞ if there exists some $r \in [1, +\infty]$ such that:

$$\int_{-\infty}^{+\infty} \ln\left(1 + \|S^{-1}\mathbf{e}\mathbf{a}^\intercal S\|_r x^2\right) \nu_L(\mathrm{d}x) \leq \Re\{\lambda_1\} \tag{7.6}$$

for some matrix S such that the matrix B is diagonalizable. If the initial condition is set to $Y_0 \overset{d}{=} Y_\infty$, then the process Y_t is strictly stationary and consequently the variance process V_t is strictly stationary as well.

In the general case, COGARCH(p, q) case, the inequality in (7.6) gives only a sufficient condition on the strict stationarity, but in the COGARCH(1, 1) case, it is also a necessary condition and can be simplified as follows:

$$\int_{-\infty}^{+\infty} \ln\left(1 + a_1 x^2\right) \nu_L(\mathrm{d}x) \leq b_1. \tag{7.7}$$

For other specific configurations of p and q, Tsai and Chan (2005) listed some useful conditions which must hold in addition to (7.6):

1. A necessary and sufficient condition to guarantee stationarity in the case of the COGARCH(2, 2) model is that the eigenvalues of B are real and $a_2 \geq 0$ and $a_1 \geq -a_2\lambda(B)$, where $\lambda(B)$ is the largest eigenvalue.
2. Under condition $2 \leq p \leq q$, all eigenvalues of B are negative and ordered in an increasing way $\lambda_1 \geq \lambda_2 \geq, \dots, \geq \lambda_{p-1}$ and γ_j are the roots of $a(z) = 0$ ordered as $0 > \gamma_1 \geq \gamma_2 \geq \dots \geq \gamma_{p-1}$. Then, a sufficient condition for stationarity is

$$\sum_{i=1}^{k} \gamma_i \leq \sum_{i=1}^{k} \lambda_i \quad \forall k \in \{1, \dots, p-1\}.$$

3. For the COGARCH(1, q) model, a sufficient condition that ensures stationarity is that all eigenvalues must be real and negative.

For some of these cases, the **yuima** package provides a diagnostic tool named `Diagnostic.Cogarch` which requires the COGARCH model specification and the parameters of the model as input:

```
# Param of the COGARCH(1,1)
paramCP11 <- list(a1 = 0.038, b1=  0.053, a0 = 0.04/0.053,
 y01 = 50.31)
check11 <- Diagnostic.Cogarch(Cog11, param=paramCP11)

##
##  COGARCH(11) model
##
##  The process is strictly stationary
##  The unconditional first moment of the Variance process exists
##
##  the Variance is a positive process

str(check11)

## List of 4
##  $ meanVarianceProc : num [1, 1] 2.67
##  $ meanStateVariable: num [1, 1] 50.3
##  $ stationary       : logi TRUE
##  $ positivity       : logi TRUE

# Param of the COGARCH(2,2)
paramCP22 <- list(a1 = 0.04, a2 = 0.001, b1 = 0.705, b2 = 0.1,
 a0 = 0.1, y01=01, y02 = 0)
check22 <- Diagnostic.Cogarch(Cog22, param=paramCP22)

##
##  COGARCH(22) model
##
##  The process is strictly stationary
##  The unconditional first moment of the Variance process exists
##
##  the Variance is a positive process

str(check22)

## List of 4
##  $ meanVarianceProc : num [1, 1] 0.167
##  $ meanStateVariable: num [1:2, 1] 1.67 0
##  $ stationary       : logi TRUE
##  $ positivity       : logi TRUE
```

7.2 Simulation Schemes

The **yuima** package implements two different schemes for the simulation of the COGARCH(p, q) model. The first is a plain Euler–Maruyama scheme which is based on the discretization of the system (7.4) on a regular grid on $[0, T]$ with N intervals of length Δt. The steps of the algorithms are as follows:

1. generate the trajectory of the underlying Lévy process L_t sampled on the grid $t_i = i \cdot \Delta t, i = 0, 1, \ldots, N$;
2. given the initial conditions $Y_0 = y_0$ and $G_0 = 0$, we iterate this equation

$$Y_n = (I + B\Delta t)\, Y_{n-1} + \mathbf{e}\,(a_0 + \mathbf{a}^\intercal Y_{n-1})\,\Delta\,[LL]^d_n\,. \tag{7.8}$$

   where $\Delta\,[LL]^d_n$ is approximated as $\Delta\,[LL]^d_n = (L_n - L_{n-1})^2$;
3. once the trajectory of the state process Y_n is available, the variance process and the COGARCH process G_n are obtained through the following equations:

$$V_n = a_0 + \mathbf{a}^\intercal Y_n$$

   and

$$G_n = G_{n-1} + \sqrt{V_n}\,(L_n - L_{n-1})\,.$$

Although the discretized version of the state process Y_n in (7.8) is a stochastic recurrence equation, the conditions for stationarity and non-negativity for the variance V_n process are not the ones of the original process. It is always possible to generate an example in which the discretized variance process V_n assumes negative values while the true process is non-negative with probability one. In fact, the following example clarifies the problem. Let us consider the following COGARCH$(1, 1)$ model driven by a variance gamma Lévy process (Madan and Seneta 1990b; Loregian et al. 2012 see). In this case, in order to have a non-negative solution for the variance process we need to check if $a_0 > 0$ and $a_1 > 0$, while the strict stationarity condition for the COGARCH(1,1) is ensured by $E[L^2] = 1$ and $a_1 - b_1 < 0$. The last two requirements guarantee also the existence of the stationary unconditional mean of the variance process V_t.

```r
model1 <- setCogarch(p = 1, q = 1,
  measure=list("rvgamma(z, 1, sqrt(2), 0, 0)"),
  measure.type = "code", Cogarch.var = "G",
  V.var = "v", Latent.var="x", XinExpr=TRUE)
```

We now simulate this model under the following choice of the parameters and sampling scheme

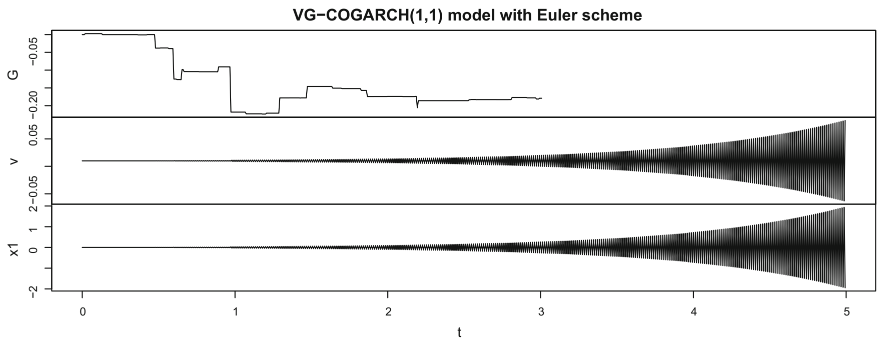


Fig. 7.1 Effect of discretization on the simulation of a COGARCH(1, 1) process under Euler scheme

```
param1 <- list(a1 = 0.038, b1 = 301, a0 =0.01, x01 = 0)
Diagnostic.Cogarch(model1, param=param1)

##
##  COGARCH(11) model
##
##  The process is strictly stationary
##  The unconditional first moment of the Variance process exists
##
##  the Variance is a positive process

Terminal1 <- 5
n1 <- 750
samp1 <- setSampling(Terminal=Terminal1, n=n1)
set.seed(123)
sim1 <- simulate(model1, sampling = samp1, true.parameter = param1,
 method="euler")
```

If we now look at the simulated trajectory in Fig. 7.1, we can see how the discretization affects the simulation although the variance process is theoretically positive

```
plot(sim1, main="VG-COGARCH(1,1) model with Euler scheme")
```

This problem can be avoided using a different discretization of Y applying Ito's Lemma for semimartingales (Protter 1990) to the transformation $e^{-Bt}Y_t$. Indeed,

$$e^{-B\Delta t}Y_t = Y_{t-\Delta t} - \int_{t-\Delta t}^{t} Be^{-Au}Y_{u-}\mathrm{d}u + \int_{t-\Delta t}^{t} e^{-Bu}\mathrm{d}Y_u$$
$$+ \sum_{s\leq t}\left[e^{-Bs}\left(Y_s - Y_{s-}\right) - e^{-Bs}\left(Y_s - Y_{s-}\right)\right].$$

We substitute the definition of Y_t in (7.4) and get

$$e^{-Bt}Y_t = Y_{t-\Delta t} - \int_{t-\Delta t}^{t} Be^{-Bu}Y_{u-}\mathrm{d}u + \int_{t-\Delta t}^{t} e^{-Bu}BY_{u-}\mathrm{d}u \\ + \int_{t-\Delta t}^{t} e^{-Bu}\mathbf{e}\left(a_0 + \mathbf{a}^\intercal Y_{u-}\right)\mathrm{d}\left[LL\right]_u^d$$

Making use of the following property for an exponential matrix

$$\begin{aligned} Be^{Bt} &= B\left(I + Bt + \frac{1}{2}B^2t^2 + \frac{1}{3!}B^3t^3 + \ldots\right) \\ &= \left(B + tB^2 + \frac{1}{2}t^2B^3 + t^3\frac{1}{3!}B^4 + \ldots\right) \\ &= \left(I + tB + \frac{1}{2}t^2B^2 + t^3\frac{1}{3!}B^3 + \ldots\right)B = e^{Bt}B, \end{aligned}$$

we get

$$Y_t = e^{Bt}Y_{t-\Delta t} + \int_{t-\Delta t}^{t} e^{B(t-u)}\mathbf{e}\left(a_0 + \mathbf{a}^\intercal Y_{u-}\right)\mathrm{d}\left[LL\right]_u^d \tag{7.9}$$

Then, (7.9) is discretized as follows:

$$Y_n = e^{B\Delta t}Y_{n-1} + e^{B\Delta t}\mathbf{e}\left(a_0 + \mathbf{a}^\intercal Y_{n-1}\right)\left([LL]_n^d - [LL]_{n-1}^d\right) \tag{7.10}$$

which can be rewritten as

$$Y_n = a_0e^{B\Delta t}\mathbf{e}\Delta\left[LL\right]_n^d + e^{B\Delta t}\left(I + \mathbf{e}\mathbf{a}^\intercal\Delta\left[LL\right]_n^d\right)Y_{n-1}. \tag{7.11}$$

with $\Delta\left[LL\right]_n^d = \left[LL\right]_n^d - \left[LL\right]_{n-1}^d$. Finally, (7.11) is used to replace (7.8) in the original Euler scheme. To invoke this discretization scheme, one should specify the argument `method ="mixed"` in the `simulate` method.

```r
set.seed(123)
sim2 <- simulate(model1, sampling = samp1, true.parameter = param1,
         method="mixed")
```

Figure 7.2 obtained with

```r
plot(sim2, main="VG-COGARCH(1,1) model with mixed scheme")
```

shows the stability of this method.

In the case of the COGARCH(p, q) driven by a compound Poisson Lévy process, the trajectory can be simulated without error directly from (7.9) as it is possible to know the jump times of the underlying Poisson process and calculate without approximation the quadratic variation of the process. Then, simple interpolation is

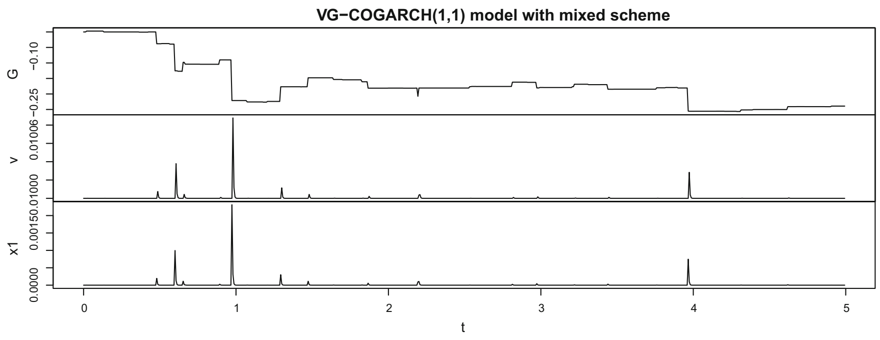
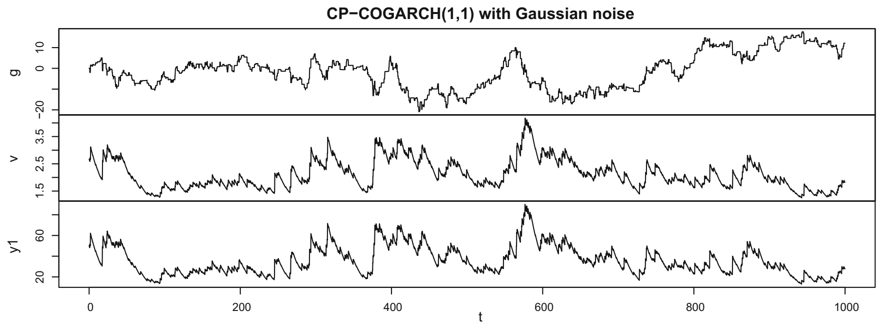


Fig. 7.2 Stability of the simulation of a COGARCH(1, 1) process under the mixed scheme



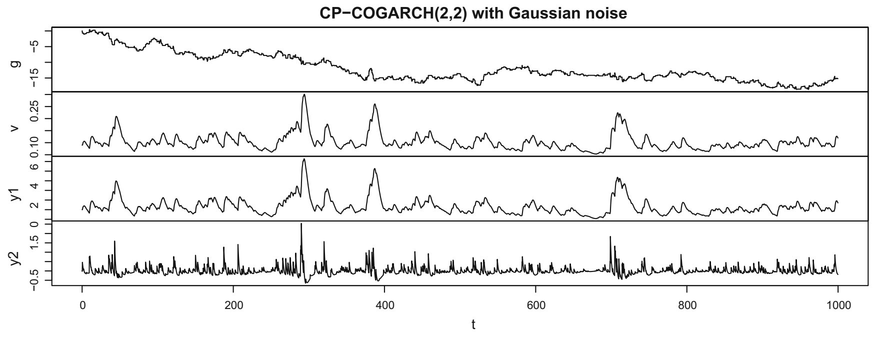


Fig. 7.3 Exact simulation of compound Poisson COGARCH(1, 1) (up) and COGARCH(2, 2) (bottom) processes under the Euler scheme

used to get observations on the given initial grid. As an example, we now plot a trajectory of the two processes of Sect. 7.1.2. The results are shown in Fig. 7.3.

```
sampCP <- setSampling(0, 1000, 5000)
simCog11 <- simulate(Cog11, true.par=paramCP11, sampling=sampCP)
simCog22 <- simulate(Cog22, true.par=paramCP22, sampling=sampCP)
```

Figure 7.2 obtained with

```
plot(simCog11, main="CP-COGARCH(1,1) with Gaussian noise")
plot(simCog22, main="CP-COGARCH(2,2) with Gaussian noise")
```

For more details about the simulation scheme and other properties of the moments of the processes V, G, and Y, refer to Iacus et al. (2017).

7.3 Generalized Method of Moments Estimation

One nice feature of the COGARCH(p, q) model is that the moments can be obtained in explicit form and these can in turn be used for a generalized method of the moment estimator. For the COGARCH$(1, 1)$ model, the moment-type estimator has been proposed in Klüppelberg et al. (2004) and Chadraa (2009) further generalized it to the COGARCH(p, q) case. The estimation procedure is similar to the CARMA(p, q) case of Chap. 6:

1. The parameters $\mathbf{a} := \left[a_1, \ldots, a_p\right]$, $\mathbf{b} := \left[b_1, \ldots, b_q\right]$ and the constant term a_0 of COGARCH(p, q) model are first estimated by the generalized method of moments by matching the empirical and the theoretical autocorrelation function.
2. Then the increments of the underlying Lévy process are obtained from the standardized residuals of the previously estimated model; and
3. Finally, maximum likelihood estimators of the parameters of the Lévy process are calculated.

We now briefly go through the steps, but full details can be found in Iacus et al. (2017).

7.3.1 *Moments Matching Step*

Let us assume that the Lévy process L is a zero-mean symmetric process. Let $G_0, G_1, \ldots, G_n, \ldots, G_T$, where $G_i = G(t_i)$, $t_i = i \cdot \Delta_t$, $i = 0, 1, \ldots, N$. Let us define the increments and the r-lagged increments of the observed COGARCH(p, q) process as

$$G_n^{(1)} = G_n - G_{n-1}, \quad G_n^{(r)} = G_n - G_{n-r}$$

where $r \geq 1$ is an integer. The increments can be obtained through time aggregation of increments of lag one, i.e., $G_n^{(r)} = \sum_{h=n-r}^{n} G_n^{(1)}$, and this fact will be used to simplify the estimation step later on. Using the lagged increments, we compute the empirical second moments

$$\hat{\mu}_r = \frac{1}{N-d-r+1} \sum_{n=r}^{N-d} \left(G_n^{(r)}\right)^2$$

and the empirical autocovariance function $\hat{\gamma}(h)$:

$$\hat{\gamma}_r(h) = \frac{1}{N-d-r+1} \sum_{n=r}^{N-d} \left(\left(G_{n+h}^{(r)} \right)^2 - \hat{\mu}_r \right) \left(\left(G_n^r \right)^2 - \hat{\mu}_r \right) \quad h = 0, 1, \ldots, d$$

where d is the maximal lag. From the above expressions, we obtain the empirical autocorrelation function:

$$\hat{\rho}_r(h) = \frac{\hat{\gamma}_r(h)}{\hat{\gamma}_r(0)}.$$

Let $\rho_r(h; \theta)$ be the theoretical correlation function, where we have put in evidence the explicit dependence on the parameters $\theta = (\mathbf{a}, \mathbf{b})$. Now let

$$\hat{g}_h(\theta) = \rho_r(h; \theta) - \hat{\rho}_r(h), \quad h = 1, \ldots, d.$$

The GMM estimator of θ is the solution of the following optimization problem

$$\min_{\theta \in \mathbb{R}^{q+p}} \|\hat{g}(\theta)\|$$

where $\|\cdot\|$ is some distance or norm. The **yuima** offers some options:

1. The L_1 norm
$$\|\hat{g}(\theta)\|_1 = \sum_{h=1}^{d} \left| \hat{g}_h(\theta) \right|.$$

2. The squared of L_2 norm
$$\|\hat{g}(\theta)\|_2^2 = \sum_{h=1}^{d} \left(\hat{g}_h(\theta) \right)^2.$$

3. The following quadratic form
$$\|\hat{g}(\theta)\|_{\mathbf{W}}^2 = \hat{g}(\theta)^{\intercal} \mathbf{W} \hat{g}(\theta) \tag{7.12}$$
where the positive definite weighting matrix $\mathbf{W}$ is chosen to obtain efficient estimators between those that belong to the class of asymptotically normal estimators.

Clearly $\|\hat{g}(\theta)\|_2^2$ is a special case of the function $\|\hat{g}(\theta)\|_{\mathbf{W}}^2$ where $\mathbf{W}$ is the identity matrix. All distances are related with the generalized method of moments (GMM) as introduced by Hansen (1982). Under some regularity conditions of Newey and McFadden (1994), the GMM estimators are consistent and **yuima** implements optimal weights (Iacus et al. 2017).

7.3.2 Lévy Distribution Estimation

Once the estimates of vector θ are obtained through the generalized method of the moments, it is possible to estimate the parameters in the Lévy distribution using the estimated increments. From

$$G_n = G_{n-1} + \sqrt{V_n}\,(L_n - L_{n-1})$$

we immediately obtain that

$$\Delta G_n \approx \sqrt{V_n}\,(\Delta L_n)$$

where $\Delta G_n = G_n - G_{n-1} = G_n^{(1)}$ and $\Delta L_n = L_n - L_{n-1}$. Taking Eq. (7.10) and noticing that $\Delta\,[LL]_n^d = [LL]_n^d - [LL]_{n-1}^d = (\Delta L_n)^2$, we obtain

$$Y_n = e^{B\Delta t} Y_{n-1} + e^{B\Delta t}\mathbf{e} V_n\,(\Delta L_n)^2 = e^{B\Delta t} Y_{n-1} + e^{B\Delta t}\mathbf{e}\,(\Delta G_n)^2\,.$$

Choosing Y_0 equal to the unconditional mean of the process Y_t, we can reconstruct its sample path form the previous recurrence equation, and through $V_n = a_0 + \mathbf{a}^\top Y_{n-1}$, we also obtain an estimate of the volatility process. At this point, we have both G_n and the estimated paths of Y_n and V_n; hence, we can estimate the Lévy increments as follows:

$$\Delta L_n = \frac{\Delta G_n}{\sqrt{V_n}}$$

and apply maximum likelihood estimation to this sequence of i.i.d. random variables.

In the next example, we try to estimate a COGARCH(1, 1) model for very high-frequency data over a long time series. The **yuima** function in this case is called `gmm` and accepts several arguments including the type objective function to optimize. We refer the reader to the manual page of the function for full details; here, we just present a simple example.

```
set.seed(123)
sampCP <- setSampling(0, 5000, 15000)
simCog11 <- simulate(Cog11, true.par=paramCP11, sampling=sampCP)
fit11 <- gmm(simCog11, start=paramCP11)
summary(fit11)

## GMM estimation
##
## Call:
## gmm(yuima = simCog11, start = paramCP11)
##
## Coefficients:
##       Estimate  Std. Error
## b1 0.05845725 0.020777149
## a1 0.03195940 0.009100367
## a0 0.75346536          NA
##
##  Log.objFun L2: -4.228533
##
##  Cogarch(1,1) model: Stationarity conditions are satisfied.
##
##  Cogarch(1,1) model: Variance process is positive.
```

```
mat <- rbind(coef(fit11), unlist(paramCP11[names(coef(fit11))]))
rownames(mat) <- c("gmm", "true")
mat
```

```
##              b1        a1        a0
## gmm  0.05845725 0.0319594 0.7534654
## true 0.05300000 0.0380000 0.7547170
```

The GMM method in this case performs adequately although we suggest the QM-LE approach for the next section. The estimated Lévy increments can be obtained specifying the argument `Est.Incr="Incr"` as we did in the previous code and subsequently extracted from the output of `gmm` accessing the slot `Incr.Lev`. In the previous example, this means typing `fit11@Incr.Lev` in the R console. As this functionality is available for the QMLE method, we will discuss this in the next section.

A strictly related approach based on prediction-based estimating functions for the COGARCH(1, 1) model can be found in Bibbona and Negri (2015), but this method is not implemented in **yuima**.

7.4 Quasi-maximum Likelihood Estimation

The quasi-maximum likelihood approach is based on a sequence of approximations of discrete stochastic processes to the COGARCH process. This method was first proposed by Maller et al. (2008) in COGARCH(1, 1) case and the extended to the COGARCH(p, q) model in Iacus et al. (2015). Let $\nu_L(y)$ be the Lévy measure of the process L_t, and such that the Lévy process is centred in zero with unitary second moment $\mu = \mathbb{E}(L_1) = 1$. Let $\tilde{B} := B + \mu \mathbf{e}\mathbf{a}^\top$, $\mu = \int_{\mathscr{R}} y^2 \mathrm{d}\nu_L(y)$. Let further

$$\Delta G_{t_i} = G_{t_i} - G_{t_{i-1}} = \int_{t_{i-1}}^{t_i} V_u \mathrm{d}L_u$$

the sequence of increments on the, possibly irregular, grid $0 = t_0 < t_1 < \ldots < t_N = T$. Then, the conditional first moment and the conditional variance of the increments of ΔG_{t_i} are respectively (Chadraa 2009):

$$\begin{aligned}&\mathbb{E}\left[\Delta G_{t_i}\left|\mathscr{F}_{t_{i-1}}\right.\right]=0\\ &\mathrm{Var}\left[\Delta G_{t_i}\left|\mathscr{F}_{t_{i-1}}\right.\right]=\mathbb{E}\left[L_1\right]\\ &\quad\times\left[\tfrac{a_0\Delta t_i b_q}{b_q-a_1\mu}+\mathbf{a}^\top e^{\tilde{B}\Delta t_i}\tilde{B}^{-1}\left(I-e^{-\tilde{B}\Delta t_i}\right)\left(Y_{t_{i-1}}-\mathbb{E}\left(Y_{t_{i-1}}\right)\right)\right]\end{aligned} \tag{7.13}$$

If the process Y_t is stationary, we have: $\mathbb{E}\left(Y_t\right)=\frac{a_0\mu}{b_q-a_1\mu}\times[1,0,\ldots,0]^\top$.

Let $G_{i,n}$ be a discrete approximation process for G_{t_i} as defined in Iacus et al. (2015), then the discretized version $Y_{i,n}$ state process Y_t takes the following form:

$$Y_{i,n}=\left(I+\frac{\left(G_{i,n}-G_{i-1,n}\right)^2}{a_0+\mathbf{a}^\top Y_{i-1,n}}\mathbf{e}\mathbf{a}^\top\right)e^{B\Delta t_{i,n}}Y_{i-1,n}+a_0\frac{\left(G_{i,n}-G_{i-1,n}\right)^2}{a_0+\mathbf{a}^\top Y_{i-1,n}}\mathbf{e}. \tag{7.14}$$

Using (7.13) and (7.14), we obtain the pseudo-likelihood function for the case of the COGARCH(p,q) model and the QMLE estimates of the model can be obtained as the solution to the following optimization problem:

$$\max_{\mathbf{a},a_0,B\in\Theta}\mathscr{L}_N\left(\mathbf{a},a_0,B\right)$$

where Θ is the parameter space and

$$\mathscr{L}_N\left(\mathbf{a},a_0,B\right)=-\frac{1}{2}\sum_{i=1}^{N}\left(\frac{\left(\Delta G_{t_i}\right)^2}{\mathrm{Var}\left[\Delta G_{t_i}\left|\mathscr{F}_{t_{i-1}}\right.\right]}+\ln\left(\mathrm{Var}\left[\Delta G_{t_i}\left|\mathscr{F}_{t_{i-1}}\right.\right]\right)\right)-\frac{N\ln(2\pi)}{2}.$$

In the next example, we show the empirical convergence of the estimates for a VG-COGARCH(1, 1) model in the case of QMLE and GMM approaches. As stationarity and high frequency are both needed in this model, we consider the setting where T is either 1000 and the number of observations is $N=15000$; therefore, $\Delta=T/N=0.06\bar{6}$.

```
param.VG <- list(a1 = 0.038,  b1 =  0.053, a0 = 0.04 / 0.053,
  y01 = 50.33)
cog.VG <- setCogarch(p = 1, q = 1, work = FALSE,
  measure=list("rvgamma(z, 1, sqrt(2), 0, 0)"),
  measure.type = "code", XinExpr = TRUE)
samp.VG <- setSampling(Terminal = 1000, n = 15000)
set.seed(123)
sim.VG <- simulate(cog.VG, true.parameter = param.VG,
  sampling = samp.VG, method = "mixed")
fit.gmm <- gmm(sim.VG, start=param.VG)
fit.qmle <- qmle(sim.VG, start=param.VG, grideq=TRUE)
nm <- names(coef(fit.gmm))
mat <- rbind(coef(fit.gmm), coef(fit.qmle)[nm],
 unlist(param.VG[nm]))
rownames(mat) <- c("gmm", "qmle", "true")
round(mat,5)
```

```
##            b1      a1      a0
## gmm  0.01283 0.01001 0.49730
## qmle 0.04973 0.03724 0.56738
## true 0.05300 0.03800 0.75472
```

As we can see, the QMLE estimates are closer to the true parameters' values than the corresponding GMM estimates. This result is not general though.

7.5 Relationship Between GARCH(1,1) and COGARCH(1,1)

Let us consider the notation from Sect. 7.2 and the Euler approximation where the increments of the Lévy process are a sequence of i.i.d. standard Gaussian distribution. For simplicity, we denote by $\sqrt{\Delta}\epsilon_n = L_n - L_{n-1} \sim N(0, \Delta)$, $n = 0, 1, \ldots, N$, with $\Delta = T/N$, and very small Δ. Consider the discretized COGARCH(1,1) process (Maller et al. 2008; Iacus et al. 2015)

$$\begin{aligned} G_n &= G_{n-1} + \sqrt{V_{n-1}}\sqrt{\Delta}\epsilon_n \\ V_n &= a_0 + a_1 Y_n \end{aligned} \tag{7.15}$$

where

$$Y_n = C_n Y_{n-1} + D_n \tag{7.16}$$

and coefficients

$$C_n = \left(1 + \epsilon_n^2 \Delta a_1\right) e^{-b_1 \Delta}, \quad D_n = a_0 \epsilon_n^2 \Delta,$$

therefore

$$Y_n = \frac{V_n - \alpha_0}{a_1}$$

and, by using (7.16), we can write

$$\frac{V_n - \alpha_0}{a_1} = \left(1 + \epsilon_n^2 \Delta a_1\right) e^{-b_1 \Delta} \frac{V_{n-1} - \alpha_0}{a_1} + \alpha_0 \epsilon_n^2 \Delta.$$

Now, using a bit of algebra, we get

$$V_n = a_0 b_1 \Delta + a_1 \Delta \epsilon_n^2 V_{n-1} + V_{n-1} e^{-b_1 \Delta} + o(\Delta)$$

and finally we can rewrite the whole system (7.15) in this GARCH(1,1) form

$$\begin{aligned} G_n &= G_{n-1} + \sigma_n \epsilon_n \\ \sigma_n^2 &= \omega + \alpha \sigma_{n-1}^2 \epsilon_{n-1}^2 + \beta \sigma_{n-1}^2 \end{aligned}$$

with

$$\begin{aligned} \sigma_n &= \sqrt{V_n}\sqrt{\Delta} \\ \omega &= a_0 b_1 \Delta^2 \\ \beta &= e^{-b_1\Delta} \\ \alpha &= a_1\Delta \end{aligned} \tag{7.17}$$

7.6 Application to Real Data

We now see an example, adapted from Bianchi et al. (2016), where this relationship is also checked on the estimates from real data for these two models. We download the data on Next Plc, and we try to fit to the de-trended log-returns (see Fig. 7.4) a GARCH(1,1) using the **rugarch** package (Ghalanos 2015) and later a COGARCH(1,1) by **yuima**.

```
require(quantmod)
getSymbols("NXT.L",  to="2016-12-31")

## [1] "NXT.L"

S <- NXT.L$NXT.L.Close
X <- na.omit(diff(log(S)))
mX <- mean(X)
X <- X - mX
plot(X,  main="Log-returns of NEXT Plc")
require(rugarch)
spec <- ugarchspec(variance.model =
 list(model = "sGARCH", garchOrder = c(1, 1)),
 mean.model = list(armaOrder = c(0, 0), include.mean = FALSE))
```

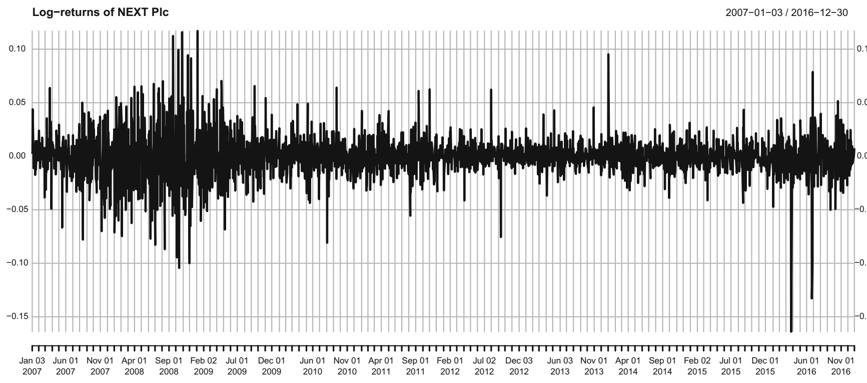


Fig. 7.4 Log-returns of the stock Next Plc exhibit volatility clustering effect

```
fitGARCH <- ugarchfit(data = X, spec = spec)
GARCH11param <- coef(fitGARCH)
GARCH11param

##        omega       alpha1        beta1
## 1.913350e-06 2.524051e-02 9.690955e-01
```

Now, we construct a function to convert the parameters from a GARCH(1,1) to a COGARCH(1,1) model according to formula (7.17). This function is similar to the implementation in the package **COGARCH.rm** (Bianchi et al. 2017).

```
Delta <- 1/252
ParGarToCog<- function(GARCH11param, dt, names=NULL){
    if(is.null(names))
     names <- names(GARCH11param)
    my.omega <- GARCH11param["omega"]
    my.alpha <- GARCH11param["alpha1"]
    my.beta <- GARCH11param["beta1"]
    a1 <- my.alpha/dt
    b1 <- -log(my.beta)/dt
    a0 <- my.omega/(b1*dt^2)
    qmleparInGARCH <- c(a0, a1, b1)
    names(qmleparInGARCH) <- c("a0", "a1", "b1")
    return(qmleparInGARCH)
}
```

Now, we use the converted values as initial point of the `qmle` optimizer

```
ParGarToCog(GARCH11param, Delta)

##         a0         a1         b1
## 0.01535941 6.36060885 7.91080657

start <- as.list(ParGarToCog(GARCH11param, Delta))

modCog11 <- setCogarch(p=1, q=1, measure =
 list(intensity="1", df=list("dnorm(z, 0, 1)")), measure.type="CP")
NXT.data <- setData(cumsum(X), delta = Delta)
Cog11 <- setYuima(data = NXT.data, model = modCog11)
Cog11.fit <- qmle(yuima = Cog11, grideq=TRUE,
 start = c(start, y1 = 0.1),
 aggregation = FALSE, method = "Nelder-Mead")
COGARCH11par <- coef(Cog11.fit)
COGARCH11par

##         a0         a1         b1
## 0.01526955 6.31562511 7.73531392
```

and we apply the reverse transform from COGARCH(1,1) to GARCH(1,1)

```
ParCogToGar<- function(COGARCH11param, dt, names=NULL){
    a0 <- COGARCH11param["a0"]
    a1 <- COGARCH11param["a1"]
    b1 <- COGARCH11param["b1"]
    my.omega <- a0*b1*dt^2
    my.alpha <- a1*dt
    my.beta <- exp(-b1*dt)
    qmleparInGARCH <- c(my.omega, my.alpha, my.beta)
    names(qmleparInGARCH) <- c("omega", "alpha1", "beta1")
    return(qmleparInGARCH)
}
ParCogToGar(COGARCH11par, Delta)

##        omega       alpha1        beta1
## 1.859958e-06 2.506200e-02 9.697706e-01

GARCH11param

##        omega       alpha1        beta1
## 1.913350e-06 2.524051e-02 9.690955e-01
```

As seen, the estimated parameters are very close for the two models after reparametrization.

References

Abramowitz, M., & Stegun, I. (1964). *Handbook of mathematical functions*. New York: Dover Publications.

Aït-Sahalia, Y. (1996). Testing continuous-time models of the spot interest rate. *The Review of Financial Studies*, *70*(2), 385–426.

Ait-Sahalia, Y., Fan, J., & Xiu, D. (2010). High-frequency covariance estimates with noisy and asynchronous financial data. *Journal of the American Statistical Association*, *105*(492), 1504–1517.

Akaike, H. (1973). Information theory and an extension of the maximum likelihood principle. In B. Petrov & F. Caski (Eds.), *Selected Papers of Hirotugu Akaike, Proceedings of the Second International Symposium on Information Theory* (pp. 267–281). Budapest: Akademiai Kiado.

Amari, S. (1985). *Differential-geometrical methods in statistics* (Vol. 28)., Lecture notes in statistics New York: Springer.

Applebaum, D. (2004). *Lévy processes and stochastic calculus*. Cambridge: Cambridge University Press.

Barndorff-Nielsen, O. (1977). Exponentially decreasing distributions for the logarithm of particle size. *Royal Society of London Proceedings Series A*, *353*.

Barndorff-Nielsen, O. (1978). Hyperbolic distributions and distributions on hyperbolae. *Scandinavian Journal of Statistics*, *5*, 151–157.

Barndorff-Nielsen, O., & Halgreen, C. (1977). Infinite divisibility of the hyperbolic and generalized inverse Gaussian distributions. *Probability Theory and Related Fields*, *38*(4), 309–311.

Barndorff-Nielsen, O. E. (1997). Processes of normal inverse Gaussian type. *Finance and Stochastics*, *2*(1), 41–68.

Barndorff-Nielsen, O. E., & Shephard, N. (2001a). Non-Gaussian ornstein-uhlenbeck-based models and some of their uses in financial economics. *Journal of the Royal Statistical Society Series B*, *63*(2), 167–241.

Barndorff-Nielsen, O. E. & Shephard, N. (2001b). Normal modified stable processes. Economics Papers 2001-W6, Economics Group, Nuffield College, University of Oxford.

Barndorff-Nielsen, O. E., Hansen, P. R., Lunde, A., & Shephard, N. (2011). Multivariate realised kernels: Consistent positive semi-definite estimators of the covariation of equity prices with noise and non-synchronous trading. *Journal of Econometrics*, *162*, 149–169.

Beckers, S. (1980). The constant elasticity of variance model and its implications in option pricing. *The Journal of Finance*, *35*(3), 661–673.

Beran, R. (1977). Minimum Hellinger estimates for parametric models. *Annals of Statistics*, *5*, 445–463.

Bertin, K., Torres, S., & Tudor, C. (2011). Drift parameter estimation in fractional diffusions driven by perturbed random walk. *Statistic and Probability Letters*, *81*(2), 243–249.

Berzin, C., & Leon, J. (2008). Estimation in models driven by fractional Brownian motion. *Annales de l'Institut Henri Poincaré*, *44*(2), 191–213.

Bianchi, F., Mercuri, L., & Rroji, E. (2016). Measuring risk with cogarch(p,q) models. In *SSRN*.

Bianchi, F., Mercuri, L., & Rroji, E. (2017). COGARCH.rm: Portfolio selection with multivariate COGARCH(p,q) models. R package version 0.1.0/r3.

Bibbona, E., & Negri, I. (2015). Higher moments and prediction-based estimation for the cogarch(1,1) model. *Scandinavian Journal of Statistics*, *42*(4), 891–910.

Bibby, B. M., & Sørensen, M. (1997). A hyperbolic diffusion model for stock prices. *Finance and Stochastics*, *1*, 25–41.

Bibinger, M. (2011). Efficient covariance estimation for asynchronous noisy high-frequency data. *Scandinavian Journal of Statistics*, *38*, 23–45.

Black, F., & Scholes, M. (1973). The pricing of options and corporate liabilities. *Journal of Political Economy*, *81*, 637–654.

Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, *31*(3), 307–327.

Brandt, A. (1986). The stochastic equation $y_n + 1 = a_n y_n + b_n$ with stationary coefficients. *Advances in Applied Probability*, *18*(1), 211–220.

Bregni, S., & Erangoli, W. (2005). Fractional noise in experimental measurements of IP traffic in a metropolitan area network. *Proceedings of IEEE GlobeCom*, *2005*(2), 781–785.

Brennan, M., & Schwartz, E. (1980). Analyzing convertible securities. *Journal of Financial and Quantitative Analysis*, *15*(4), 907–929.

Brockwell, P. (2001). Lévy-driven CARMA processes. *Annals of the Institute of Statistical Mathematics*, *53*(1), 113–124.

Brockwell, P., & Marquardt, T. (2005). Lévy-driven and fractionally integrated arma processes with continuous time parameter. *Statistica Sinica*, *15*(2), 477–494.

Brockwell, P., Chadraa, E., & Lindner, A. (2006). Continuous-time GARCH processes. *Annals of Applied Probability*, *16*(2), 790–826.

Brockwell, P. J., & Schlemm, E. (2013). Parametric estimation of the driving Lévy process of multivariate CARMA processes from discrete observations. *Journal of Multivariate Analysis*, *115*, 217–251.

Brockwell, P. J., Davis, R. A., & Yang, Y. (2007). Estimation for non-negative Lévy-driven Ornstein–Uhlenbeck processes. *Journal of Applied Probability*, *44*, 987–989.

Brockwell, P. J., Davis, R. A., & Yang, Y. (2011). Estimation for non-negative Lévy-driven CARMA processes. *Journal of Business and Economic Statistics*, *29*(2), 250–259.

Brouste, A., & Iacus, S. (2013). Parameter estimation for the discretely observed fractional Ornstein–Uhlenbeck process and the yuima R package. *Computational Statistics*, *28*(4), 1529–1547.

Brouste, A., Fukasawa, M., Hino, H., Iacus, S. M., Kamatani, K., Koike, Y., et al. (2014). The yuima project: A computational framework for simulation and inference of stochastic differential equations. *Journal of Statistical Software*, *57*(4), 1–51.

Carr, P., Geman, H., Madan, D., & Yor, M. (2002). The fine structure of asset returns: An empirical investigation. *The Journal of Business*, *75*, 305–332.

Chadraa, E. (2009). Statistical modelling with COGARCH(P,Q) processes. Ph.D. thesis, Colorado State University.

Chambers, J. M. (1998). *Programming with data: A guide to the S language*. New York: Springer.

Chan, K., Karolyi, G., Longstaff, F., & Sanders, A. (1992). An empirical investigation of alternative models of the short-term interest rate. *The Journal of Finance*, *47*, 1209–1227.

Chan, K.-S., & Ripley, B. (2012). TSA: Time series analysis. R package version 1.01.

Chen, S., Gao, J., & Cheng, Y. (2008). A test for model specification of diffusion processes. *Annals of Statistics*, *36*(1), 167–198.

Cheridito, P., Kawaguchi, H., & Maejima, M. (2003). Fractional Ornstein–Uhlenbeck processes. *Electronic Journal of Probability*, *8*(3), 1–14.

Chhikara, R. S., & Folks, J. L. (1989). *The inverse Gaussian distribution: Theory, methodology, and applications*. New York, NY, USA: Marcel Dekker Inc.

Chiao, C., Hung, K., & Lee, C. (2004). The price adjustment and lead-lag relations between stock returns: Microstructure evidence from the Taiwan stock market. *Empirical Finance*, *11*, 709–731.

Christensen, K., Kinnebrock, S., & Podolskij, M. (2010). Pre-averaging estimators of the ex-post covariance matrix in noisy diffusion models with non-synchronous data. *Journal of Econometrics*, *159*, 116–133.

Çınlar, E. (2011). *Probability and stochastics*., Graduate texts in mathematics New York: Springer.

Comte, F., & Renaut, E. (1996). Non-causality in continuous time models. *Econometric Theory*, *12*, 215–256.

Cont, R., & Tankov, P. (2004). *Financial modelling with jump processes*. Boca Raton: Chapman & Hall/CRC.

Cox, J. (1996). The constant elasticity of variance option pricing model. *The Journal of Portfolio Management*, *3*, 15–17.

Cox, J., Ingersoll, J., & Ross, S. (1980). An analysis of variable rate loan contracts. *The Journal of Finance*, *35*(2), 389–403.

Cox, J., Ingersoll, J., & Ross, S. (1985). A theory of the term structure of interest rates. *Econometrica*, *53*, 385–408.

Cressie, N., & Read, T. (1984). Multinomial goodness of fit tests. *Journal of the Royal Statistical Society Series B*, *46*, 440–464.

Crouse, M. S., & Baraniuk, R. G. (1999). Fast, exact synthesis of Gaussian and NonGaussian long-range-dependent processes. *IEEE Transactions on Information Theory*, *X*, X.

Csiszár, I. (1967). On topological properties of f-divergences. *Studia Scientific Mathematicae Hungarian*, *2*, 329–339.

Dachian, S., & Kutoyants, Y. A. (2008). On the goodness-of-fit tests for some continuous time processes. In F. Vonta, M. Nikulin, N. Limnios, & C. Huber-Carol (Eds.), *Statistical models and methods for biomedical and technical systems* (pp. 395–413). Boston: Birkhuser.

Dalgaard, P. (2008). *Introductory statistics with R* (2nd ed.). New York: Springer.

Daubechies, I. (1992). *Ten lectures on wavelets*. SIAM.

De Gregorio, A., & Iacus, S. M. (2012). Adaptive lasso-type estimation for ergodic diffusion processes. *Econometric Theory*, *28*, 1–23.

De Gregorio, A., & Iacus, S. (2013). On a family of test statistics for discretely observed diffusion processes. *Journal of Multivariate Analysis*, *122*, 292–316.

de Jong, F., & Nijman, T. (1997). High frequency analysis of lead-lag relationships between financial markets. *Journal of Empirical Finance*, *4*, 259–277.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, *39*(1), 1–38.

Dimitris, K. (2002). An EM type algorithm for maximum likelihood estimation of the normal-inverse Gaussian distribution. *Statistics & Probability Letters*, *57*(1), 43–52.

Doob, J. (1944). The elementary Gaussian process. *Annals of Mathematical Statistics*, *15*(3), 229–282.

Dothan, U. (1978). On the term structure of interest rates. *Journal of Financial Economics*, *6*, 59–69.

Eberlein, E. (2001). *Application of generalized hyperbolic Lévy motions to finance* (pp. 319–336). Boston, MA: Birkhäuser.

Eberlein, E., & Keller, U. (1995). Hyperbolic distributions in finance. *Bernoulli*, *1*, 281–299.

Eberlein, E., & von Hammerstein, E. A. (2004). *Generalized hyperbolic and inverse Gaussian distributions: Limiting cases and approximation of processes* (pp. 221–264). Basel: Birkhäuser.

Efron, B., Hastie, T., Johnstone, I., & Tibshirani, R. (2004). Least angle regression. *The Annals of Statistics*, *32*, 407–489.

Feller, W. (1951a). Diffusion processes in genetics. In J. Neyman (Ed.), *Proceedings of the 2nd Berkeley Symposium on Mathematical Statistics and Probability* (Vol. 54, pp. 227–246). Berkeley: University of California Press.

Feller, W. (1951b). Two singular diffusion problems. *Annals of Mathematics*, *54*, 173–182.

Fukasawa, M. (2011). Discretization error of stochastic integrals. *The Annals of Applied Probability*, *21*, 1436–1465.

Genon-Catalot, V., & Jacod, J. (1993). On the estimation of the diffusion coefficient for multi-dimensional diffusion processes. *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, *29*, 119–151.

Ghalanos, A. (2015). rugarch: Univariate GARCH models. R package version 1.3-6.

Giet, L., & Lubrano, M. (2008). A minimum Hellinger distance estimator for stochastic differential equations: An application to statistical inference for continuous time interest rate models. *Computational Statistics and Data Analysis*, *52*(6), 2945–2965.

Graybill, D., & Shiyatov, S. (1992). Dendroclimatic evidence from the northern Soviet Union. In R. S. Bradley & P. D. Jones (Eds.), *Climate since A.D. 1500*. London: Routledge.

Hansen, L. (1982). Large sample properties of generalized method of moments estimators. *Econometrica*, *50*(4), 1029–1054.

Hawkes, A. G. (1971). Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, *58*.

Hayashi, T., & Yoshida, N. (2005). On covariance estimation of non-synchronously observed diffusion processes. *Bernoulli*, *11*, 359–379.

Hayashi, T. & Yoshida, N. (2006). Nonsynchronous covariance estimator and limit theorem. *Institute of Statistical Mathematics*, Research Memorandum No. 1020, 1–40.

Hayashi, T., & Yoshida, N. (2008a). Asymptotic normality of a covariance estimator for nonsynchronously observed diffusion processes. *Annals of the Institute of Statistical Mathematics*, *60*, 367–406.

Hayashi, T. & Yoshida, N. (2008b). Nonsynchronous covariance estimator and limit theorem ii. *Institute of Statistical Mathematics*, Research Memorandum No. 1067, 1–40.

Hinde, J. (1982). Compound Poisson regression models. In *GLIM 82: Proceedings of the International Conference on Generalised Linear Models*. New York: Springer.

Hoffmann, M., Rosenbaum, M., & Yoshida, N. (2013). Estimation of the lead-lag parameter from non-synchronous data. *Bernoulli*, *19*(2), 426–461.

Hu, Y., & Nualart, D. (2010). Parameter estimation for fractional Ornstein–Uhlenbeck processes. *Statistics and Probability Letters*, *80*(11–12), 1030–1038.

Hu, Y., Nualart, D., Xiao, W., & Zhang, W. (2011). Exact maximum likelihood estimator for drift fractional brownian motion at discrete time. *Acta Mathematica Scientia*, *31*(5), 1851–1859.

Iacus, S. M. (2008). *Simulation and inference for stochastic differential equations: With R examples*. New York: Springer.

Iacus, S. M., & Yoshida, N. (2012). Estimation for the change point of the volatility in a stochastic differential equation. *Stochastic Processes and Their Applications*, *122*, 1068–1092.

Iacus, S. M. & Mercuri, L. (2014). Estimation of Lévy CARMA models in the yuima package: Application on the financial time series. In *COMPSTAT 2014 21st International Conference on Computational Statistics* (pp. 451–458).

Iacus, S. M., & Mercuri, L. (2015). Implementation of Lévy CARMA model in yuima package. *Computational Statistics*, *30*(4), 1111–1141.

Iacus, S. M., Mercuri, L., & Rroji, E. (2015). Discrete time approximation of a COGARCH(p,q) model and its estimation.

Iacus, S. M., Mercuri, L., & Rroji, E. (2017). Cogarch(p,q): Simulation and inference with `yuima` package. Accepted for publication on *Journal of Statistical Software* **80**, 1–49.

Istas, J., & Lang, G. (1997). Quadratic variations and estimation of the local Hölder index of a Gaussian process. *Annales de l'Institut Henri Poincaré*, *23*(4), 407–436.

Itô, K. (2013). *Stochastic processes: Lectures given at Aarhus university*. Berlin, Heidelberg: Springer.

Kang, J., Lee, C., & Lee, S. (2006). Empirical investigation of the lead-lag relations of returns and volatilities among the KOSPI200 spot, futures and options markets and their explanations. *Journal of Emerging Market Finance*, *5*, 235–261.

Kawai, R., & Masuda, H. (2011). On simulation of tempered stable random variates. *Journal of Computational and Applied Mathematics*, *235*(8), 2873–2887.

Kesten, H. (1973). Random difference equations and renewal theory for products of random matrices. *Acta Mathematica*, *131*(1), 207–248.

Klüppelberg, C., Lindner, A., & Maller, R. (2004). A continuous-time garch process driven by a Lévy process: Stationarity and second-order behaviour. *Journal of Applied Probability*, *41*(3), 601–622.

Knight, K., & Fu, W. (2000). Asymptotics for Lasso-type estimators. *The Annals of Statistics*, *28*(5), 1356–1378.

Koike, Y. (2014). An estimator for the cumulative co-volatility of asynchronously observed semimartingales with jumps. *Scandinavian Journal of Statistics*, *41*(2), 460–481.

Kolmogorov, A. (1940). Winersche Spiralen und einige andere interessante Kurven in Hilbertschen Raum. *Academic Science USSR*, *26*, 115–118.

Koponen, I. (1995). Analytics approach to the problem of convergence of truncated Lévy flights towards the Gaussian stochastic process. *Physics Review E*, *52*, 1197–1199.

Küchler, U., & Tappe, S. (2008a). Bilateral gamma distributions and processes in financial mathematics. *Stochastic Processes and Their Applications*, *118*(2), 261–283.

Küchler, U., & Tappe, S. (2008b). On the shapes of bilateral gamma densities. *Statistics & Probability Letters*, *78*(15), 2478–2484.

Küchler, U., Neumann, K., Sørensen, M., & Streller, A. (1999). Stock returns and hyperbolic distributions. *Mathematical and Computer Modelling*, *29*, 1–15.

Kunitomo, N., & Takahashi, A. (2001). The asymptotic expansion approach to the valuation of interest rate contingent claims. *Mathematical Finance*, *11*(1), 117–151.

Kunitomo, N. & Sato, S. (2008). Separating information maximum likelihood estimation of realized volatility and covariance with micro-market noise. CIRJE Discussion papers CIRJE-F-581, University of Tokyo.

Kutoyants, Y. (1994). *Identification of dynamical systems with small noise*. Dordrecht: Kluwer.

Kutoyants, Y. (1998). *Statistical inference for spatial Poisson processes*., Lecture notes in statistics New York: Springer.

Kutoyants, Y. (2004). *Statistical inference for Ergodic diffusion processes*. London: Springer.

Kyprianou, A., Schoutens, A., & Wilmott, P. (2005). *Exotic option pricing and advanced Lévy models*. Chichester: Wiley.

Lee, S., & Wee, I. (2008). Residual emprical process for diffusion processes. *Journal of Korean Mathematical Society*, *45*(3), 683–693.

Levendorskii, S., & Boyarchenko, S. (2002). *Non-Gaussian Merton–Black–Scholes theory*. Singapore: World Scientific.

Levy, E. (1992). Pricing European average rate currency options. *Journal of International Money and Finance*, *11*, 474–491.

Lewis, A., & Shedler, G. (1979). Simulation of nonhomogeneous poisson processes by thinning. *naval Research Logistics Quarterly*, *26*(3), 403–413.

Liese, F., & Vajda, I. (1987). *Convex statistical distances*. Leipzig: Tuebner.

Loregian, A., Mercuri, L., & Rroji, E. (2012). Approximation of the variance gamma model with a finite mixture of normals. *Statistics & Probability Letters*, *82*(2), 217–224.

Ludena, C. (2004). Minimum contrast estimation for fractional diffusion. *Scandinavian Journal of Statistics*, *31*, 613–628.

Madan, D. B., & Seneta, E. (1990a). The variance gamma (v.g.) model for share market returns. *Journal of Business*, *64*(4), 511–524.

Madan, D. B., & Seneta, E. (1990b). The variance gamma (v.g.) model for share market returns. *The Journal of Business*, *63*(4), 511–524.

Madan, D., Carr, P., & Change, E. (1998). The variance gamma process and option pricing. *European Finance Review*, *2*, 79–105.

Maller, R., Müller, G., & Szimayer, A. (2008). Garch modelling in continuous time for irregularly spaced time series data. *Bernoulli*, *14*, 519–542.

Mancini, C., & Gobbi, F. (2012). Identifying the brownian covariation from the co-jumps given discrete observations. *Econometric Theory*, *28*, 249–273.

Mandelbrot, B., & Ness, J. V. (1968). Fractional Brownian motions, fractional noises and application. *SIAM Review*, *10*, 422–437.

Masuda, H. (2002). Analytical properties of GIG and GH distributions. *Proceedings of the Institute of Statistical Mathematics*, *50*(2), 165–199.

Merton, R. C. (1973a). Theory of rational option pricing. *Bell Journal of Economics and Management Science*, *4*(1), 141–183.

Merton, R. C. (1973b). Theory of rational option pricing. *Bell Journal of Economics and Management Science*, *4*(1), 141–183.

Michael, J. R., Schucany, W. R., & Haas, R. W. (1976). Generating random variates using transformations with multiple roots. *The American Statistician*, *30*(2), 88–90.

Mishura, Y., & Shevchenko, G. (2008). The rate of convergence for Euler approximations of solutions of stochastic differential equations driven by fractional Brownian motion. *Stochastics*, *80*, 489–511.

Morales, D., Pardo, L., & Vajda, I. (1997). Some new statistics for testing hypotheses in parametric models. *Journal of Multivariate Analysis*, *67*, 137–168.

Neuenkirch, A., & Nourdin, I. (2007). Exact rate of convergence of some approximation schemes associated to SDEs driven by a fractional Brownian motion. *Journal of Theoretical Probability*, *20*(4), 871–899.

Neuenkirch, A. & Tindel, S. (2011). A least square-type procedure for parameter estimation in stochastic differential equations with additive fractional noise.

Newey, W. K., & McFadden, D. (1994). Large sample estimation and hypothesis testing. *Handbook of Econometrics*, *4*, 2111–2245.

Ogata, Y. (1981). On Lewis' simulation method for point processes. *IEEE Transactions on Information Theory*, *27*(1), 23–31.

Ogihara, T., & Yoshida, N. (2011). Quasi-likelihood analysis for the stochastic differential equation with jumps. *Statistical Inference for Stochastic Processes*, *14*(3), 189.

Ogihara, T., & Yoshida, N. (2014). Quasi-likelihood analysis for nonsynchronously observed diffusion processes. *Stochastic Processes and Their Applications*, *124*(9), 2954–3008.

Osborne, M. (1959). Brownian motion in the stock market. *Operations Research*, *7*, 145–173.

Pardo, L. (2006). *Statistical inference based on divergence measures*. London: Chapman & Hall/CRC.

Protter, P. (1990). *Stochastic integration and differential equations*. New York: Springer.

R Core Team. (2017). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing.

Rényi, A. (1961). On measures of entropy and information. *Proceedings of the Fourth Berkeley Symposium on Probability and Mathematical Statistics, University of California, Berkeley* (Vol. 1, pp. 547–561).

Robert, C., & Rosenbaum, M. (2011). A new approach for the dynamics of ultra high frequency data: The model with uncertainty zones. *Journal of Financial Econometrics*, *9*, 344–366.

Ryan, J. A. (2013). quantmod: Quantitative financial modelling framework. R package version 0.4-0.

Ryan, J. A. & Ulrich, J. M. (2014). xts: eXtensible time series. R package version 0.9-7.

Ryder, T. H. (1999). Generalized hyperbolic diffusion processes with applications in finance. *Mathematical Finance*, *9*(2), 183–201.

Sakamoto, Y., Ishiguro, M., & Kitagawa, G. (1986). *Akaike information criterion statistics*. Dordrecht: D. Reidel Publishing Company.

Sato, K. (1999). *Lévy processes and infinitely divisible distributions*. Cambridge: Cambridge University Press.

Schlemm, E., & Stelzer, R. (2012). Quasi maximum likelihood estimation for strongly mixing state space models and multivariate Lévy-driven CARMA processes. *Electronic Journal of Statistics [electronic only]*, *6*, 2185–2234.

Schoutens, W. (2003). *Lévy processes in finance*. Chichester: Wiley.

Schroder, M. (1989). Computing the constant elasticity of variance option pricing formula. *The Journal of Finance*, *44*(1), 211–219.

Seneta, E. (2007). *The early years of the variance-gamma process* (pp. 3–19). Boston, MA: Birkhäuser.

Shimizu, Y., & Yoshida, N. (2006). Estimation of parameters for diffusion processes with jumps from discrete observations. *Statistical Inference for Stochastic Processes*, *9*(3), 227–277.

Simpson, D. (1989). Hellinger deviance tests: Efficiency, breakdown points, and examples. *Journal of the American Statistical Association*, *84*, 107–113.

Takahashi, A. (1999). An asymptotic expansion approach to pricing financial contingent claims. *Asia-Pacific Financial Markets*, *6*(2), 115–151.

Tibshirani, R. (1996). Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society Series B*, *58*, 267–288.

Todorov, V. (2011). Econometric analysis of jump-driven stochastic volatility models. *Journal of Econometrics*, *160*(1), 12–21.

Todorov, V., & Tauchen, G. (2006). Simulation methods for Lévy-driven continuous-time autoregressive moving average (CARMA) stochastic volatility models. *Journal of Business and Economic Statistics*, *24*, 455–469.

Tómasson, H. (2013). Some computational aspects of Gaussian CARMA modelling. *Statistics and Computing*, 1–13.

Trapletti, A. & Hornik, K. (2013). tseries: Time series analysis and computational finance. R package version 0.10-32.

Tsai, H., & Chan, K. S. (2005). A note on non-negative continuous time processes. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *67*(4), 589–597.

Uchida, M. (2010). Contrast-based information criterion for ergodic diffusion processes from discrete observations. *Annals of the Institute of Statistical Mathematics*, *62*, 161–187.

Uchida, M., & Yoshida, N. (2012). Adaptive estimation of an ergodic diffusion process based on sampled data. *Stochastic Processes and Their Applications*, *122*, 2885–2924.

Uchida, M., & Yoshida, N. (2013). Quasi likelihood analysis of volatility and nondegeneracy of statistical random field. *Stochastic Processes and Their Applications*, *123*(7), 2851–2876. A Special Issue on the Occasion of the 2013 International Year of Statistics.

Uhlenbeck, G., & Ornstein, L. (1930). On the theory of Brownian motion. *Physical Review*, *36*, 823–841.

Vasicek, O. (1977). An equilibrium characterization of the term structure. *Journal of Financial Economics*, *5*, 177–188.

Watanabe, S. (1987). Analysis of Wiener functionals (Malliavin Calculus) and its applications to heat kernels. *The Annals of Probability*, *15*, 1–39.

Willinger, W., Taqqu, M., Leland, W., & Wilson, D. (1995). Self-similarity in high-speed packet traffic: Analysis and modeling of ethernet traffic measurements. *Statistical Science*, *10*, 67–85.

Wood, A., & Chan, G. (1994). Simulation of stationary Gaussian processes. *Journal of Computational and Graphical Statistics*, *3*(4), 409–432.

Wuertz, D. (2012). fExoticOptions: Exotic option valuation. *R Package Version*, *2152*, 78.

Wuertz, D., & Chalabi, Y. (2013). timeSeries: Rmetrics - financial time series objects. *R Package Version*, *3010*, 97.

Wuertz, D., Chalabi, Y., with contributions from Byers, J. W. Maechler, M., & others. (2013). timeDate: Rmetrics - chronological and calendar objects. R package version 3010.98.

Wuertz, D., & many others. (2013). fImport: Rmetrics - economic and financial data import. R package version 3000.82.

Xiao, W., Zhang, W., & Xu, W. (2011). Parameter estimation for fractional Ornstein–Uhlenbeck processes at discrete observation. *Applied Mathematical Modelling*, *35*, 4196–4207.

Yoshida, N. (1992a). Asymptotic expansion for statistics related to small diffusions. *Journal of the Japan Statistical Society*, *22*, 139–159.

Yoshida, N. (1992b). Estimation for diffusion processes from discrete observation. *Journal of Multivariate Analysis*, *41*(2), 220–242.

Yoshida, N. (2011). Polynomial type large deviation inequalities and quasi-likelihood analysis for stochastic differential equations. *Annals of the Institute of Statistical Mathematics*, *63*, 431–479.

Zeileis, A., & Grothendieck, G. (2005). zoo: S3 infrastructure for regular and irregular time series. *Journal of Statistical Software*, *14*(6), 1–27.

Zhang, L. (2011). Estimating covariation: Epps effect, microstructure noise. *Journal of Econometrics*, *160*, 33–47.

Zou, H. (2006). The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, *101*(476), 1418–1429.

Index