Overview of the 'yuima' and 'yuimaGUI' R packages

Emanuele Guidotti

Contents

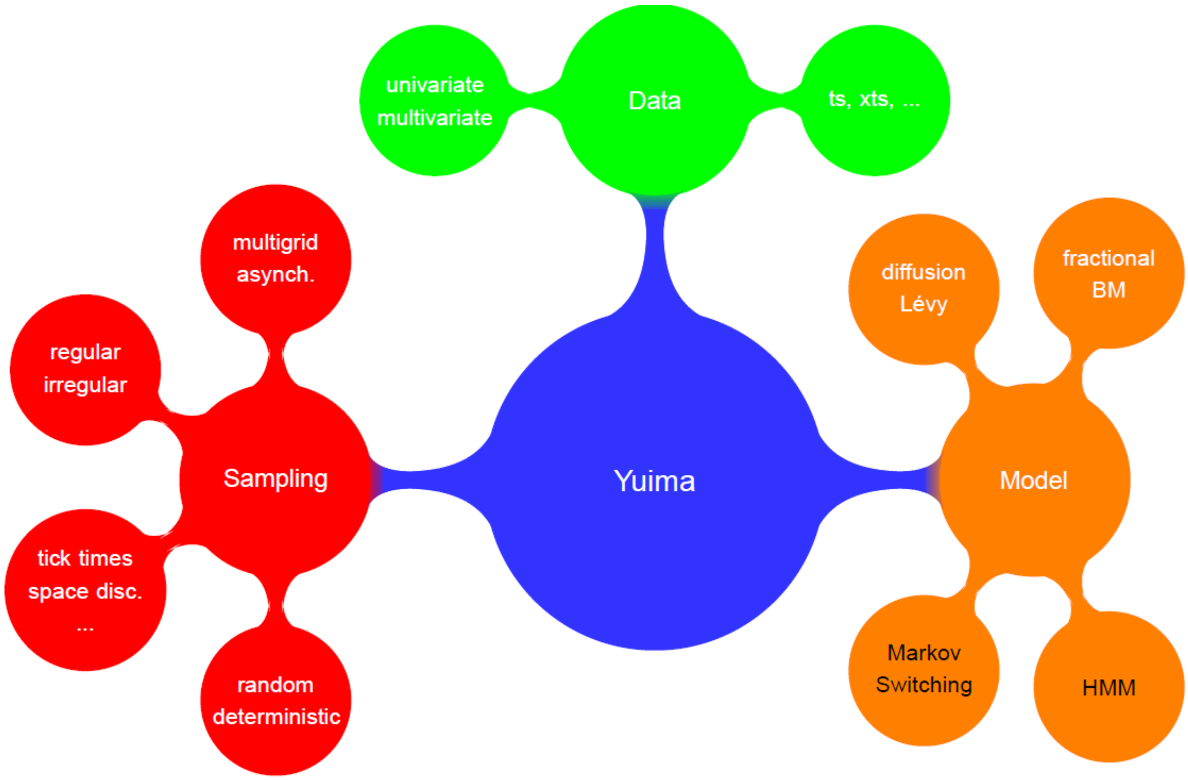
The Yuima Project aims at implementing, via the yuima package, a very abstract framework to describe probabilistic and statistical properties of stochastic processes in a way which is the closest as possible to their mathematical counterparts but also computationally efficient.

```r
# install the package
install.packages('yuima')
```

```r
# load the package
require(yuima)
```

The YUIMA Object

The main object is the `yuima` object which allows to describe the **model** in a mathematically sound way. Then the **data** and the **sampling** structure can be included as well for estimation and simulation purposes.



Model

The 'setModel' function

The `setModel()` function defines a stochastic differential equation with or without jumps of the following form:

$$dX_t = a(t, X_t, \alpha)dt + b(t, X_t, \beta)dW_t^H + c(t, X_t, \gamma)dZ_t$$

where

- $a(t, X_t, \alpha)$ is the drift term. Described by the `drift` argument
- $b(t, X_t, \beta)$ is the diffusion term. Described by the `diffusion` argument
- $c(t, X_t, \gamma)$ is the jump term. Described by the `jump.coeff` argument
- H is the Hurst coefficient. Described by the `hurst` argument
- Z_t is the Levy noise. Described by the `measure.type` and `measure` arguments

Deterministic Model

$$dU_t = \sin(\alpha t)dt$$

```
setModel(drift = "sin(alpha*t)", # the drift term
         solve.variable = "u",    # the solve variable
         time.variable = "t")     # the time variable
```

Geometric Brownian Motion

$$dX_t = \mu X_t\, dt + \sigma X_t\, dW_t$$

```r
setModel(drift = "mu*x",          # the drift term
         diffusion = "sigma*x",   # the diffusion term
         solve.variable = "x")    # the solve variable
```

CKLS Model

$$dX_t = (\theta_1 + \theta_2 X_t)\ dt + \theta_3 X_t^{\theta_4}\ dW_t$$

```r
setModel(drift = "theta1+theta2*x",       # the drift term
         diffusion = "theta3*x^theta4",   # the diffusion term
         solve.variable = "x")            # the solve variable
```

2-Dimensional Diffusion with 3 Noises

$$\begin{cases} dX_t^1 = -3X_t^1\ dt + dW_t^1 + X_t^2 dW_t^3 \\ dX_t^2 = -(X_t^1 + 2X_t^2)\ dt + X_t^1 dW_t^1 + 3dW_t^2 \end{cases}$$

```r
setModel(drift = c("-3*x1","-x1-2*x2"),                                # the drift vector
         diffusion = matrix(c("1","x1","0","3","x2","0"), 2, 3),   # the diffusion matrix
         solve.variable = c("x1","x2"))                                # the solve variables
```

Fractional Ornstein-Uhlenbeck

$$dX_t = -\theta X_t\ dt + \sigma\ dW_t^H$$

```r
setModel(drift = "-theta*x",     # the drift term
         diffusion="sigma",      # the diffusion term
         hurst = NA,             # the hurst coefficient
         solve.variable = "x")   # the solve variable
```

Jump Process with Compound Poisson Measure

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

```r
setModel(drift = "-theta*x",                            # the drift term
         diffusion="sigma",                             # the diffusion term
         jump.coeff = "1",                              # the jump term
         measure.type = "CP",                           # the measure type
         measure = list(                                # the measure
          intensity = "lambda",                           # constant intensity
          df = "dnorm(z, mu_jump, sigma_jump)"            # jump density function
         ),
         solve.variable = "x")                          # the solve variable
```

The 'setPoisson' Function

Defines a generic Compound Poisson model.

Compound Poisson with constant intensity and Gaussian jumps

$$X_t = X_0 + \sum_{i=0}^{N_t} Y_i \ :\quad N_t \sim Poi\Big(\int_0^t \lambda(t)dt\Big),\quad Y_i \sim N(\mu_{jump},\ \sigma_{jump})\lambda(t) = \lambda$$

```r
setPoisson(intensity = "lambda",                                # the intensity function
           df = "dnorm(z, mean = mu_jump, sd = sigma_jump)",   # the density function
           solve.variable = "x")                                # the solve variable
```

Compound Poisson with exponentially decaying intensity and Student-t jumps

$$X_t = X_0 + \sum_{i=0}^{N_t} Y_i \ : \quad N_t \sim Poi\Big(\int_0^t \lambda(t) dt\Big), \quad Y_i \sim t(\nu_{jump},\ \mu_{jump})\lambda(t) = \alpha\ e^{-\beta t}$$

```r
setPoisson(intensity = "alpha*exp(-beta*t)",             # the intensity function
           df = "dt(z, df = nu_jump, ncp = mu_jump)",    # the density function
           solve.variable = "x")                         # the solve variable
```

The 'setCarma' Function

Defines a generic Continuous ARMA model.

Continuous ARMA(3,1) process driven by a Brownian Motion

$$CARMA(3,1)$$

```r
setCarma(p = 3,   # autoregressive coefficients
         q = 1)   # moving average coefficients
```

Continuous ARMA(3,1) process driven by a Compound Poisson with Gaussian jumps

$$CARMA(3,1)$$

```r
setCarma(p = 3,                              # autoregressive coefficients
         q = 1,                              # moving average coefficients
         measure.type = "CP",                # compound poisson
         measure = list(                     # cp measure
           intensity = "lambda",                 # intensity function
           df = "dnorm(z, 'mu', 'sigma')"        # density function
         ))
```

The 'setCogarch' Function

Defines a generic Continuous GARCH model.

Continuous COGARCH(1,1) process driven by a Compound Poisson with Gaussian jumps

$$COGARCH(1,1)$$

```r
setCogarch(p = 1,                              # autoregressive coefficients
           q = 1,                              # moving average coefficients
           measure.type = "CP",                # compound poisson
           measure = list(                     # cp measure
             intensity = "lambda",                 # intensity function
             df = "dnorm(z, 'mu', 'sigma')"        # density function
           ))
```

Data

The `setData()` function prepares the data for model estimation. The `delta` argument describes the time increment between observations. If we have monthly data and want to measure time in years, then `delta` should be 1/12. If we have daily data and want to measure time in months, then `delta` should be 1/30. If we have financial daily data and want to measure time in years, then `delta` should be 1/252, since 252 is the average number of trading days in one year. In general, if we want to measure time in unit T, `delta` should be 1 over the average number of observations in a period T. The unit of measure of time affects the estimated value of the model parameters.

The following example downloads and sets some financial data (see tutorial on Data Acquisition in R).

```
# Install the quantmod package if needed:
# install.packages('quantmod')

# load quantmod
require(quantmod)

# download Facebook quotes
fb <- getSymbols(Symbols = 'META', src = 'yahoo', auto.assign = FALSE)

# setData with time in years -> delta = 1/252
# (there are 252 observations in 1 year)
setData(fb$META.Close, delta = 1/252, t0 = 0)
```

```
##
##
## Number of original time series: 1
## length = 3276, time range [2012-05-18 ; 2025-05-29]
##
## Number of zoo time series: 1
##            length time.min time.max       delta
## META.Close   3276        0   12.996 0.003968254
```

Sampling

The `setSampling()` function describes the simulation grid. If `delta` is not specified, it is calculated as `(Terminal-Initial)/n`. If `delta` is specified, the `Terminal` is adjusted to be equal to `Initial+n*delta`.

```
# define a regular grid using delta
setSampling(Initial = 0, delta = 0.01, n = 1000)
```

```
# define a regular grid using Terminal
setSampling(Initial = 0, Terminal = 2, n = 1000)
```

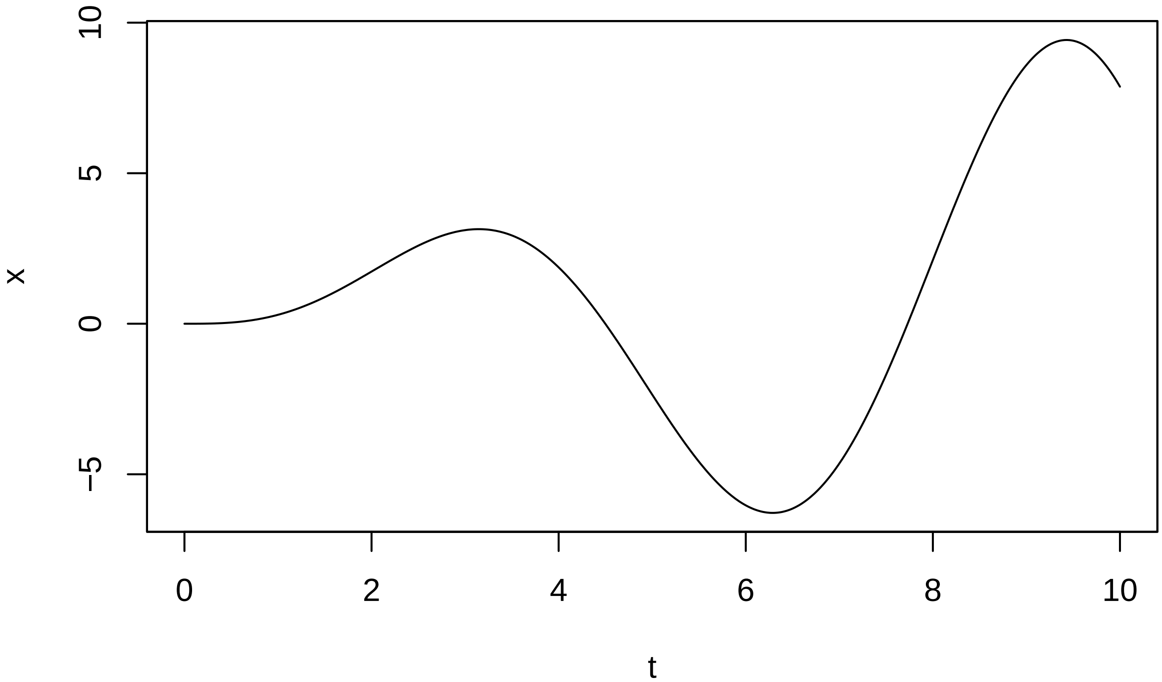
Simulation

Simulation of a generic model is perfomed with the `simulate()` function.

Example Solve an Ordinary Differential Equation

```
# model: ordinary differential equation
model <- setModel(drift = 'sin(t)*t', solve.variable = 'x', time.variable = 't')
# simulation scheme
sampling <- setSampling(Initial = 0, Terminal = 10, n = 1000)
# yuima object
yuima <- setYuima(model = model, sampling = sampling)
```

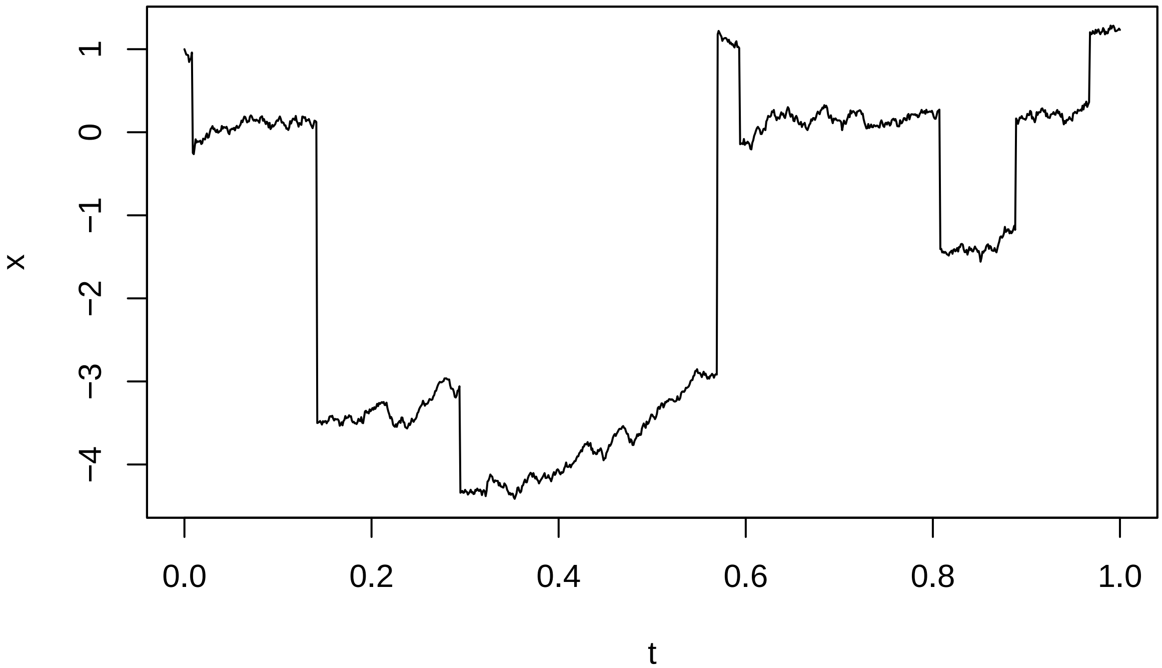
```r
# simulation
sim <- simulate(yuima)
# plot
plot(sim)
```



Example Simulate one trajectory of a jump diffusion model

```r
# model: jump diffusion
model <- setModel(drift = "-theta*x",
                  diffusion="sigma",
                  jump.coeff = "1",
                  measure.type = "CP",
                  measure = list(
                    intensity = "lambda",
                    df = "dnorm(z, mu_jump, sigma_jump)"
                  ),
                  solve.variable = "x")

# simulation scheme
sampling <- setSampling(Initial = 0, Terminal = 1, n = 1000)

# yuima object
yuima <- setYuima(model = model, sampling = sampling)

# simulation
sim <- simulate(yuima,                    # the yuima object
                xinit = 1,                # the initial value
                true.parameter = list(    # specify the parameters:
                  theta = 1,                # value for the 'theta' parameter
                  sigma = 1,                # value for the 'sigma' parameter
                  lambda = 10,              # value for the 'lambda' parameter
                  mu_jump  = 0,             # value for the 'mu_jump' parameter
```

```
                  sigma_jump = 2          # value for the 'sigma_jump' parameter
                ))

# plot
plot(sim)
```



Estimation

The 'qmle' Function

The `qmle()` function calculates the quasi-likelihood and estimate of the parameters of the stochastic differential equation by the maximum likelihood method or least squares estimator of the drift parameter.

Example Simulate a Geometric Brownian Motion and estimate its parameters

```
# model: geometric brownian motion
model <- setModel(drift = 'mu*x', diffusion = 'sigma*x', solve.variable = 'x')

# simulation scheme
sampling <- setSampling(Initial = 0, Terminal = 1, n = 1000)

# yuima object
yuima <- setYuima(model = model, sampling = sampling)

# simulation
sim <- simulate(yuima, true.parameter = list(mu = 1.3, sigma = 0.25), xinit = 100)

# estimation
estimation <- qmle(sim,                           # the yuima object
              start = list(mu = 0, sigma = 1),    # starting values for optimization
              lower = list(sigma = 0))            # lower bounds

# estimates and standard errors
```

```r
summary(estimation)
```

```
## Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = sim, start = list(mu = 0, sigma = 1), lower = list(sigma = 0))
##
## Coefficients:
##        Estimate  Std. Error
## sigma 0.2476376 0.005623821
## mu    1.1498516 0.247637569
##
## -2 log L: 3182.872
```

Example Estimate the yearly volatility (σ in the Geometric Brownian Motion) of Google stock quotes

```r
# Install the quantmod package if needed:
# install.packages('quantmod')

# load quantmod
require(quantmod)

# download Google quotes
goog <- getSymbols(Symbols = 'GOOG', src = 'yahoo', auto.assign = FALSE)

# setData with time in years -> delta = 1/252
# (there are 252 observations in 1 year)
data <- setData(goog$GOOG.Close, delta = 1/252, t0 = 0)

# model: geometric brownian motion
model <- setModel(drift = 'mu*x', diffusion = 'sigma*x', solve.variable = 'x')

# yuima object
yuima <- setYuima(model = model, data = data)

# estimation
estimation <- qmle(yuima,                              # the yuima object
              start = list(mu = 0, sigma = 0.5),  # starting values for optimization
              lower = list(sigma = 0))            # lower bounds

# estimates and standard errors
summary(estimation)
```

```
## Quasi-Maximum likelihood estimation
##
## Call:
## qmle(yuima = yuima, start = list(mu = 0, sigma = 0.5), lower = list(sigma = 0))
##
## Coefficients:
##        Estimate  Std. Error
## sigma 0.2984614 0.003103375
## mu    0.1911261 0.069630259
##
## -2 log L: 9813.243
```

yuimaGUI

The yuimaGUI package provides a user-friendly interface for yuima. It simplifies tasks such as estimation and simulation of stochastic processes, including additional tools related to quantitative finance such as data retrieval of stock prices and economic indicators, time series clustering, change point analysis, lead-lag estimation.

The yuimaGUI is available online for free, but it is strongly recommended to install the application via the R package on your local machine for better performance and less downtime.

```r
# install the package
install.packages('yuimaGUI')
```

```r
# load the package
require(yuimaGUI)
```

```r
# run the interface
yuimaGUI()
```

Code Download

Download the full code to generate this document and reproduce the examples. The file is in R Markdown, format for making dynamic documents with R. An R Markdown document is written in markdown, an easy-to-write plain text format, and contains chunks of embedded R code.
Download: https://storage.googleapis.com/emanueleguidotti/R/yuima-and-yuimaGUI.zip