The “yuima” package: and R framework for simulation and inference of SDEs

Stefano M. Iacus (University of Milan & R Core Team)
on behalf of Yuima Core Team

CREST and Sakigake International Symposium, Tokyo Institute of Technology, 18 Dec. 2010

Overview of the Yuima Project

The Yuima Project Team

A. Brouste (Univ. Le Mans, FR)
M. Fukasawa (Osaka Univ. JP)
H. Hino (Waseda Univ., Tokyo, JP)
S.M. Iacus (Milan Univ., IT)
K. Kengo (Tokyo Univ., JP)
H. Masuda (Kyushu Univ., JP)
Y. Shimitzu (Osaka Univ., JP)
M. Uchida (Osaka Univ., JP)
N. Yoshida (Tokyo Univ., JP)

. . . more to come

The `yuima` package[1] is written by people working in mathematical statistics and finance, who actively publish results in the field, have some knowledge of R, and have the feeling on “what’s next” in the field.

Aims at filling the gap between theory and practice!

[1] The Yuima Project is funded by the Japan Science Technology (JST) Basic Research Programs PRESTO, Grants-in-Aid for Scientific Research No. 19340021.

The `yuima` package goal: fill the gap between theory and practice

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.

- it is an R package, using `S4` classes and methods, where the basic class extends to SDE's with jumps (simple Poisson, Lévy), SDE's driven by fBM, Markov switching regime processes, HMM, etc.
- separates the data description from the inference tools and simulation schemes
- the design allows for multidimensional, multi-noise processes specification
- it includes a variety of tools useful in finance, like asymptotic expansion of functionals of stochastic processes via Malliavin calculus

Why the R language?

- R is now developed by a number of academics (18 people) called the R Core Team. Founded in 1999. I joined in 2001.
- R is free and open source. R is cross platform (M$-Windog, Unix & the alikes, Mac)
- No headheaches when transfer files, scripts, etc from one system to another. It works out-of-the-box.

Ok, it is free. So what?

“Free” is ok, but not enough!

Why the R language?

- the real added value of R is that it is open source, so completely transparent and hence continuously peer-reviewed by people in academia, users, etc. No black-boxes or magics behind R.
- this means: as soon as you release your code and there is a bug in it, a hundred of people will pester you to fix it!
- and if you don't fix it, your software is dead! So software is not just supposed to work, it has to work!
- R itself is about 2M lines of C and Fortran and R code!
- R has now more than 2600 (well, 2693 as of today) contributed packages in almost all fields of stats (but not limited to)
- 2693 plus 1 : the `yuima` package!

Overview of the `yuima` package

The `yuima` object

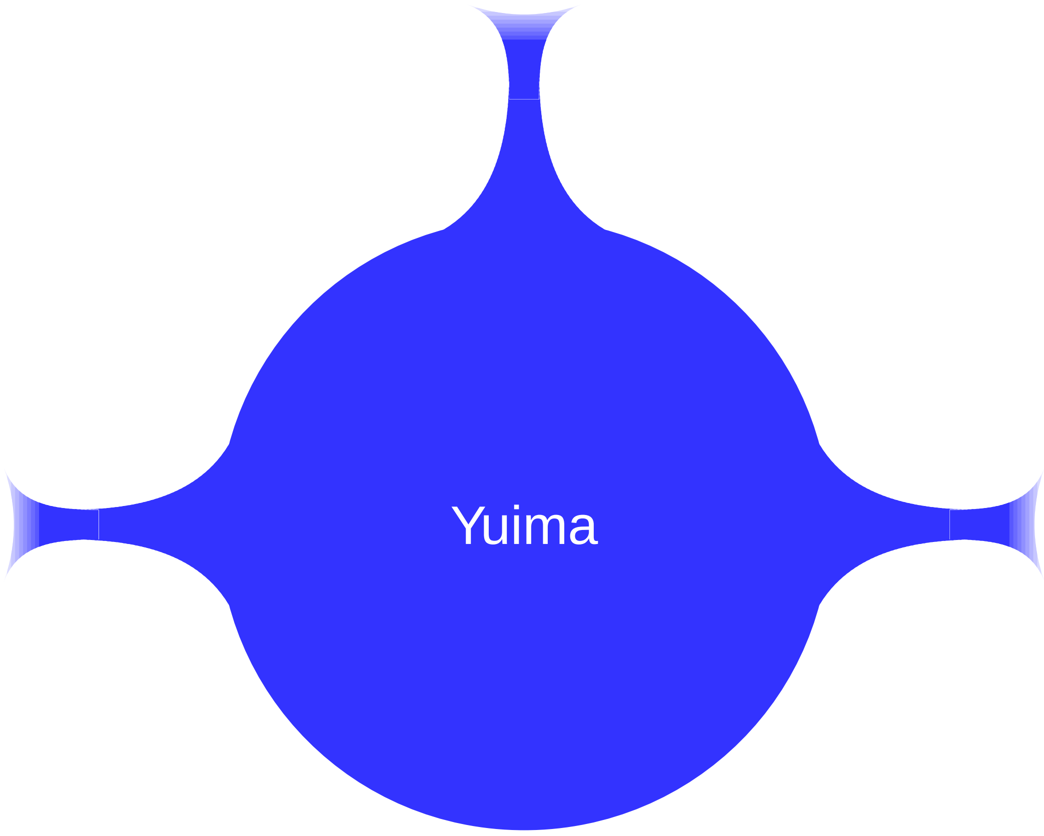
The main object is the `yuima` object which allows to describe the model in a mathematically sound way.

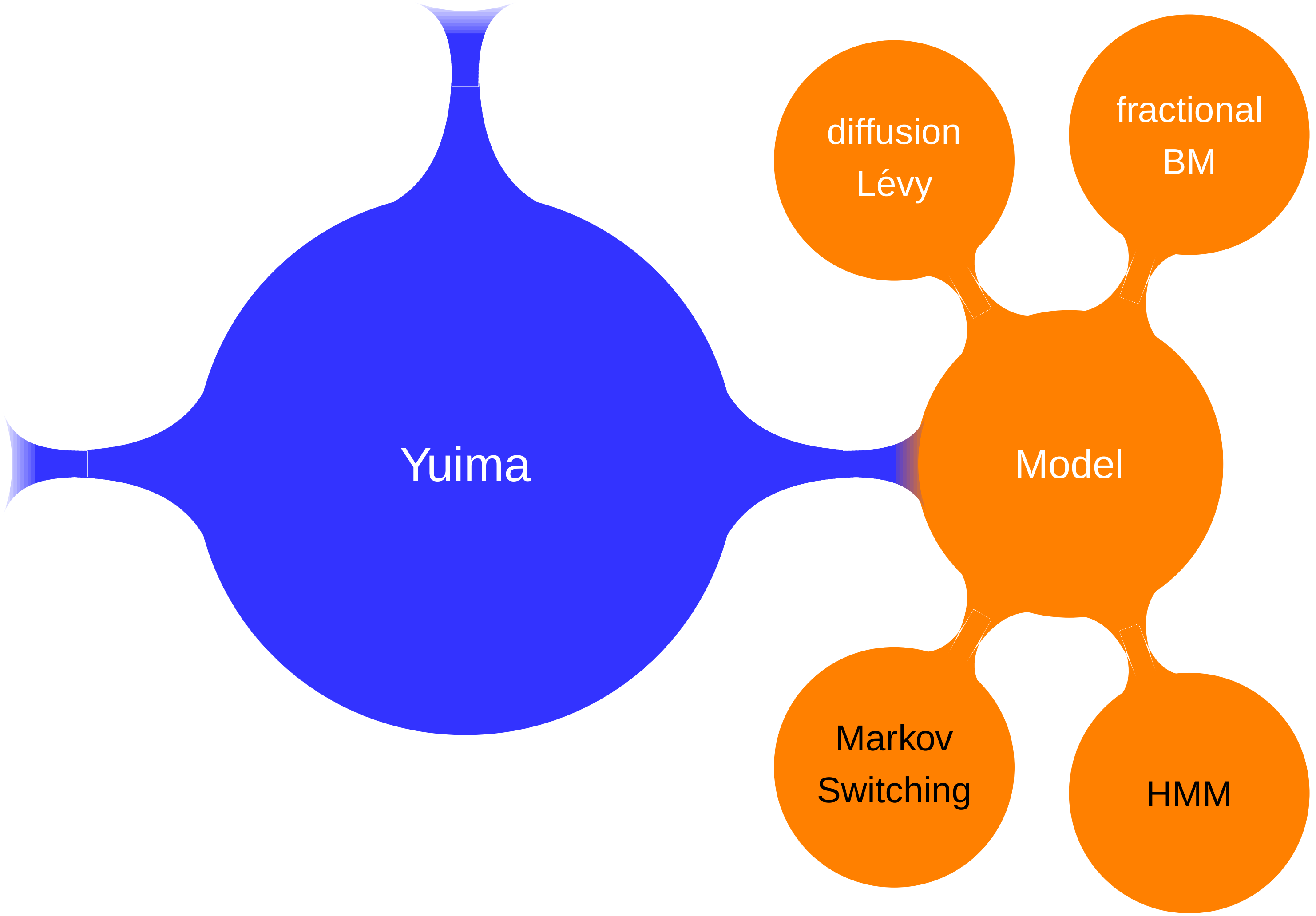
Then the data and the sampling structure can be included as well or, just the sampling scheme from which data can be generated according to the model.

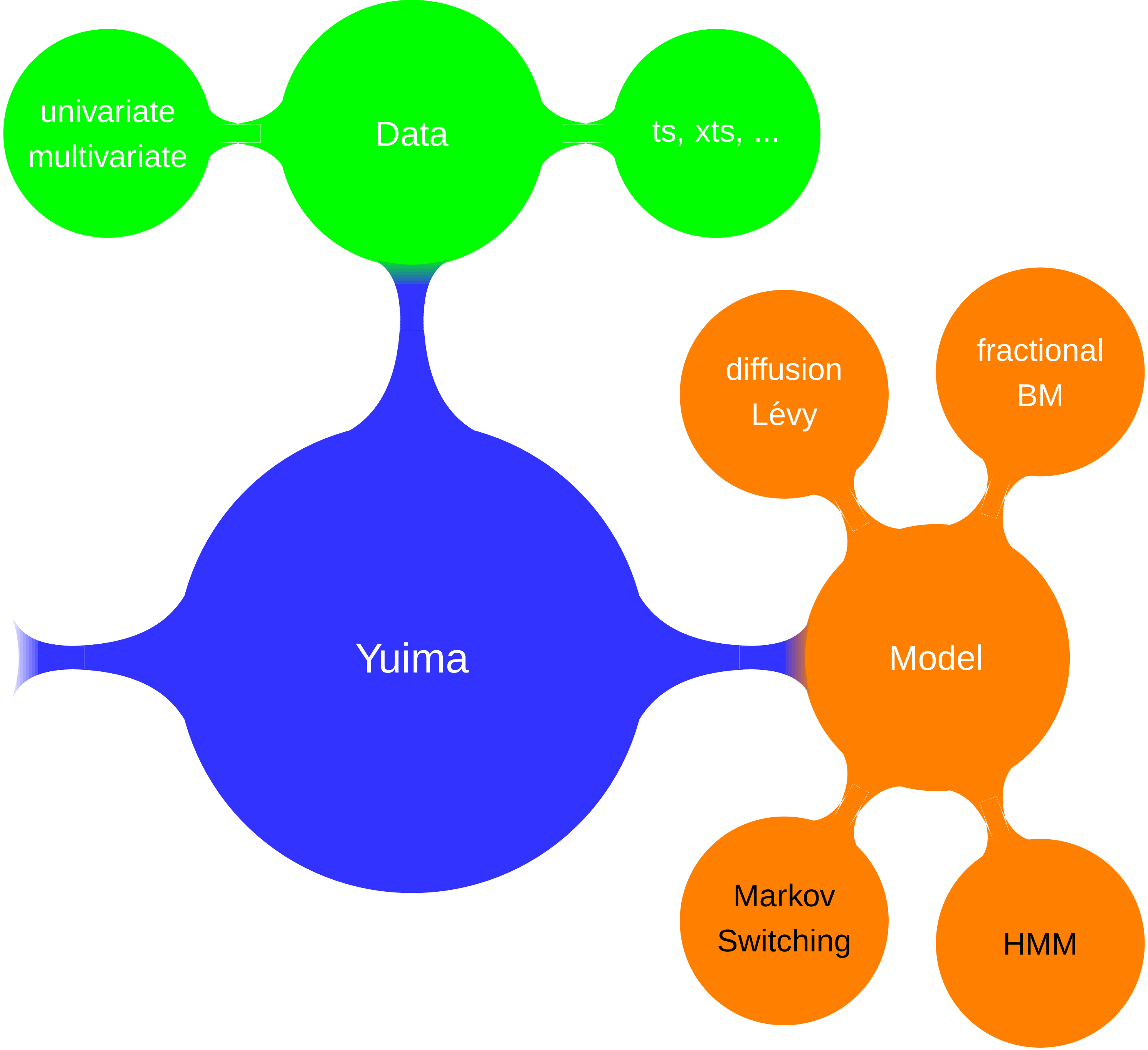
The package exposes very few generic functions like `simulate`, `qmle`, `plot`, etc. and some other specific functions for special tasks.

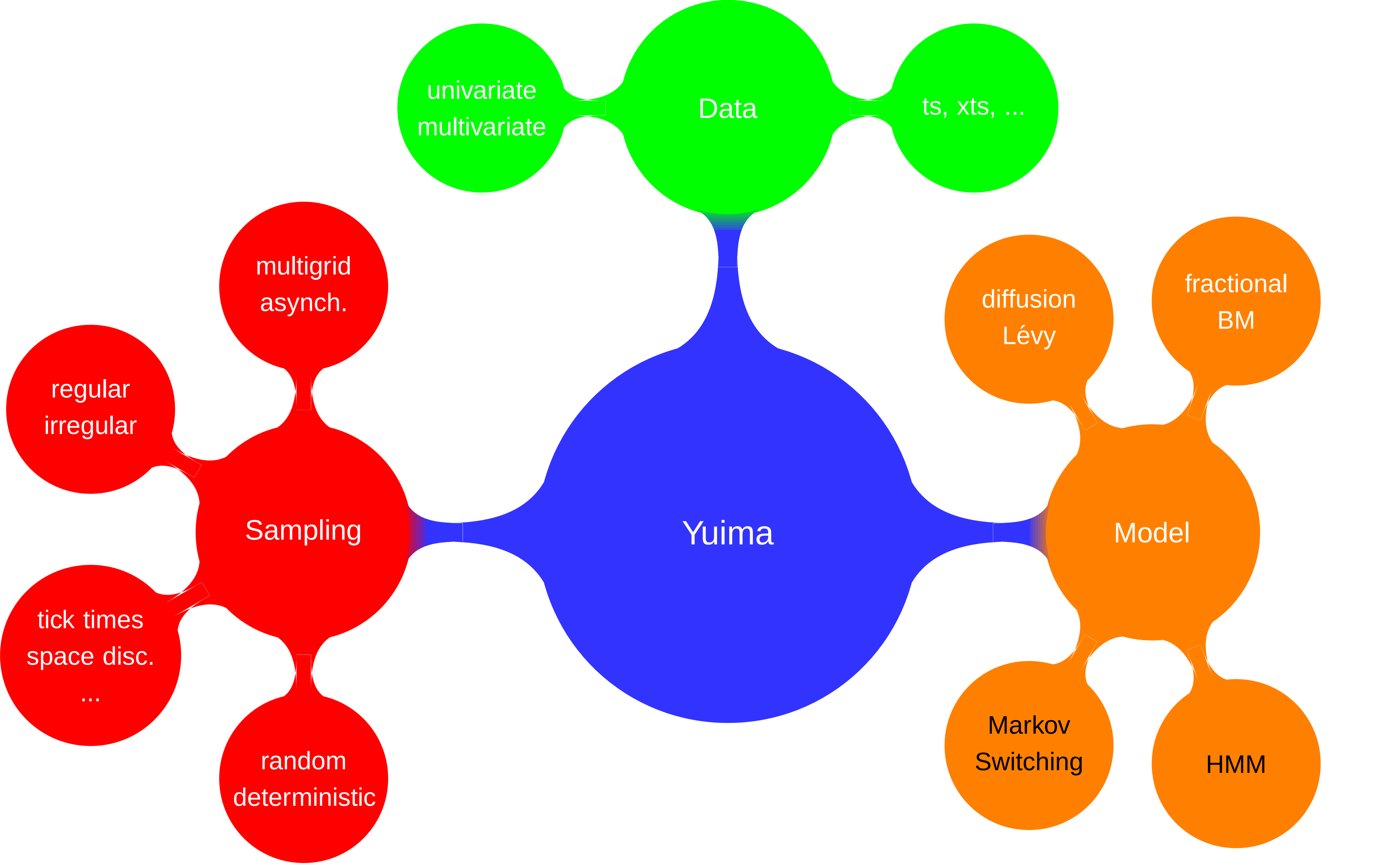
Before looking at the details, let us see an overview of the main object.

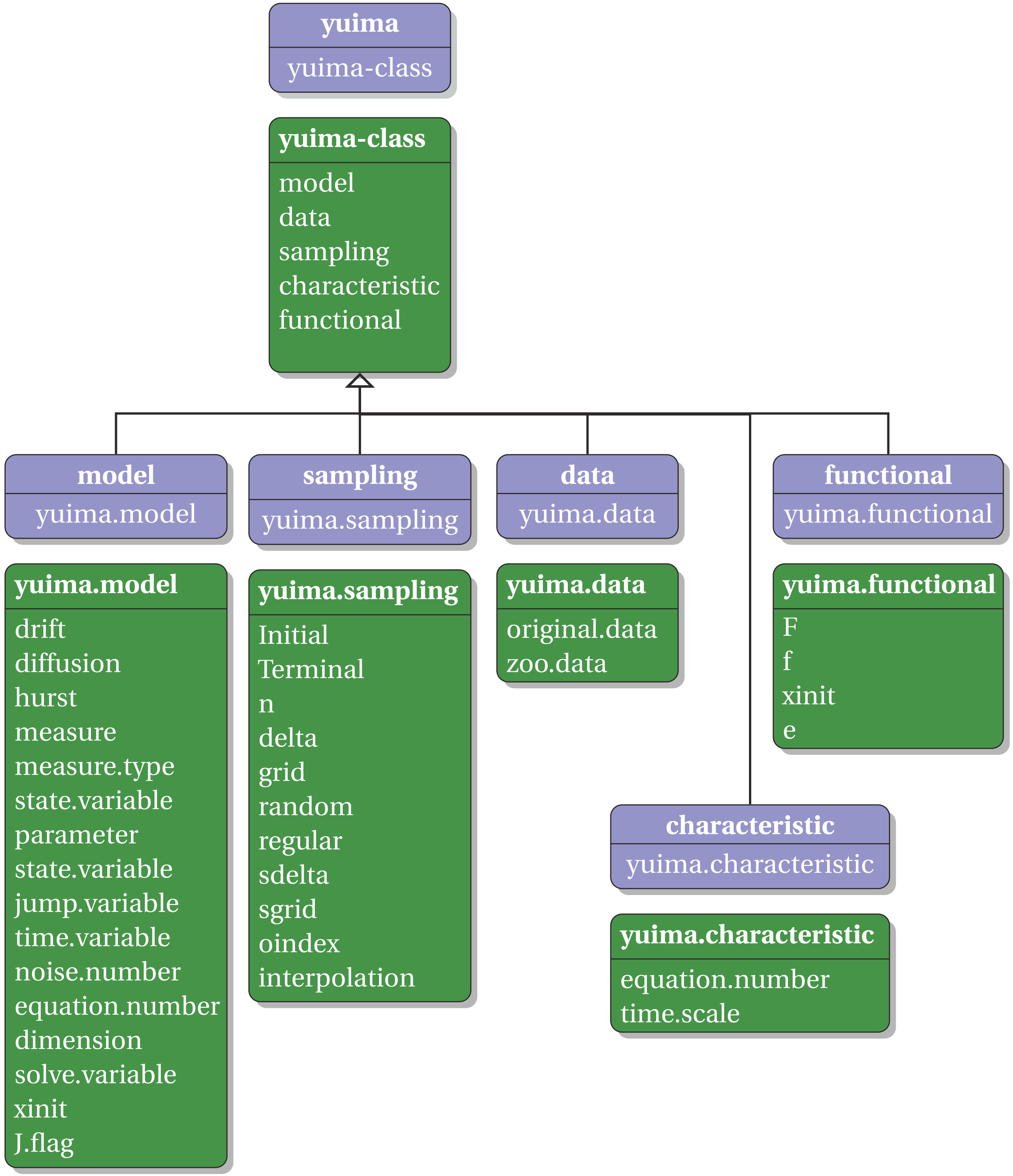
What contains a `yuima` object ?


Yuima

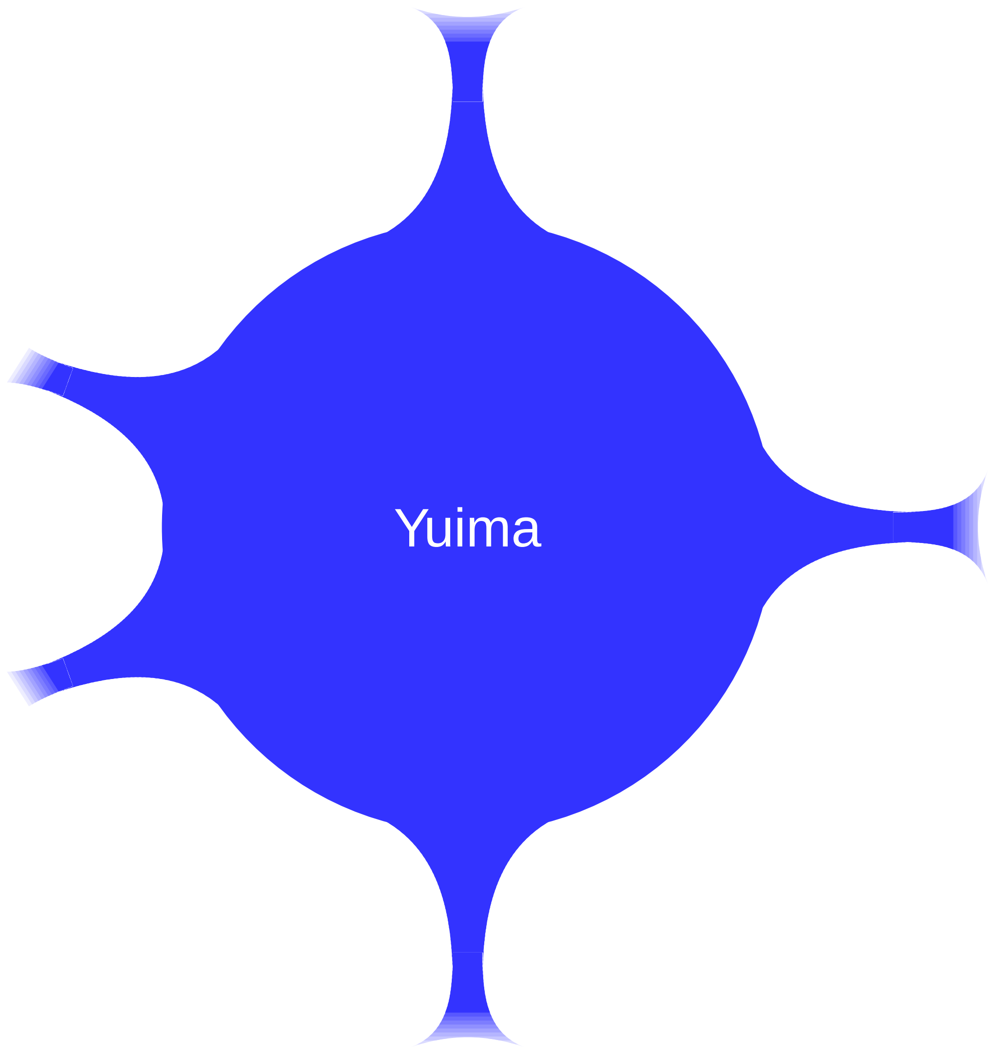

Yuima
Model
diffusion
Lévy
fractional
BM
Markov
Switching
HMM

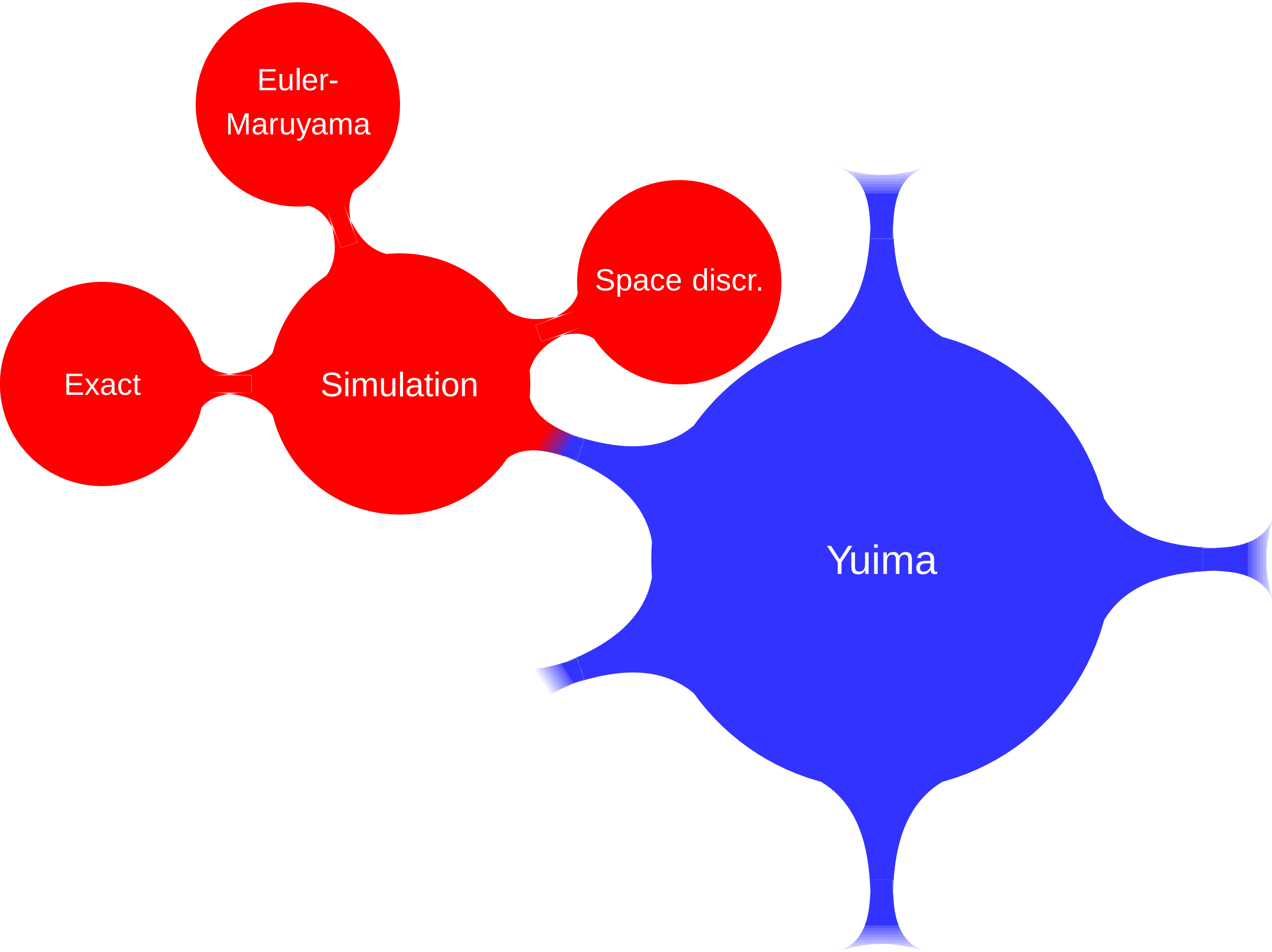

univariate
multivariate
Data
ts, xts, ...
diffusion
Lévy
fractional
BM
Yuima
Model
Markov
Switching
HMM

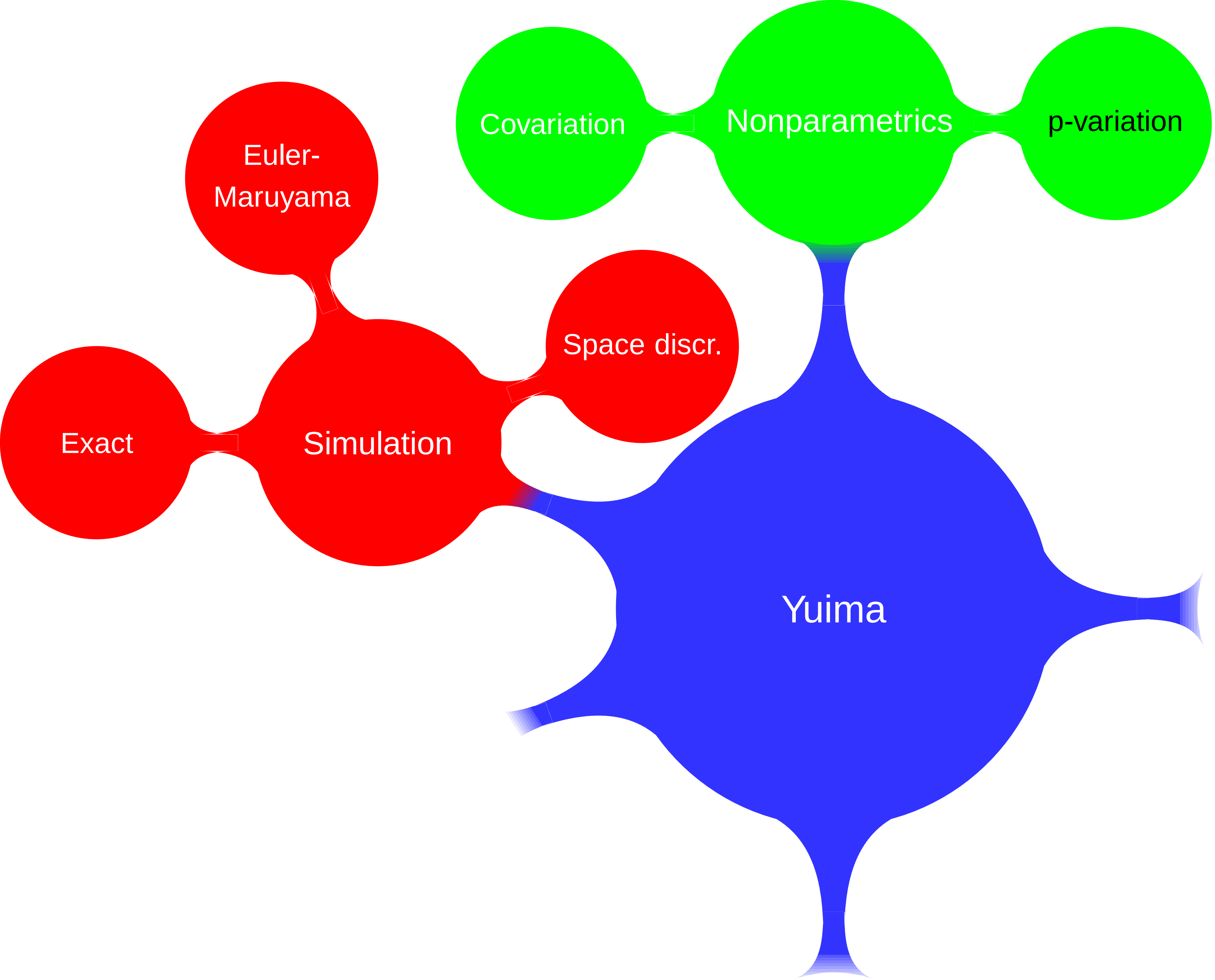

univariate
multivariate
Data
ts, xts, ...
multigrid
asynch.
regular
irregular
Sampling
tick times
space disc.
...
random
deterministic
Yuima
diffusion
Lévy
fractional
BM
Model
Markov
Switching
HMM

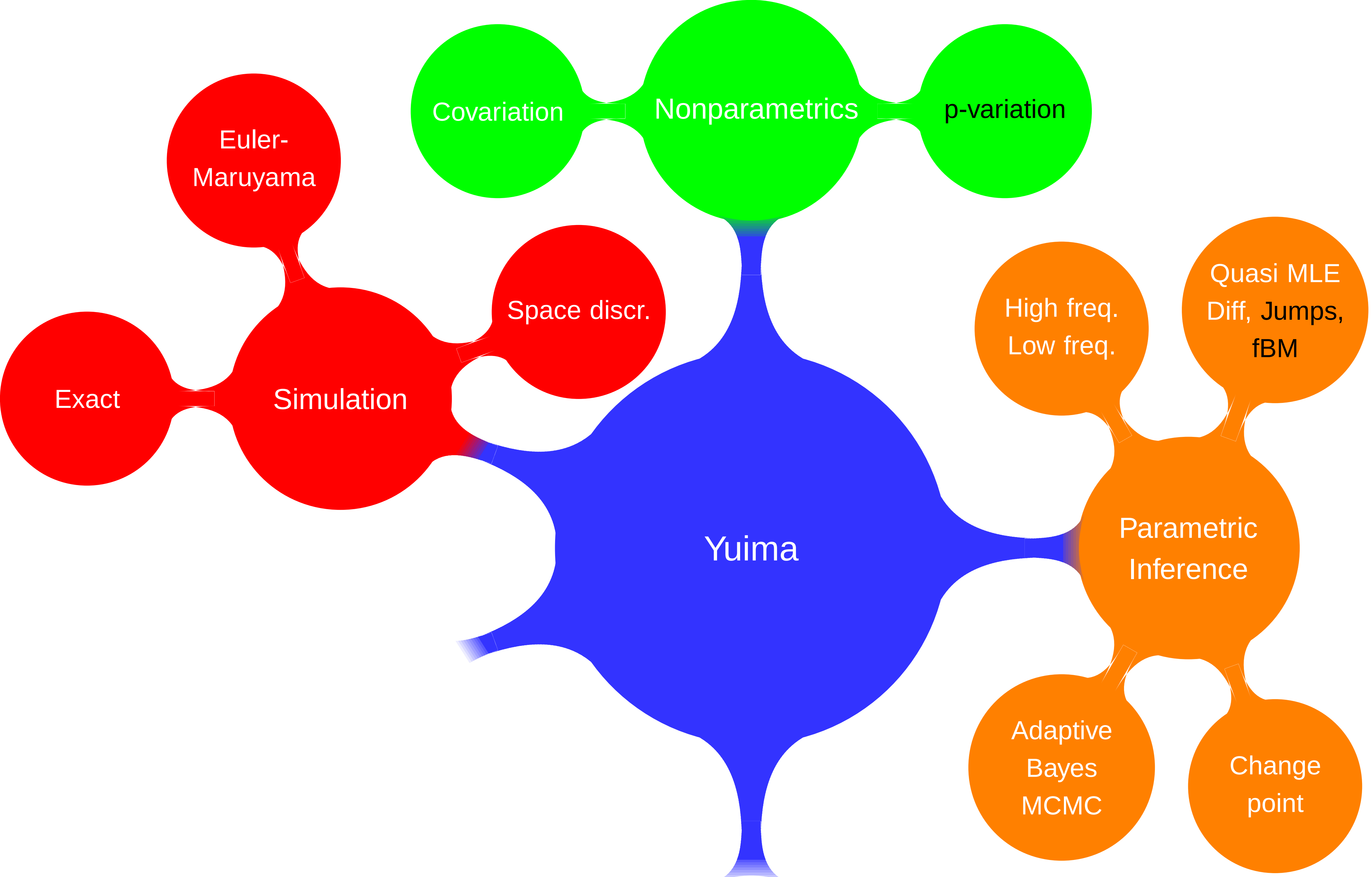

yuima
yuima-class
yuima-class
model
data
sampling
characteristic
functional
model
yuima.model
sampling
yuima.sampling
data
yuima.data
functional
yuima.functional
yuima.model
drift
diffusion
hurst
measure
measure.type
state.variable
parameter
state.variable
jump.variable
time.variable
noise.number
equation.number
dimension
solve.variable
xinit
J.flag
yuima.sampling
Initial
Terminal
n
delta
grid
random
regular
sdelta
sgrid
oindex
interpolation
yuima.data
original.data
zoo.data
yuima.functional
F
f
xinit
e
characteristic
yuima.characteristic
yuima.characteristic
equation.number
time.scale

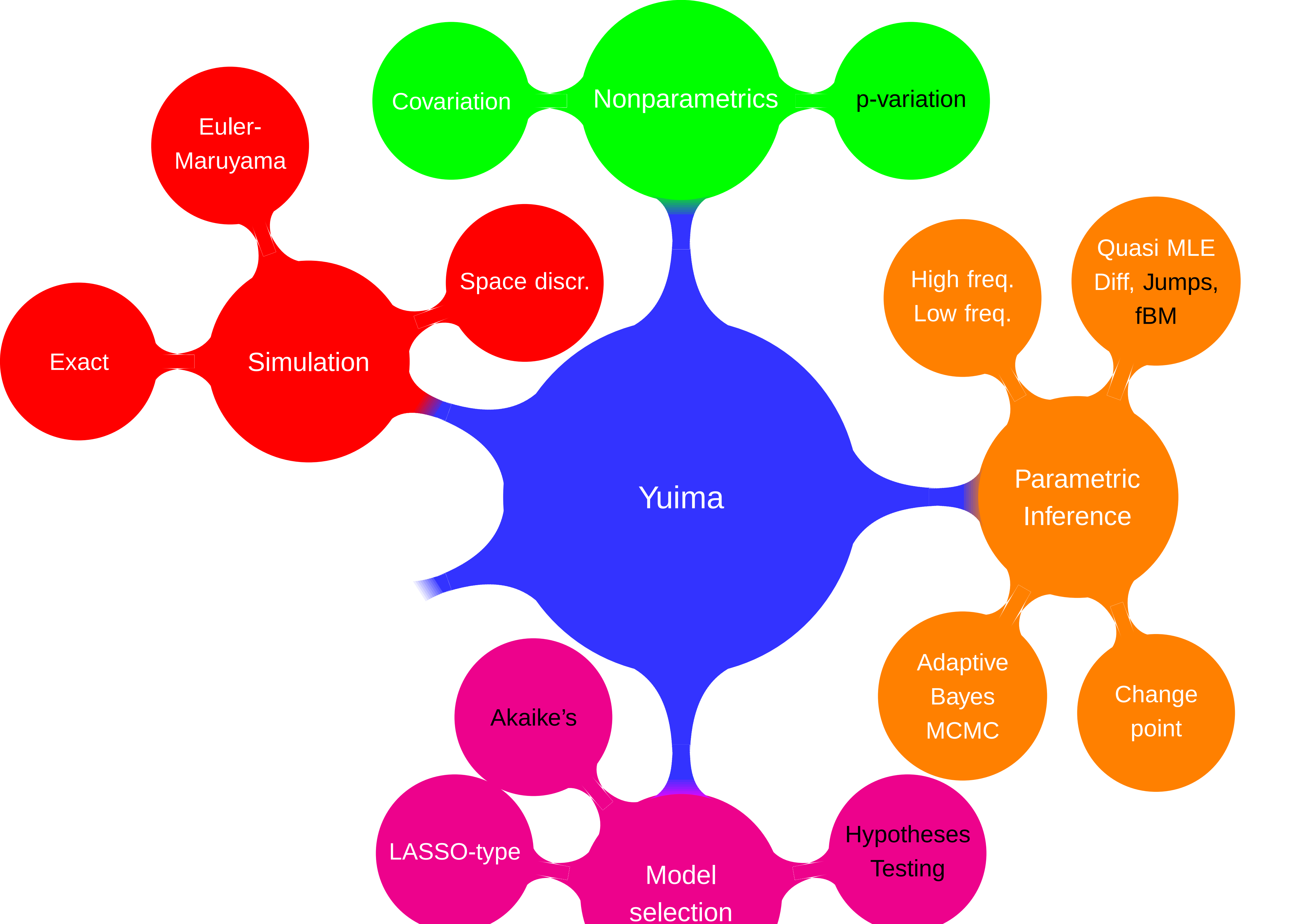
What is possible to do with a `yuima` object in hands?

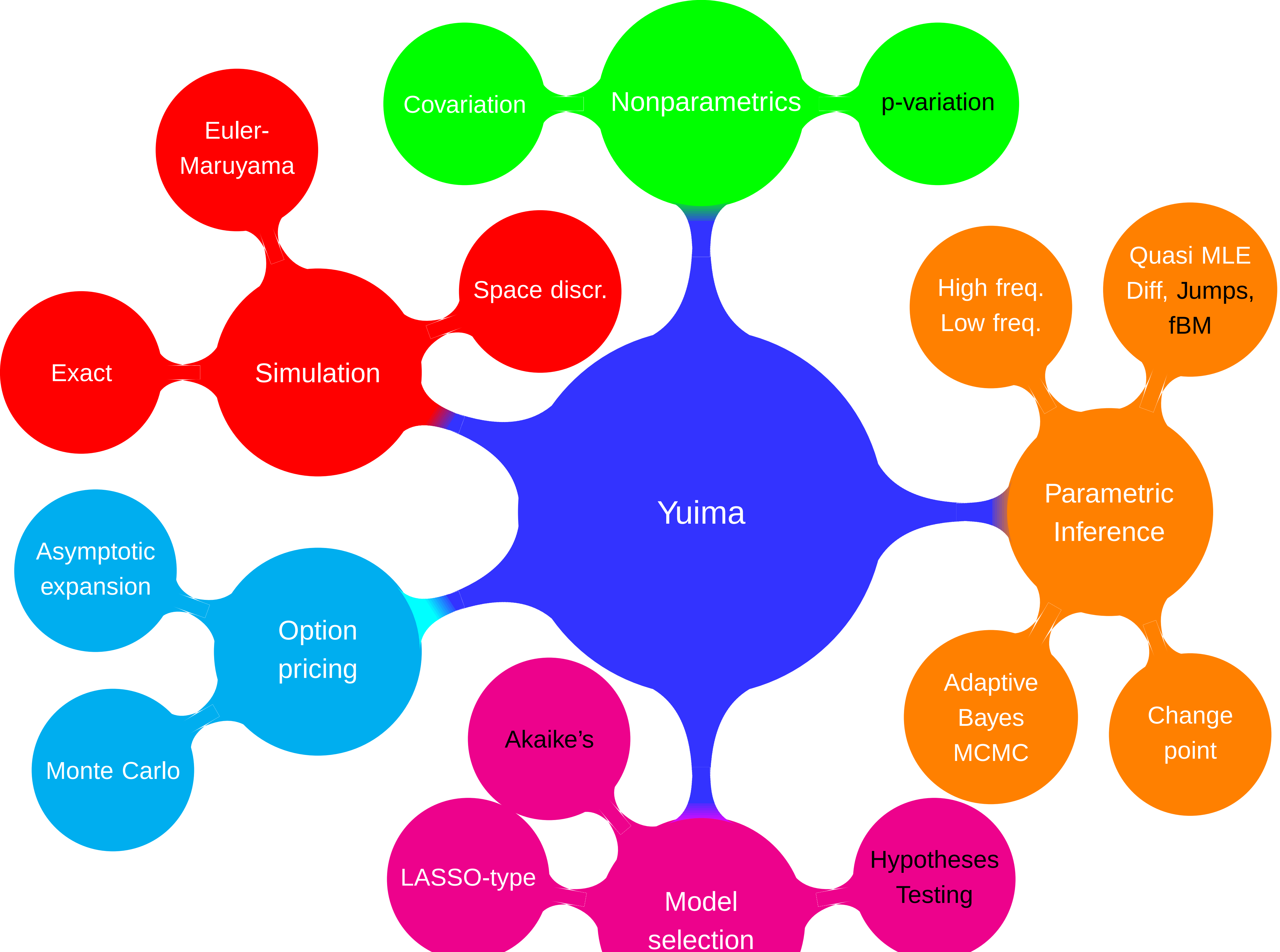

Yuima


Euler-
Maruyama
Space discr.
Exact
Simulation
Yuima


Covariation
Nonparametrics
p-variation
Euler-
Maruyama
Space discr.
Exact
Simulation
Yuima


Covariation
Nonparametrics
p-variation
Euler-
Maruyama
Space discr.
Exact
Simulation
High freq.
Low freq.
Quasi MLE
Diff, Jumps,
fBM
Yuima
Parametric
Inference
Adaptive
Bayes
MCMC
Change
point


Covariation
Nonparametrics
p-variation
Euler-
Maruyama
Space discr.
Exact
Simulation
Yuima
High freq.
Low freq.
Quasi MLE
Diff, Jumps,
fBM
Parametric
Inference
Adaptive
Bayes
MCMC
Change
point
Akaike's
LASSO-type
Model
selection
Hypotheses
Testing


Yuima
Nonparametrics
Covariation
p-variation
Simulation
Euler-
Maruyama
Space discr.
Exact
Parametric
Inference
High freq.
Low freq.
Quasi MLE
Diff, Jumps,
fBM
Adaptive
Bayes
MCMC
Change
point
Option
pricing
Asymptotic
expansion
Monte Carlo
Model
selection
Akaike's
LASSO-type
Hypotheses
Testing

How it is supposed to work?

The model specification

We consider here the three main classes of SDE's which can be easily specified. All multidimensional and eventually parametric models.

We consider here the three main classes of SDE's which can be easily specified. All multidimensional and eventually parametric models.

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

The model specification

We consider here the three main classes of SDE's which can be easily specified. All multidimensional and eventually parametric models.

- Diffusions $\mathrm{d}X_t = a(t, X_t)dt + b(t, X_t)\mathrm{d}W_t$
- Fractional Gaussian Noise, with H the Hurst parameter

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

We consider here the three main classes of SDE's which can be easily specified. All multidimensional and eventually parametric models.

- Diffusions $\mathrm{d}X_t = a(t, X_t)dt + b(t, X_t)\mathrm{d}W_t$
- Fractional Gaussian Noise, with H the Hurst parameter

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

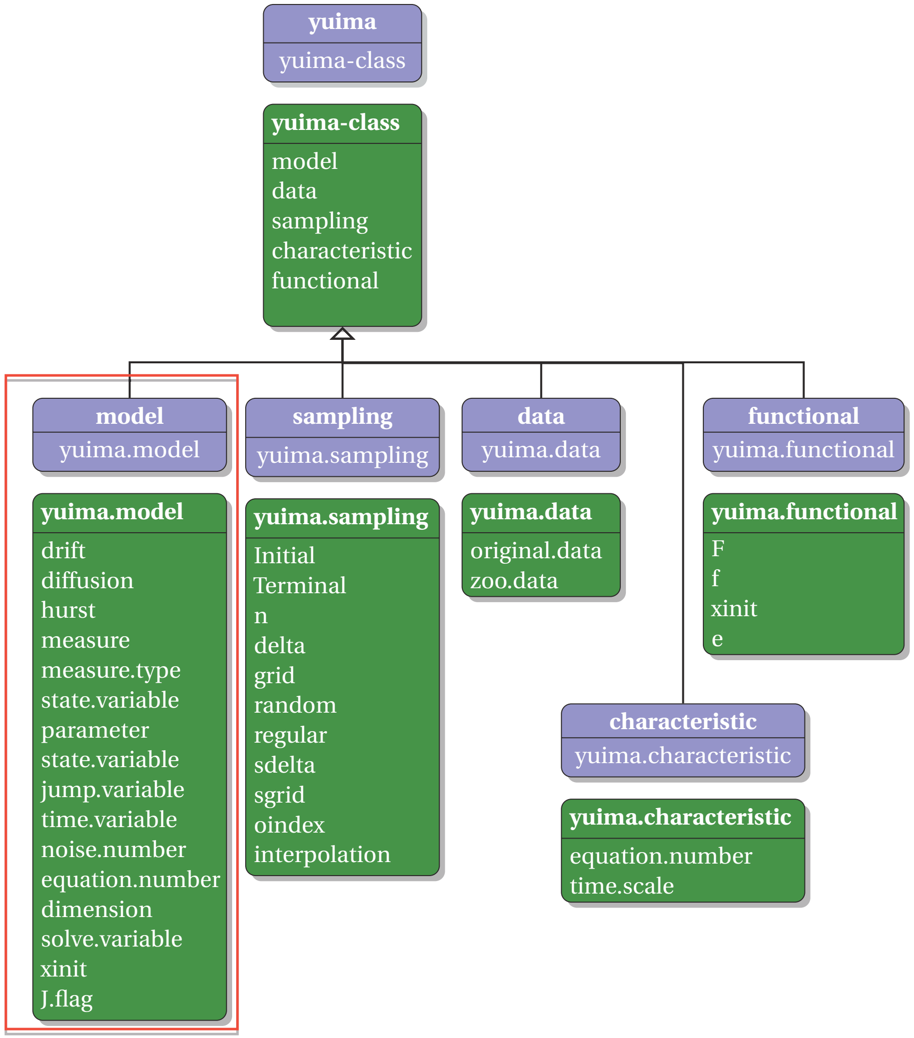
- Diffusions with jumps, Lévy

$$\mathrm{d}X_t = a(X_t)dt + b(X_t)\mathrm{d}W_t + \int_{|z|>1} c(X_{t-}, z)\mu(\mathrm{d}t, \mathrm{d}z)$$

$$+ \int_{0<|z|\le 1} c(X_{t-}, z)\{\mu(\mathrm{d}t, \mathrm{d}z) - \nu(\mathrm{d}z)\mathrm{d}t\}$$

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

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



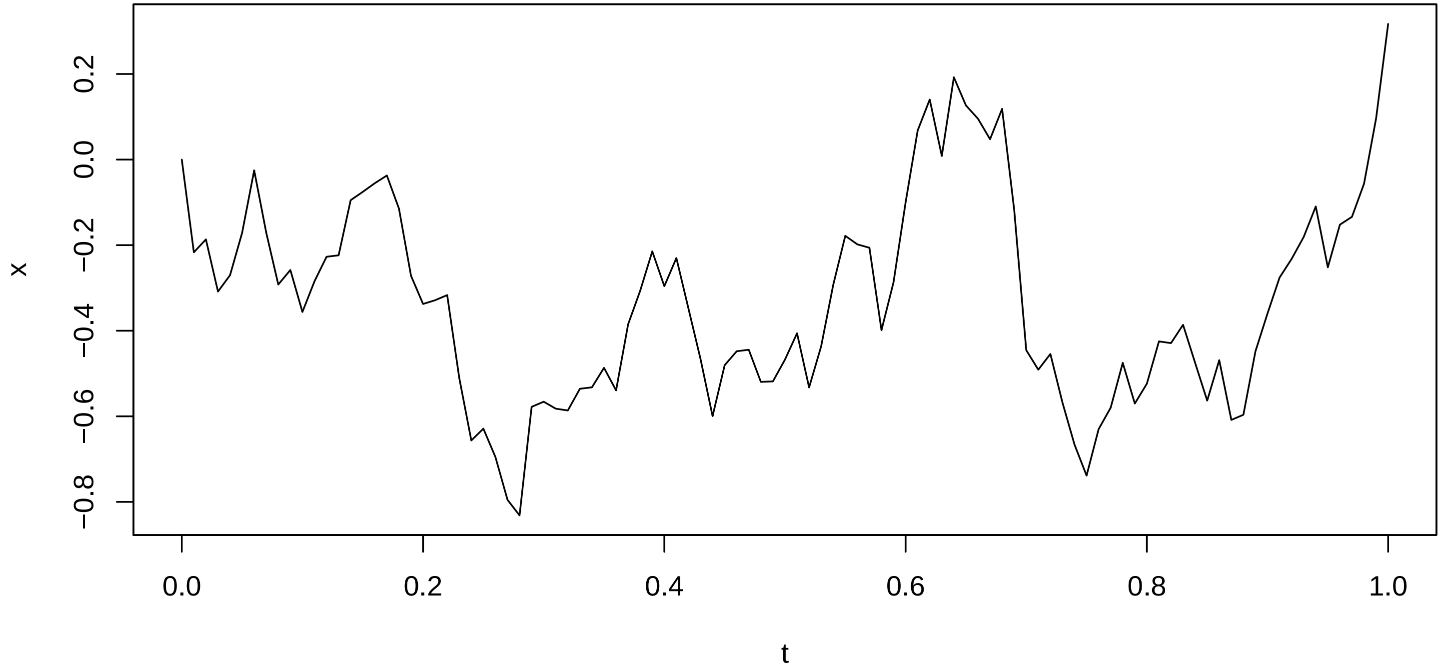
$$dX_t = -3X_t dt + \frac{1}{1+X_t^2} dW_t$$


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

```
> str(mod1)
Formal class 'yuima.model' [package "yuima"] with 16 slots
  ..@ drift          :  expression((-3 * x))
  ..@ diffusion      :List of 1
  .. ..$ :  expression(1/(1 + x^2))
  ..@ hurst          : num 0.5
  ..@ jump.coeff     :  expression()
  ..@ measure        : list()
  ..@ measure.type   : chr(0)
  ..@ parameter      :Formal class 'model.parameter' [package "yuima"] with 6 slots
  .. .. ..@ all      : chr(0)
  .. .. ..@ common   : chr(0)
  .. .. ..@ diffusion: chr(0)
  .. .. ..@ drift    : chr(0)
  .. .. ..@ jump     : chr(0)
  .. .. ..@ measure  : chr(0)
  ..@ state.variable : chr "x"
  ..@ jump.variable  : chr(0)
  ..@ time.variable  : chr "t"
  ..@ noise.number   : num 1
  ..@ equation.number: int 1
  ..@ dimension      : int [1:6] 0 0 0 0 0 0
  ..@ solve.variable : chr "x"
  ..@ xinit          : num 0
  ..@ J.flag         : logi FALSE
```

And we can easily simulate and plot the model like

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

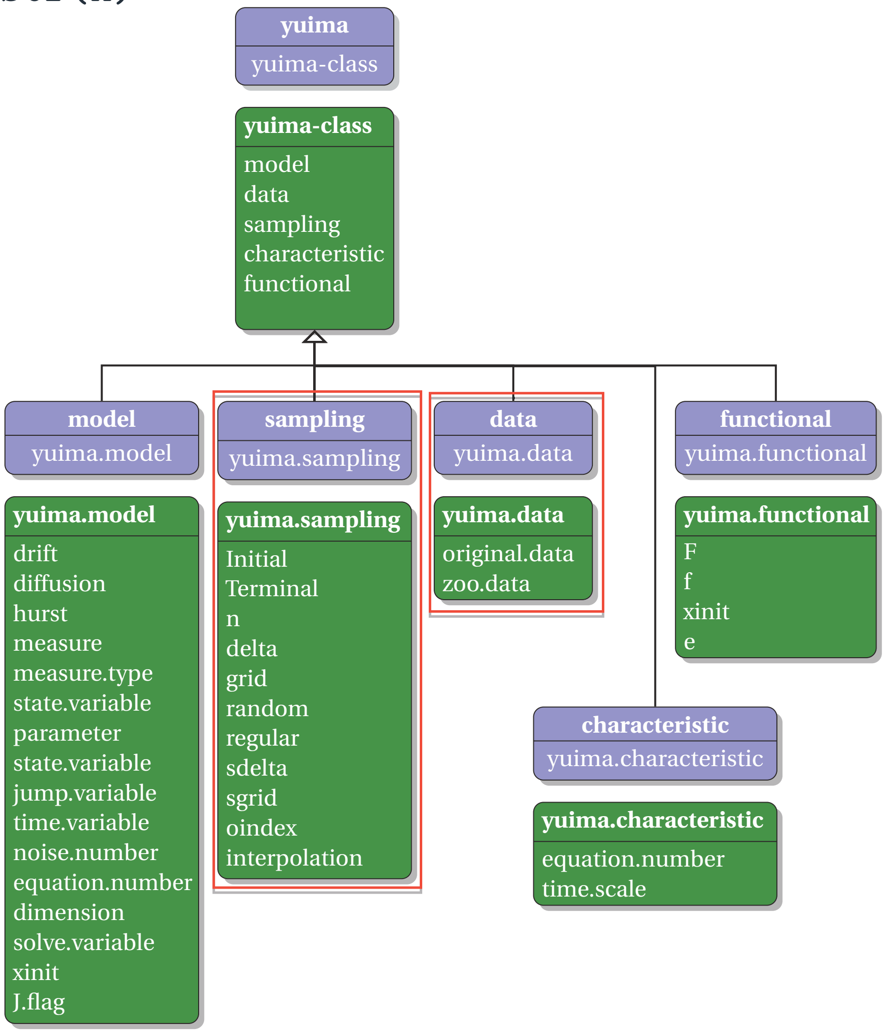


The `simulate` function fills the slots `data` and `sampling`

```
> str(X)
```

The `simulate` function fills the slots `data` and `sampling`

```
> str(X)
```



The `simulate` function fills the slots `data` and `sampling`

```
> str(X)

Formal class 'yuima' [package "yuima"] with 5 slots
  ..@ data          :Formal class 'yuima.data' [package "yuima"] with 2 slots
  .. .. ..@ original.data: ts [1:101, 1] 0 -0.217 -0.186 -0.308 -0.27 ...
  .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. .. ..$ : NULL
  .. .. .. .. ..$ : chr "Series 1"
  .. .. .. ..- attr(*, "tsp")= num [1:3] 0 1 100
  .. .. ..@ zoo.data     :List of 1
  .. .. .. ..$ Series 1:'zooreg' series from 0 to 1
 ..@ model          :Formal class 'yuima.model' [package "yuima"] with 16 slots

(...) output dropped

 ..@ sampling       :Formal class 'yuima.sampling' [package "yuima"] with 11 slots
  .. .. ..@ Initial      : num 0
  .. .. ..@ Terminal     : num 1
  .. .. ..@ n            : num 100
  .. .. ..@ delta        : num 0.1
  .. .. ..@ grid         : num(0)
  .. .. ..@ random       : logi FALSE
  .. .. ..@ regular      : logi TRUE
  .. .. ..@ sdelta       : num(0)
  .. .. ..@ sgrid        : num(0)
  .. .. ..@ oindex       : num(0)
  .. .. ..@ interpolation: chr "none"
```

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

Automatic extraction of the parameters for further inference

```
> str(mod2)
Formal class 'yuima.model' [package "yuima"] with 16 slots
  ..@ drift           :  expression((-theta * x))
  ..@ diffusion       :List of 1
  .. ..$ :  expression(1/(1 + x^gamma))
  ..@ hurst           : num 0.5
  ..@ jump.coeff      :  expression()
  ..@ measure         : list()
  ..@ measure.type    : chr(0)
  ..@ parameter       :Formal class 'model.parameter' [package "yuima"] with 6 slots
  .. .. ..@ all       : chr [1:2] "theta" "gamma"
  .. .. ..@ common    : chr(0)
  .. .. ..@ diffusion: chr "gamma"
  .. .. ..@ drift     : chr "theta"
  .. .. ..@ jump      : chr(0)
  .. .. ..@ measure   : chr(0)
  ..@ state.variable : chr "x"
  ..@ jump.variable  : chr(0)
  ..@ time.variable  : chr "t"
  ..@ noise.number    : num 1
  ..@ equation.number: int 1
  ..@ dimension       : int [1:6] 2 0 1 1 0 0
  ..@ solve.variable : chr "x"
  ..@ xinit           : num 0
  ..@ J.flag          : logi FALSE
```

Parametric model: $\mathrm{d}X_t = -\theta X_t \mathrm{d}t + \frac{1}{1+X_t^\gamma}\mathrm{d}W_t$

And this can be simulated specifying the parameters

```
> simulate(mod2,true.param=list(theta=1,gamma=3))
```

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

has to be organized into matrix form

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

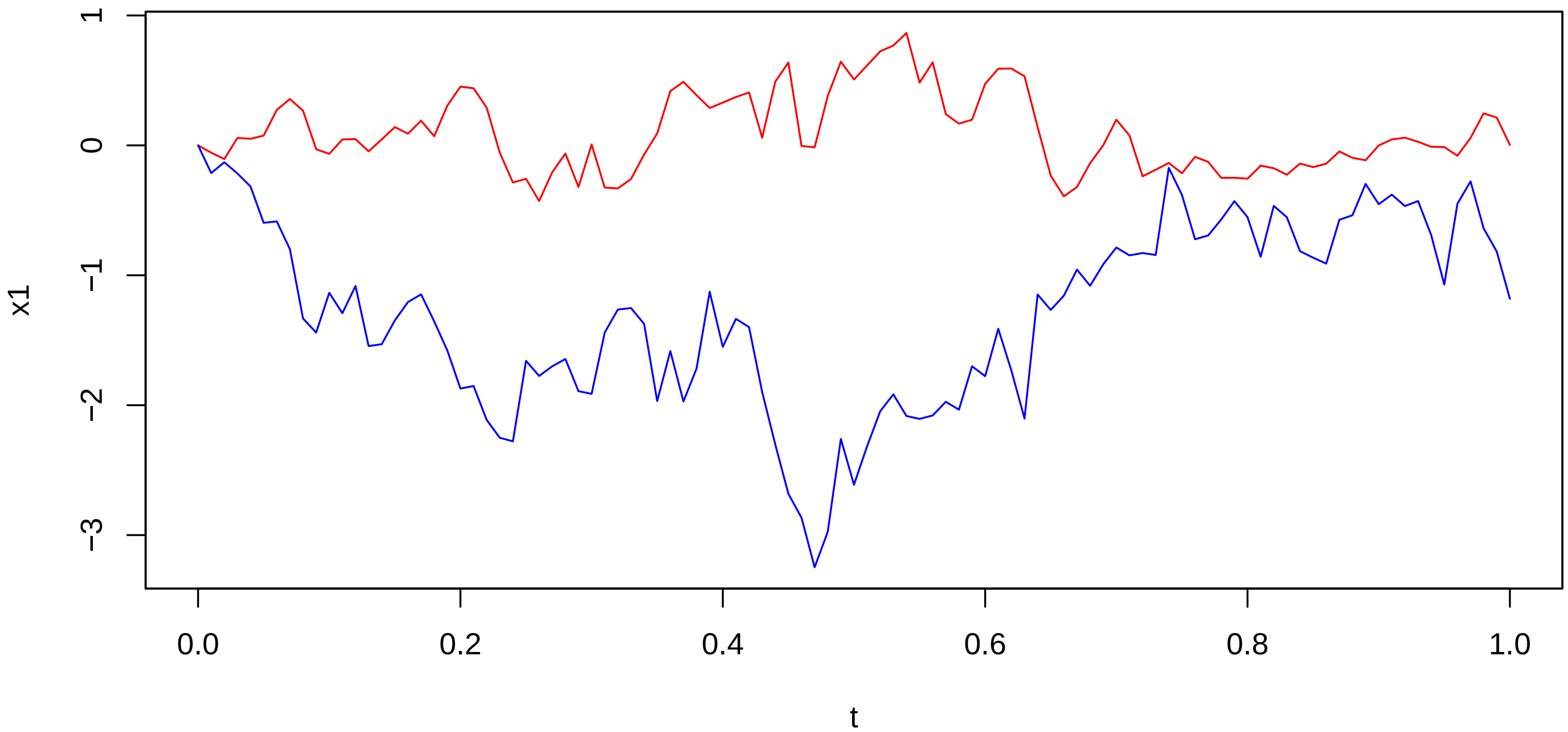
```
> sol <- c("x1","x2") # variable for numerical solution
> a <- c("-3*x1","-x1-2*x2")   # drift vector
> b <- matrix(c("1","x1","0","3","x2","0"),2,3)  #  diffusion matrix
> mod3 <- setModel(drift = a, diffusion = b, solve.variable = sol)
```

2-dimensional diffusions with 3 noises

$$
\begin{aligned}
\mathrm{d}X_t^1 &= -3X_t^1\mathrm{d}t + \mathrm{d}W_t^1 + X_t^2\mathrm{d}W_t^3 \\
\mathrm{d}X_t^2 &= -(X_t^1 + 2X_t^2)\mathrm{d}t + X_t^1\mathrm{d}W_t^1 + 3\mathrm{d}W_t^2
\end{aligned}
$$

```
> str(mod3)
Formal class 'yuima.model' [package "yuima"] with 16 slots
  ..@ drift          :  expression((-3 * x1), (-x1 - 2 * x2))
  ..@ diffusion      :List of 2
  .. ..$ :  expression(1, 0, x2)
  .. ..$ :  expression(x1, 3, 0)
  ..@ hurst          : num 0.5
  ..@ jump.coeff     :  expression()
  ..@ measure        : list()
  ..@ measure.type   : chr(0)
  ..@ parameter      :Formal class 'model.parameter' [package "yuima"] with 6 slots
  .. .. ..@ all       : chr(0)
  .. .. ..@ common    : chr(0)
  .. .. ..@ diffusion: chr(0)
  .. .. ..@ drift     : chr(0)
  .. .. ..@ jump      : chr(0)
  .. .. ..@ measure   : chr(0)
  ..@ state.variable : chr "x"
  ..@ jump.variable  : chr(0)
  ..@ time.variable  : chr "t"
  ..@ noise.number   : int 3
  ..@ equation.number: int 2
  ..@ dimension      : int [1:6] 0 0 0 0 0 0
  ..@ solve.variable : chr [1:2] "x1" "x2"
  ..@ xinit          : num [1:2] 0 0
  ..@ J.flag         : logi FALSE
```

```
> set.seed(123)
> X <- simulate(mod3)
> plot(X,plot.type="single",col=c("red","blue"))
```



Also models likes this can be specified

$$\left\{\begin{aligned} \mathrm{d}X_t^1 &= X_t^2\left|X_t^1\right|^{2/3}\mathrm{d}W_t^1, \\ \mathrm{d}X_t^2 &= g(t)\mathrm{d}X_t^3, \\ \mathrm{d}X_t^3 &= X_t^3(\mu\mathrm{d}t + \sigma(\rho\mathrm{d}W_t^1 + \sqrt{1-\rho^2}\mathrm{d}W_t^2)) \end{aligned}\right. ,$$

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

The above is an example of parametric SDE with more equations than noises.

```
> mod4 <- setModel(drift="3*y", diffusion=1, hurst=0.3, solve.var="y")
```

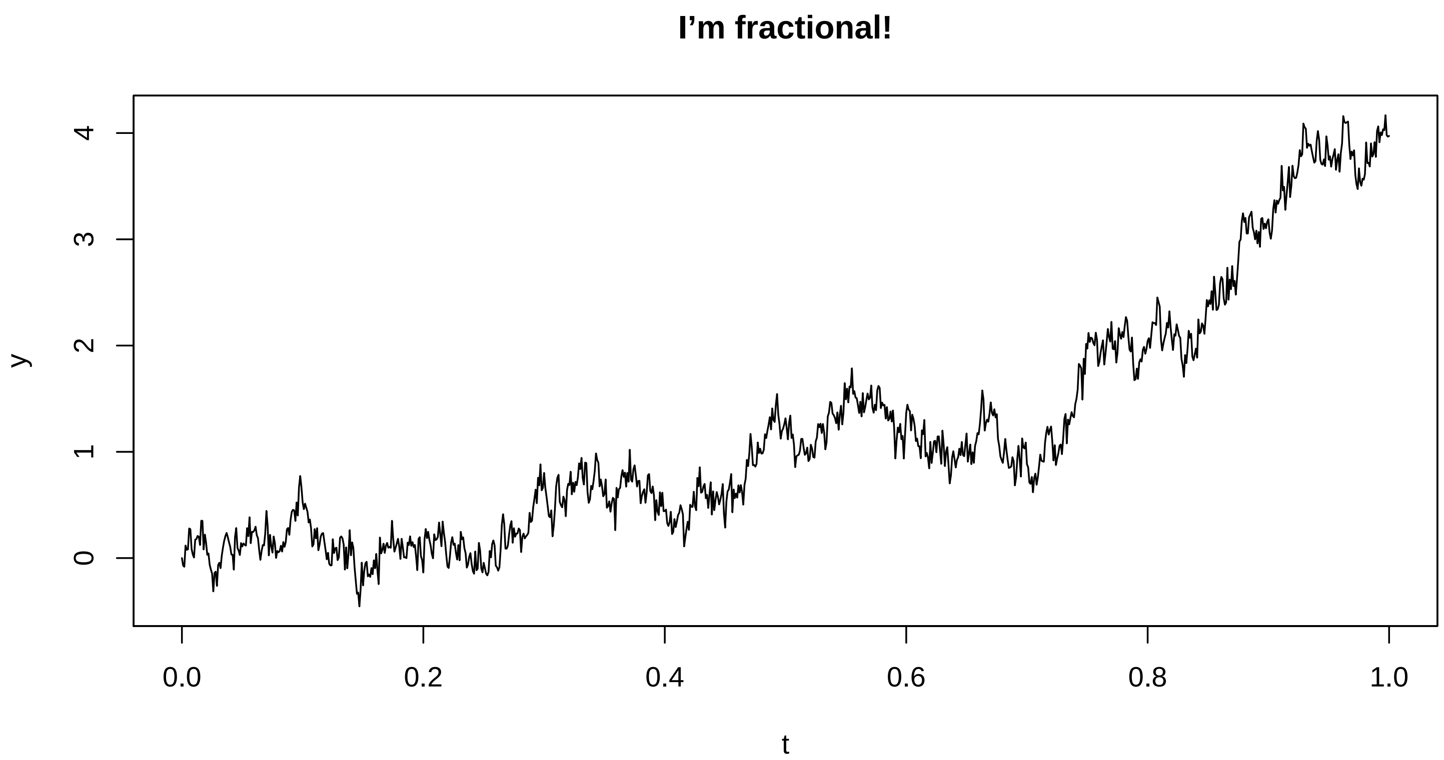
```
> mod4 <- setModel(drift="3*y", diffusion=1, hurst=0.3, solve.var="y")
```

The `hurst` slot is filled

```
> str(mod4)
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     :  expression()
  ..@ measure        : list()
  ..@ measure.type   : chr(0)
  ..@ parameter      :Formal class 'model.parameter' [package "yuima"] with 6 slots
  .. .. ..@ all       : chr(0)
  .. .. ..@ common    : chr(0)
  .. .. ..@ diffusion: chr(0)
  .. .. ..@ drift     : chr(0)
  .. .. ..@ jump      : chr(0)
  .. .. ..@ measure   : chr(0)
  ..@ state.variable : chr "x"
  ..@ jump.variable  : chr(0)
  ..@ time.variable  : chr "t"
  ..@ noise.number   : num 1
  ..@ equation.number: int 1
  ..@ dimension      : int [1:6] 0 0 0 0 0 0
  ..@ solve.variable : chr "y"
  ..@ xinit          : num 0
  ..@ J.flag         : logi FALSE
```

Fractional Gaussian Noise $\mathrm{d}Y_t = 3Y_t\mathrm{d}t + \mathrm{d}W_t^H$


```
> set.seed(123)
> X <- simulate(mod4, n=1000)
> plot(X, main="I'm fractional!")
```



Jump processes can be specified in different ways in mathematics (and hence in yuima package).

Let Z_t be a Compound Poisson Process (i.e. jumps follow some distribution, e.g. Gaussian)

Then is is possible to consider the following SDE which involves jumps

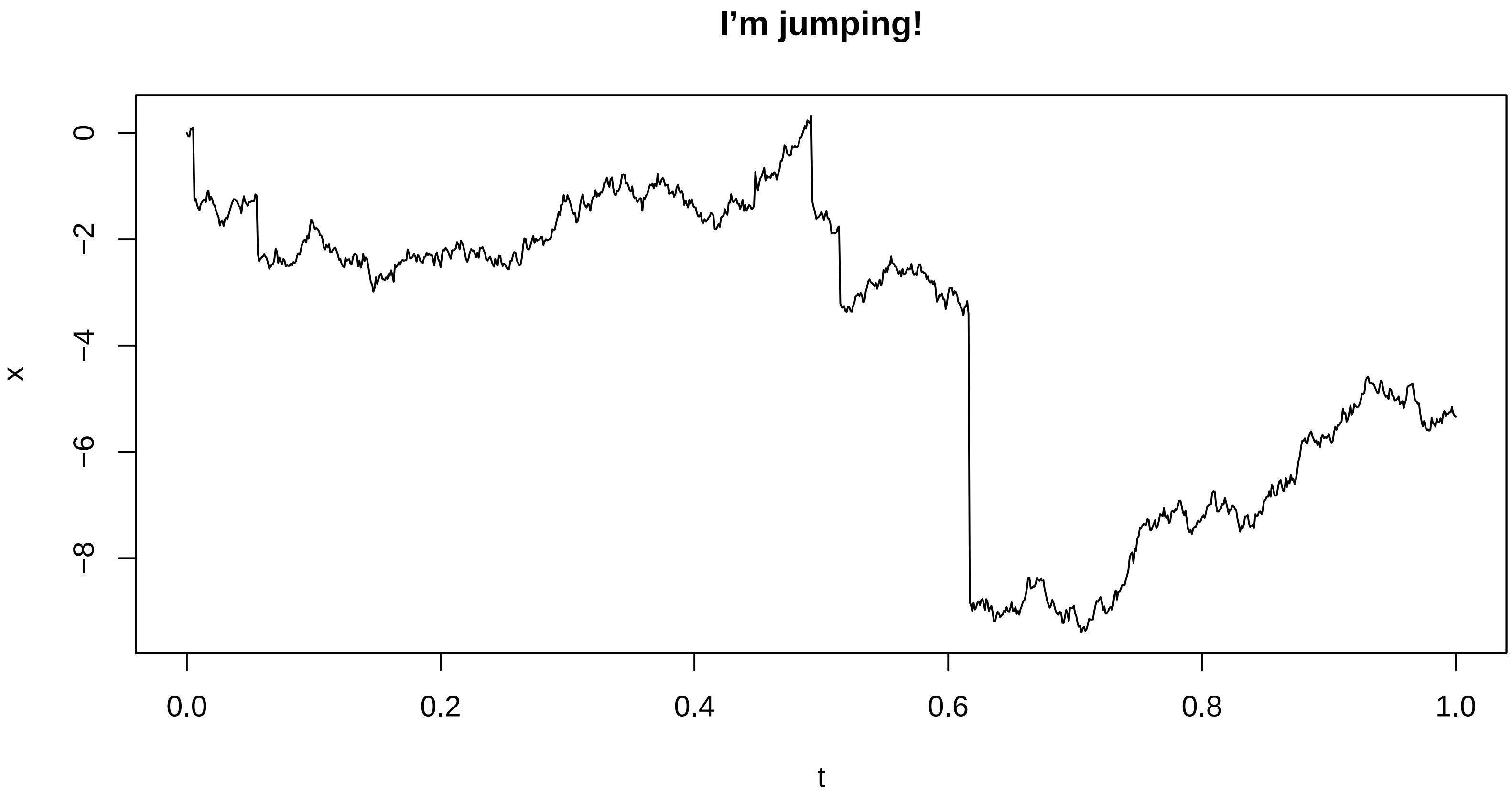
$$\mathrm{d}X_t = a(X_t)dt + b(X_t)\mathrm{d}W_t + \mathrm{d}Z_t$$

Next is an example of Poisson process with intensity $\lambda = 10$ and Gaussian jumps.

In this case we specify `measure.type` as “CP” (Compound Poisson)

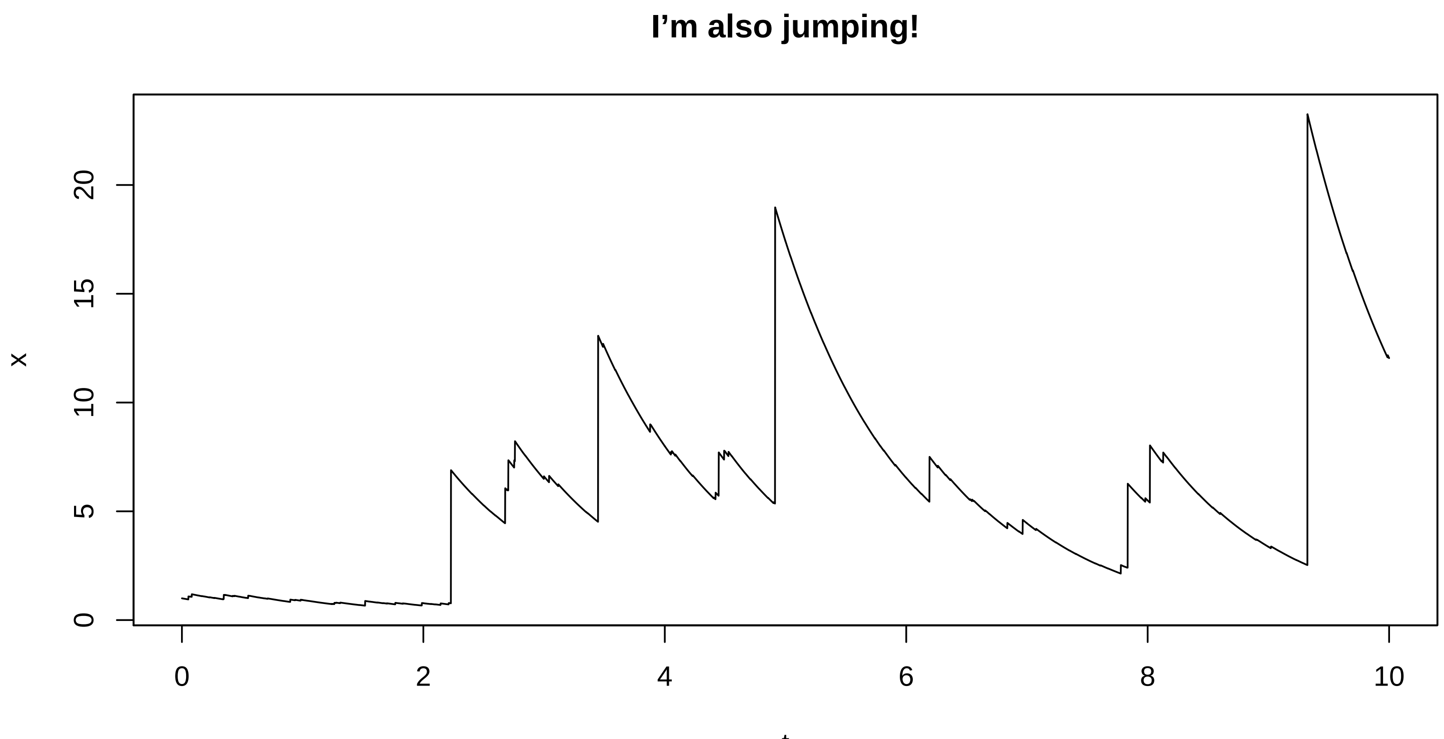
Jump process: $dX_t = -\theta X_t dt + \sigma dW_t + Z_t$


```
> mod5 <- setModel(drift=c("-theta*x"), diffusion="sigma",
 jump.coeff="1", measure=list(intensity="10", df=list("dnorm(z, 0, 1)")),
 measure.type="CP", solve.variable="x")
> set.seed(123)
> X <- simulate(mod5, true.p=list(theta=1,sigma=3),n=1000)
> plot(X, main="I'm jumping!")
```



Another way is to specify the Lévy measure. Without going into too much details, here is an example of a simple OU process with IG Lévy measure $\mathrm{d}X_t = -X_t dt + dZ_t$

```
> mod6 <- setModel(drift="-x", xinit=1, jump.coeff="1",
  measure.type="code", measure=list(df="rIG(z, 1, 0.1)"))
> set.seed(123)
> plot( simulate(mod6, Terminal=10, n=10000), main="I'm also jumping!")
```



The `setModel` method

Models are specified via

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

in

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

The package implements many multivariate RNG to simulate Lévy paths including `rIG`, `rNIG`, `rbgamma`, `rngamma`, `rstable`.

Other user-defined or packages-defined RNG can be used freely.

The `setModel` method

Models are specified via

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

in

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

The package implements many multivariate RNG to simulate Lévy paths including `rIG`, `rNIG`, `rbgamma`, `rngamma`, `rstable`.

Other user-defined or packages-defined RNG can be used freely.

Models are specified via

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

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

The package implements many multivariate RNG to simulate Lévy paths including `rIG`, `rNIG`, `rbgamma`, `rngamma`, `rstable`.

Other user-defined or packages-defined RNG can be used freely.

The `setModel` method

Models are specified via

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

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

The package implements many multivariate RNG to simulate Lévy paths including `rIG`, `rNIG`, `rbgamma`, `rngamma`, `rstable`.

Other user-defined or packages-defined RNG can be used freely.

Models are specified via

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

$$dX_t = a(X_t)dt + b(X_t)dW_t + c(X_t)dZ_t$$

The package implements many multivariate RNG to simulate Lévy paths including `rIG`, `rNIG`, `rbgamma`, `rngamma`, `rstable`.
Other user-defined or packages-defined RNG can be used freely.

A sampling or subsampling structure can be created via the `setSampling` constructor.

This allow to specify regular or irregular multidimensional grids (i.e. each equation has its own grid), possibly a random distribution of times.

A sampling or subsampling structure can be created via the `setSampling` constructor.

This allow to specify regular or irregular multidimensional grids (i.e. each equation has its own grid), possibly a random distribution of times.

The `sampling` slot in Yuima is also used during the inference. For example, one can specify the “`model`”, the “`data`” and then explicit the `sampling` which will contain informations about how these data have been collected. In this case, the tools for inference in Yuima will act differently upon this information.

The `setSampling` method

A sampling or subsampling structure can be created via the `setSampling` constructor.

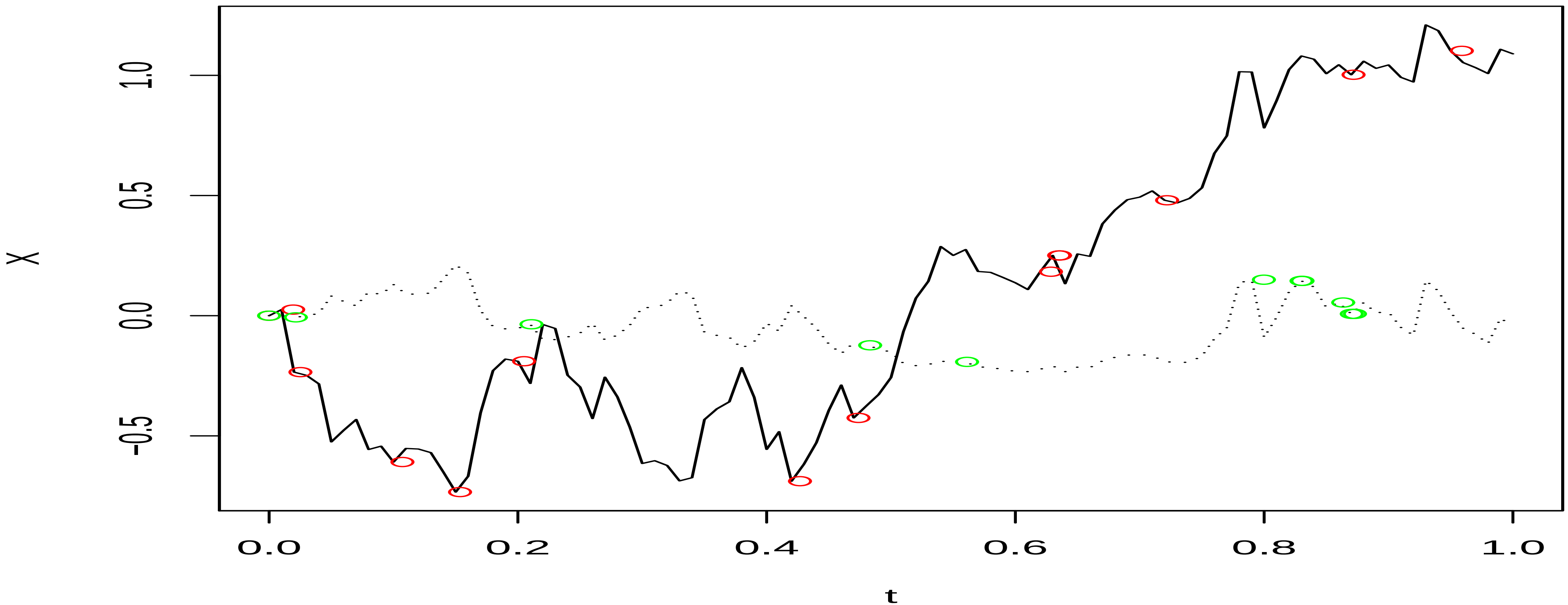
This allow to specify regular or irregular multidimensional grids (i.e. each equation has its own grid), possibly a random distribution of times.

The `sampling` slot in Yuima is also used during the inference. For example, one can specify the "`model`", the "`data`" and then explicit the `sampling` which will contain informations about how these data have been collected. In this case, the tools for inference in Yuima will act differently upon this information.

In simulation studies, one can decide to simulate the processes at high frequency and then resample the simulated data according to different subsampling schemes: random, irregular, space grids, etc and verifty the effect of different subsampling on the estimation or the calibration of financial product.

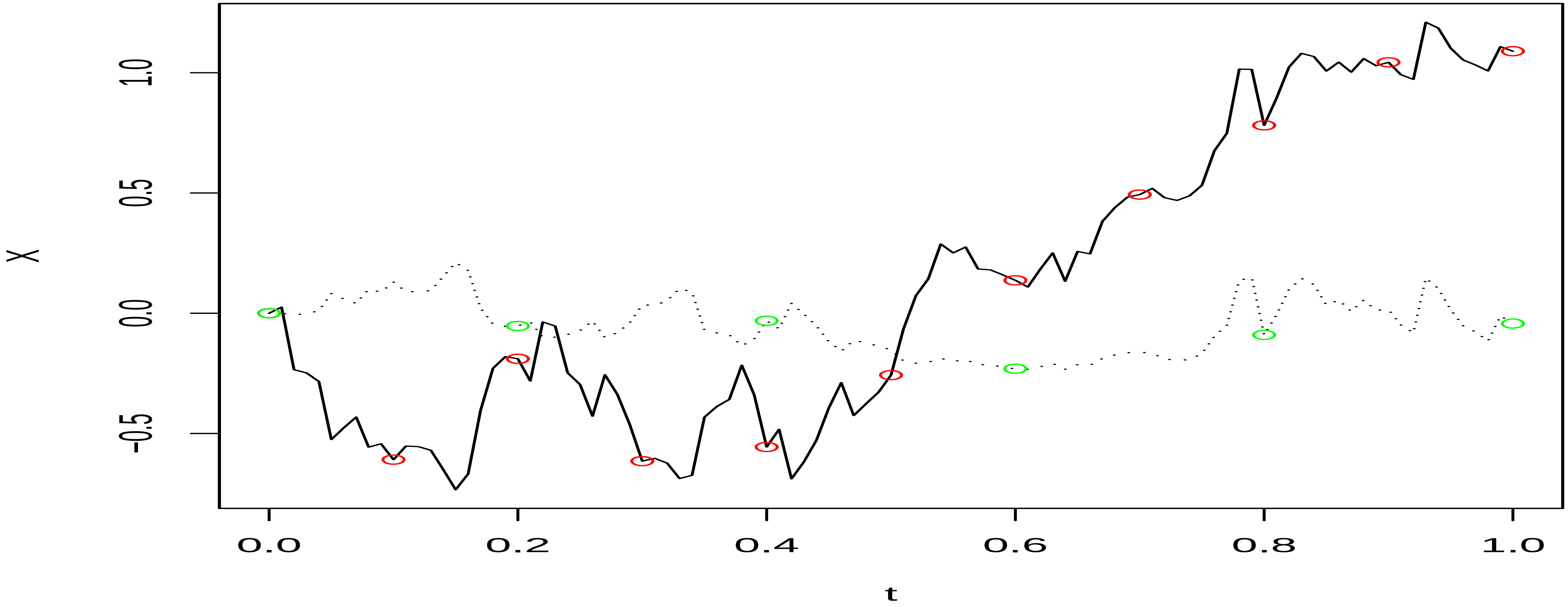
The sampling structure can be used to operate subsampling. Next example shows how to perform Poisson random sampling, with two independent Poisson processes.

```
> newsamp <- setSampling(random = list(rdist = c(function(x) rexp(x, rate = 10),
+     function(x) rexp(x, rate = 20))))
> newdata <- subsampling(X, sampling = newsamp)
> plot(X, plot.type = "single", lty = c(1, 3), ylab = "X")
> points(get.zoo.data(newdata)[[1]], col = "red")
> points(get.zoo.data(newdata)[[2]], col = "green")
```



But you can also do deterministic subsampling

```
> newsamp <- setSampling(delta = c(0.1, 0.2))
> newdata <- subsampling(X, sampling = newsamp)
> plot(X, plot.type = "single", lty = c(1, 3), ylab = "X")
> points(get.zoo.data(newdata)[[1]], col = "red")
> points(get.zoo.data(newdata)[[2]], col = "green")
```



Inference & Finance

Which tools have been developed

- quasi-likelihood estimation for multidimensional diffusions (Yoshida, 1992, 2005)
- Adaptive Bayes type estimators (Yoshida, 2005)
- change point estimation for the volatility in a multidimensional Itô process (Iacus & Yoshida, 2009)
- Asymptotic expansion of functional of diffusion processes (Yoshida, 2005)
- LASSO-type model selection (De Gregorio & Iacus, 2010)
- the covariance estimator of Yoshida-Hayashi (2005) for multidimensional Itô processes with asynchronous data

Just not to be too vague, let us consider the exact fomulations of some of the problems which can be handled by the `yuima` package.

Quasi Maximum Likelihood Analysis

Consider the mutldimensional diffusion process

$$\mathrm{d}X_t = b(\theta_2, X_t)\mathrm{d}t + \sigma(\theta_1, X_t)\mathrm{d}W_t$$

where W_t is an r-dimensional standard Wiener process independent of the initial value $X_0 = x_0$. Quasi-MLE assumes the following approximation of the true log-likelihood for multidimensional diffusions

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

where $\theta = (\theta_1, \theta_2)$, $\Delta X_i = X_{t_i} - X_{t_{i-1}}$, $\Sigma_i(\theta_1) = \Sigma(\theta_1, X_{t_i})$, $b_i(\theta_2) = b(\theta_2, X_{t_i})$, $\Sigma = \sigma^{\otimes 2}$, $A^{\otimes 2} = A^T A$ and A^{-1} the inverse of A. Then the QML estimator of θ is

$$\tilde{\theta}_n = \arg\min_{\theta} \ell_n(\mathbf{X}_n, \theta)$$

QMLE example

To estimate a model we make use of the `qmle` function. Consider the model

$$\mathrm{d}X_t = -\theta_2 X_t \mathrm{d}t + \theta_1 \mathrm{d}W_t$$

with $\theta_1 = 0.3$ and $\theta_2 = 0.1$

```
> diff.matrix <- matrix(c("theta1"), 1, 1)
> ymodel <- setModel(drift = c("(-1)*theta2*x"), diffusion = diff.matrix,
+     time.variable = "t", state.variable = "x", solve.variable = "x")
> n <- 100
> 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 = 0.3, theta2 = 0.1))
```

Now `yuima` contains information about the model and the simulated data.
The true values of the parameters θ_1 and θ_2 were specified for the simulation, but unknown to the `yuima` object.

QMLE example

we can now call `qmle` on the `yuima` object which now contains informations about the model and the data.

```
> mle1 <- qmle(yuima, start = list(theta1 = 0.8, theta2 = 0.7),
+       lower = list(theta1=0.05, theta2=0.05), upper = list(theta1=0.5, theta2=0.5),
+       method = "L-BFGS-B")
> coef(mle1)
    theta1     theta2
0.30766981 0.05007788
> summary(mle1)
Maximum likelihood estimation

Call:
qmle(yuima = yuima, start = list(theta1 = 0.8, theta2 = 0.7),
    method = "L-BFGS-B", lower = list(theta1 = 0.05, theta2 = 0.05),
    upper = list(theta1 = 0.5, theta2 = 0.5))

Coefficients:
         Estimate Std. Error
theta1 0.30766981 0.02629925
theta2 0.05007788 0.15144393

-2 log L: -280.0784
```

Adaptive Bayes Estimation

QMLE example

Consider again the diffusion process solution to

$$\mathrm{d}X_t = b(X_t, \theta_2)\mathrm{d}t + \sigma(X_t, \theta_1)\mathrm{d}W_t, \tag{2}$$

The adaptive Bayes type estimator is defined as follows.

$$\tilde{\theta}_1 = \Big[\int_{\Theta_1} \ell_n(\mathbf{x}_n, (\theta_1, \theta_2^\star))\pi_1(\theta_1)d\theta_1\Big]^{-1} \int_{\Theta_1} \theta_1 \ell_n(\mathbf{x}_n, (\theta_1, \theta_2^\star))\pi_1(\theta_1)d\theta_1 \tag{3}$$

where π_1 is a prior density on Θ_1. For 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

$$\tilde{\theta}_2 = \Big[\int_{\Theta_2} \ell_n(\mathbf{x}_n, (\tilde{\theta}_1, \theta_2))\pi_2(\theta_2)d\theta_2\Big]^{-1} \int_{\Theta_2} \theta_2 \ell_n(\mathbf{x}_n, (\tilde{\theta}_1, \theta_2))\pi_2(\theta_2)d\theta_2 \tag{4}$$

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 $\theta = (\theta_1, \theta_2)$.

ADABayes example

Adaptive Bayes aestimation is developed in `yuima` via the method `adaBayes`. Consider again the model

$$\mathrm{d}X_t = -\theta_2 X_t \mathrm{d}t + \theta_1 \mathrm{d}W_t$$

with $\theta_1 = 0.3$ and $\theta_2 = 0.1$ In order to perform Bayesian estimation, we need to prepare the prior densities for the parameters. For simplicity we use uniform distributions in $[-2, 2]$

```
> prior.theta1 <- function(theta2) 1 * (theta2 > 0 & theta2 < 1)
> prior.theta2 <- function(theta1) 1 * (theta1 > 0 & theta1 < 1)
> prior <- list(theta1 = list(measure.type = "density", density = prior.theta1,
+     domain = c(-2, 2)), theta2 = list(measure.type = "density",
+     density = prior.theta2, domain = c(-2, 2)))
```

The we call `adaBayes` as follows

```
> param.init <- list(theta2 = 0.35, theta1 = 0.52)
> lower = c(0, 0)
> upper = c(1, 1)
> bayes1 <- adaBayes(yuima, start = param.init, lower = lower,
+     upper = upper, prior = prior, method = "nomcmc")

> bayes1
$theta2
[1] 0.1131918
$theta1
[1] 0.2856613
```

and if you compare with joint QMLE estimator

```
> coef(mle1)
    theta1     theta2
0.30766981 0.05007788
```

True values $\theta_1 = 0.3$ and $\theta_2 = 0.1$

Change-point Analysis

Volatility Change-Point Estimation

The theory works for SDEs of the form

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

where W_t a r-dimensional Wiener process and b_t and X_t are multidimensional processes and σ is the diffusion coefficient (volatility) matrix.

When $Y = X$ the problem is a diffusion model.

The process b_t may have jumps but should not explode and it is treated as a nuisance in this model.

The change-point problem for the volatility is formalized as follows

$$Y_t = \begin{cases} Y_0 + \int_0^t b_s ds + \int_0^t \sigma(X_s, \theta_1^*) dW_s & \text{for } t \in [0, \tau^*) \\ Y_{\tau^*} + \int_{\tau^*}^t b_s ds + \int_{\tau^*}^t \sigma(X_s, \theta_2^*) dW_s & \text{for } t \in [\tau^*, T]. \end{cases}$$

The change point τ^* instant is unknown and is to be estimated, along with θ_1^* and θ_2^*, from the observations sampled from the path of (X, Y).

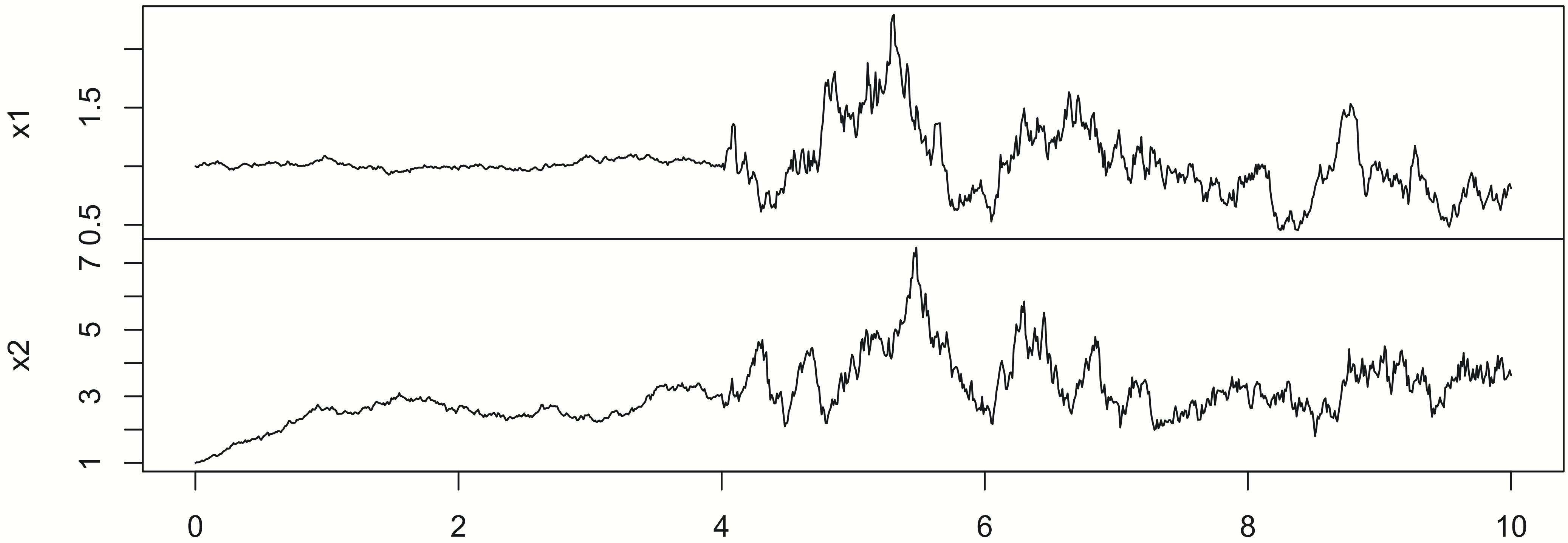
Example of Volatility Change-Point Estimation

Consider the 2-dimensional stochastic differential equation

$$\begin{pmatrix} dX_t^1 \\ dX_t^2 \end{pmatrix} = \begin{pmatrix} 1 - X_t^1 \\ 3 - X_t^2 \end{pmatrix} dt + \begin{bmatrix} \theta_{1.1} \cdot X_t^1 & 0 \cdot X_t^1 \\ 0 \cdot X_t^2 & \theta_{2.2} \cdot X_t^2 \end{bmatrix}' \begin{pmatrix} dW_t^1 \\ dW_t^2 \end{pmatrix}$$

$$X_0^1 = 1.0, \quad X_0^2 = 1.0,$$

with change point instant at time $\tau = 0.4$

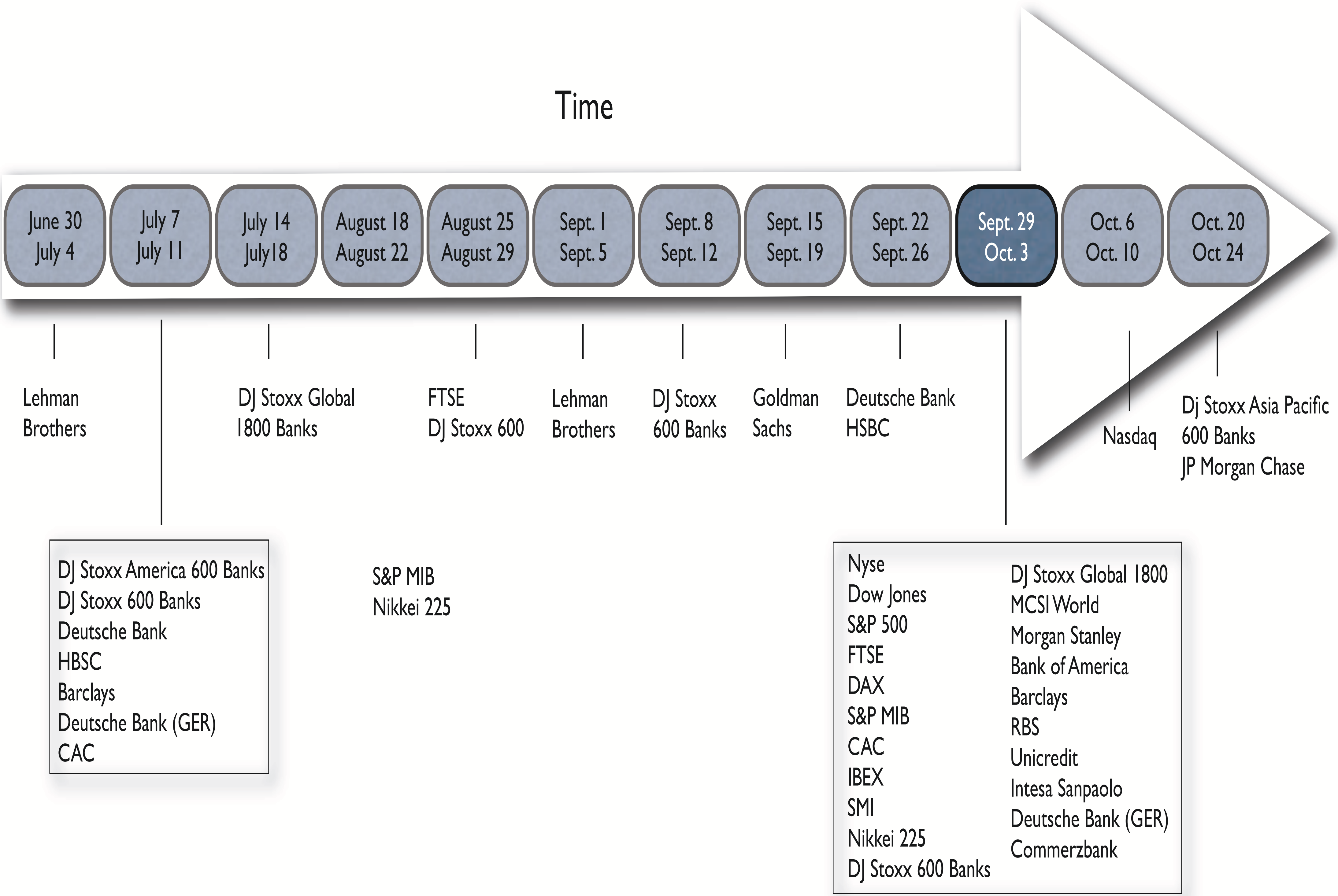


Example of Volatility Change-Point Estimation

the object `yuima` contains the model and the data from the previous slide so we can call `CPoint` on this `yuima` object

```
> t.est <- CPoint(yuima,param1=t1,param2=t2, plot=TRUE)
>
> t.est$tau
[1] 3.99
```

An application to the recent financial crisis showed that...


Time
June 30 July 4
July 7 July 11
July 14 July18
August 18 August 22
August 25 August 29
Sept. 1 Sept. 5
Sept. 8 Sept. 12
Sept. 15 Sept. 19
Sept. 22 Sept. 26
Sept. 29 Oct. 3
Oct. 6 Oct. 10
Oct. 20 Oct 24
Lehman Brothers
DJ Stoxx America 600 Banks
DJ Stoxx 600 Banks
Deutsche Bank
HBSC
Barclays
Deutsche Bank (GER)
CAC
DJ Stoxx Global 1800 Banks
S&P MIB
Nikkei 225
FTSE
DJ Stoxx 600
Lehman Brothers
DJ Stoxx 600 Banks
Goldman Sachs
Deutsche Bank
HSBC
Nyse
Dow Jones
S&P 500
FTSE
DAX
S&P MIB
CAC
IBEX
SMI
Nikkei 225
DJ Stoxx 600 Banks
DJ Stoxx Global 1800
MCSI World
Morgan Stanley
Bank of America
Barclays
RBS
Unicredit
Intesa Sanpaolo
Deutsche Bank (GER)
Commerzbank
Nasdaq
Dj Stoxx Asia Pacific 600 Banks
JP Morgan Chase

Asymptotic Expansion

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

$$\mathrm{d}X_t^\varepsilon = a(X_t^\varepsilon, \varepsilon)\mathrm{d}t + b(X_t^\varepsilon, \varepsilon)\mathrm{d}W_t, \qquad \varepsilon \in (0,1]$$

with W_t and r-dimensional Wiener process, i.e. $W_t = (W_t^1, \ldots, W_t^r)$.

The functional is expressed in the following abstract form

$$F^\varepsilon(X_t^\varepsilon) = \sum_{\alpha=0}^{r} \int_0^T f_\alpha(X_t^\varepsilon, \mathrm{d})\mathrm{d}W_t^\alpha + F(X_t^\varepsilon, \varepsilon), \qquad W_t^0 = t$$

Example: B&S asian call option

$$\mathrm{d}X_t^\varepsilon = \mu X_t^\varepsilon \mathrm{d}t + \varepsilon X_t^\varepsilon \mathrm{d}W_t$$

and the B&S price is related to $\mathbb{E}\left\{\max\left(\frac{1}{T}\int_0^T X_t^\varepsilon \mathrm{d}t - K, 0\right)\right\}$. Thus the functional of interest is

$$F^\varepsilon(X_t^\varepsilon) = \frac{1}{T}\int_0^T X_t^\varepsilon \mathrm{d}t, \qquad r = 1$$

Example: B&S asian call option

$$\mathrm{d}X_t^\varepsilon = \mu X_t^\varepsilon \mathrm{d}t + \varepsilon X_t^\varepsilon \mathrm{d}W_t$$

and the B&S price is related to $\mathbb{E}\left\{\max\left(\frac{1}{T}\int_0^T X_t^\varepsilon \mathrm{d}t - K, 0\right)\right\}$. Thus the functional of interest is

$$F^\varepsilon(X_t^\varepsilon) = \frac{1}{T}\int_0^T X_t^\varepsilon \mathrm{d}t, \qquad r = 1$$

with

$$f_0(x,\varepsilon) = \frac{x}{T}, \quad f_1(x,\varepsilon) = 0, \quad F(x,\varepsilon) = 0$$

in

$$F^\varepsilon(X_t^\varepsilon) = \sum_{\alpha=0}^{r}\int_0^T f_\alpha(X_t^\varepsilon, \mathrm{d})\mathrm{d}W_t^\alpha + F(X_t^\varepsilon, \varepsilon)$$

So, the call option price requires the composition of a smooth functional

$$F^{\varepsilon}(X_t^{\varepsilon}) = \frac{1}{T}\int_0^T X_t^{\varepsilon} \mathrm{d}t, \qquad r = 1$$

with the irregular function

$$\max(x - K, 0)$$

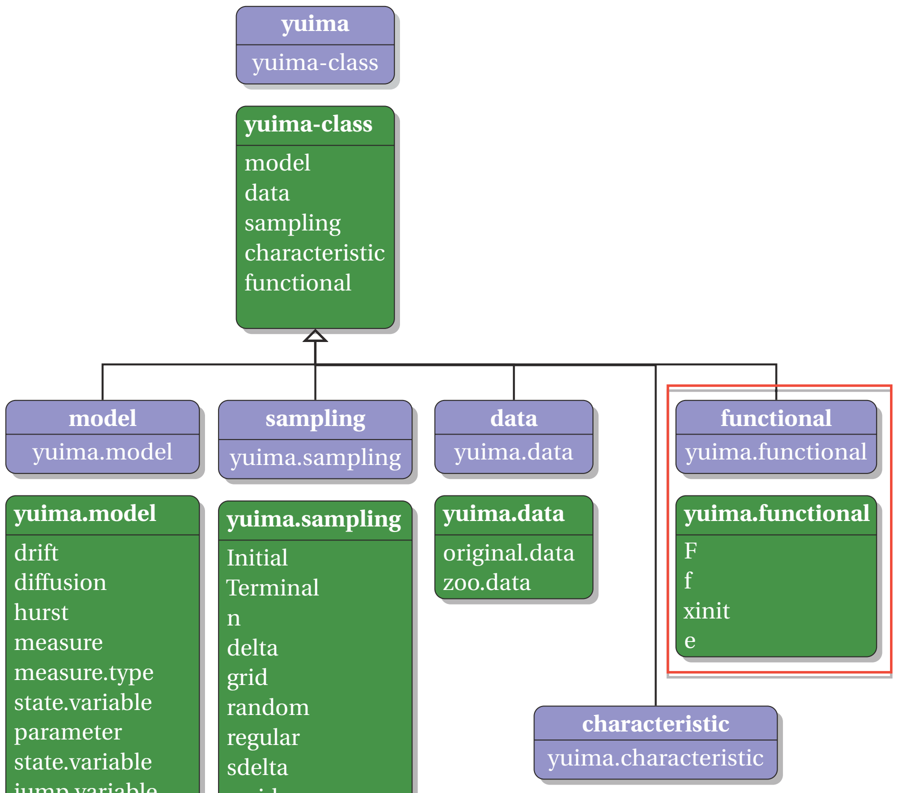
Monte Carlo methods require a HUGE number of simulations to get the desired accuracy of the calculation of the price, while asymptotic expansion of F^{ε} provides unexpectedly accurate approximations.

The `yuima` package provides functions to construct the functional F^{ε}, and automatic asymptotic expansion based on Malliavin calculus starting from a `yuima` object.

```
> diff.matrix <- matrix( c("x*e"), 1,1)
> model <- setModel(drift = c("x"), diffusion = diff.matrix)
> T <- 1
> xinit <- 1
> f <- list( expression(x/T), expression(0))
> F <- 0
> e <- .3
> yuima <- setYuima(model = model, sampling = setSampling(Terminal=T, n=1000))
> yuima <- setFunctional( yuima, f=f,F=F, xinit=xinit,e=e)
```

`setFunctional` method


```
> diff.matrix <- matrix( c("x*e"), 1,1)
> model <- setModel(drift = c("x"), diffusion = diff.matrix)
> T <- 1
> xinit <- 1
> f <- list( expression(x/T), expression(0))
> F <- 0
> e <- .3
> yuima <- setYuima(model = model, sampling = setSampling(Terminal=T, n=1000))
> yuima <- setFunctional( yuima, f=f,F=F, xinit=xinit,e=e)
```



```
> diff.matrix <- matrix( c("x*e"), 1,1)
> model <- setModel(drift = c("x"), diffusion = diff.matrix)
> T <- 1
> xinit <- 1
> f <- list( expression(x/T), expression(0))
> F <- 0
> e <- .3
> yuima <- setYuima(model = model, sampling = setSampling(Terminal=T, n=1000))
> yuima <- setFunctional( yuima, f=f,F=F, xinit=xinit,e=e)
```

the definition of the functional is now included in the yuima object (some output dropped)

```
> str(yuima)
Formal class 'yuima' [package "yuima"] with 5 slots
  ..@ data          :Formal class 'yuima.data' [package "yuima"] with 2 slots
  ..@ model         :Formal class 'yuima.model' [package "yuima"] with 16 slots
  ..@ sampling      :Formal class 'yuima.sampling' [package "yuima"] with 11 slots
  ..@ functional    :Formal class 'yuima.functional' [package "yuima"] with 4 slots
  .. .. ..@ F    : num 0
  .. .. ..@ f    :List of 2
  .. .. .. ..$ :  expression(x/T)
  .. .. .. ..$ :  expression(0)
  .. .. ..@ xinit: num 1
  .. .. ..@ e    : num 0.3
```

Estimation of functionals. Example.

Then, it is as easy as

```
> F0 <- F0(yuima)
> F0
[1] 1.716424
> max(F0-K,0)  # asian call option price
[1] 0.7164237
```

Estimation of functionals. Example.

Then, it is as easy as

```
> F0 <- F0(yuima)
> F0
[1] 1.716424
> max(F0-K,0)  # asian call option price
[1] 0.7164237
```

and back to asymptotic expansion, the following script may work

```
> rho <- expression(0)
> get_ge <- function(x,epsilon,K,F0){
+   tmp <- (F0 - K) + (epsilon * x)
+   tmp[(epsilon * x) < (K-F0)] <- 0
+   return( tmp )
+ }
> K <- 1 # strike
> epsilon <- e  # noise level
> g <- function(x) {
+   tmp <- (F0 - K) + (epsilon * x)
+   tmp[(epsilon * x) < (K-F0)] <- 0
+   tmp
+ }
```

The expansion of previous functional gives

```
> asymp <- asymptotic_term(yuima, block=10, rho, g)
calculating d0 ...done
calculating d1 term ...done
> asymp$d0 + e * asymp$d1  # asymp. exp. of asian call price
[1] 0.7148786

> library(fExoticOptions) # From RMetrics suite
> TurnbullWakemanAsianApproxOption("c", S = 1, SA = 1, X = 1,
+        Time = 1, time = 1, tau = 0.0, r = 0, b = 1, sigma = e)
Option Price:
[1] 0.7184944

> LevyAsianApproxOption("c", S = 1, SA = 1, X = 1,
+       Time = 1, time = 1, r = 0, b = 1, sigma = e)
Option Price:

[1] 0.7184944
> X <- sde.sim(drift=expression(x), sigma=expression(e*x), N=1000,M=1000)
> mean(colMeans((X-K)*(X-K>0)))  # MC asian call price based on M=1000 repl.
[1] 0.707046
```

Multivariate Asymptotic Expansion

Asymptotic expansion is now also available for multidimensional diffusion processes like the Heston model

$$\begin{aligned} \mathrm{d}X_t^{1,\varepsilon} &= aX_t^{1,\varepsilon}\mathrm{d}t + \varepsilon X_t^{1,\varepsilon}\sqrt{X_t^{2,\varepsilon}}\mathrm{d}W_t^1 \\ \mathrm{d}X_t^{2,\varepsilon} &= c(d - X_t^{2,\varepsilon})\mathrm{d}t + \varepsilon\sqrt{X_t^{2,\varepsilon}}\left(\rho\mathrm{d}W_t^1 + \sqrt{1-\rho^2}\mathrm{d}W_t^2\right) \end{aligned}$$

i.e. functionals of the form $F(X^{1,\varepsilon}, X^{2,\varepsilon})$.

Asynchronous covariance estimation

Consider a two-dimensional Itô process (X^1, X^2) satisfying the stochastic differential equations

$$\begin{aligned} \mathrm{d}X_t^i &= \mu_t^i \mathrm{d}t + \sigma_t^i \mathrm{d}W_t^i, \quad t \in [0,T] \\ X_0^i &= x_0^i \end{aligned}$$

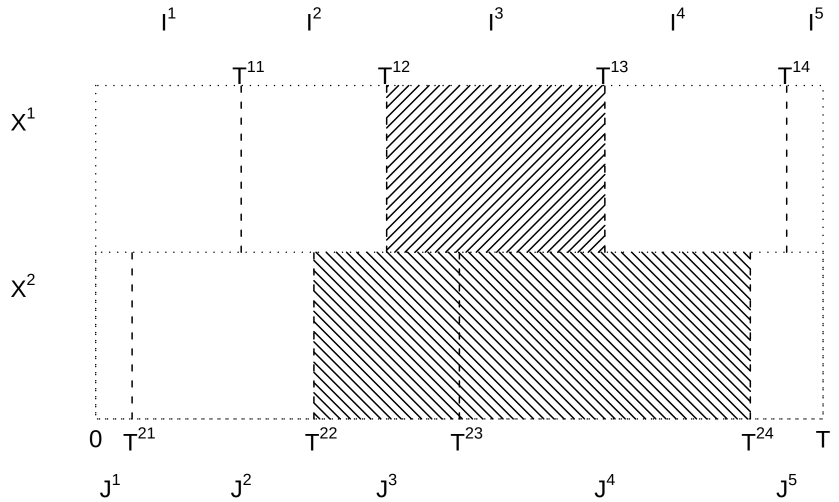
for $i = 1, 2$. Here W^i denote standard Wiener processes with a progressively measurable correlation process $\mathrm{d}\langle W_1, W_2 \rangle_t = \rho_t \mathrm{d}t$, μ_t^i and σ_t^i are progressively measurable processes, and x_0^i are initial random variables independent of (W^1, W^2).

We are interested in

$$\theta = \langle X^1, X^2 \rangle_T = \int_0^T \sigma_t^1 \sigma_t^2 \rho_t \mathrm{d}t. \tag{5}$$

from discrete asynchronous observations

$$U_n = \sum_{i,j: T^{1i} \le T, T^{2j} \le T} (X^1_{T^{1i}} - X^1_{T^{1\{i-1\}}})(X^2_{T^{2j}} - X^2_{T^{2\{j-1\}}}) \mathbf{1}_{\{I^i \bigcap J^j \neq \emptyset\}}. \tag{6}$$



Two sequences of stopping times T^{1k} and T^{2j} are *asynchronous* times of observations from the two processes X^i_t. I^k and J^j are the intervals determined respectively by subsequent elements of the sequences of random times T^{1k} and T^{2j}.

Asynchronous data: example

Consider a two-dimensional stochastic process (X_t^1, X_t^2) satisfying

$$\begin{aligned} dX_t^1 &= \sigma_{1,t} dB_t^1, \\ dX_t^2 &= \sigma_{2,t} dB_t^2. \end{aligned} \tag{7}$$

Here B_t^1 and B_t^2 denote two standard Wiener processes; however we take them correlated in the following way:

$$B_t^1 = W_t^1, \tag{8}$$

$$B_t^2 = \int_0^t \rho_s dW_s^1 + \int_0^t \sqrt{1-\rho_s^2} dW_s^2, \tag{9}$$

where W_t^1 and W_t^2 are independent Wiener processes, and ρ_t is the correlation function between B_t^1 and B_t^2. We consider $\sigma_{i,t}, i = 1, 2$ and ρ_t of the following form in this example:

$$\sigma_{1,t} = \sqrt{1+t}, \quad \sigma_{2,t} = \sqrt{1+t^2}, \quad \rho_t = \frac{1}{\sqrt{2}}.$$

The parameter we want to estimate is the quadratic covariation between X^1 and X^2:

$$\theta = \langle X_1, X_2 \rangle_T = \int_0^T \sigma_{1,t}\sigma_{2,t}\rho_t \mathrm{d}t = 1. \tag{10}$$

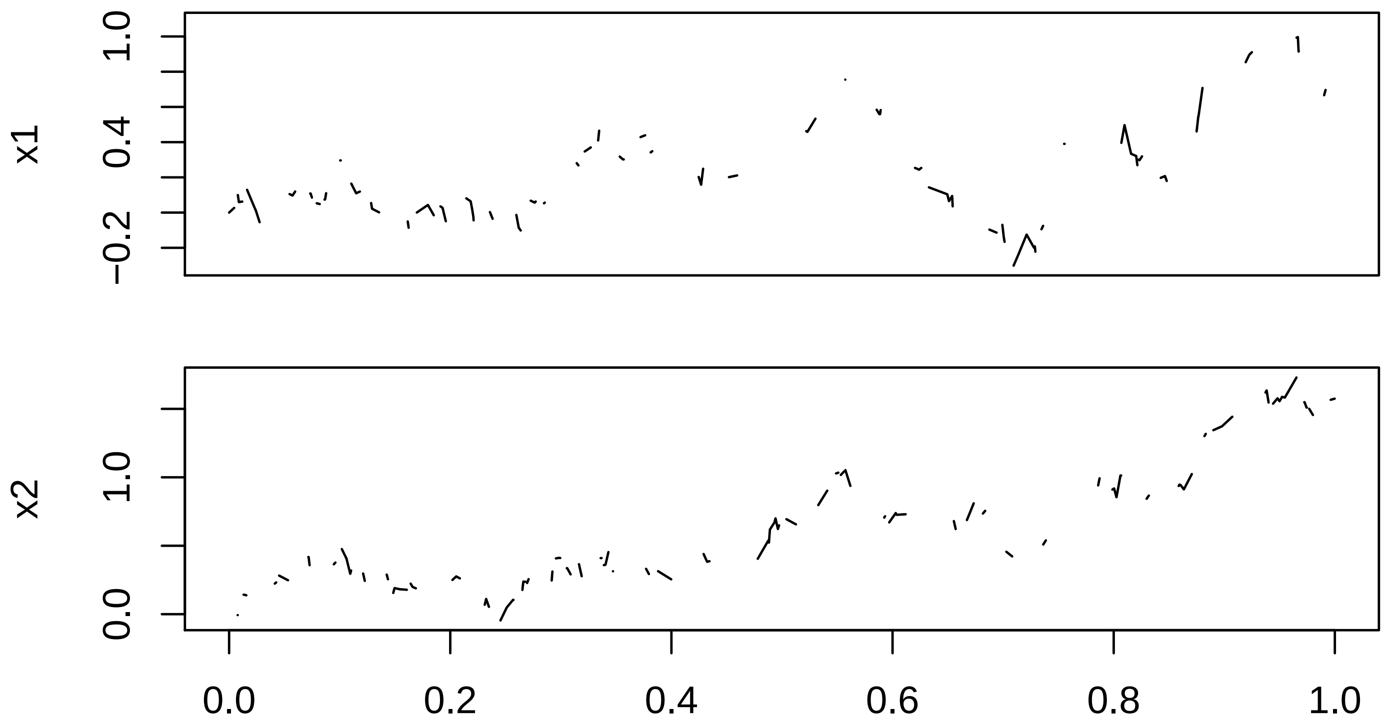
```
> diff1 <- function(t,x1=0, x2=0) sqrt(1+t)
> diff2 <- function(t,x1=0, x2=0) sqrt(1+t^2)
> rho <- function(t,x1=0, x2=0) sqrt(1/2)
> diff.matrix <- matrix( c( "diff1(t,x1,x2)",
+ "diff2(t,x1,x2) * rho(t,x1,x2)", "",
+ "diff2(t,x1,x2) * sqrt(1-rho(t,x1,x2)^2)"),2,2)
> cor.mod <- setModel(drift = c("",""),  diffusion = diff.matrix,
+ solve.variable=c("x1","x2"))
```

we generate the data

```
> Terminal <- 1
> n <- 1000
> yuima.samp <- setSampling(Terminal=Terminal,n=n)
> yuima <- setYuima(model=cor.mod, sampling=yuima.samp)
> set.seed(123)
> X <- simulate(yuima)
```

And now we subsample using Poisson random sampling

```
> newsamp <- setSampling(
 random=list(rdist=c( function(x) rexp(x, rate=0.2*n/Terminal),
  function(x) rexp(x, rate=0.3*n/Terminal))) )
> Y <- subsampling(X, sampling=newsamp)
> cce(X)$covmat[1,2]
[1] 1.086078
> cce(Y)$covmat[1,2]
[1] 1.070313
```



LASSO estimation & model selection

LASSO estimation

LASSO is nothing but estimation under constraints on the parameters. Usually studied for the least squares estimation method, can be applied here using the QMLE approach for the following diffusion model

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

where $\alpha \in R^p$, $\beta \in R^q$, $p, q \geq 1$
The target function is the minimization of $H_n(\alpha, \beta)$ = minus the $\log$ of the approximated likelihood,

$$\min_{\alpha,\beta} H_n(\alpha, \beta) + \sum_{j=1}^{p} \lambda_{n,j}|\alpha_j| + \sum_{k=1}^{q} \gamma_{n,k}|\beta_k|$$

Lasso tries to set the maximal number of parameters to 0. In this sense operates model selection jointly with estimation.

Interest rates LASSO estimation examples

LASSO estimation of the U.S. Interest Rates monthly data from 06/1964 to 12/1989. These data have been analyzed by many author including Nowman (1997), Aït-Sahalia (1996), Yu and Phillips (2001) and it is a nice application of LASSO.

Reference	Model	α	β	γ
Merton (1973)	$dX_t = \alpha dt + \sigma dW_t$		0	0
Vasicek (1977)	$dX_t = (\alpha + \beta X_t)dt + \sigma dW_t$			0
Cox, Ingersoll and Ross (1985)	$dX_t = (\alpha + \beta X_t)dt + \sigma\sqrt{X_t}dW_t$			$1/2$
Dothan (1978)	$dX_t = \sigma X_t dW_t$	0	0	1
Geometric Brownian Motion	$dX_t = \beta X_t dt + \sigma X_t dW_t$	0		1
Brennan and Schwartz (1980)	$dX_t = (\alpha + \beta X_t)dt + \sigma X_t dW_t$			1
Cox, Ingersoll and Ross (1980)	$dX_t = \sigma X_t^{3/2} dW_t$	0	0	$3/2$
Constant Elasticity Variance	$dX_t = \beta X_t dt + \sigma X_t^{\gamma} dW_t$	0		
CKLS (1992)	$dX_t = (\alpha + \beta X_t)dt + \sigma X_t^{\gamma} dW_t$			

Interest rates LASSO estimation examples

Model	Estimation Method	α	β	σ	γ
Vasicek	MLE	4.1889	-0.6072	0.8096	–
CKLS	Nowman	2.4272	-0.3277	0.1741	1.3610
CKLS	Exact Gaussian	2.0069	-0.3330	0.1741	1.3610
	(Yu & Phillips)	(0.5216)	(0.0677)		
CKLS	QMLE	2.0822	-0.2756	0.1322	1.4392
		(0.9635)	(0.1895)	(0.0253)	(0.1018)
CKLS	QMLE + LASSO	1.5435	-0.1687	0.1306	1.4452
	with mild penalization	(0.6813)	(0.1340)	(0.0179)	(0.0720)
CKLS	QMLE + LASSO	0.5412	0.0001	0.1178	1.4944
	with strong penalization	(0.2076)	(0.0054)	(0.0179)	(0.0720)

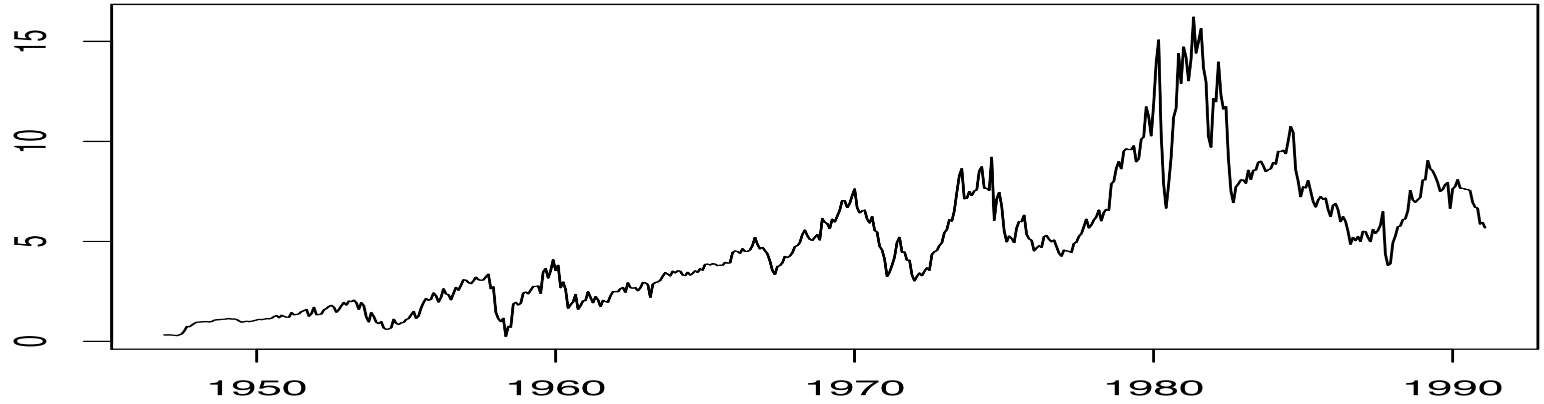
LASSO selected: Cox, Ingersoll and Ross (1980) model

$$\mathrm{d}X_t = \frac{1}{2}\mathrm{d}t + 0.12 \cdot X_t^{3/2}\mathrm{d}W_t$$

An example of Lasso use on real data with CKLS model

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

```
> library(Ecdat)
> data(Irates)
> rates <- Irates[,"r1"]
> plot(rates)
> require(yuima)
> X <- window(rates, start=1964.471, end=1989.333)
> mod <- setModel(drift="alpha+beta*x", diffusion=matrix("sigma*x^gamma",1,1))
> yuima <- setYuima(data=setData(X), model=mod)
```



Adaptive sequences: $\lambda_n = \lambda_0/\tilde{\theta}_n$; $\tilde{\theta}_n$ = QMLE.


```
> lambda0 <- list(alpha=10, beta =10, sigma =10, gamma =10)
> 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)
> lasso10 <- lasso(yuima, lambda0, start=start, lower=low, upper=upp,
   method="L-BFGS-B")

Looking for MLE estimates...
Performing LASSO estimation...

> round(lasso10$mle, 3)  # QMLE
 sigma  gamma  alpha   beta
 0.133  1.443  2.076 -0.263

> round(lasso10$lasso, 3)  # LASSO
sigma gamma alpha  beta
0.117 1.503 0.591 0.000
```

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

$$dX_t = 0.6dt + 0.12 X_t^{\frac{3}{2}} dW_t$$

For more informations and software see

http://R-Forge.R-Project.org/projects/yuima

In the near future also

- filtering
- estimation for fractional OU process
- quasi likelihood analysis for some classes of Lévy models
- Graphical User Interface for option pricing